USERS

DESARROLLE UNA RED SOCIAL COMPLETA

SISTEMAS WEB ESCALABLES

DESARROLLO DE SITIOS QUE SE ADAPTAN A DEMANDAS CRECIENTES

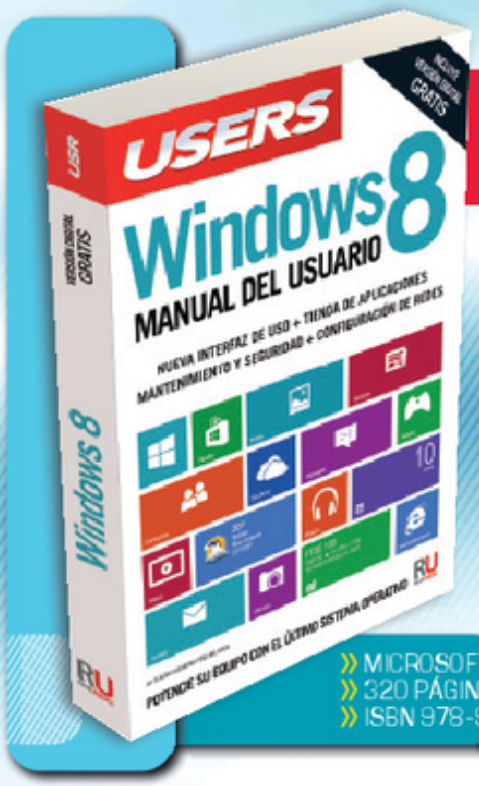
- INTEGRACIÓN PHP + REDIS
- PUBLICACIÓN DE MÓDULOS CON NODE.JS
- BASES DE DATOS NOSQL
- IMPLEMENTACIÓN DE UN CHAT Y DE UN SISTEMA DE NOTIFICACIÓN
- CREACIÓN DE UN SERVIDOR WEB

por CARLOS ALBERTO BENITEZ

MEJORE EL RENDIMIENTO SIN REDISEÑAR LA ARQUITECTURA

RU RedUSERS

CONÉCTESE CON LOS MEJORES LIBROS DE COMPUTACIÓN



POTENCIE SU EQUIPO CON EL ÚLTIMO SISTEMA OPERATIVO

» MICROSOFT / WINDOWS
» 320 PÁGINAS
» ISBN 978-987-1949-09-0

APRENDA A DESARROLLAR INCREÍBLES JUEGOS 3D

» DESARROLLO
» 320 PÁGINAS
» ISBN 978-987-1857-81-4

MONTE SU PROPIO NEGOCIO DE REPARACIÓN DE EQUIPOS

» HARDWARE
» 320 PÁGINAS
» ISBN 978-987-1949-02-1

DESCUBRA CÓMO DESARROLLAR UNA ESTRATEGIA BASADA EN MEDIOS SOCIALES

» EMPRESAS / INTERNET
» 192 PÁGINAS
» ISBN 978-987-1857-62-3

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

MÁS INFORMACIÓN / CONTÁCTENOS

usershop.redusers.com +54 (011) 4110-8700 usershop@redusers.com

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

SISTEMAS WEB ESCALABLES

DESARROLLO DE SITIOS QUE SE ADAPTAN A DEMANDAS CRECIENTES

por Carlos Alberto Benitez

RedUSERS

TÍTULO:	Sistemas web escalables
AUTOR:	Carlos Benitez
COLECCIÓN:	Manuales USERS
FORMATO:	24 x 17 cm
PÁGINAS:	320

ISBN 978-987-1949-20-5

Benítez, Carlos Alberto

Sistemas web escalables. - 1a ed. - Buenos Aires : Fox Andina; Dalaga, 2013.

320 p. ; 24x17 cm. - (Manual users; 255)

ISBN 978-987-1949-20-5

1. Informática. I. Título

CDD 005.3

VISITE NUESTRA WEB

EN NUESTRO SITIO PODRÁ ACCEDER A UNA PREVIEW DIGITAL DE CADA LIBRO Y TAMBIÉN OBTENER, DE MANERA GRATUITA, UN CAPÍTULO EN VERSIÓN PDF, EL SUMARIO COMPLETO E IMÁGENES AMPLIADAS DE TAPA Y CONTRATAPA.

RedUSERS
COMUNIDAD DE TECNOLOGIA

redusers.com

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios y todos los elementos necesarios para asegurar un aprendizaje exitoso.

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com usershop@redusers.com + 54 (011) 4110-8700

Carlos Benitez

Analista de Sistemas recibido en la Universidad Nacional de Misiones en 2008, trabaja con tecnologías web desde el 2005. Ha orientado su perfil a realizar desarrollos con PHP, JS, Ajax, CSS, HTML5, jQuery, MySQL, Linux y servidores HTTP, SMTP y FTP, entre otros.

Es autodidacta y entusiasta de la investigación e implementación de nuevas tecnologías, como sistemas de tiempo real, bases de datos NoSQL y técnicas de escalabilidad horizontal.

Ha dictado varios seminarios de desarrollo de sistemas web, así como también ha participado de todo tipo de cursos que añadieron valor a su conocimiento y experiencia.

Actualmente, trabaja como UNIX Developer en una compañía de marketing digital US/ARG. Paralelamente, se encuentra desarrollando un sistema SaaS de gestión de tareas para equipos, con características como eventos en tiempo real y conexión a APIs de terceros.

Agradecimientos

Este libro está dedicado, principalmente, a Eve y Tomás, mis compañeros en el día a día y quienes apoyaron desde el primer momento este proyecto realizado.

También agradezco a mi familia y amigos, por el tiempo y la ayuda brindada.

Prólogo

La redacción de esta obra me entregó la posibilidad de investigar y profundizar mis conocimientos sobre las tecnologías que se están implementando en el desarrollo de sistemas, donde el principal protagonista es el usuario. Hoy en día, es necesario crear sistemas aptos para evolucionar en el tiempo, que a la vez puedan soportar alto tráfico.

Tener en cuenta la capacidad que tendrá nuestro sistema para escalar nos hace pensar en alternativas antes de desarrollar una aplicación; por ejemplo, qué tipo de base de datos necesitaremos y a qué le daremos más importancia: a la cantidad, a la calidad de los datos o a mecanismos que informen en tiempo real al usuario sobre una situación.

Como desarrollador de sistemas, me he encontrado en situaciones donde tuve que optar entre varias tecnologías para resolver un problema (por ejemplo, entre una base de datos relacional, una base de datos NoSQL o una combinación de ambas) pero, como veremos más adelante, ninguna tecnología es mejor que otra sino que, simplemente, algunas se ajustan mejor a la necesidad a resolver.

Estamos acostumbrados a interactuar con otros usuarios e incluso con el sistema mismo en tiempo real, pero antes esta situación era diferente, ya que el circuito de la comunicación era un ida y vuelta entre el cliente, que solicitaba algo al servidor, y la devolución que éste le hacía en algún formato.

Gracias a Node.js podemos desarrollar sistemas escalables de una manera muy rápida y, debido a que Node utiliza la herramienta NPM para gestionar módulos, podemos instalar y crear los propios casi sin esfuerzo.

Finalmente, otra característica importante que ofrece Node es la posibilidad de utilizar frameworks como Express. Gracias a Express, es posible crear sistemas independientes sin necesidad de contar con un servidor web externo.

Carlos Alberto Benitez
cabenitez83@gmail.com
Twitter: @betustwit

El libro de un vistazo

En la obra conoceremos de qué se tratan los sistemas web escalables. Haremos una introducción a los sistemas de base de datos NoSQL que nos servirá de punto de partida para aprender a trabajar con Redis y PHP. Luego haremos una breve explicación acerca de los sistemas orientados a eventos, donde estará enmarcado Node.js, y para finalizar crearemos una red social utilizando las herramientas mencionadas en el libro.

*01 INTRODUCCIÓN A LOS SISTEMAS ESCALABLES

En este capítulo aprenderemos los tipos de escalabilidad y los aspectos que debemos tener en cuenta para diseñar un sistema que escale. Además, conoceremos algunas de las bases de datos NoSQL más importantes, como MongoDB, Cassandra y Redis.

*02 REDIS

Haremos una introducción al motor de base de datos Redis, cuándo es conveniente usarlo y cómo instalarlo. Luego veremos los tipos de datos que define y los comandos que podemos utilizar, tanto para interactuar con los datos como para administrar el servidor.

*03 CLIENTES REDIS

Aquí conoceremos dos de los clientes de Redis disponibles para PHP, y nos enfocaremos en phpredis. Veremos cómo instalarlo y estudiaremos una herramienta de administración de la base de datos.

*04 ADMINISTRACIÓN DE REDIS

En esta sección aprenderemos cómo configurar la replicación y los tipos de persistencia de Redis y veremos las opciones de seguridad que podemos implementar en esta base de datos. Luego conoceremos una herramienta muy útil, que nos servirá para evaluar el rendimiento en diferentes condiciones.

*05 PROGRAMACIÓN ORIENTADA A EVENTOS

Este capítulo nos mostrará los diferentes paradigmas del desarrollo de sistemas, para terminar haciendo foco en la programación orientada a eventos, que nos servirá de introducción para entender lo que veremos en el siguiente capítulo.

*06 NODE.JS

Aquí conoceremos Node.js, las soluciones que ofrece y sus características. Veremos cuándo utilizar esta tecnología y cuándo es conveniente optar por otra, junto con las ventajas y empresas que la utilizan.

*07 MANEJO DE PETICIONES EN NODE.JS

Inicialmente estructuraremos el código escrito en Node. Luego, convertiremos nuestra aplicación en un servidor web y aprenderemos cómo transformarlo en un módulo. Al final del capítulo hablaremos acerca de ruteo y manejador de peticiones.

*08 GESTIÓN DE MÓDULOS EN NODE.JS

Conoceremos algunos de los módulos de Node.js y sabremos cómo estructurar y crear módulos propios. También veremos de qué se trata el gestor de paquetes conocido como NPM, cómo instalarlo y cuáles son sus comandos más importantes.

*09 MÓDULOS MÁS IMPORTANTES DE NODE

Nos adentraremos en algunos de los módulos más importantes de Node.js, como el framework MVC Express, Socket.IO (para manejar eventos de tiempo real), Nodemailer (para envío de correos electrónicos), node_redis y Nodemon.

*Ap ON WEB DESARROLLO DE UNA RED SOCIAL

Finalmente, en esta parte de la obra aplicaremos los conceptos de desarrollo aprendidos en los capítulos anteriores. El objetivo será desarrollar una red social, un tipo de sistema que refleja todas las características que deben tener los sistemas escalables.

INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual, podrá encontrar una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados. Para que pueda distinguirlos en forma más sencilla, cada recuadro está identificado con diferentes iconos:


CURIOSIDADES E IDEAS


ATENCIÓN


DATOS ÚTILES Y NOVEDADES


SITIOS WEB

RedUSERS
COMUNIDAD DE TECNOLOGÍA

La red de productos sobre tecnología más importante del mundo de habla hispana

Libros
Desarrollos temáticos en profundidad

Coleccionables
Cursos intensivos con gran despliegue visual

Revistas
Las últimas tecnologías explicadas por expertos

RedUSERS
redusers.com
Noticias actualizadas minuto a minuto, reviews, entrevistas y trucos

Newsletters
Regístrese en redusers.com para recibir un resumen con las últimas noticias

RedUSERS PREMIUM
premium.redusers.com
Nuestros productos en versión digital, con contenido adicional y a precios increíbles

Usershop
usershop.redusers.com
Revistas, libros y fascículos a un clic de distancia y con entregas a todo el mundo

Contenido

Sobre el autor 4
Prólogo 5
El libro de un vistazo 6
Información complementaria 7
Introducción 12

*01 Introducción a los sistemas escalables

Sistemas escalables 14
Escalabilidad vertical 18
Escalabilidad horizontal 20
¿Cómo diseñar un sistema escalable? 22
Aspectos fundamentales para el diseño 23
Descentralización 25
Bases de datos NoSQL 29
Cuándo utilizar NoSQL 32
Tipos de bases de datos NoSQL 33
Resumen 41
Actividades 42

*02 Redis

Introducción 44
Cuándo usar Redis 46
Instalación 47
Tipos de datos 55
Strings 56
Lists 56
Sets 57
Sorted sets 58
Hashes 60
Comandos 61
Publicación y suscripción 68
Formato de los mensajes 69
Suscripciones mediante un patrón de coincidencia 71
Suscripciones coincidentes con canales y con patrones 73
Resumen 73
Actividades 74

*03 Clientes Redis

Clientes PHP 76
Predis 76
phpredis 77
phpRedisAdmin 82
Ejercicio de prueba con PHP 86
Creación del archivo index.html 90
Creación del archivo style.css 97
Creación del archivo script.js 103
Creación del archivo funciones.php 112
Resumen 123
Actividades 124

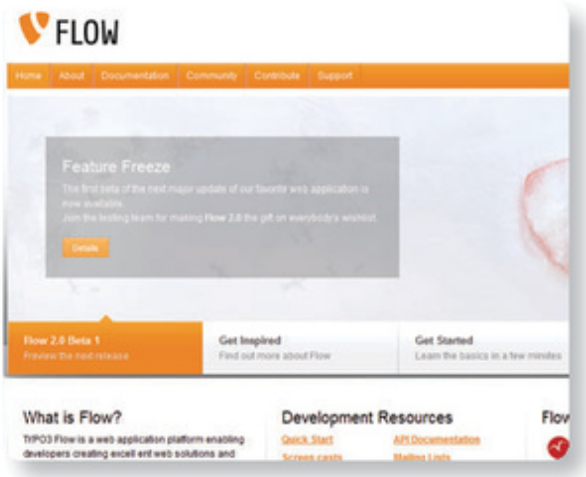
*04 Administración de Redis

Configuración de Redis 126
Pasaje de parámetros por la línea de comandos 126
Reconfigurar Redis en tiempo de ejecución 127
Configurar Redis como caché 128
Replicación 129
Funcionamiento de la replicación de Redis 130
Configuración de un esclavo 130
Configuración de un esclavo de solo lectura 131
Configuración de autenticación de esclavo a maestro 132
Tipos de persistencia 132

Persistencia mediante RDB....133
Persistencia mediante AOF....134
Recomendación de persistencia....136
Habilitar AOF mientras se utiliza RDB....136
Seguridad....139
Seguridad de acceso....139
Autenticación....140
Deshabilitar comandos....140
Rendimiento de Redis (benchmarks)....141
Ejemplo de prueba: Mini-twt....142
Resumen....169
Actividades....170

*05
Programación orientada a eventos

Paradigmas en el desarrollo de sistemas....172
Programación orientada a objetos....172
Programación imperativa....173
Programación declarativa....174
Programación estructurada....175
Programación orientada a aspectos....176
Programación orientada a eventos....178
Resumen....179
Actividades....180



*06
Node.js

Node.js....182
Soluciones....183
Características....183
Funcionamiento....184
Diferencias entre Node y Apache....185
Ajax versus Comet....186
Comparación de rendimiento....187
Cuándo usar Node....188
Cuándo no usar Node....189
Ventajas del uso de Node....189
Empresas que utilizan Node....190
Instalación de Node en Linux, Mac y Windows....195
Instalación en Linux (distribución Ubuntu)....195
Instalación en Mac....196
Instalación en Windows....197
Primeros pasos con Node....202
Resumen....207
Actividades....208

*07
Manejo de peticiones en Node

Estructuración de código....210
Servidor web....210
Convertir el servidor en un módulo....213
Ruteo....215
Manejadores....219
Registro de accesos....229
Servidor de documentos HTML....231
Ruteo por defecto....234
Cambio de la codificación....235
Manejo de peticiones POST....238
Resumen....243
Actividades....244

*08 Gestión de módulos en Node

Módulos **246**
Módulos propios de Node 247
Módulo File 247
Incluir módulos desde el directorio node_modules 248
Cacheo de módulos 248
Estructuración y creación de módulos **249**
Gestor de paquetes npm **258**
Comandos de npm 258
README.md 265
package.json 267
Resumen **273**
Actividades **274**

*09 Módulos más importantes de Node

Frameworks MVC **276**
Express **277**
Instalación 278
Creación de una aplicación 279
Instalación de dependencias 280
Prueba de la aplicación 281
El archivo app.js 283
Ruteo 284
Socket.IO **291**
Nodemailer **298**
Ejemplo de prueba 298
node_redis **302**
Redis Commander **305**
Nodemon **306**
Resumen **309**
Actividades **310**

* Servicios al lector

Índice temático **312**

*Ap on web Desarrollo de una red social

El porqué del proyecto
Definición del proyecto
Instalación de herramientas previas
Creación del proyecto
Configuración del archivo app.js
Definición de la apariencia del sistema
Registro y login de usuarios
Creación de la página principal
Definición de los eventos para un usuario logueado
Envío de una solicitud de amistad
Respuesta a una solicitud de amistad
Lista de amigos conectados
Informar cuando se conecta un usuario
Informar cuando se desconecta un usuario
Creación del sistema de chat
Creación del sistema de posts
Vista de la base de datos
Resumen
Actividades

Introducción

Son cada vez más las compañías que desarrollan productos digitales con la idea de poder escalar a medida que pasa el tiempo. Esta tarea no es fácil ya que, inicialmente, es necesario saber en qué consiste la escalabilidad vertical y horizontal. Un sistema escalable debe ser capaz de tolerar fallos y debe estar descentralizado, de manera que la caída de un nodo no signifique la pérdida total del sistema.

El aumento de la accesibilidad a Internet ha provocado la necesidad de implementar sistemas de persistencia óptimos y que ofrezcan una mayor flexibilidad de almacenamiento. Inicialmente, fueron las compañías más importantes las que optaron por desarrollar bases de datos no relacionales, ya que eran las que contaban con infraestructura y capacidad económica para hacerlo. Pero hoy, gracias a proyectos bajo licencias libres como MongoDB y Redis, cualquier persona es capaz de desarrollar un sistema con bases de datos NoSQL.

Cuando hablamos de bases de datos por lo general solo pensamos en cómo persistir los datos y no dónde hacerlo. Redis es una alternativa muy interesante, ya que opera en memoria, por lo que brinda una performance realmente muy buena para lectura y escritura de datos. Ofrece varios clientes que permiten interactuar con los lenguajes de programación más populares, entre ellos, PHP y Node.js.

Debido a la gran popularidad de PHP en el mundo de los sistemas web, es una alternativa a tener en cuenta en la integración con las bases de datos NoSQL. Como veremos a lo largo del libro, Redis dispone de un cliente para operar de manera nativa con el intérprete de PHP.

Las redes sociales han evolucionado de una manera muy significativa, generando en los usuarios la necesidad de información actualizada, en todo momento y desde cualquier lugar. Node.js ofrece un entorno independiente, capaz de responder en tiempo real a las acciones del usuario.

*01

Introducción a los sistemas escalables

Conoceremos los aspectos fundamentales del diseño de sistemas web de gran tráfico, a los que llamaremos sistemas escalables. Veremos las técnicas de desarrollo más adecuadas y diferenciaremos dos tipos de escalabilidad. Por último, nos encargaremos de realizar una breve introducción a algunas bases de datos NoSQL.

- **Sistemas escalables** 14
- **Escalabilidad vertical** 18
- **Escalabilidad horizontal** 20
- **¿Cómo diseñar un sistema escalable?** 22
  - Aspectos fundamentales para el diseño 23
- **Descentralización** 25
- **Bases de datos NoSQL** 29
  - Cuándo utilizar NoSQL 32
  - Tipos de bases de datos NoSQL 33
- **Resumen** 41
- **Actividades** 42

Sistemas escalables

Hace unos años, con la aparición de Ajax, muchos desarrolladores pensábamos que quedaba poco por innovar en el mundo de los sistemas web. Sin embargo, este surgimiento dio lugar a nuevos desafíos a la hora de satisfacer las necesidades de los usuarios durante la implementación. Cada vez es más importante desarrollar sistemas web perdurables en el tiempo y con la capacidad de soportar mayor demanda sin perder rendimiento. Uno de los grandes retos es planificar una arquitectura adecuada, que no sea propensa a fallas o a congestiones en el servidor, para mantener un rendimiento óptimo.

Podemos decir que el desarrollo web es un proceso continuo, que no termina cuando está publicado el sistema en cuestión, sino que se extiende a través de la implementación de nuevas funcionalidades, optimizaciones y gestión de contenidos. A partir del desarrollo posterior a la primera implementación, los sistemas web deberían adaptarse a la demanda y al tráfico, que se encuentran en constante cambio.

> GOOGLE ANALYTICS ES UNA DE LAS OPCIONES DE ANÁLISIS EXISTENTES

Gracias a las herramientas de análisis actuales, como la famosa **Google Analytics**, es posible conocer cómo se comporta un sistema web y cómo debe ir ajustando su rendimiento dependiendo del tráfico que recibe, el nivel de aceptación de los usuarios, el tipo de contenido de mayor impacto y los patrones de navegación.

Una buena planificación en la construcción de un sistema web es fundamental para el futuro a largo plazo. El análisis y la comprensión del funcionamiento de grandes sitios pueden dar lugar a decisiones más inteligentes sobre cómo debemos crear los nuestros.
A continuación, veremos los principios que influyen en el diseño de sistemas web de gran escala:

- **Disponibilidad**: el tiempo de funcionamiento de un sistema web es algo primordial para la reputación y la funcionalidad de muchas empresas. Tanto es así que, para algunos de los sitios más importantes de venta en línea, el hecho de no estar disponibles

por unos pocos minutos puede significar una pérdida de miles en ingresos, por lo que el diseño es muy importante para que los sistemas estén siempre en uso.

- **Rendimiento**: el rendimiento es un factor cada vez más importante, ya que la velocidad de navegación y la facilidad de uso son determinantes para que un usuario decida recurrir a nuestro sistema. Por otro lado, los resultados en los motores de búsqueda se relacionan directamente con los ingresos de las empresas. Por lo tanto, es importante saber que un sistema web deberá dar respuesta a los usuarios tan rápido como esto sea posible.
- **Manejabilidad**: es necesario tener en cuenta que el diseño de un sistema fácil de manejar es una consideración determinante, ya que la capacidad de administración es equivalente a la capacidad de ampliación de las operaciones de mantenimiento y actualización. Cuestiones que inciden en la capacidad de administración son la facilidad de diagnóstico y la comprensión de los problemas para poder hacer cambios o modificaciones.
- **Costo**: el costo es un factor importante pero no determinante. Incluye tanto el hardware como el software, además de otros aspectos para implementar y mantener los sistemas, como el tiempo de desarrollo y el que se necesita para operarlos.

> TANTO EL RENDIMIENTO COMO LA MANEJABILIDAD SON CLAVES AL DISEÑAR UN SISTEMA

De todo lo anterior, obtenemos el concepto de **escalabilidad**, que es la habilidad que tiene un sistema web para procesar mayor información sin perder rendimiento, principalmente en los tiempos de acceso a la

RED USERS PREMIUM

Para obtener material adicional gratuito, ingrese a la sección **Publicaciones/Libros** dentro de **http://premium.redusers.com**. Allí encontrará todos nuestros títulos y podrá acceder a contenido extra de cada uno, como sitios web relacionados, programas recomendados, ejemplos utilizados por el autor, apéndices y archivos editables o de código fuente. Todo esto ayudará a comprender mejor los conceptos desarrollados en la obra.

> MÁS QUE UN BUEN DESARROLLO, LO IMPORTANTE ES TENER UN SISTEMA ESCALABLE

información. Es un error muy frecuente dejar de lado este aspecto, ya que nuestro sistema puede llegar a tener un éxito comparable a los diferentes servicios que ofrece Google. Por lo tanto, es más importante tener una adecuada capacidad de escalar que disponer de un desarrollo excesivamente bueno, porque estaríamos limitando el crecimiento del sitio.

Antes de continuar, vamos a ver un ejemplo de escalabilidad real. **Facebook** nos muestra sus números para entender la magnitud de información que maneja.

Figura 1. La escalabilidad de Facebook para la gran demanda de información que maneja. Detalles en: **http://infographiclabs.com/infographic/facebook-2012.**

FOURSQUARE, UN GIGANTE NO TAN CONOCIDO

Aunque no sea tan popular como Facebook o Twitter, **Foursquare**, un servicio para marcar lugares específicos que está basado en localización web aplicada a las redes sociales, no se queda atrás a la hora de escalar. Al mes de abril de 2012 ya contaba con más de 20 millones de usuarios. Podemos conocerlo mejor en la dirección web **https://foursquare.com**.

Por otra parte, tenemos a otro gigante como **Twitter**, que también es un exponente importante de escalabilidad. A simple vista, parece un sistema sencillo, pero cuando lo investigamos podemos ver que es muy complejo y que requiere de mucho trabajo para soportar el gran volumen de información que crece exponencialmente.

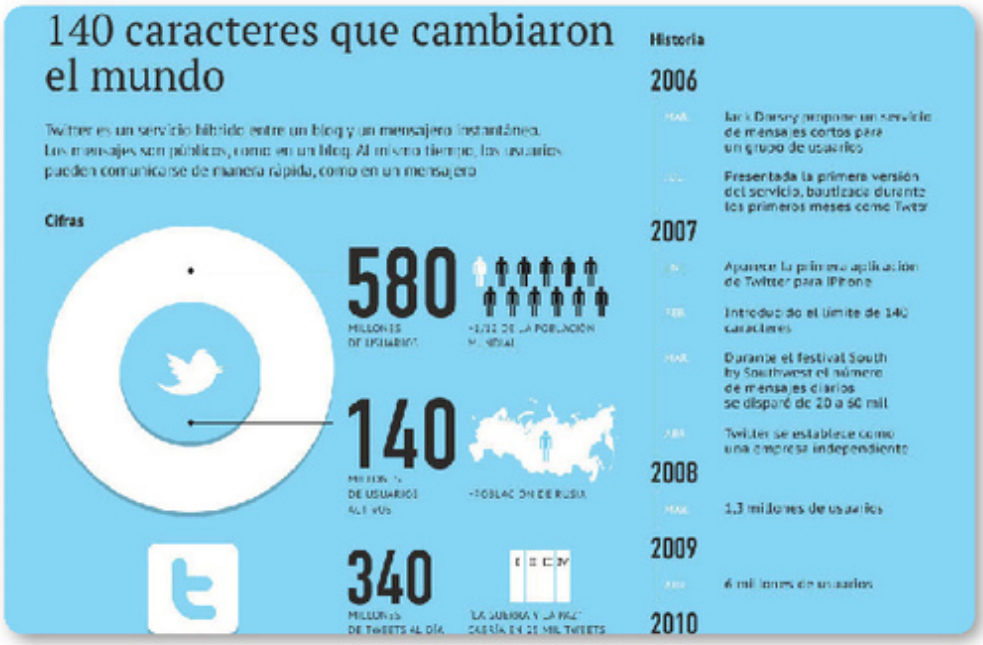


Figura 2. En esta imagen podemos ver un ejemplo de la complejidad del diseño de una red social.

La escalabilidad de un sistema requiere de un análisis cuidadoso desde el principio del desarrollo, ya que de otra manera, a medida que éste evolucione, irán apareciendo problemas que serán difíciles de solucionar. Para esto, debemos definir las estrategias de escalabilidad en el marco tecnológico que nos permitan satisfacer dos aspectos importantes: la **demanda de los usuarios** y el **almacenamiento de contenido**. El factor de crecimiento del contenido es la capacidad que tendrá nuestro sistema de almacenar la información de los usuarios y de dar soporte a las grandes necesidades en determinado tiempo. Consideremos que el crecimiento de la demanda es la capacidad que debe afrontar el sistema para poder mantener la calidad del servicio frente a un incremento del acceso de usuarios.

En resumen, tengamos en cuenta que para que un sistema sea escalable, tiene que cumplir estas tres condiciones:

- Adaptarse al aumento de la demanda de usuarios.
- Adaptarse al incremento de la información que maneja.
- Posibilitar su mantenimiento.

DEBEMOS CONSIDERAR LA EXISTENCIA DE DIFERENTES TIPOS DE ESCALABILIDAD

En este punto, nos encontramos ante la situación que más confusión genera cuando hablamos de escalabilidad, porque no es lo mismo la escalabilidad que el rendimiento del sistema.

Consideremos que el rendimiento se presenta como la velocidad en que se procesan las solicitudes provenientes desde los usuarios, por lo tanto en los sistemas que son escalables es fácil mejorar el rendimiento. Sin embargo, en los sistemas que no lo son, no se puede tener un rendimiento óptimo. A continuación, nos encargaremos de describir los diferentes tipos de escalabilidad que debemos tener en cuenta en nuestros sistemas.

Escalabilidad vertical

La escalabilidad vertical es la estrategia más común y es utilizada de manera frecuente para escalar sistemas. Consiste en actualizar el hardware actual por uno más potente y costoso, ya sea agregando procesadores o reemplazando los existentes por otros más rápidos, como adicionando memoria o simplemente migrando el sistema a un servidor más potente. Es decir, si el hardware actual puede atender 1000 peticiones por segundo, cuando estemos cerca de este valor podemos reemplazarlo por uno de mayor potencia y quizás hasta

LEY DE MOORE

La escalabilidad vertical cuenta con la ayuda de la **Ley de Moore** que dice que para un precio fijo la máquina que podemos comprar será cada año más potente. Por lo tanto, simplemente se puede esperar y, cuando veamos que es necesario, reemplazar el servidor al mismo precio del servidor actual.

duplicar el rendimiento de nuestro sistema. Pero la escalabilidad vertical tiene un límite, ya que alcanzado un punto simplemente no hay máquina más potente que comprar y el precio suele aumentar exponencialmente junto con la potencia.

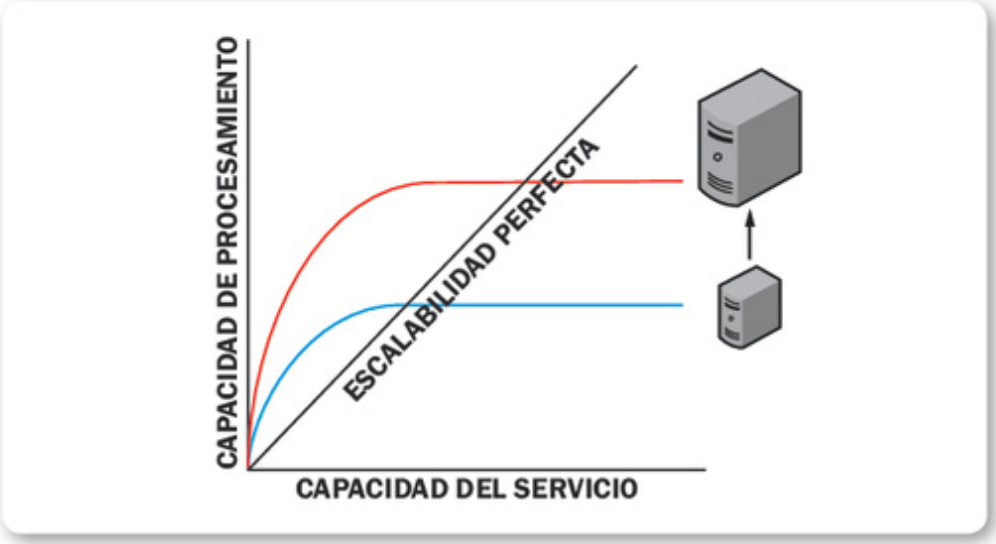


Figura 3. El crecimiento no lineal del rendimiento se irá estabilizando hasta un punto donde ya no será posible escalar el sistema.

Es necesario tener en cuenta que la escalabilidad vertical tiene la desventaja de presentar un alto costo pero, por otro lado, una de sus ventajas en la sencillez de implementación, lo que da como resultado que no debamos hacer ningún cambio en el desarrollo del sistema sino, simplemente, invertir más dinero para realizar la compra de una máquina más grande. Una vez que hayamos actualizado todos los componentes del equipo al máximo posible, llegaremos al límite real de la capacidad de procesamiento, y es ahí desde donde debemos escalar verticalmente, reemplazando el equipo completo por uno más potente.

> UNA DE LAS DESVENTAJAS DE LA ESCALABILIDAD VERTICAL ES SU ALTO COSTO

Aunque parezca una solución muy básica, efectivamente es utilizada, por ejemplo, para realizar la escalabilidad en servidores de bases de datos. Pero para sistemas de gran tamaño no es la mejor opción, por lo que empresas referentes de la Web, como Google y Facebook, optan por escalar horizontalmente.

Escalabilidad horizontal

La escalabilidad horizontal consiste en distribuir la carga de procesamiento, es decir, nos permite aumentar el número de servidores pero no necesariamente su potencia. Aunque el escalado horizontal se logra utilizando varios servidores, el conjunto opera como un único equipo que, al dedicarse a una tarea en común, logra maximizar la tolerancia a fallas; al mismo tiempo, representa un desafío mayor al administrador del sistema.

LA ESCALABILIDAD HORIZONTAL USA TÉCNICAS DE GESTIÓN DE BALANCEO DE CARGA

En este tipo de escalabilidad se suelen utilizar una gran variedad de técnicas en la gestión del balanceo de carga que necesitan los sistemas para escalar horizontalmente. En el caso de los sistemas que funcionan en los clústeres de servidores, para lograr una mayor capacidad simplemente se agregan más equipos, provocando que la información se duplique, lo que genera una redundancia de datos que a la vez brinda una mayor capacidad de recuperación de errores.

Con esta técnica, el servicio está siempre disponible por más que uno o varios servidores hayan fallado, o en caso de que se necesite retirar algunos por tareas de mantenimiento. De este modo, el escalado horizontal proporciona un mecanismo sin limitaciones en cuestión de hardware, ya que cada servidor adicional proporciona una mejora casi lineal en la escalabilidad.

El aspecto más importante para que un sistema sea escalable es la transparencia de la ubicación. Esto se conoce como **afinidad de ubicación** y requiere cambios en el código para poder escalar un

MEJORES PRÁCTICAS PARA ESCRIBIR CÓDIGO

Ya sea para proyectos simples o sistemas complejos con códigos **PHP**, **JavaScript**, **Ajax** o **MySQL**, debemos darle real importancia a la optimización y a las buenas prácticas. Podemos ver algunos consejos interesantes en inglés en el sitio que encontramos en la siguiente dirección: **http://net.tutsplus.com/tutorials/html-css-techniques/top-15-best-practices-for-writing-super-readable-code**.

sistema horizontal de un servidor a varios, demandando un costo de refactorización del código fuente.

El mejor ejemplo de este paradigma es la arquitectura de Google, que posee una plataforma compuesta por cientos de miles de máquinas domésticas interconectadas, donde constantemente sucede que decenas de máquinas dejan de funcionar por problemas de hardware. Pero esto no representa un problema, ya que el sistema redirige de manera automática el tráfico a los nodos que siguen activos, de manera que los usuarios nunca experimenten algún cambio en el funcionamiento. Para aumentar la capacidad solamente es preciso comprar más máquinas y ponerlas a trabajar.

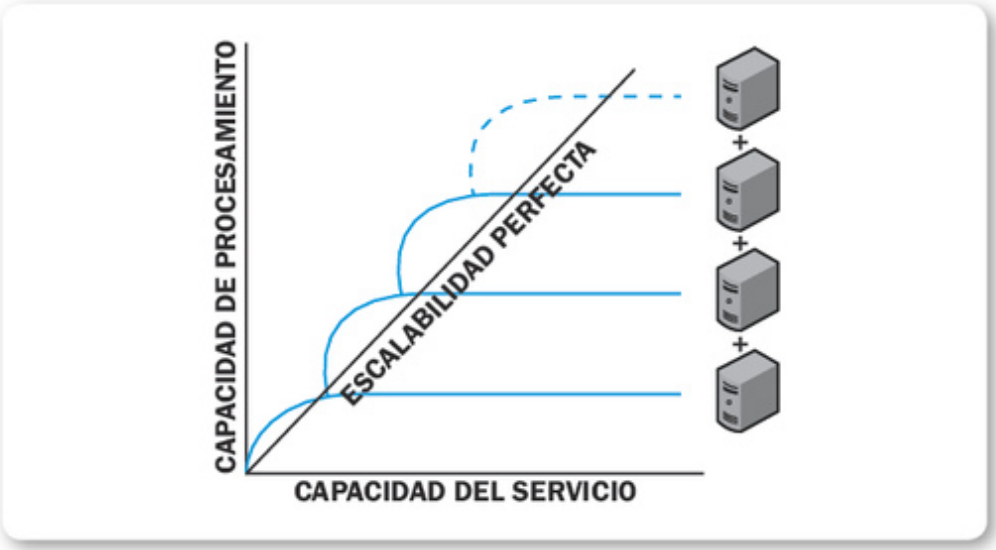


Figura 4. En la escalabilidad horizontal, a medida que se agregan equipos, el sistema escalará indefinidamente.

Ahora que sabemos cómo funciona Google, podemos decir que la escalabilidad horizontal no tiene límites. Pero, para que el sistema escale indefinidamente, es necesario definir un diseño adecuado.

ESCALABILIDAD DE GOOGLE

MapReduce es un framework de desarrollo de aplicaciones paralelas, que simplifica el trabajo con los enormes volúmenes de datos que manejan sus servicios. Podemos ver la definición completa y descargar la versión en pdf desde: **http://research.google.com/archive/mapreduce.html**.

¿Cómo diseñar un sistema escalable?

Para que un sistema sea escalable lo más importante es diseñarlo correctamente. Si pasamos esto por alto, en cualquier otro momento del modelo de desarrollo tendremos un problema muy costoso de resolver. Veamos las consideraciones que debemos tener en cuenta.

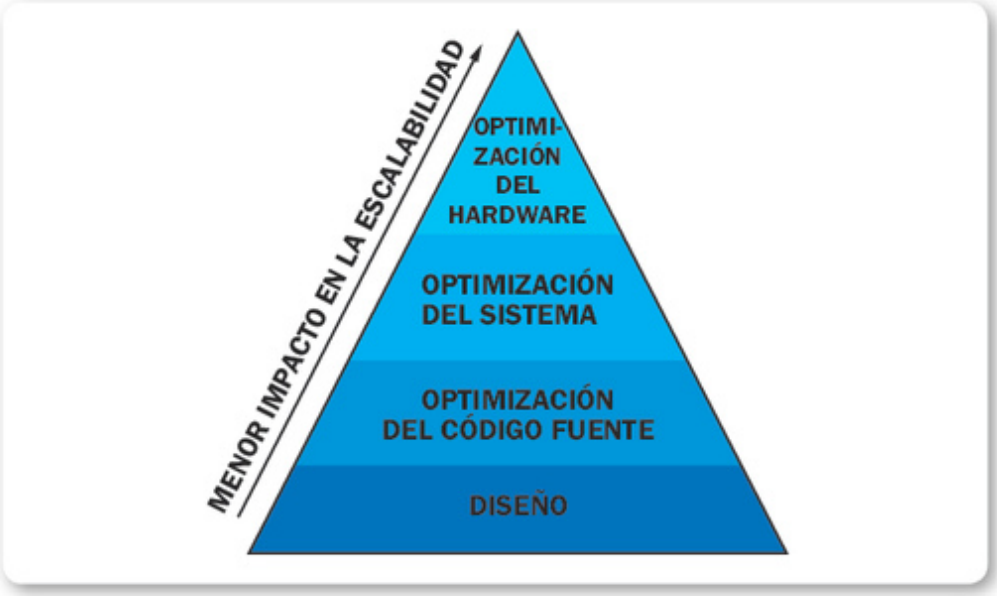


Figura 5. La pirámide de escalabilidad muestra las tareas más importantes para lograr que un sistema sea escalable.

Como vemos en la pirámide de la **Figura 5**, la base de un sistema debe ser el diseño, ya que es lo que tiene mayor influencia en la escalabilidad. A medida que nos desplazamos hacia la cima, nos encontramos con los factores de menor importancia, que son los que causan menos impacto en el desarrollo del sistema escalable. En otras palabras, un diseño bien pensado puede brindar mayor escalabilidad que cualquier hardware que se pueda agregar.

El aspecto fundamental cuando diseñamos sistemas escalables debe ser el poder garantizar una administración eficaz de los recursos. Como el diseño no debe estar limitado a ningún componente o etapa del sistema tenemos que considerar la escalabilidad en todos los niveles, desde la interfaz de usuario hasta el almacenamiento de los datos.

Aspectos fundamentales para el diseño

Es importante tomar las decisiones adecuadas a la hora de diseñar un sistema, pero para garantizar un buen resultado debemos conocer los factores a tener en cuenta. A continuación, detallaremos cuatro aspectos fundamentales que son la base del desarrollo de los sistemas escalables.

Tiempos de espera adecuados

Debemos considerar que todos los procesos deben tener un lapso de espera razonable, ya que el período durante el que un proceso utiliza un recurso es tiempo de espera para otro proceso por ese mismo recurso. Para resolver esto, podemos clasificar los procesos en sincrónicos y asincrónicos. Una alternativa para lograr escalabilidad en los sistemas es realizar procesos de manera asincrónica, donde las acciones de larga duración se almacenan para terminar más adelante en un proceso separado.

> UN TIEMPO DE ESPERA ADECUADO ES ESENCIAL PARA EL DESARROLLO DE UN SISTEMA ESCALABLE

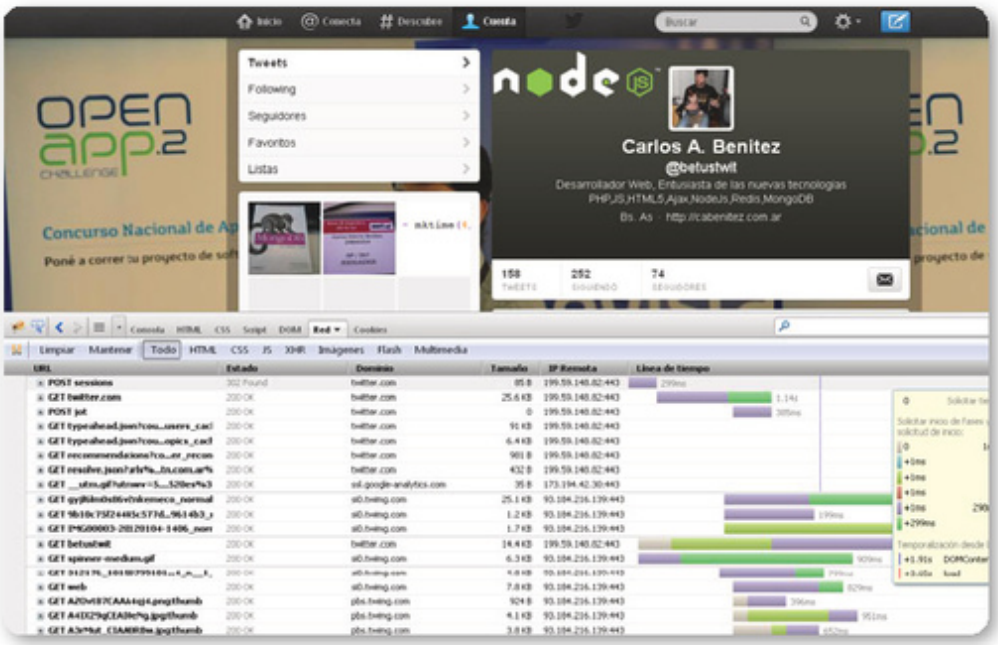


Figura 6. `Firebug` es un complemento de Firefox que permite saber el tiempo que demora cada solicitud de un sitio web.

Procesos que no bloquean

Independientemente del diseño, todos los sistemas poseen una cantidad determinada de recursos que son distribuidos a los procesos según nuestra capacidad para priorizar. Es una buena práctica usar en menor medida los procesos que bloquean recursos, como la memoria, el ancho de banda, el procesador o las conexiones a las bases de datos.

Sistemas conmutables

Se trata del aspecto que menos se tiene en cuenta al diseñar y desarrollar un sistema escalable. Se dice que un sistema es conmutable si dos o más procesos se pueden aplicar en cualquier orden y aún así obtener el mismo resultado. Por lo general, los sistemas que no ejecutan transacciones son conmutables.

Sistemas intercambiables

Este aspecto significa que debemos desplazar el estado de cualquier método fuera de los componentes que utiliza. Por ejemplo, cuando se realizan llamadas a algún método es posible pasar mediante un parámetro el estado que tendrá o leerse desde una base de datos externa.

Procesos y recursos particionados

El último aspecto fundamental es particionar los recursos y los procesos. Esto quiere decir que debemos minimizar las relaciones entre los recursos y los procesos, ya que al hacerlo reducimos la posibilidad de aparición de cuellos de botella cuando algún participante de la relación tarda más que otro.

¿LOS PATRONES DE DISEÑO SON NECESARIOS?

Los patrones de diseño son la estructura y la solución estándar para los problemas que surgen en el desarrollo de sistemas. Brindan soluciones documentadas y se clasifican en **patrones de creación**, **patrones estructurales** y **patrones de comportamiento**. En el siguiente enlace tenemos más detalles: **http://patronesdediseno.net16.net**.

Descentralización

Existen varias alternativas en arquitecturas de hardware y software para el manejo de sistemas de gran disponibilidad, con tolerancia a fallas y soporte para escalabilidad. A continuación veremos tres arquitecturas dominantes diferentes en la construcción de sistemas escalables, pero que comparten un concepto en común: **clúster**. Específicamente, hablaremos del término **clusterización**, que es el agrupamiento de máquinas para operar como un único recurso. El lugar donde se encuentran los clústeres conectados se llama **datacenter** (en español, centro de datos).

> EL DATACENTER ES EL LUGAR DONDE ENCONTRAMOS LOS CLÚSTERES CONECTADOS

Figura 7. Para darnos la idea de cómo es un datacenter por dentro, nada mejor que conocer el de Google.

Un requisito necesario para cada nodo del clúster es la posibilidad de acceso a un canal de comunicación de alta velocidad; es decir, que posean gran ancho de banda disponible, de modo que se facilite la comunicación entre cada uno de los nodos. Otro requisito es que todos deben tener acceso simultáneo a los discos compartidos.

Ahora detallaremos las tres arquitecturas más importantes:

- **Shared Memory (SM):** es un tipo de arquitectura en la que múltiples procesadores comparten la memoria. Cada procesador tiene acceso completo a ella mediante un bus para lectura y escritura de datos como también para la comunicación entre cada uno de ellos. La performance del sistema está limitada por varios factores, como el ancho de banda del bus de la memoria, el ancho de banda de la comunicación entre los procesadores, la cantidad de memoria disponible en el sistema, e incluso el ancho de banda de entrada y salida del nodo.

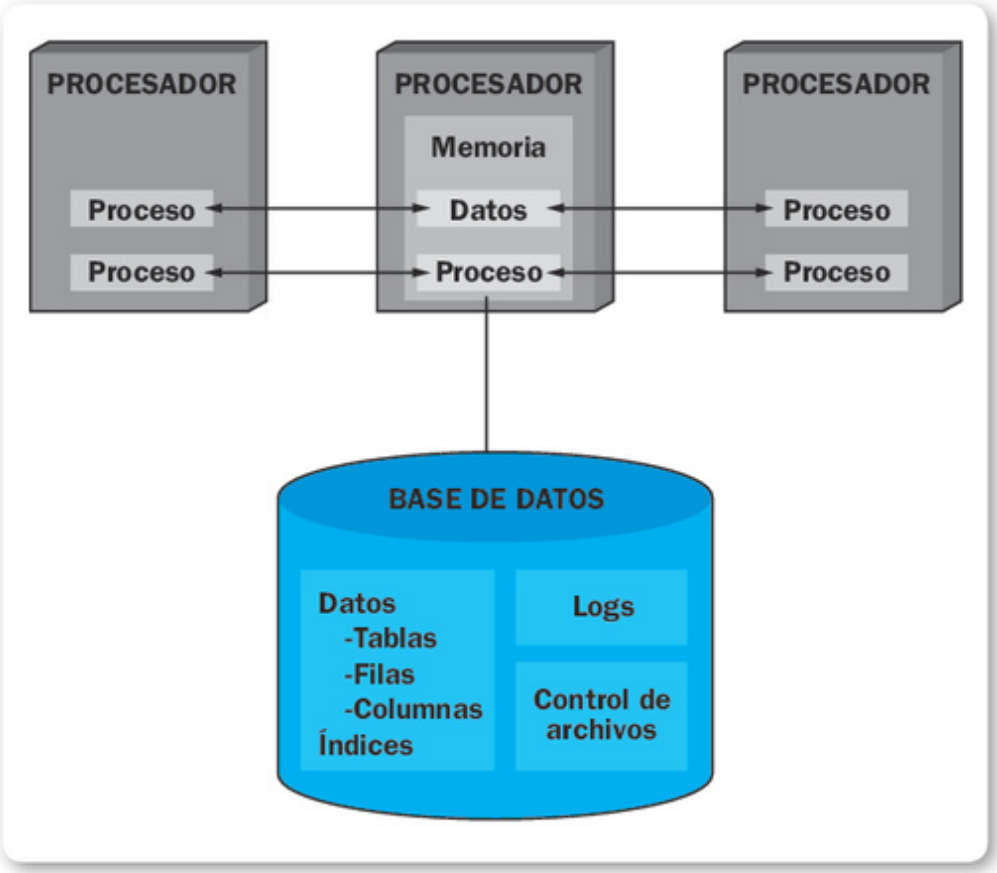


Figura 8. La arquitectura Shared Memory comparte la memoria entre todos los procesadores del clúster.

- **Shared Disk (SD):** en esta arquitectura varios servidores e instancias de bases de datos son accedidas como una sola. Un servidor puede ejecutar varias instancias de una base de datos y, a diferencia de **SM**, la arquitectura **SD** también puede agrupar

varias instancias en un solo servidor. Cada nodo del sistema tiene su propio procesador y su memoria asociada, y la comunicación es establecida a través de un bus de alta velocidad que permite a cada nodo acceder al mismo disco. En este tipo de arquitectura, las redes muy grandes pueden operar con un único conjunto de datos, lo que conlleva la desventaja de crear un solo punto de falla, aunque se puede recurrir a la duplicación de la base para solucionarlo. Shared Disk es una arquitectura que requiere un diseño y una implementación cuidadosa, ya que necesita de una gran infraestructura y de una buena administración para mantener todo en funcionamiento.

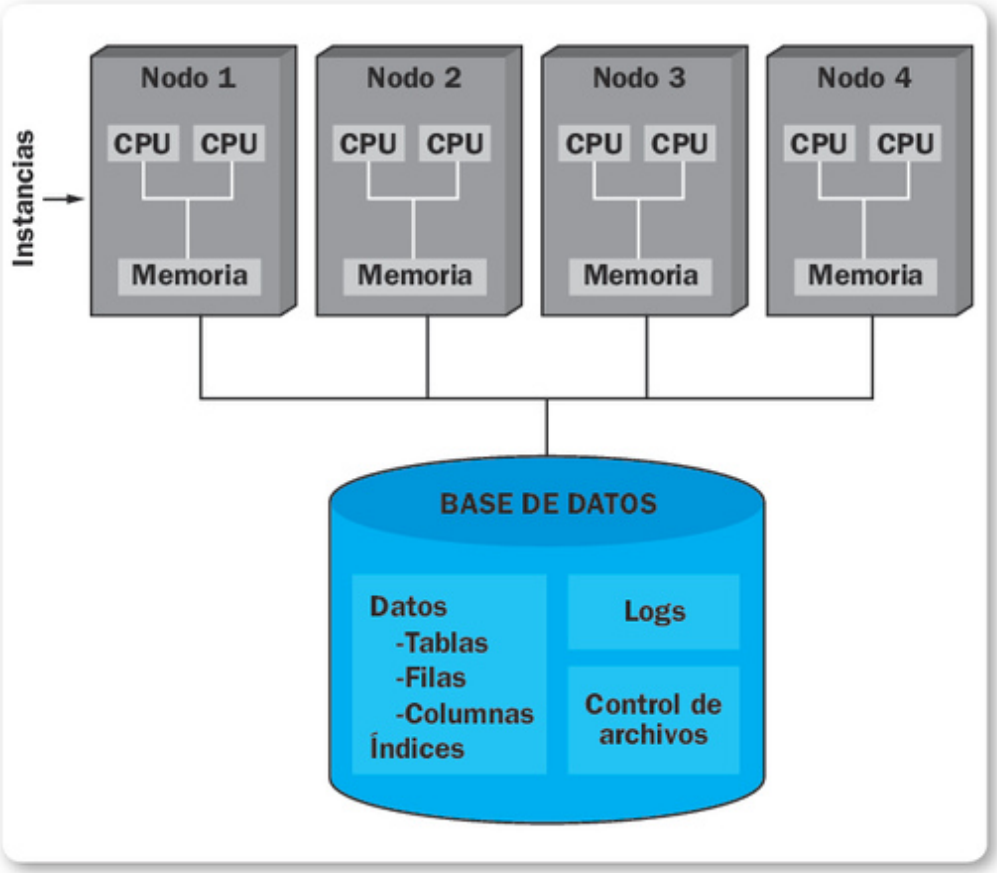


Figura 9. Shared Disk comparte la base de datos o sus instancias. Los datos son accedidos por todos los nodos del sistema.

- **Shared Nothing (SN)**: en este tipo de arquitectura no se comparten ni la memoria ni los discos de almacenamiento. Cada nodo del sistema es independiente, autosuficiente y autocontenido, lo que quiere decir que cada servidor puede trabajar sin depender de los

LA DESCENTRALIZACIÓN PERMITE LA ESCALABILIDAD HORIZONTAL

demás nodos ni de sistemas externos. En español, se la denomina **descentralización**, término que se refiere al tipo de arquitectura que permite que los sistemas escalen horizontalmente. Como ya dijimos, a medida que aumenta el tráfico en nuestro sistema se necesita mayor capacidad de procesamiento y, simplemente, se deben agregar tantas máquinas como sean necesarias para ser capaces de ofrecer un servicio de calidad.

Para implementar Shared Nothing se debe crear una imagen de un disco del sistema y copiar al nuevo nodo, que debe ser programado para arrancar automáticamente.

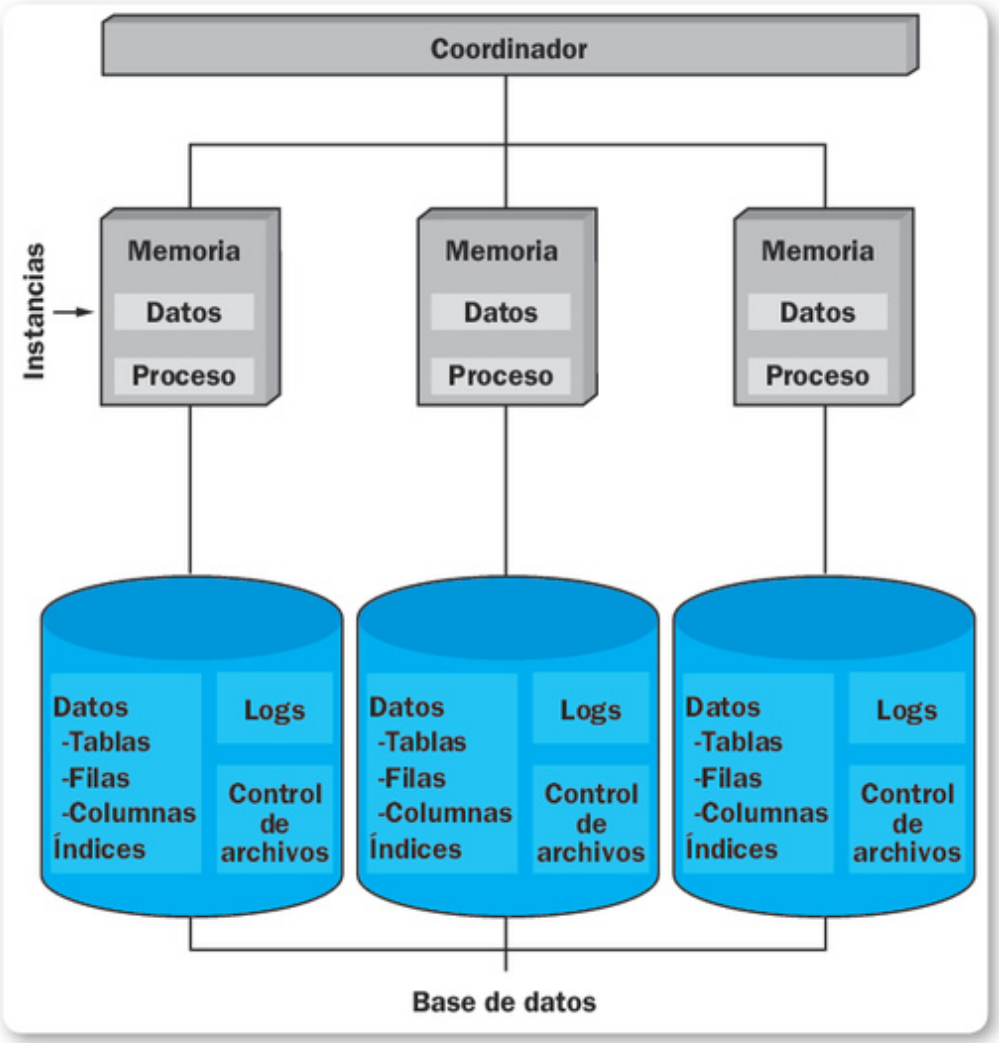


Figura 10. En Shared Nothing cada nodo funciona de manera independiente, pero bajo el control de un coordinador.

La característica más importante de esta arquitectura es que los estados del sistema no se guardan solo en un nodo individual, sino que son compartidos. Esto garantiza la presencia de calidad y estabilidad, de manera que si algún nodo falla no se pierde información y el sistema automáticamente redirige las solicitudes hacia algún otro nodo que se encuentre activo.

Bases de datos NoSQL

Todos los sistemas necesitan de algún método de persistencia de datos. Aproximadamente desde los años 70, los datos se almacenan en sistemas de bases de datos relacionados llamados **RDBMS** (**Relational Database Management System**), como **Oracle** y **MySQL**, que utilizan un esquema de tablas con columnas para identificar cada campo y definir el tipo de dato que contendrá. Por ejemplo, para la tabla **Persona**, tenemos los campos id, nombre, apellido, dirección y teléfono, y cada registro representa una nueva fila. Para acceder a los datos se debe utilizar un lenguaje de consultas estructurado conocido como **SQL** (**Structured Query Language**). Como vimos en la sección anterior, existen varias técnicas eficientes para escalar este tipo de bases de datos, como **Sharding** (o replicación), en sitios como **eBay** o **Flickr**, que funcionan con una excelente performance utilizando MySQL como base de datos (lo que sugiere una muy buena capacidad de escalar en MySQL).

> TODOS LOS SISTEMAS NECESITAN UN MÉTODO DE PERSISTENCIA DE DATOS

USABILIDAD DE LOS LENGUAJES WEB

PHP: usabilidad 50%, sigue un enfoque clásico y está bien documentado. **RUBY**: usabilidad 90%, posee un código limpio y potente y es fácil de usar. **PYTHON**: usabilidad 70%, utiliza sangría estricta, que lo hace muy legible. Para una comparación más amplia podemos visitar el siguiente sitio web: **http://udemy.com/blog/modern-language-wars**.

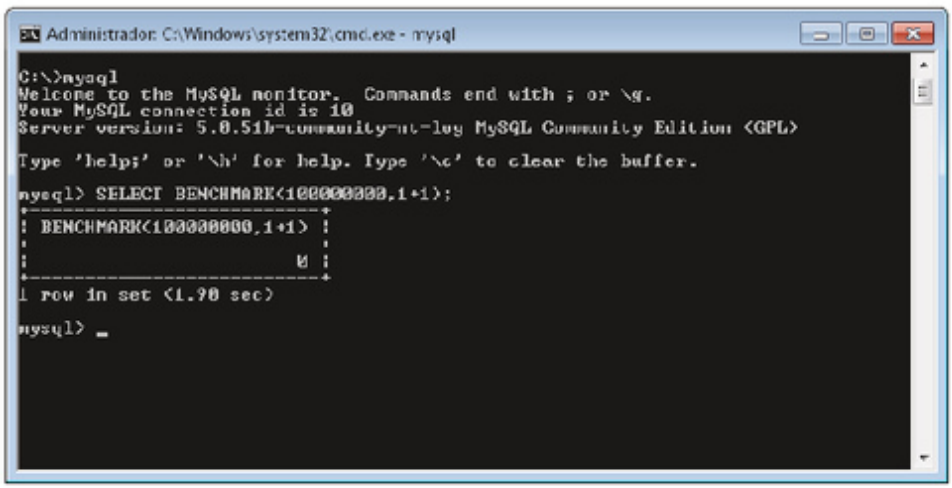


Figura 11. En MySQL es posible ejecutar el comando **BENCHMARK** para obtener un índice de performance.

Algunos gigantes de la Web que poseen una gran capacidad de desarrollo e investigación han optado por dejar de lado las bases de datos relacionales, ya que tienen problemas de escalabilidad y, en su lugar, han diseñado bases de datos no relacionales. El primero en innovar fue Google con **BigTable**, luego lo han seguido otros como **Amazon** con **DynamoDB** o Facebook con **Cassandra**. Solo empresas con estructuras gigantes podían abordar tal inversión para desarrollar sus propias bases de datos no relacionales. Sin embargo, hoy en día existen cada vez más bases de datos de esta clase al alcance de los programadores y las empresas de pequeña escala. Varias alternativas son **open source** (código abierto), como por ejemplo Cassandra, **MongoDB**, **Redis**, etc. Estos proyectos, recientes, son conocidos como **NoSQL (Not Only SQL)**.

> GRANDES EMPRESAS HAN OPTADO POR UTILIZAR BASES DE DATOS NO RELACIONALES

NoSQL se puede definir como una categoría que comprende gran variedad de bases de datos. Estas bases no tienen una estructura de datos en forma de tablas y relaciones entre ellas, sino que son más flexibles. Es decir, poseen estructuras dinámicas donde es posible agregar atributos solo a algunos registros y seguir manteniendo la

agrupación de la información. Por ejemplo, podemos encontrarnos con personas que poseen más atributos que otras sin que sea necesario que rediseñemos la estructura nuevamente.

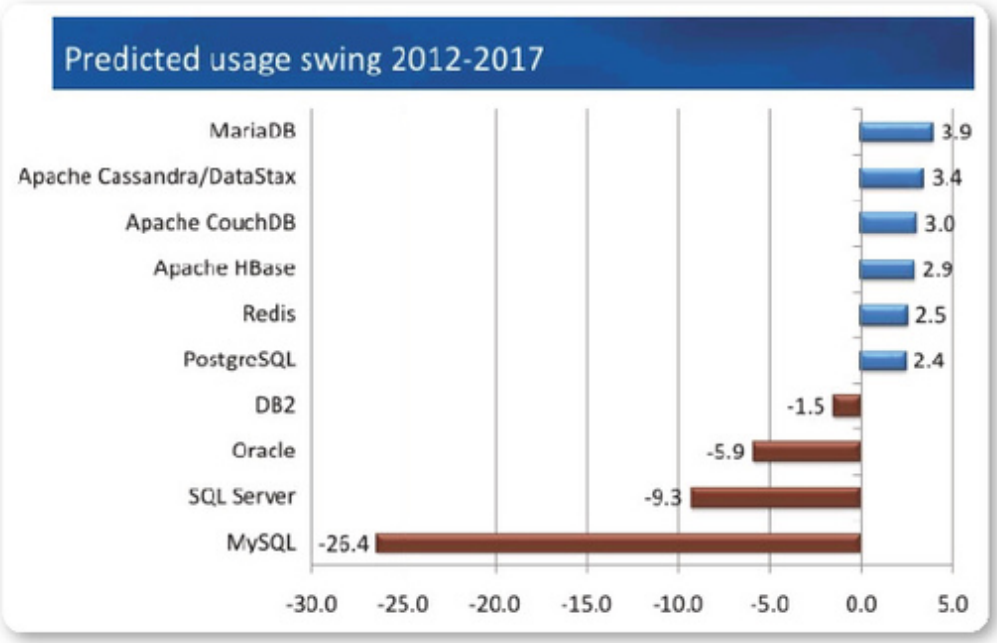


Figura 12. Un estudio de **451 Research** predice un uso cada vez mayor de bases de datos NoSQL y cada vez menor de relacionales.

Una de las principales características de las bases de datos NoSQL es que están diseñadas para operar con grandes cantidades de datos de manera extremadamente rápida. Para esto, suelen almacenar la información en memoria utilizando el disco solo como un mecanismo de persistencia. Además, debemos considerar que las operaciones de lectura y escritura de datos están altamente optimizadas, permitiendo escalar horizontalmente sin perder rendimiento.

ESCALABILIDAD DE GOOGLE

Google File System se presenta como un sistema de archivos distribuido, que posee una gran capacidad de almacenamiento (hasta cinco petabytes) y una velocidad de lectura y escritura de hasta 40 gigabytes por segundo. Si necesitamos más información podemos ver la definición completa y también descargar el documento adecuado visitando la página que encontramos en la dirección web **http://research.google.com/archive/gfs.html**.

Cuándo utilizar NoSQL

Podemos tener la necesidad de desarrollar un sistema que soporte enormes operaciones de lectura y escritura de datos y que pueda brindar un servicio a millones de usuarios sin perder rendimiento o, incluso, tener que hacer una reingeniería de un sistema existente que necesita escalar pero que opera con una base de datos relacional. Tengamos en cuenta que en estas situaciones es necesario considerar que se debe utilizar una base de datos NoSQL.

Empresas grandes, como Facebook, Twitter o Google, utilizan las bases de datos NoSQL como principal medio de almacenamiento, aunque esto no implica que no puedan usar bases de datos relacionales también. Es decir, es posible utilizar una base de datos NoSQL para almacenar toda la información de un sistema combinándola con las bases de datos relacionales. El sistema NoSQL estará destinado a funcionalidades que requieran millones de operaciones en tiempo real mientras que los sistemas relacionales, a consultas simples.

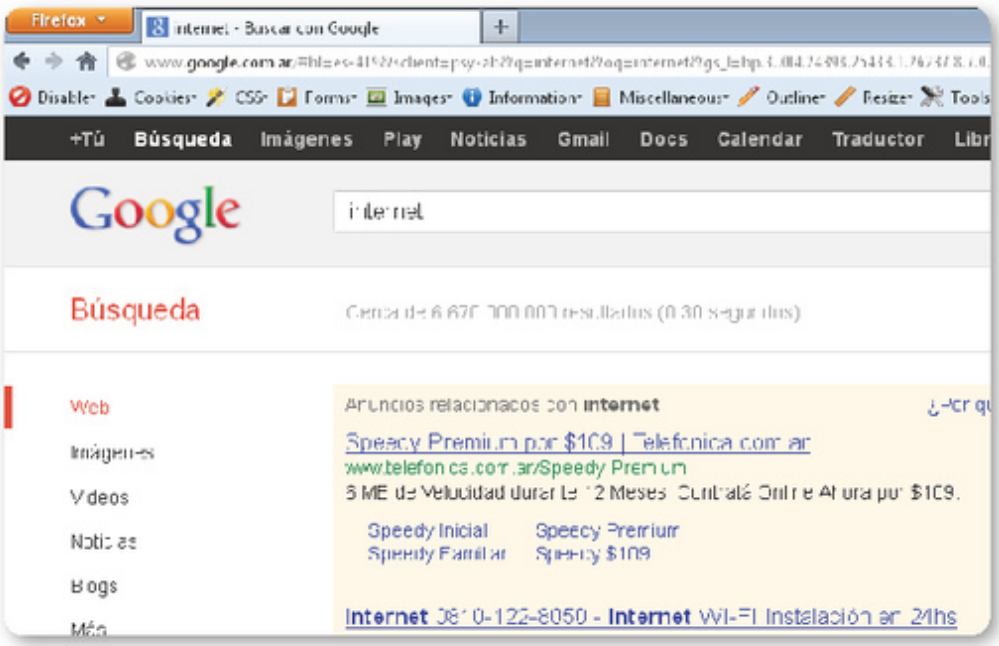


Figura 13. La palabra **internet** en Google arroja más de 6 mil millones de resultados en solo 0.30 segundos.

Como mencionamos anteriormente, Google es un referente de este tipo de implementaciones debido al enorme volumen de información que procesa y almacena diariamente, ya que no puede permitirse ningún cuello de botella que lentifique su sistema.

Tipos de bases de datos NoSQL

La clasificación de las bases de datos NoSQL depende de la forma en que se almacenan los datos, y define categorías como clave-valor, familia de columnas, documentos o grafos. Un aspecto importante es que no necesariamente utilizan SQL como principal lenguaje de consultas para acceder a los datos.

BASES DE DATOS NOSQL

CATEGORÍA	BASE DE DATOS	OPEN SOURCE
Orientadas a familias de columnas	Cassandra	Sí
	Hypertable	Sí
	Hadoop	Sí
Orientadas a documentos	MongoDB	Sí
	CouchDB	Sí
	RavenDB	Sí
Orientadas a clave-valor	Redis	Sí
	Riak	Sí
	DynamoDB	Sí
Orientadas a grafos	Neo4j	No
	HyperGraphDB	Sí
	InfoGrid	No
Orientadas a objetos	Db4o	Sí
	EyeDB	Sí
	Perst	Sí

Tabla 1. Clasificación por categoría de algunas de las bases de datos NoSQL más importantes.

WIKIPEDIA POR DENTRO

Wikipedia.org contiene más de 8 millones de artículos. En promedio, recibe 30.000 peticiones por segundo, que se distribuyen en tres datacenters compuestos por 350 servidores. Un solo datacenter es el primario y los otros dos son solo caché. El sistema operativo empleado es GNU/Linux, la plataforma está escrita en PHP y la base de datos usada es MySQL.

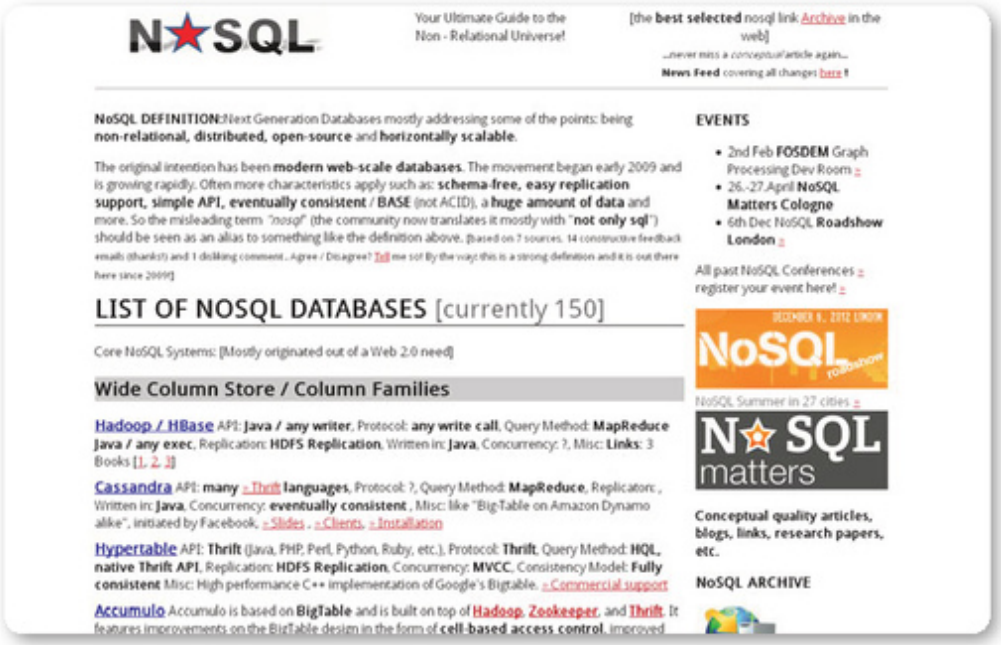


Figura 14. En **http://nosql-database.org** encontraremos la lista completa de bases de datos agrupadas por tipos.

A continuación, haremos una introducción a tres bases de datos NoSQL de las categorías: una orientada a familia de columnas, otra orientada a documentos y, la última, orientada a clave-valor.

MongoDB

MongoDB se presenta como una base de datos NoSQL de alta performance y escalable. Fue desarrollada por **10gen** a mediados del año 2007. Se encuentra escrita utilizando el lenguaje **C++** bajo el concepto de código abierto, por lo que podemos modificar sus archivos fuente. Antes de ver algunas de sus características, vamos a definir dos conceptos previos: JSON y BSON.

EL MOVIMIENTO NOSQL

Se puede resumir la definición de NoSQL en que es un sistema de base de datos de última generación, no-relacional, distribuido, de código abierto y altamente escalable. Es interesante saber que en este momento existen 150 bases NoSQL. Para más información podemos visitar el sitio oficial, que se encuentra en la dirección **http://nosql-database.org**.

- **JSON**: es el acrónimo de **JavaScript Object Notation** y define un formato simple de intercambio de datos. Debido a su sencillez, se presenta como alternativa a XML. Su principal ventaja es la posibilidad de escribir fácilmente un analizador sintáctico JSON. Por ejemplo, en JavaScript un JSON se puede analizar de la siguiente forma:

```
{"auto": {
"marca": "Marca1",
"modelo": "Modelo1"
}
}
```

- **BSON**: es el acrónimo de **Binary JSON**, que significa que los datos se serializan de forma binaria en formato JSON. A diferencia de JSON, posee extensiones que permiten representar los tipos de datos que contiene. Es decir, cada elemento en BSON consta de un campo nombre, un tipo y un valor. Los nombres son de clase string y los tipos pueden ser string, entero, fecha o booleano, entre otros. BSON es usado principalmente para almacenar y transferir información a la base de datos MongoDB.

Ahora que hemos definido los términos JSON y BSON, estamos listos para entender el funcionamiento de MongoDB. Esta base de datos está orientada a documentos, por lo que, a diferencia de los sistemas de bases de datos relacionales que guardan los datos en tablas, MongoDB los almacena en documentos con el formato **JSON** en un esquema dinámico llamado **BSON**. Esto establece que no existe un esquema predefinido. Cada uno de los elementos de los datos se denominan documentos y son guardados en las colecciones. Comparando este esquema con un sistema de base de datos relacional, podemos decir que las colecciones son las tablas y los documentos son los registros, con la diferencia de que en una base de datos relacional todos los registros tienen la misma cantidad de campos y en MongoDB cada documento de una colección puede definir sus propios campos.

> MONGODB ES UNA BASE DE DATOS ORIENTADA A DOCUMENTOS, EN FORMATO JSON

La estructura de un documento sigue el formato JSON compuesto por el par **clave-valor**, donde la clave es el nombre del campo y el valor es su contenido, separados mediante el carácter : (dos puntos). Es importante destacar que el valor puede contener números, cadenas de caracteres o datos binarios como imágenes.
A continuación, veremos un ejemplo de sintaxis:

```
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
   "Nombre": "Linus",
   "Apellido": "Torvalds",
   "Edad": 43,
   "Domicilio": {
      "Estado": "Oregón",
      "Ciudad": "Portland"
   }
}
```

En este caso, el campo **Domicilio** contiene otro documento con los campos **Estado** y **Ciudad**.

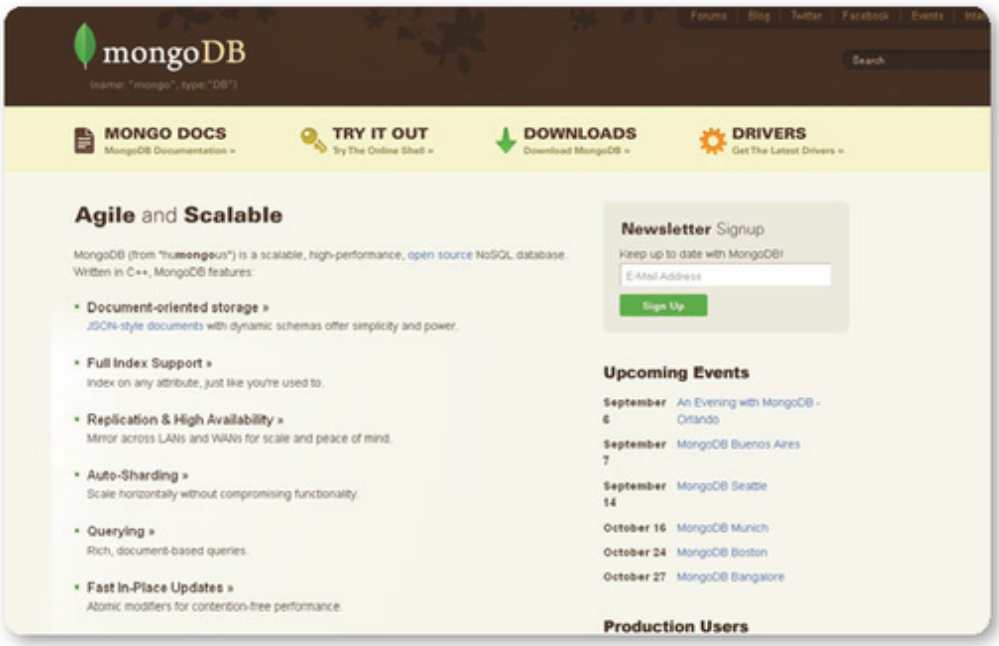


Figura 15. En **http://mongodb.org** encontraremos documentación, un enlace de descarga y un tutorial en línea.

MongoDB puede escalar horizontalmente utilizando la arquitectura **auto-sharding**, que provee balanceo de carga automática, permite agregar nuevas máquinas sin detener el sistema y puede escalar a miles de nodos, generando que no exista un único punto de falla.

A continuación se listan los casos en los cuales es adecuado usar esta base de datos:

> GRACIAS A AUTO-SHARDING, MONGODB PUEDE ESCALAR EN FORMA HORIZONTAL

- Sistemas de administración de contenido
- Sitios de comercio electrónico
- Juegos en línea
- Aplicaciones móviles
- Sistema de estadísticas en tiempo real

Actualmente la base de datos MongoDB tiene drivers oficiales para los lenguajes de programación, como C, C++, JavaScript, Node.JS, PHP y Python, entre otros, y es usado por una gran cantidad de empresas, entre ellas MTV, SourceForge, Disney, EA, The New York Times, ShareThis, GitHub, Foursquare y Justin.tv.

Cassandra

Cassandra es una base de datos NoSQL diseñada por Facebook para gestionar de manera eficiente su gran cantidad de datos. En el año 2008, Facebook libero el código de Cassandra cediéndolo a **Apache Software Fundation**, quien lo distribuye bajo una licencia open source.

Cassandra está escrita en Java y su modelo de almacenamiento está basado en el par clave-valor, siguiendo al sistema BigTable de Google y Dynamo de Amazon. Cassandra permite almacenar registros de una

SOPORTE DE BASES DE DATOS NOSQL

Las alternativas NoSQL de código abierto no suelen tener el mismo soporte que los proveedores de bases de datos privativas. Pero esta cuestión no representa un problema para los seguidores del enfoque, ya que estas bases de datos ofrecen una nutrida documentación y la mayoría tiene una comunidad activa.

manera continua y ordenada, mediante el modelo de datos de familias de columnas, el cual ofrece la posibilidad de crear **columnas índices**, brindando una gran performance.

Figura 16. Cassandra, como alternativa de bases de datos NoSQL. El sitio oficial es: **`http://cassandra.apache.org`**.

Cassandra posee un gran poder de escalabilidad mediante la gestión de los datos entre los nodos que forman parte de un clúster. Los nodos nuevos pueden añadirse de forma horizontal permitiendo que la información sea procesada automáticamente por el sistema. En este caso, Cassandra se encarga del balanceo de carga y la consistencia del nodo nuevo con respecto a los que ya existían. Así, si algún nodo deja de funcionar, el sistema seguirá trabajando de manera que los datos no se verán afectados. Recordemos que esto le otorga alta tolerancia a las fallas, una de sus características más importantes.

TERCER PILAR DE LA ESCALABILIDAD

Bigtable es un sistema de almacenamiento distribuido diseñado para gestionar petabytes de datos a través de miles de servidores. Podemos ver la definición completa y descargar la versión en pdf desde: **http://research.google.com/archive/bigtable.html**.

Es necesario considerar que la base de datos Cassandra es adecuada para ser utilizada en sistemas que no pueden perder datos, incluso cuando un nodo deja de funcionar.

Es interesante tener en cuenta que algunas de las empresas que han optado por usar esta base son las sigueintes: Netflix, Twitter, Constant Contact, Cisco y Digg. En este sentido, el mayor clúster conocido con esta base de datos tiene más de 300 TB de datos en más de 400 máquinas.

> CASSANDRA PUEDE SER UTILIZADA POR SISTEMAS QUE NO DEBEN PERDER DATOS

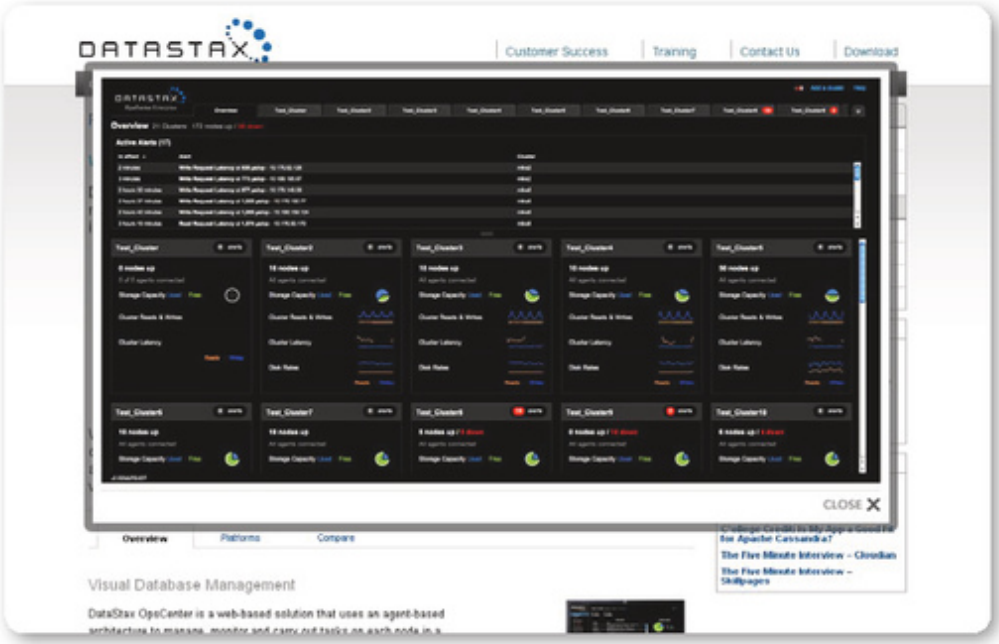


Figura 17. Con **DataStax OpsCenter** podemos administrar Cassandra. Lo descargamos desde **http://datastax.com/products/opscenter**.

Redis

Redis es un motor de base de datos NoSQL que opera en memoria, basado en el almacenamiento en tablas de **hashes** de tipo clave-valor. Está escrito en ANSI C y liberado bajo licencia open source, y actualmente soporta varios tipos de datos, como **strings**, **listas**, **sets**, **sorted sets** y **hashes**. En cuanto a la persistencia, debemos considerar que guarda la información correspondiente en memoria, pero puede ser configurada en el disco rígido.

Figura 18. Desde el sitio oficial, podemos descargar Redis y ver los comandos disponibles: **`http://redis.io`**.

EL RENDIMIENTO DE REDIS ES MEJOR QUE EL DE OTROS SISTEMAS DE BASES DE DATOS

Una característica interesante es que admite la replicación de tipo **maestro-esclavo**, y a su vez un esclavo puede ser maestro de otro esclavo. Esto permite la posibilidad de tener una replicación en forma de árbol. Otra función importante es el soporte de **publicación** y **suscripción**. Cuando un cliente esclavo se suscribe a un canal, recibe un mensaje de estado completo de las publicaciones del maestro, que es replicado a todo el árbol.

Como sabemos, debido a que Redis mantiene los datos en memoria, el rendimiento que nos ofrece es extremadamente bueno comparado con otros sistemas de bases de datos.

TENDENCIA DE EMPLEOS

Gracias al sitio **Indeed** podemos saber cuáles son las tendencias del mercado con respecto a las bases de datos NoSQL y cuáles tienen mayor incidencia. Por ejemplo, para comparar Cassandra, Redis y MongoDb podemos acceder a: **http://indeed.com/jobtrends/cassandra%2C+redis%2C+mongoDb.html**.

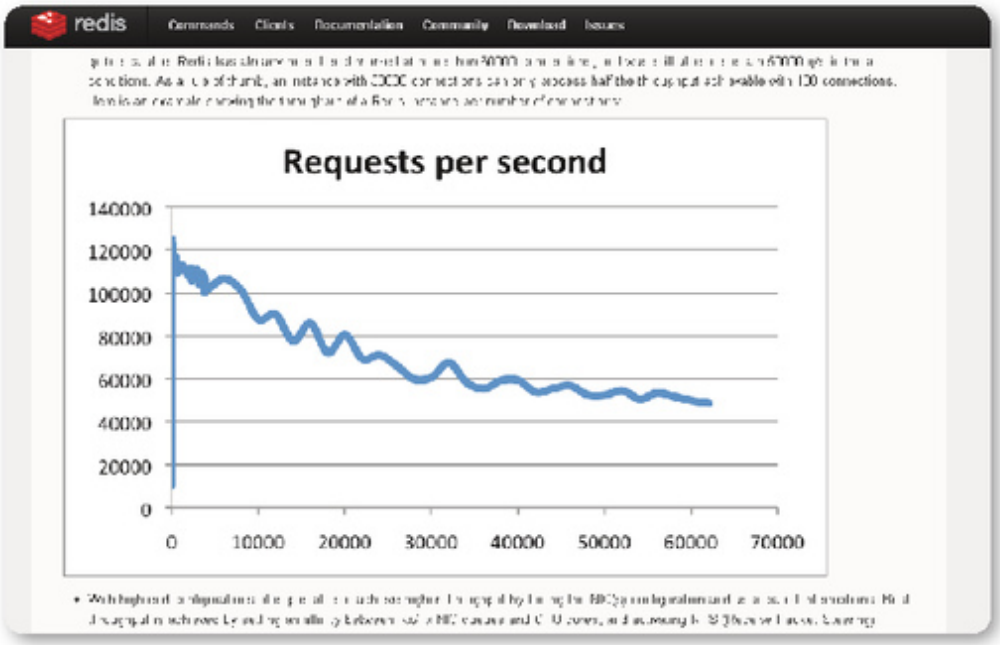


Figura 19. Podemos ver el rendimiento de Redis en la dirección **`http://redis.io/topics/benchmarks`**.

En los próximos capítulos nos concentraremos en el funcionamiento de este motor de bases de datos en particular, y descubriremos el potencial que posee al implementarlo en el ejemplo final.

RESUMEN

Como vimos en este capítulo, para desarrollar sistemas escalables primero necesitamos entender la importancia de un buen diseño, que logre soportar un crecimiento ilimitado. Cada vez son más frecuentes los sistemas que interactúan con redes sociales que manejan gran cantidad de información, por lo que se necesitan sistemas de almacenamiento con tolerancia a fallas y de gran performance para soportar la demanda. Una vez entendido el concepto de escalabilidad estamos en condiciones de implementar una solución NoSQL como herramienta de persistencia, junto a la tecnología que nos permitirá crear desarrollos orientados a eventos, Node.JS.

Actividades

TEST DE AUTOEVALUACIÓN

1. Mencione dos diferencias entre escalar verticalmente y horizontalmente.
2. ¿Qué es lo que se comparte en la arquitectura **Shared Memory**?
3. ¿A qué se denomina **Shared Nothing**?
4. ¿Según qué criterio se clasifican las bases de datos NoSQL?
5. ¿En qué formato almacena los datos MongoDB?
6. ¿Cassandra opera en memoria?
7. ¿Es posible almacenar videos en Cassandra?
8. ¿Es posible almacenar archivos de audio en MongoDB?
9. ¿Redis es open source?
10. En Redis ¿un nodo esclavo puede ser maestro y tener otros nodos esclavos? ¿Soporta la estructura de árbol?

EJERCICIOS PRÁCTICOS

1. Haga una lista con las características principales que deben tener los sistemas para que sean escalables.
2. Defina base de datos relacional y NoSQL.
3. Proponga diferentes sistemas y las bases de datos NoSQL que deberían utilizar.
4. Exponga las características que una base de datos NoSQL debe cumplir.
5. Haga una lista de los sitios de Internet que escalan, según un criterio propio.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*02

Redis

Conoceremos Redis, una de las bases de datos NoSQL más relevantes y usadas en la actualidad para el desarrollo de sistemas escalables. Una de sus características más importantes es que todas las bases de datos creadas operan en memoria, otorgándole un rendimiento casi único en el manejo de grandes volúmenes de datos.

- **Introducción 44**
- **Cuándo usar Redis 46**
- **Instalación 47**
- **Tipos de datos 55**
  - Strings 56
  - Lists 56
  - Sets 57
  - Hashes 60
- **Comandos 61**
- **Publicación y suscripción 68**
  - Suscripciones mediante un patrón de coincidencia 71
  - Suscripciones coincidentes con canales y con patrones 73
- **Resumen 73**
- **Actividades 74**

↗↗↗

Introducción

Redis es un servidor de estructuras de datos que opera en memoria, basado en el almacenamiento de tipo clave-valor (denominado tabla de hashes), y que ofrece la posibilidad de persistencia de los datos en el disco rígido. Fue desarrollado por Salvatore Sanfilippo en el año 2009 en el lenguaje C y obtuvo gran popularidad desde que personas involucradas en el mundo del desarrollo de sistemas le dieron soporte.

Este motor de base de datos es patrocinado por **VMware** y está liberado bajo la licencia BSD, por lo que es considerado un software de código abierto. Funciona en la mayoría de los sistemas POSIX –como Linux, BSD y OSX– y también en sistemas operativos Windows sin soporte oficial, aunque existen variadas alternativas de instalación y funcionamiento.

Existe una gran variedad de lenguajes que se pueden usar para el manejo de datos, como **C**, **C#**, **Erlang**, **Go**, **Io**, **Java**, **Node.js**, **Perl**, **PHP**, **Python** y **Ruby**, entre otros. Nosotros nos vamos a enfocar más específicamente en PHP y Node.js.

Redis ha sido diseñado para escalar horizontalmente, ya que ofrece un mecanismo de replicación maestro-esclavo, donde los servidores esclavos obtienen las copias exactas directamente del maestro. La replicación tiene estas características:

- Un maestro puede tener múltiples esclavos.
- Un esclavo puede tener otros esclavos.
- Posee un método de sincronización no bloqueante para el maestro: es decir, un maestro puede seguir atendiendo peticiones mientras los esclavos se sincronizan.
- Para configurar un servidor como esclavo de otro solo hay que especificar el siguiente comando en el archivo de configuración del esclavo:

```
(: slaveof [IP SERVIDOR MAESTRO][PUERTO] :)

slaveof 192.168.1.1 6379
```

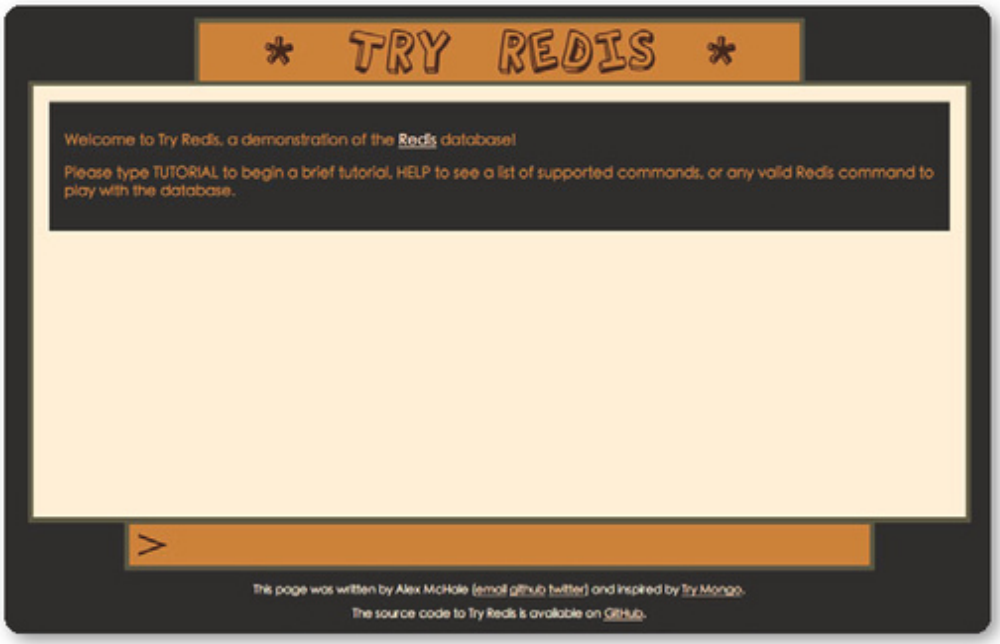


Figura 1. Redis ofrece un tutorial interactivo para conocer y consultar las características más importantes: **`http://try.redis-db.com`**.

Los tipos de datos que soporta son strings, lists, sets, sorted sets y hashes. En ellos se pueden ejecutar operaciones atómicas, como agregar caracteres a un string, incrementar/decrementar el valor de un hash o agregar valores a un list, junto con otras operaciones como intersección, unión o diferencia. Incluso es posible obtener un elemento con un alto ranking en un sorted set y, por otro lado, cuenta con un mecanismo de publicación/suscripción y expiración de datos para usarlo como sistema de caché. Debido a que el motor opera con toda la base de datos en memoria se crea una limitación en torno a este requerimiento físico.

> REDIS SOPORTA LOS TIPOS DE DATOS STRINGS, SETS, SORTED SETS Y HASHES

HACIA DÓNDE VA LA WEB

Metamarkets.com ha publicado un artículo donde define tres eras de la Web. La primera, de 1991 a 1999, a la que denominó **era HTML**; la segunda, de 2000 a 2009, que llamó **era LAMP** y la tercera, desde 2010 hasta la actualidad, la **era JavaScript**. El artículo completo, en inglés, lo podemos ver en: **http://gigaom.com/cloud/node-js-and-the-javascript-age**.

Con respecto al rendimiento, Redis es de un solo hilo, a diferencia de otros sistemas que crean un proceso por cada petición. En los servidores actuales es muy común tener múltiples núcleos, por lo que se produce un desperdicio de potencial de procesamiento; para mejorar el rendimiento en estos sistemas es posible iniciar varias instancias en el mismo servidor, pero librerías como **Predis** (cliente PHP para Redis) ya realizan esto de forma automática.

Cuándo usar Redis

Cuando necesitamos implementar una solución que se ajuste a nuestras necesidades nos encontramos el interrogante de cuál es la mejor opción. Para respondernos a esta pregunta debemos tener en cuenta varias consideraciones importantes como, por ejemplo, la naturaleza de las estructuras de datos; es decir, si vamos a necesitar una estructura definida, como es el caso de las bases de datos relacionales, o si podremos tener mayor flexibilidad implementando motores NoSQL.

Si no estamos seguros de que una base de datos relacional posee las características necesarias y requerimos buena performance y escalabilidad, Redis es una buena alternativa.

A la hora de implementar una solución también debemos tener en cuenta las diferencias entre las bases de datos NoSQL. Por ejemplo, MongoDB permite realizar consultas de rangos, búsquedas mediante expresiones regulares, indexación e implementar **MapReduce** (un framework para la manipulación de grandes volúmenes de datos), mientras que Redis es extremadamente rápido y adecuado para sistemas que requieren escritura de datos de manera continua, como videojuegos,

SCROLL INFINITO

Como estamos acostumbrados a ver en las redes sociales, en el área de las publicaciones se muestra una especie de scroll infinito de manera que, a medida que se avanza al final de la página, van apareciendo más y más contenidos. Con el uso de scroll infinito se deja de lado el uso del pie de pagina, esto nos permitirá cargar contenidos al final de la página en forma sencilla.

mensajería instantánea, redes sociales, etcétera (aunque no se recomienda su uso si se posee un gran conjunto de datos con modificaciones mínimas en largos intervalos de tiempo).

Las bases de datos como Redis son rápidas, escalan fácilmente y se ajustan perfectamente a las necesidades modernas, pero es importante elegir la herramienta adecuada para el trabajo requerido. Teniendo en cuenta esto, es totalmente valido inclusive combinar un sistema relacional con un sistema **NoSQL**.

> REDIS ES UNA BASE DE DATOS RÁPIDA Y QUE ESCALA EN FORMA FÁCIL Y SENCILLA

Instalación

Por definición, Redis está diseñado para operar en sistemas UNIX, tanto para entornos de desarrollo como de producción. Aunque no es oficialmente soportado en sistemas win32/win64, por motivos didácticos vamos a presentarlo como alternativa para entorno de desarrollo. Vale la pena aclarar que no es recomendable usarlo en sistemas Windows para entornos de producción, debido a las limitaciones que existen entre el sistema operativo y el motor de base de datos, que en este momento no son objetos de nuestro estudio.

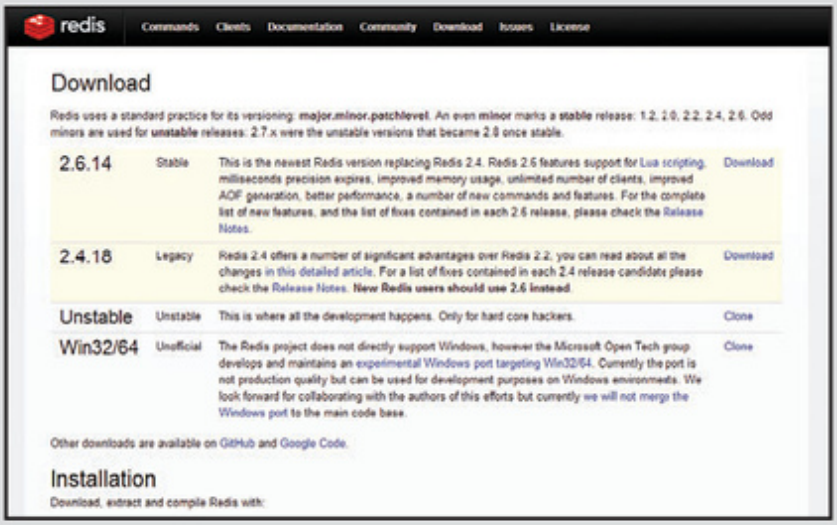
Para la instalación en sistemas UNIX podemos acceder a la dirección **http://redis.io/download**, donde encontraremos los pasos y un pequeño ejemplo para comprobar que tenemos Redis funcionando. A continuación, veremos cómo instalar Redis en nuestro sistema Windows. Debemos tener en cuenta que primero necesitamos Redis y una herramienta adicional que crea un servicio para nuestro motor.

CREAR ESCENARIOS DE PRUEBA

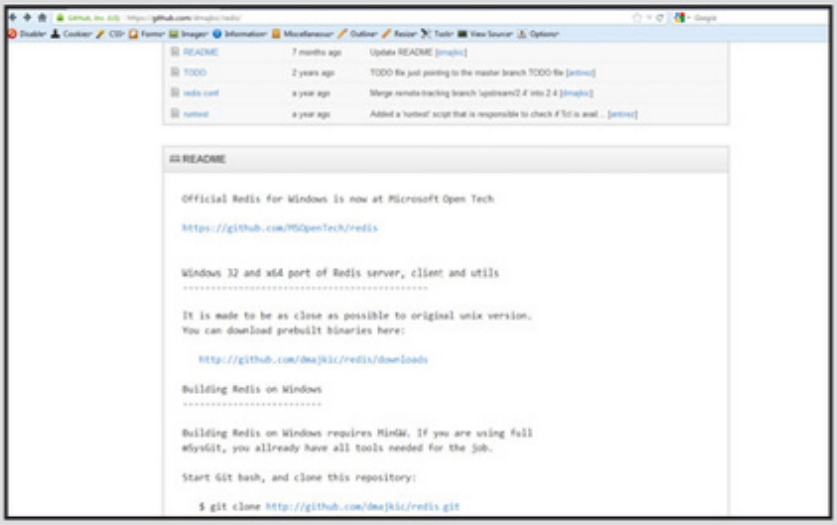
Siempre es importante probar nuestros sistemas, pero muchas veces no sabemos cómo hacerlo. **SimpleTest** es un framework para PHP de código abierto, usado para automatizar los procesos de prueba. Podemos descargarlo desde la página oficial: **http://simpletest.org**.

PAP: DESCARGAR LOS INSTALADORES

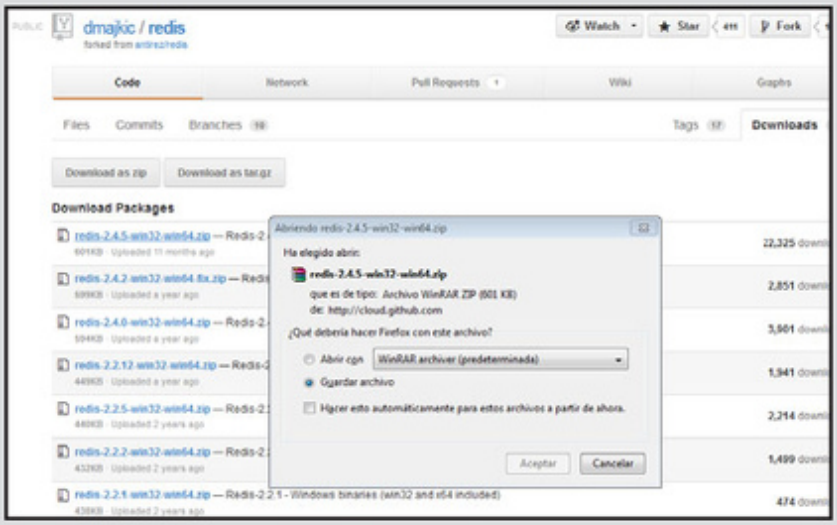
01 Diríjase a **http://redis.io**, el sitio de Redis. En el menú, encontrará el enlace Download; haga clic en él y verá una página con las opciones de descarga.



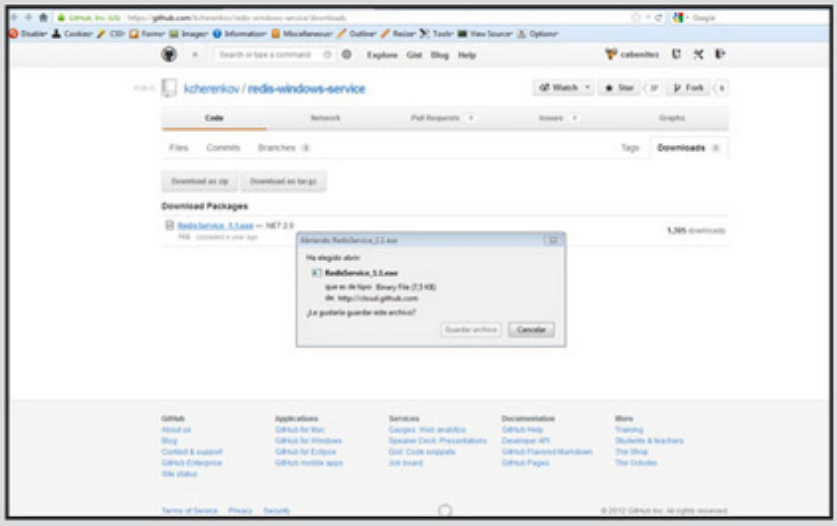
02 Dentro de Win32/64 haga clic en `Windows port targeting Win32/64`. Busque la sección `Acknowledgements` y haga clic en **https://github.com/dmajkic/redis**, luego presione sobre **http://github.com/dmajkic/redis/downloads**.



03 Se abre la página de descarga de los instaladores de Redis para Windows, que se encuentran comprimidos en archivos ZIP y, por lo general, contienen tanto la versión para 32 bits como para 64 bits.



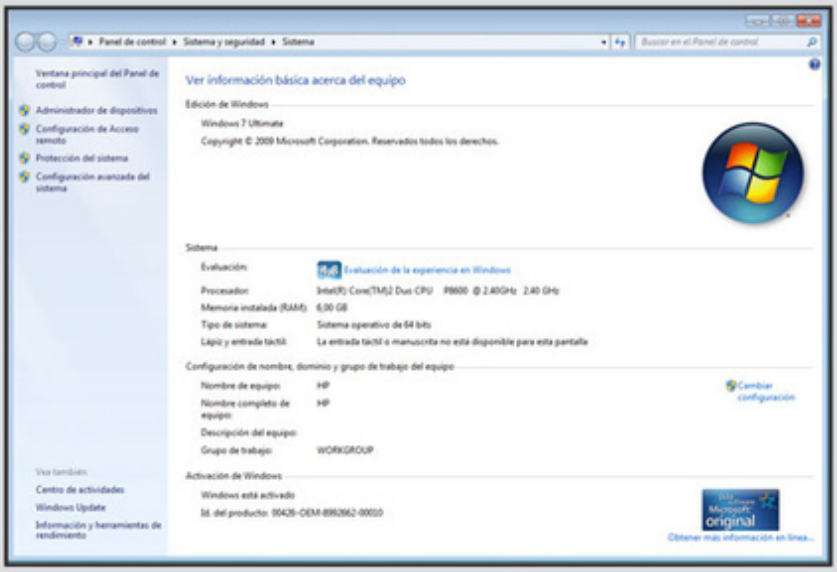
04 Ahora debe descargar el programa que hace que Redis se ejecute como un servicio. En **https://github.com/kcherenkov/redis-windows-service** haga clic en `Compiled executable`. Esto lo llevará a la página de descarga.



Una vez descargados los instaladores es momento de preparar el entorno, instalar Redis y hacer una prueba para verificar que todo salió bien. Para realizar estas tareas será necesario que completemos una serie de procedimientos, descritos en el siguiente **Paso a paso**. No se trata de un procedimiento complicado, solo es necesario observar las recomendaciones que entregamos.

PAP: INSTALACIÓN DE REDIS

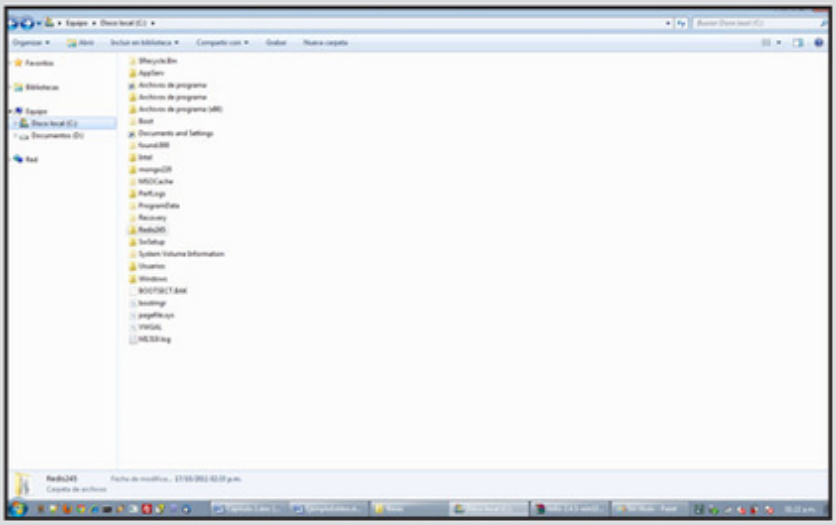
01 Antes que nada, es necesario que identifique qué sistema de archivos tiene instalado, si de 32 o 64 bits. Para esto, deberá acceder a `Panel de control/Sistema y seguridad/Sistema`; posteriormente, encárguese de realizar la verificación de la configuración en la sección `Tipo de sistema`.



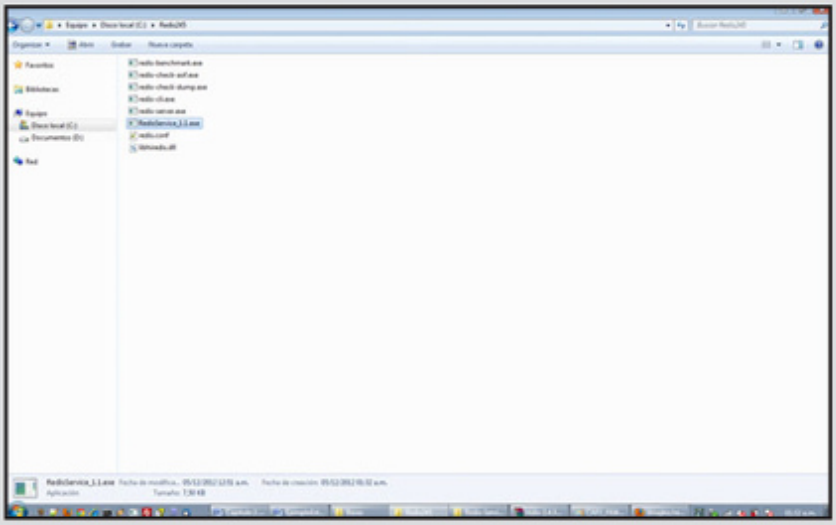
ANTEOJOS PARA NAVEGAR POR INTERNET

Investigadores alemanes han desarrollado anteojos que utilizan realidad aumentada para consultar manuales y visitar páginas web sin necesidad de moverse de lugar. Podemos ver más información en el siguiente enlace: **http://tendencias21.net/Nuevas-gafas-de-realidad-aumentada-permiten-consultar-manuales-sin-tocarlos_a14261.html**.

02 A continuación, diríjase al directorio donde descargó el instalador de Redis y abra el archivo ZIP, que contiene las versiones de 32 y 64 bits. Extraiga la carpeta que corresponde a su sistema y péguela en la unidad C.

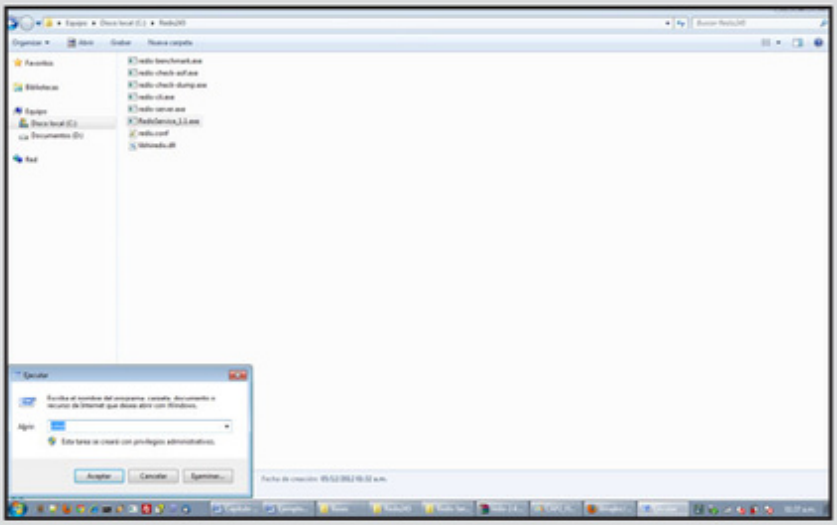


03 Para que sea posible realizar una mejor identificación, cambie el nombre de la carpeta que copió en el paso anterior por **Redis**, más la versión que corresponde a la instalación que se encuentra realizando, por ejemplo: **Redis245**.



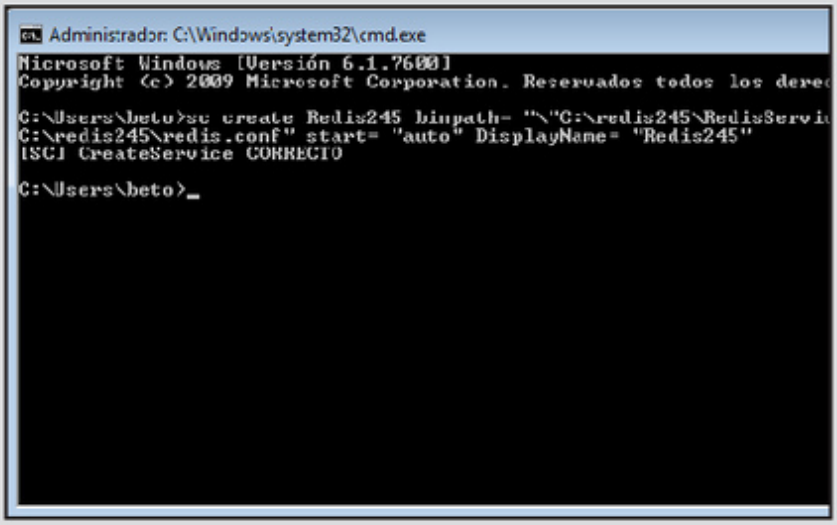
04

Luego, vaya al directorio donde descargó el instalador `RedisService_1.1.exe` y posteriormente cópielo y péguelo dentro de la carpeta que ha creado anteriormente (es decir, **Redis245**).

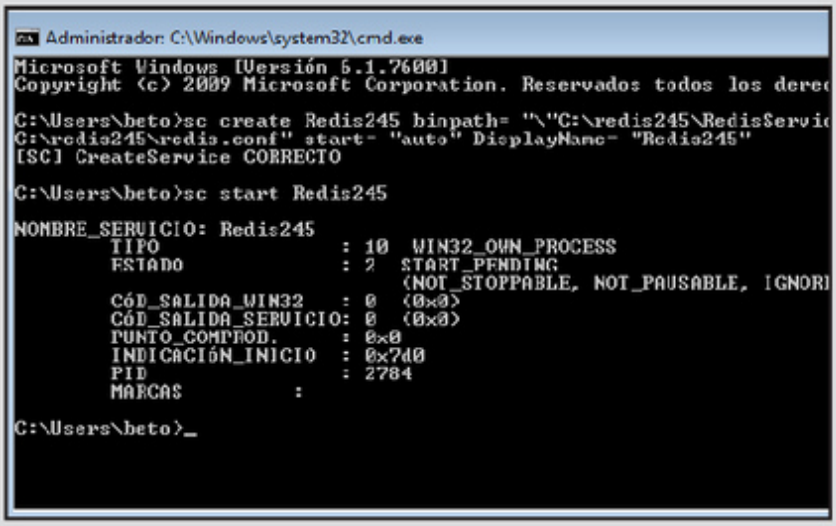


05

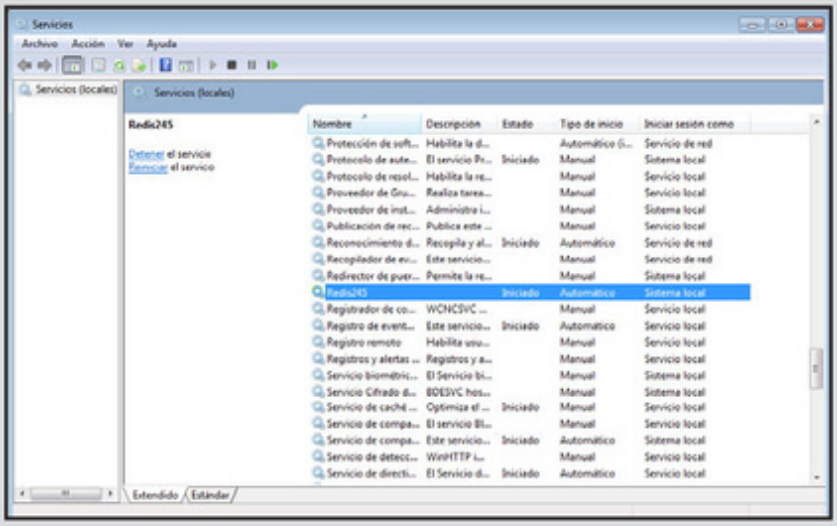
Ahora que ya tiene todo en su lugar, solo le queda crear el servicio para Windows. Para esto, abra una consola presionando las teclas `WINDOWS + R`. En la ventana `Ejecutar`, escriba `cmd` y presione `ENTER` o `Aceptar`.



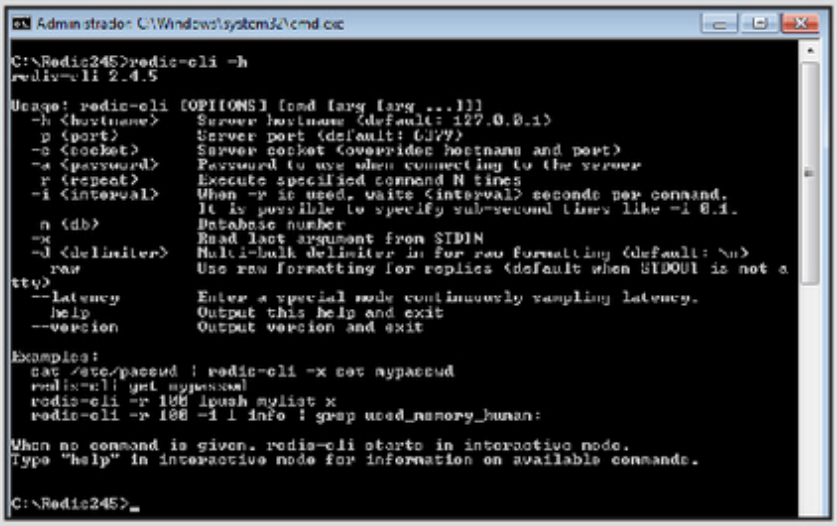
06 En la consola que se abre, escriba: `sc create %name% binpath= "\"%binpath%\" %configpath%" start= "auto" DisplayName= "Redis"`. Referencia: %name% es el nombre del servicio, %binpath% es la ruta del archivo de servicio y %configpath% es la ruta del archivo de configuración.



07 Es momento de iniciar el servicio. En la consola, escriba: `sc start Redis245`. Si el servicio se inició, deberá ver una pantalla como la que sigue.



08 Para asegurarse de que el servicio está creado e iniciado, puede acceder al visor de servicios de Windows a través de `Panel de control/Sistema y seguridad/Herramientas administrativas/Servicios`. Ahí verá una pantalla como la siguiente; ubíquese en la lista y presione la letra R. Debería tener un servicio llamado **Redis245** con el estado `Iniciado`.



Una vez que verificamos que el servicio está ejecutándose, nos queda hacer la prueba de fuego. Para esto, abrimos una consola, nos situamos en el directorio donde tenemos Redis (que, siguiendo nuestro ejemplo, es **C:\Redis245**) e iniciamos el servidor con el siguiente comando: **redis-server.exe**

Luego, abrimos otra consola, nos situamos en **C:/Redis245** y ejecutamos el siguiente comando: **redis-cli.exe**

Probamos almacenando una clave con su respectivo valor y lo volvemos a obtener:

```
redis 127.0.0.1:6379> set var1 "hola mundo"
OK
redis 127.0.0.1:6379> get var1
"hola mundo"
```

De este modo, tenemos listo el servidor de bases de datos para realizar pruebas y empezar a conocerlo mejor. Es necesario aclarar que, de ahora en más, ya no es necesario ejecutar el programa **redis-server.exe**, ya que lo tenemos ejecutándose como servicio de Windows y podemos utilizarlo cuando lo necesitemos. Si reiniciamos nuestra computadora, podemos abrir una consola, acceder al directorio donde tenemos la base de datos y ejecutar **redis-cli**; luego, cuando ejecutamos el comando **get var1**, debemos obtener el valor **"hola mundo"**.

Tipos de datos

A diferencia de otras bases de datos NoSQL, Redis ofrece varios tipos de datos incorporados, donde cada uno tiene un significado semántico útil que adhiere beneficios en cuanto a rendimiento. Pero antes de hablar de cada uno de los tipos de datos, es importante tener en cuenta algunas cuestiones cuando diseñamos una estructura clave-valor:

- Ser consistentes cuando definimos las claves. Como pueden contener cualquier carácter es una buena práctica definir espacios de nombres mediante separadores. Estos espacios de nombres deben ser representativos de cada contexto. Una práctica útil puede ser usar el carácter **:** (dos puntos) como separador, por ejemplo, **cliente:proyecto:1:tareas**.
- Cuando definimos las claves, debemos limitarlas a un tamaño razonable. Es buena práctica usar nombres tan cortos como sea posible, ya que al obtener una clave desde el motor se efectúan operaciones de comparación más eficientes con claves cortas.
- Así como las claves no deben ser largas, tampoco deben ser excesivamente cortas, ya que deben ser legibles y representativas para un mejor entendimiento del contexto.

Con estas cuestiones en mente, claves como **c:p:1:t** o **cliente 10** serían una mala opción, porque la primera no tiene ninguna representación semántica y la segunda incluye espacios en blanco. Ahora vamos a definir y analizar en profundidad los tipos de datos que ofrece Redis: strings, lists, sets, hashes y sorted sets.

Strings

El tipo de dato string es el más común y frecuentemente usado. Es un binario seguro, lo que significa que puede almacenar una cadena de cualquier tipo de dato, como una imagen **JPG**, un video **MPEG** o un objeto serializado, y debe tener un tamaño máximo de 512 Mb. Un ejemplo son las imágenes almacenadas para los avatares de una red social.

Redis ofrece varias operaciones útiles que se pueden realizar con este tipo de datos. Por ejemplo, se puede usar un string como un contador y realizar operaciones de incremento o decremento.

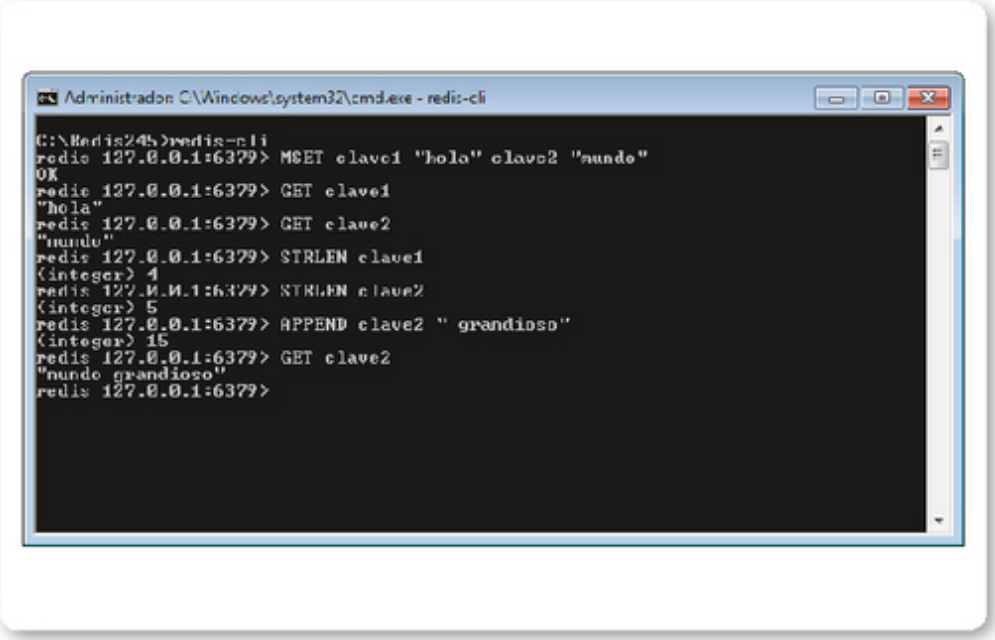


Figura 2. Vemos los comandos que podemos usar con los strings para agregar dos claves, obtener los valores para cada uno, la longitud y agregar un valor a cada clave.

Lists

Este tipo de datos es una lista ordenada de strings binarios, que establece un criterio de orden por inserción basado en la estructura de una lista enlazada. Es posible agregar elementos tanto al final como al inicio de la lista. Su longitud máxima es de 2^{32} -1 elementos, es decir, más de 4 billones de elementos por lista.

Redis ofrece un gran rendimiento en las operaciones de inserción como también de borrado constante, tanto al inicio como al final de las listas, inclusive cuando éstas contienen varios millones de elementos.

El acceso en los extremos es rápido. Sin embargo, si se desea obtener un elemento del medio, suele ser más lento en listas muy grandes.

Existen utilidades para este tipo de datos. Se pueden crear colas de eventos (**timeline**, usadas en las redes sociales, como Twitter) en las cuales se van ubicando los tweets que van escribiendo los usuarios.

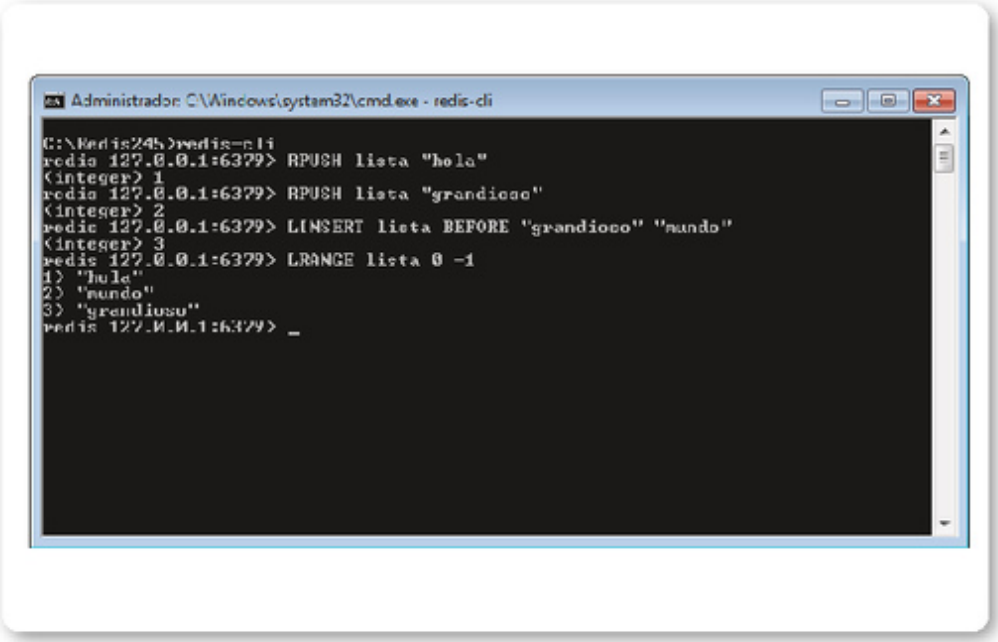


Figura 3. Vemos cómo insertar un elemento a la derecha, luego otro más y finalmente, uno entre ambos.

Sets

El tipo de datos set es una colección desordenada de strings. Al igual que el tipo de datos list, el número máximo de elementos es de 2^{32} -1, o sea, más de 4 millones de elementos por set. En este tipo de datos se pueden realizar operaciones como adición, eliminación o verificación de existencia de algún elemento, entre otras.

VELOCIDAD DE NUESTRO SITIO WEB

Google ofrece un complemento para **Chrome** que sirve para medir la velocidad que tienen los sitios y ofrece una lista de las mejores prácticas ordenadas por importancia, donde cada una incluye un puntaje de incidencia. Más información en: **https://developers.google.com/speed/pagespeed**.

Un aspecto interesante es que los sets no permiten elementos repetidos, es decir, si se intenta agregar un mismo elemento dos veces, Redis ignorará la segunda operación. Al mismo tiempo, Redis provee para los sets operaciones matemáticas a partir de conjuntos ya existentes, como unión, intersección o diferencias en muy poco tiempo.

Al igual que los tipos de datos vistos anteriormente, con los sets se pueden hacer varias implementaciones. Por ejemplo, un sistema de almacenamiento de acceso a un sitio, teniendo como dato la dirección IP del visitante: cada vez que el visitante ingresa, almacenamos su IP y, en caso de que los accesos se repitan, no se duplicarán en la base de datos.

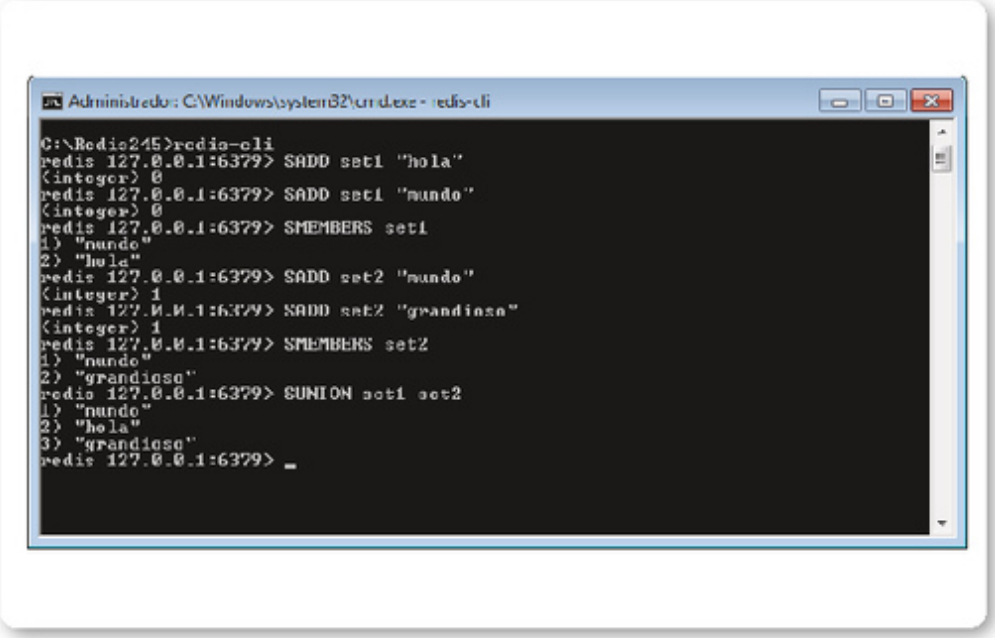


Figura 4. Es posible crear dos sets y luego realizar una operación de unión entre ellos.

Sorted sets

Este tipo de datos es una colección no repetida de strings similar a los sets, con la diferencia de que cada elemento está asociado a un puntaje, usado para obtener un conjunto ordenado de menor a mayor. Es importante tener en cuenta que los elementos son únicos pero los puntajes pueden ser repetidos.

Las operaciones que podemos realizar con este tipo de datos son adición, eliminación y modificación, de una manera muy rápida ya que los elementos se encuentran ordenados. Es posible obtener rangos por

puntaje o por posición, así como también acceder a elementos que se encuentran en el medio del conjunto; consideremos que para ambos casos tendremos un gran desempeño.

En síntesis, cualquier operación sobre este tipo de datos tiene una gran velocidad de procesamiento debido a su característica de puntaje. Un ejemplo de uso es un sistema para actualizar el ranking de líderes de un juego online, en el que cada vez que el usuario avanza un nivel se actualiza su puntuación. De esta manera, se puede obtener la lista de todo el conjunto, donde veremos cada elemento y su puntuación, o la posición para cada elemento en particular.

> LAS OPERACIONES SOBRE SORTED SETS OBTIENEN GRAN VELOCIDAD DE PROCESAMIENTO

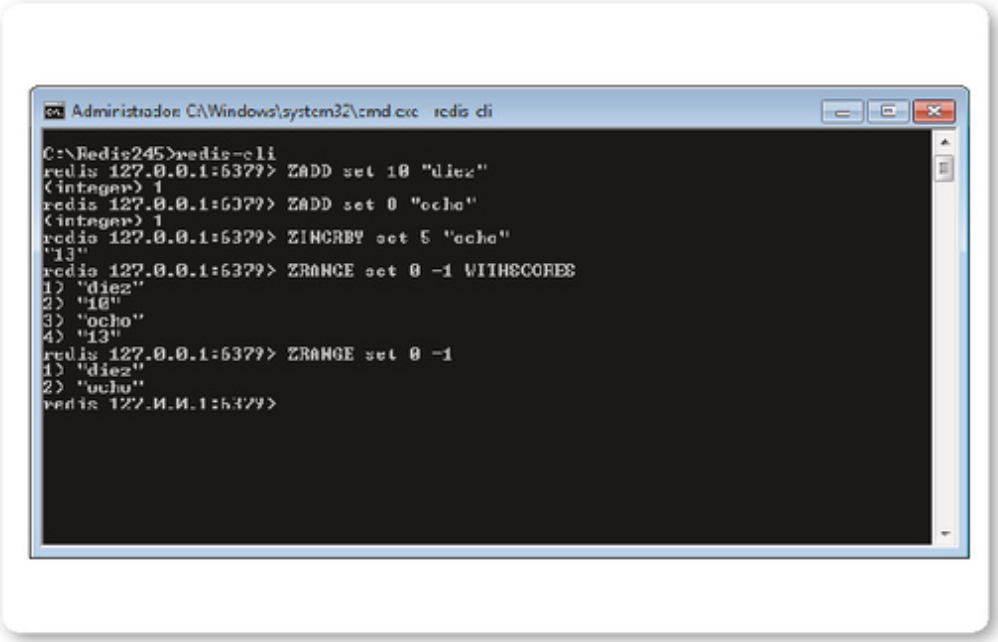


Figura 5. Agregamos dos elementos con sus respectivos puntajes, que luego obtenemos con **ZRANGE**.

REDIS CONTRA MYSQL

El sitio **hammerprinciple.com** ofrece una gran variedad de comparaciones de tecnologías, entre las cuales podemos encontrar una de las más buscadas: Redis contra MySQL. Para ver cada uno de los parámetros comparados podemos acceder a: **http://hammerprinciple.com/databases/items/redis/mysql**.

Hashes

Los hashes son lo más parecido a las estructuras de bases de datos relacionales. Un hash en Redis puede almacenar varios campos con sus respectivos valores dentro de una clave específica. Este tipo de datos es perfecto para representar objetos, por ejemplo usuarios, con campos como nombre, apellido, nombre de usuario y correo electrónico.

Redis provee un mecanismo de almacenamiento de hashes que requiere muy poco espacio, de manera que se pueden almacenar millones de objetos en una mínima instancia. Hablamos de almacenar objetos porque el tipo de dato hash es usado principalmente para este fin, pero también se pueden almacenar muchos otros elementos. Y, al igual que los list y los sets, cada hash puede almacenar 2^{32} -1, es decir más de 4 millones de pares campo-valor.

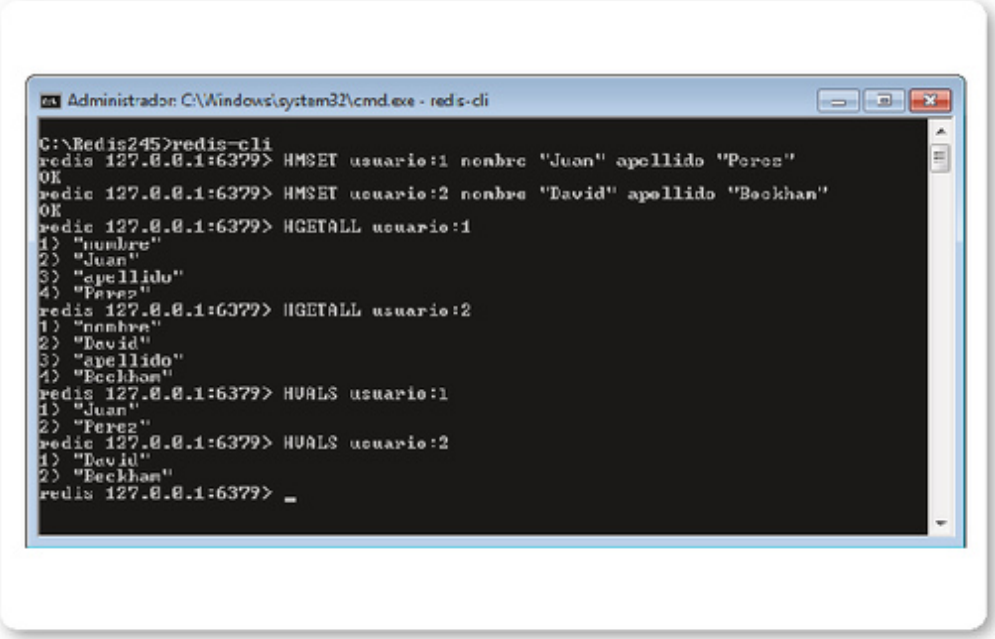


Figura 6. Ejemplo de almacenamiento de dos hashes y del modo en que podemos obtener los valores.

COMPUTACIÓN EN LA NUBE

Más comúnmente conocida como **cloud computing**, esta tecnología es cada vez más utilizada por las personas comunes, ya que, a diferencia de las computadoras personales, brinda a los usuarios la posibilidad de acceder a su información y a los programas dondequiera que estén y desde cualquier tipo de dispositivo.

Comandos

Debemos considerar que, como todo motor de base de datos, Redis dispone de comandos tanto para realizar la manipulación de los diferentes tipos de datos como para efectuar la configuración de parámetros del servidor, claves y opciones de conexión.

Es importante aclarar que para ejecutar cualquier comando debemos tener abierta una consola y estar ubicados en el directorio donde se encuentra nuestro motor, por ejemplo **C:\Redis245**.

En la siguiente tabla vamos a conocer los comandos más importantes que nos ofrece Redis, agrupados por categoría.

PRINCIPALES COMANDOS

CATEGORÍA	COMANDO	DESCRIPCIÓN
Servidor	BGREWRITEAOF	Reescribe asincrónicamente un archivo AOF (append-only file).
	BGSAVE	Guarda de manera asincrónica la base de datos que está en el disco duro.
	CLIENT KILL	Cierra la conexión a un cliente específico.
	CLIENT LIST	Devuelve información acerca de la conexión del cliente.
	CONFIG GET	Devuelve los valores de los parámetros de configuración del servidor.
	CONFIG SET	Establece valores para los parámetros de configuración del servidor.
	CONFIG RESETSTAT	Reinicia los valores devueltos por el comando INFO.
	DBSIZE	Devuelve la cantidad de claves en la base de datos seleccionada.
	FLUSHALL	Elimina todas las claves de todas las bases de datos existentes.
	FLISHDB	Elimina todas las claves de la base de datos actual.
	INFO	Devuelve información y estadísticas acerca del servidor.
	LASTSAVE	Devuelve la fecha en formato UNIX de la última vez que se guardó la base de datos en el disco.
	SAVE	Guarda sincrónicamente la base de datos en el disco.

Servidor	SHUTDOWN	Guarda sincrónicamente la base de datos en el disco y apaga el servidor.
	SLAVEOF	Cambia la configuración del servidor como esclavo de otro y, en caso de que ya sea un esclavo, lo cambia a maestro.
Conexión	AUTH	Solicita autenticación a un servidor Redis mediante una contraseña.
	ECHO	Imprime un mensaje.
	PING	Testea la conexión a un servidor.
	QUIT	Cierra una conexión.
	SELECT	Selecciona una base de datos para trabajar en la conexión actual.
Keys	DEL	Borra una clave.
	DUMP	Devuelve una versión serializada del valor almacenado en una clave especifica.
	EXISTS	Determina si una clave existe.
	EXPIRE	Configura el tiempo de vida en segundos de una clave.
	EXPIREAT	Configura la expiración de una clave en un formato UNIX timestamp.
	KEYS	Busca todas las claves que coinciden con un patrón.
	MIGRATE	Transfiere una clave de una instancia de Redis a otra.
	MOVE	Mueve una clave de base de datos a otra.
	PERSIST	Elimina el tiempo de expiración de una clave.
	PEXPIRE	Configura el tiempo de vida de una clave.
	PEXPIREAT	Configura el tiempo de vida en formato UNIX timestamp de una clave.
	PTTL	Obtiene el tiempo de vida de una clave en milisegundos.
	RANFOMKEY	Devuelve aleatoriamente una clave.
	RENAME	Renombra una clave.
	RENAMENX	Renombra una clave, solo si la nueva clave no existe.
	SORT	Ordena los elementos de una lista, set o sorted set.
	TTL	Obtiene el tiempo de vida de una clave.
	TYPE	Determina el tipo de clave.

Strings	APPEND	Agrega un valor a una clave.
	DECR	Decrementa el valor de una clave de tipo entero.
	DECRBY	Decrementa el valor de una clave de tipo entero mediante un valor establecido.
	GET	Devuelve el valor de una clave.
	GETBIT	Devuelve el valor del bit de una clave.
	GETRANGE	Obtiene una porción de un string almacenado en una clave.
	GETSET	Asigna un valor a una clave y devuelve el valor anterior.
	INCR	Incrementa en uno el valor de una clave de tipo entero.
	INCRBY	Incrementa en uno el valor de una clave de tipo entero mediante un valor establecido.
	INCRBYFLOAT	Incrementa el valor de una clave de tipo flotante mediante un valor establecido.
	MGET	Devuelve los valores de todas las claves especificadas.
	MSET	Asigna múltiples valores a múltiples claves.
	MSETNX	Asigna múltiples valores a múltiples claves, solo si ninguna clave existe.
	PSETEX	Asigna el valor y un tiempo de expiración en milisegundos de una clave.
	SET	Asigna un valor string a una clave.
	SETEX	Asigna el valor y un tiempo de expiración de una clave.
	SETNX	Asigna el valor de una clave, solo si la clave no existe.
	SETRANGE	Sobrescribe una porción de un string para una clave dada, especificando una posición de partida.
	STRLEN	Obtiene la longitud del valor de una clave.
Hashes	HDEL	Elimina uno o más campos de un hash.
	HEXISTS	Determina si existe un campo en una clave.
	HGET	Obtiene el valor de un campo de un hash.
	HGETALL	Obtiene todos los campos y valores de un hash.

Hashes	HINCRBY	Incrementa el valor entero de un campo mediante un valor establecido.
	HINCRBYFLOAT	Incrementa el valor flotante de un campo mediante un valor establecido.
	HKEYS	Devuelve todos los campos de un hash.
	HLEN	Obtiene el número de campos de un hash.
	HMGET	Obtiene los valores mediante los campos establecidos.
	HMSET	Establece múltiples campos con sus respectivos valores.
	HSET	Establece un valor a un campo específico.
	HSETNX	Establece un valor a un campo específico, solo si el campo no existe.
	HVALS	Obtiene todos los valores de un hash.
Lists	BLPOP	Obtiene y elimina el primer elemento de una lista, y en caso de que no existan elementos se bloquea la conexión.
	BRPOP	Obtiene y elimina el último elemento de una lista, y en caso de que no existan elementos se bloquea la conexión.
	BRPOPLPUSH	Obtiene y elimina un elemento de una lista, lo pone en otra lista y lo retorna. En caso de que no existan elementos se bloquea la conexión.
	LINDEX	Obtiene un elemento de una lista mediante un índice especificado.
	LINSERT	Inserta un elemento antes o después de otro elemento.
	LLEN	Obtiene la longitud de una lista.
	LPOP	Elimina y devuelve el primer elemento de una lista.
	LPUSH	Agrega uno o más elementos al principio de una lista.
	LPUSHX	Agrega uno o más elementos al principio de una lista, solo si la lista existe.
	LRANGE	Obtiene un rango de elementos de una lista.
	LREM	Elimina elementos de una lista.
	LSET	Asigna un valor a un elemento de una lista mediante un índice.
	LTRIM	Recorta un elemento de una lista mediante un rango especificado.

Lists	RPOP	Elimina y obtiene el último elemento de una lista.
	RPOPLPUSH	Elimina el último elemento de una lista, lo pone en otra lista y lo retorna.
	RPUSH	Agrega uno o más elementos a una lista.
	RPUSHX	Agrega uno o más elementos a una lista, solo si la lista existe.
Sets	SADD	Agrega uno o más miembros a un set.
	SCARD	Obtiene el número de elementos de un set.
	SDIFF	Devuelve los miembros de un set que resultan de una diferencia entre el primer set y los sets sucesivos.
	SDIFFSTORE	Es similar a SDIFF pero en lugar de retornar el set resultante lo almacena en una clave.
	SINTER	Realiza una intersección entre múltiples sets.
	SINTERSTORE	Realiza una intersección entre múltiples sets y almacena el set resultante en una clave.
	SISMEMBER	Determina si un valor obtenido es un miembro de un set.
	SMEMBERS	Obtiene todos los miembros de un set.
	SMOVE	Mueve un miembro de un set a otro.
	SPOP	Elimina y retorna un elemento aleatorio de un set.
	SRANDMEMBER	Obtiene uno o múltiples miembros aleatoriamente de un set.
	SREM	Elimina uno o más miembros de un set.
	SUNION	Devuelve los miembros resultantes de la unión de los sets especificados.
	SUNIONSTORE	Devuelve los miembros resultantes de la unión de los sets especificados y los almacena en una clave.
Sorted Sets	ZADD	Agrega uno o más miembros a un sorted set o actualiza su puntuación en caso de que el miembro ya exista.
	ZCARD	Obtiene el número de miembros de un sorted set.
	ZCOUNT	Obtiene el número de elementos con su puntuación mediante un rango establecido.
	ZINCRBY	Incrementa la puntuación de un miembro.

Sorted Sets	ZINTERSTORE	Realiza una intersección entre múltiples sorted sets y almacena el resultado en una nueva clave.
	ZRANGE	Retorna un rango de miembros de un sorted set ordenados por clave.
	ZRANGEBYSCORE	Retorna un rango de miembros de un sorted set ordenados por puntuación.
	ZRANK	Determina el índice de un miembro en un sorted set.
	ZREM	Elimina uno o más miembros de un sorted set.
	ZREMRANGEBYRANK	Elimina todos los miembros de un sorted set mediante índices establecidos.
	ZREMRANGEBYSCORE	Elimina todos los miembros de un sorted set mediante puntuaciones establecidas.
	ZREVRANGE	Retorna un rango de miembros con sus puntuaciones ordenado de mayor a menor mediante índices establecidos.
	ZREVRANGEBYSCORE	Retorna un rango de miembros con sus puntuaciones ordenado de mayor a menor mediante puntuaciones establecidas.
	ZREVRANK	Retorna el índice de un miembro con su puntuación ordenado de mayor a menor.
	ZSCORE	Obtiene la puntuación asociada a un miembro.
	ZUNIONSTORE	Une múltiples sorted sets y almacena el resultado en una nueva clave.
Pub/Sub	PSUBSCRIBE	Escucha mensajes publicados de canales que coinciden con los patrones establecidos.
	PUBLISH	Publica mensajes en un canal.
	PUNSUBSCRIBE	Detiene el proceso de escucha de mensajes publicados de canales que coinciden con los patrones establecidos.
	SUBSCRIBE	Escucha mensajes publicados de canales que coinciden con los canales establecidos.
	UNSUBSCRIBE	Detiene el proceso de escucha de mensajes que coinciden con los canales establecidos.

Tabla 1. Principales comandos de Redis.

Además de los comandos que vimos anteriormente, disponemos de otros que debemos considerar para ser usados en los clientes. Si escribimos **redis-cli -h** se muestran diferentes opciones interesantes. Entre los comandos más importantes se encuentran los que mencionamos a continuación:

> LA OPCIÓN REDIS-CLI -H NOS MUESTRA UNA SERIE DE COMANDOS ÚTILES

- **-h <hostname>**: este comando es adecuado para conectarnos a una instancia de Redis que haya sido instalada en un servidor remoto.
- **-p <port>**: se trata del comando que permite especificar un puerto diferente al puerto por defecto que es **6379**.
- **-a <password>**: para especificar la clave que se usará cuando intentemos conectarnos al servidor.
- **--version**: muestra la versión de Redis que tenemos instalada.

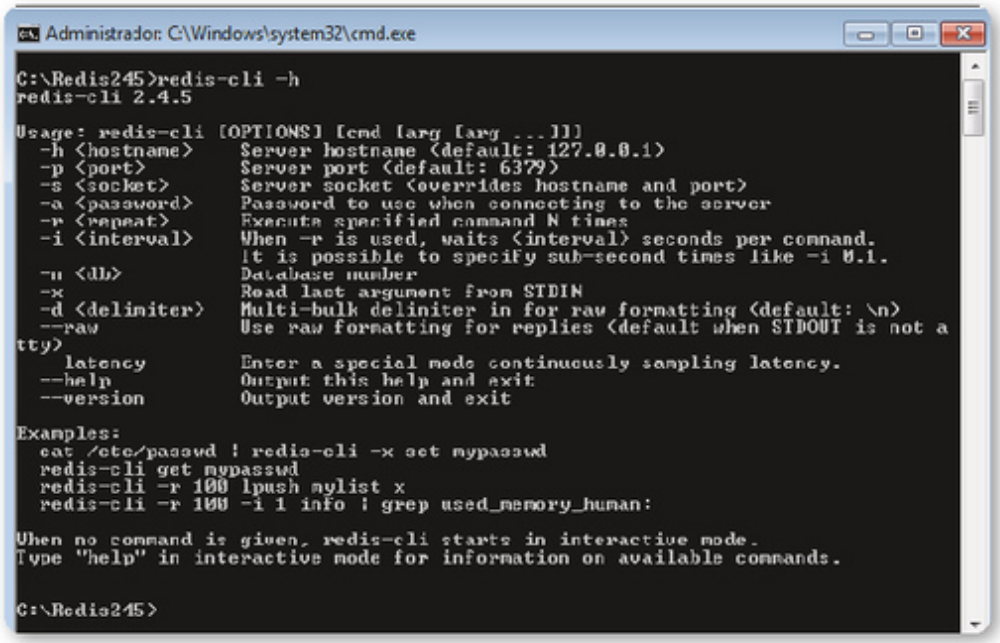


Figura 7. Listado de los comandos que se encuentran disponibles luego de ejecutar **`redis-cli -h`**.

Debemos tener en cuenta que, para ver la lista completa de comandos, debemos acceder al sitio oficial de Redis y buscar entre sus vínculos. También podemos hacerlo directamente desde la siguiente ditrección web: **`http://redis.io/commands`**.

Publicación y suscripción

Redis implementa el paradigma de publicación y suscripción de mensajes, el cual define que los publicadores no son programados para enviar mensajes a receptores específicos (suscriptores), sino que los mensajes publicados solo pertenecen a canales que no tienen conocimiento de qué o quién los recibirá. Por otro lado, los suscriptores solo recibirán los mensajes de los canales a los cuales están suscriptos sin conocer qué publicador los envía.

Como vimos en la tabla de comandos, **SUBSCRIBE**, **UNSUBSCRIBE** y **PUBLISH** son implementados para establecer la comunicación entre los suscriptores y los publicadores. Para entender mejor lo que estamos comentando vamos a desarrollar un pequeño ejemplo.

Para todos los casos es necesario abrir una consola y situarnos en el directorio donde tenemos la instancia de Redis, por ejemplo, **C:\Redis245**. Primero ejecutamos el comando **redis-cli** y, una vez iniciado el cliente, nos suscribimos al canal **canal_uno**:

```
C:\Redis245>redis-cli
redis 127.0.0.1:6379> SUBSCRIBE canal_uno
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "canal_uno"
3) (integer) 1
```

Luego, en otra ventana de comandos, ejecutamos el comando **redis-cli** y publicamos un mensaje en **canal_uno**:

```
C:\Redis245>redis-cli
redis 127.0.0.1:6379> PUBLISH canal_uno "hola mundo"
(integer) 1
redis 127.0.0.1:6379>
```

Después de haber publicado el mensaje, automáticamente lo reciben todos los clientes suscriptos a este canal:

```
C:\Redis245>redis-cli
redis 127.0.0.1:6379> SUBSCRIBE canal_uno
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "canal_uno"
3) (integer) 1
1) "message"
2) "canal_uno"
3) "hola mundo"
```

Consideremos que, cuando deseemos dejar de recibir mensajes de este canal, solo debemos desuscribirnos del siguiente modo:

```
C:\Redis245>redis-cli
redis 127.0.0.1:6379> UNSUBSCRIBE canal_uno
1) "unsubscribe"
2) "canal_uno"
3) (integer) 0
redis 127.0.0.1:6379>
```

Formato de los mensajes

Dado que los mensajes necesitan retornar tres valores, son de tipo **Multi-bulk reply**, los cuales tienen la capacidad de contener múltiples elementos en la misma salida. El primer elemento indica el tipo de mensaje, que puede ser **subscribe**, **unsubscribe** o **message**.

REDIS CONTRA MONGODB

Si bien Redis y MongoDB son dos motores de bases de datos de tipo NoSQL, tienen características diferentes que determinan cuándo usar cada uno. En **hammerprinciple.com** podemos ver una comparación entre ambos, que es útil consultar para futuros desarrollos: **http://hammerprinciple.com/databases/items/redis/mongodb**.

- **subscribe**: muestra que nos hemos suscripto con éxito al canal que tenemos como segundo elemento, mientras que el tercero indica la cantidad de canales a los cuales estamos suscriptos.

```
1) "subscribe"
2) "canal_uno"
3) (integer) 1
```

- **unsubscribe**: indica que nos hemos desuscripto con éxito del canal que tenemos como segundo elemento. El tercero indica la cantidad de canales a los cuales estamos suscriptos. Si la cantidad es igual a cero, quiere decir que no estamos suscriptos a ningún canal y el cliente puede ejecutar cualquier tipo de comando.

```
1) "unsubscribe"
2) "canal_uno"
3) (integer) 0
```

- **message**: este es el mensaje recibido como resultado del comando **PUBLISH** ejecutado por otro cliente. El segundo elemento indica el canal del cual proviene el mensaje y el tercero es el mensaje propiamente dicho.

```
1) "message"
2) "canal_uno"
3) "hola mundo"
```

HOJAS DE AYUDA

Es muy común que necesitemos ejecutar algún comando, acceder a una propiedad o ejecutar un método propio del lenguaje, y no recordemos la sintaxis. En el enlace que encontramos en la siguiente dirección, es posible realizar la descarga de varias hojas de ayuda, que pueden ser de gran utilidad: **http://creativasfera.com/las-mejores-cheat-sheets-para-desarrollo-web**.

Suscripciones mediante un patrón de coincidencia

Redis soporta la suscripción mediante un patrón de coincidencia en el nombre del canal. Esto quiere decir que los clientes pueden suscribirse mediante patrones globales, de modo que reciben todos los mensajes coincidentes con el patrón establecido.

El asterisco en el patrón indica un comodín, es decir que puede tomar cualquier valor, por ejemplo el patrón **canal_*** coincide con **canal_uno**, **canal_dos** y **canal_de_comunicacion**. Veamos un ejemplo: un cliente se suscribe a todos los canales que contengan el patrón **canal_***:

```
C:\Redis245>redis-cli
redis 127.0.0.1:6379> PSUBSCRIBE canal_*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "canal_*"
3) (integer) 1
```

Otro cliente publica un mensaje para el **canal_uno** y el **canal_dos**:

```
C:\Redis245>redis-cli
redis 127.0.0.1:6379> PUBLISH canal_uno "hola mundo"
(integer) 1
redis 127.0.0.1:6379> PUBLISH canal_dos "hola amigos!"
(integer) 1
redis 127.0.0.1:6379>
```

APLICACIONES MOBILE CON HTML5

Debido a la gran variedad de sistemas operativos que existen, es cada vez mayor el desafío de desarrollar aplicaciones para los dispositivos móviles. La esperanza en este ámbito es **HTML5**, ya que permite desarrollar aplicaciones no nativas para todas las plataformas que están basadas en la Web.

El cliente suscripto al canal con el patrón **canal_*** recibe ambos mensajes, el que proviene de **canal_uno** y el que proviene **canal_dos**. Los parámetros del mensaje son cuatro: el primero es el tipo (o sea **pmessage**), el segundo es el patrón coincidente a la suscripción, el tercero es el nombre del canal que envió el mensaje y el cuarto es el mensaje propiamente dicho.

```
C:\Redis245>redis-cli
redis 127.0.0.1:6379> PSUBSCRIBE canal_*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "canal_*"
3) (integer) 1
1) "pmessage"
2) "canal_*"
3) "canal_uno"
4) "hola mundo"
1) "pmessage"
2) "canal_*"
3) "canal_dos"
4) "hola amigos!"
```

Con el comando **PUNSUBSCRIBE**, un cliente se desuscribe de los canales coincidentes con el patrón establecido:

```
C:\Redis245>redis-cli
redis 127.0.0.1:6379> PUNSUBSCRIBE canal_*
1) "punsubscribe"
2) "canal_*"
3) (integer) 0
redis 127.0.0.1:6379>
```

Así, un cliente se desuscribe del canal coincidente con el patrón **canal_*** y obtiene un mensaje donde el primer parámetro es el tipo, el segundo es el patrón coincidente y el tercero es la cantidad de canales a los cuales sigue suscripto.

Suscripciones coincidentes con canales y con patrones

Los clientes que estén suscriptos a varios patrones coincidentes entre sí, o a canales mediante patrones y canales simples, recibirán mensajes duplicados. Podemos verlo en el ejemplo siguiente:

```
SUNSCRIBE canal_uno
PSUBSCRIBE canal*
```

En este caso, si un mensaje es enviado al **canal_uno**, el cliente recibirá dos mensajes, uno de tipo **message** y otro de tipo **pmessage**.

RESUMEN

Hemos empezado a analizar Redis, ya que es indispensable conocer sus características además de cómo y cuándo usarlo. Este motor de base de datos fue desarrollado para funcionar en entornos UNIX, aunque existe soporte no oficial para funcionar en Windows como entorno de desarrollo. Dispone de tipos de datos propios que brindan una excelente performance tanto para almacenamiento como para recuperación de datos. Una característica importante es el mecanismo de publicación y suscripción, a partir del cual se pueden tener canales de comunicación entre clientes de manera muy simple, como por ejemplo, un sistema de mensajería instantánea.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Dónde operan las bases de datos creadas en Redis?
2. ¿En qué tipo de almacenamiento se basa Redis?
3. ¿Redis tiene soporte para Windows?
4. ¿Se puede configurar una instancia de Redis como esclavo de otro?
5. ¿**Integer** es un tipo de dato definido en Redis?
6. ¿Es posible almacenar un archivo PDF en Redis?
7. ¿Los espacios de nombres para las claves son recomendados?
8. ¿Se puede usar el carácter **:** como separador de espacios de nombres?
9. ¿Varios clientes pueden suscribirse al mismo tiempo en un canal? ¿Por qué?
10. ¿Es posible que un cliente se suscriba a un canal sin conocer su nombre?

EJERCICIOS PRÁCTICOS

1. Mencione los tipos de datos que ofrece Redis.
2. Ejecute diez comandos para cada tipo de dato string, list, set, sorted set y hash.
3. Abra varias ventanas de comando y cree un canal de comunicación utilizando las funciones de publicación/suscripción.
4. Exponga las características que hacen que Redis tenga gran rendimiento.
5. Cree una estructura similar a un contador de visitas, teniendo como dato un nombre y el contador.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*03

Clientes Redis

Aprenderemos a trabajar con Redis y con el lenguaje de programación PHP. Con este objetivo en mente, estudiaremos dos clientes pero enfocándonos solo en uno de ellos, que nos servirá para desarrollar todos los ejercicios de aquí en adelante. También veremos cómo utilizar una interfaz web de administración y, por último, crearemos un ejemplo simple para aplicar lo estudiado.

- **Clientes PHP 76**
  - Predis 76
  - phpredis 77
  - phpRedisAdmin 82
- **Ejercicio de prueba con PHP 86**
  - Creación del archivo index.html 90
  - Creación del archivo style.css 97
  - Creación del archivo script.js 103
  - Creación del archivo funciones.php 112
- **Resumen 123**
- **Actividades 124**

Clientes PHP

Antes de comenzar a estudiar cada uno de los clientes, debemos dar su definición: un cliente es llamado por la interfaz que permite la comunicación entre el lenguaje de programación y la base de datos. Como mencionamos en el **Capítulo 2**, Redis ofrece una amplia lista de clientes para los lenguajes más populares, como C, Go, Java, Node.js, Perl, Python, Ruby y PHP. Para ver la lista completa podemos acceder a **http://redis.io/clients**,donde encontraremos varias alternativas para cada lenguaje.

Predis

Predis se presenta como un cliente maduro, soportado y flexible que requiere tener instalada una versión de PHP igual o superior a 5.3, pero que no necesita extensiones adicionales. Entre las características principales de este cliente se encuentran:

- Provee soporte estable desde la versión 1.2 hasta la 2.6 y también para las versiones de Redis que aún no son estables.
- Ofrece soporte para **redis-cluster**.
- Brinda soporte para replicación maestro/esclavo.
- Se pueden ejecutar comandos solapados o simples.
- Tiene soporte de transacciones.
- Se conecta a Redis usando los protocolos TCP/IP o mediante sockets en UNIX, el cual permite conexiones persistentes.
- Ofrece la posibilidad de conectarse a diferentes tipos de redes o protocolos mediante el uso de clases de conexión.
- Brinda un sistema flexible para definir y registrar su propio conjunto de comandos y perfiles del servidor al cliente.

ARQUITECTURAS TECNOLÓGICAS

Las arquitecturas tecnológicas que se utilizaban para desarrollar aplicaciones no son suficientemente escalables como para soportar millones de usuarios, apoyándose en productos innovadores como Redis, que sustituyen a las antiguas arquitecturas cliente-servidor y las bases de datos relacionales.

Predis puede ser descargado de **GitHub**, desde la dirección **https://github.com/nrk/predis**. En esta dirección encontraremos las indicaciones para su instalación y configuración.

phpredis

phpredis es un cliente escrito en C que ofrece un muy buen rendimiento, ya que se integra de manera directa al lenguaje PHP. Fue desarrollado y mantenido por **Owlient** desde fines de 2009, y a principios de 2011 fue liberado bajo la licencia **PHP License**, versión 3.01. A continuación veremos cómo instalarlo en Linux y en Windows.

Instalación en GNU/Linux

Para instalar phpredis, primero debemos descargarlo del sitio **GitHub** desde la dirección: **https://github.com/nicolasff/phpredis**. Luego, nos ubicamos en el directorio donde lo guardamos, y ejecutamos en una consola los comandos que detallamos a continuación (es importante aclarar que, como se compilará un módulo de PHP, es necesario que tengamos instalado **php5-dev**):

```
phpize
./configure
make&& make install
```

Si los comandos anteriores se ejecutaron correctamente, vamos a obtener un mensaje como el siguiente:

```
Build complete.
Don't forget to run 'make test'.
Installing shared extensions: /usr/lib/php5/20060613/
```

Para finalizar, debemos cargar el nuevo módulo a PHP. Buscamos el archivo **php.ini**, lo editamos y agregamos **extension=redis.so** en la sección **Dynamic Extensions**. Luego, reiniciamos Apache con el comando **sudo service apache2 restart**. Para verificarlo, escribimos en una consola lo siguiente:

```
php -m | grep redis
php -i | grep 'Redis Support'
```

Para comprobar si hemos instalado correctamente la extensión creamos un archivo **PHP** con el nombre **phpinfo.php** y escribimos el código que mostramos a continuación:

```
<?
        phpinfo();
?>
```

A continuación, accedemos desde un navegador al archivo **phpinfo.php** y buscamos la palabra **redis**. Deberemos obtener algo similar a lo que muestra la imagen que vemos a continuación.

Figura 1. Redis se ha cargado como extensión de PHP y ahora podemos ejecutar los comandos disponibles desde el lenguaje.

Instalación en Windows

La instalación en Windows suele ser un poco más sencilla que en sistemas UNIX, porque la librería ya se encuentra compilada. Debemos

dirigirnos a la siguiente dirección: **https://github.com/nicolasff/phpredis/downloads** y descargar el archivo correspondiente a nuestra versión de PHP. Por ejemplo, para la versión 5.2.6, descargamos el archivo **php_redis-2.1.3-5.2-vc6-ts-4350b2a.zip**. Si observamos el nombre, luego de **php_redis-2.1.3-** viene **5.2**, que coincide con la versión de PHP para la cual fue compilada.

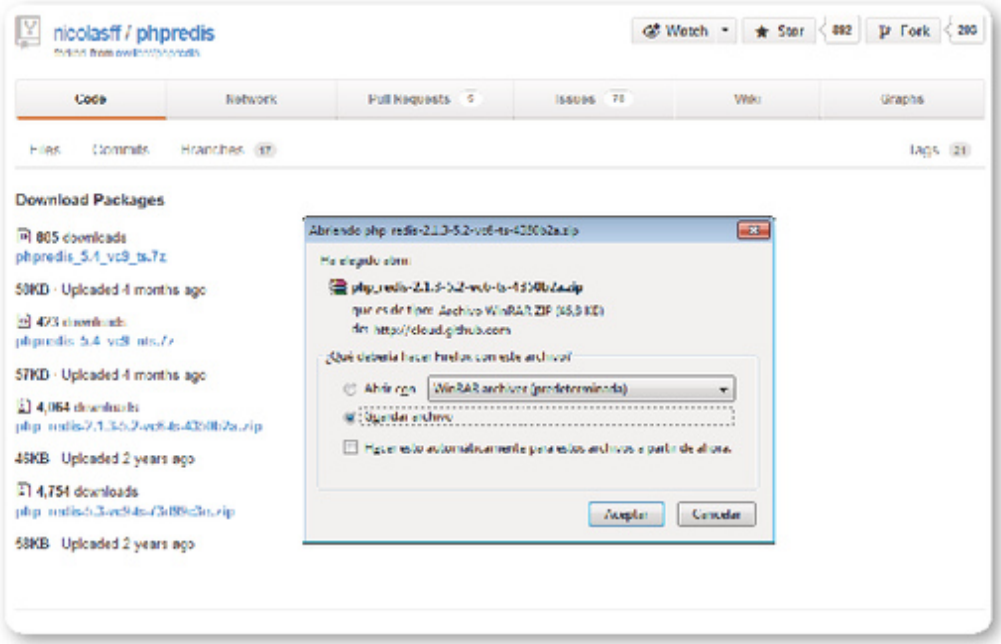


Figura 2. La librería phpredis se encuentra compilada para sistemas Windows, solo es cuestión de descargarla y agregarla a PHP.

Una vez descargado el archivo, lo abrimos y extraemos **php_redis.dll** dentro del directorio de extensiones de PHP, por ejemplo **C:\AppServ\php5\ext**. Luego, agregamos la nueva extensión a PHP. Para esto, localizamos el archivo **php.ini**, lo abrimos y agregamos el módulo **extension=php_redis.dll** en la sección **Dynamic Extensions**.

DETECTAR SOPORTE DE HTML5 Y CSS3

Cada día es necesario aplicar nuevas características a nuestros proyectos, pero con la gran diversidad de navegadores la compatibilidad de funciones suele ser un problema. Para esto existe **Modernizr**, una librería JavaScript que detecta el soporte que tiene el navegador del usuario y nos ayuda a tener el control sobre el uso de HTML5 y CSS3. Más información en: **http://modernizr.com**.

Para verificar que se ha realizado la instalación de la extensión de manera correcta procedemos a crear un archivo **PHP** con el nombre **phpinfo.php** y escribimos lo siguiente:

```
<?
        phpinfo();
?>
```

Luego, accedemos desde un navegador al archivo **phpinfo.php** y buscamos la palabra **redis**. Debemos ver lo que muestra la **Figura 3**.

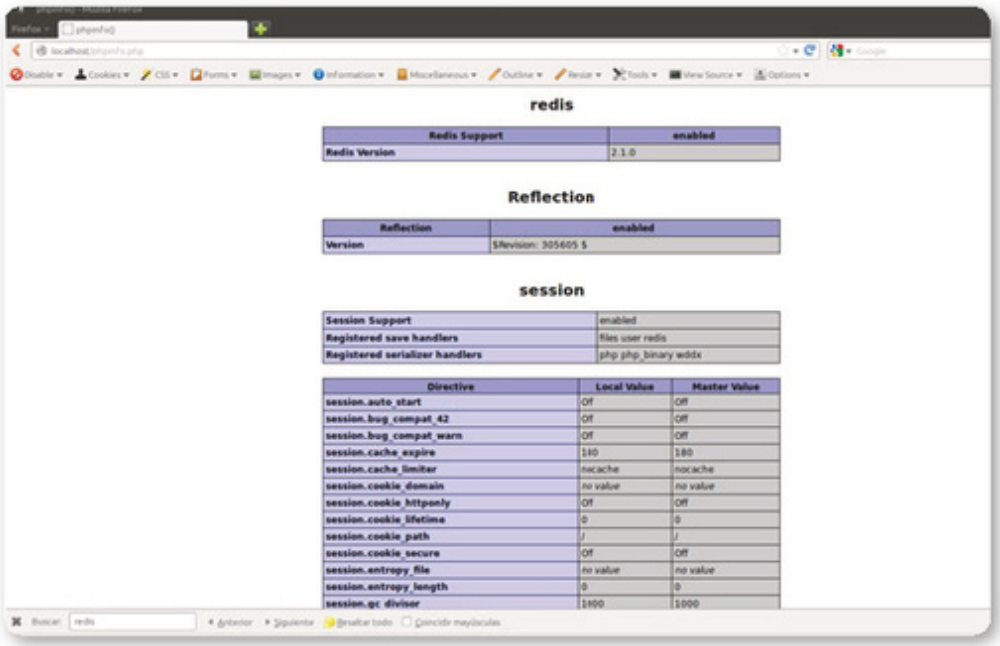


Figura 3. PHP ya se puede comunicar con Redis mediante la librería que hemos agregado.

DISEÑOS PLANOS

Cada vez son más las empresas que optan por rediseñar el estilo de sus sitios, proponiendo un diseño más limpio, sencillo y minimalista con colores que contrastan, donde las fuentes por lo general son más grandes y legibles. Las sombras, degradados, elementos con volúmenes, relieves y texturas ya no son usados en el diseño web actual.

Verificar el funcionamiento

Ahora que tenemos instalado Redis, funcionando con PHP, vamos a escribir nuestro primer programa, que nos servirá para verificar que todo funciona como debería. Para esto, creamos un archivo y lo guardamos en el directorio público de Apache, y lo nombramos **test-phpredis.php**.

```
<?php

$redis = new Redis();
try {
        $redis->connect('127.0.0.1', 6379);
        $redis->set('saludo', 'Hola Mundo!');
        $saludo_valor = $redis->get('saludo');

        if($saludo_valor){
                echo "<h1>Test cliente phpredis:</h1>";
                echo "1. La clave 'saludo' almacena el valor: '{$saludo_valor}'.";
                if($redis->delete('saludo'))
                        echo "2. Hemos borrado la clave 'saludo'";
        }else
                echo "imposible obtener 'saludo'.";
} catch (RedisException $e){
        echo $e->getMessage();
}

?>
```

Vamos a describir qué hace este primer ejemplo:

- En la línea 3 instanciamos Redis para hacer uso de sus comandos. Luego, generamos un fragmento **try/catch** para manejar las excepciones. En la línea 5 intentamos conectarnos a Redis especificando la dirección IP y el puerto del servidor. Como estamos intentando conectarnos al servidor local usamos la IP **127.0.0.1** y el puerto por defecto, que es **6379**.
- En la línea 6, mediante el comando **set**, almacenamos el valor **Hola Mundo!** en la clave **saludo**.

- En la línea 7, a través del comando **get**, obtenemos el valor que hemos almacenado en la clave **saludo** y lo asignamos a la variable **saludo_valor**.
- En la línea 9 verificamos si la variable contiene un valor devuelto por Redis. En el caso contrario damos un mensaje de error.
- En la línea 11 solo imprimimos un encabezado y en la siguiente mostramos el valor que contiene la clave.
- En las líneas 13 y 14 eliminamos la clave y damos un mensaje.

Para ejecutar el programa, accedemos a través de un navegador.

Figura 4. Aquí vemos una implementación básica para el tipo de datos **`string`** de Redis.

En este ejemplo hemos visto cómo asignar, obtener y eliminar una clave en Redis. Es una buena idea probar diferentes comandos con todos los tipos de datos en el mismo archivo, para ir familiarizándonos con la sintaxis y los métodos disponibles.

phpRedisAdmin

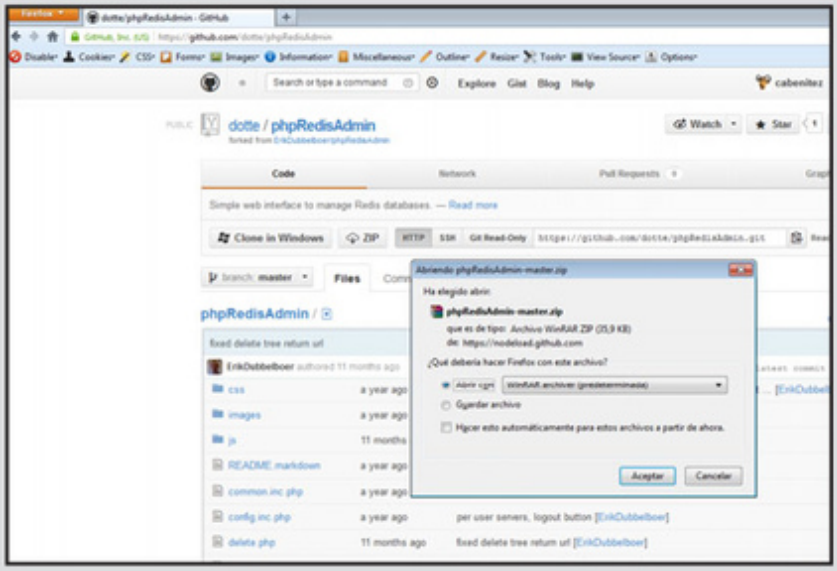
Para realizar el manejo de cualquier base es necesario contar con una interfaz de administración que nos brinde la posibilidad de visualizar y operar con los datos.

Para administrar Redis, usaremos **phpRedisAdmin**, que está escrita en PHP y, por lo tanto, funciona en un entorno web. El proyecto original ha sido modificado por su creador y ahora utiliza Predis para su funcionamiento. Como ya hemos compilado el módulo **phpredis** no vamos a utilizar esta versión sino una paralela mantenida por **Dotte**.

A continuación, veremos cómo instalar y configurar phpRedisAdmin.

PAP: INSTALACIÓN DE PHPREDISADMIN

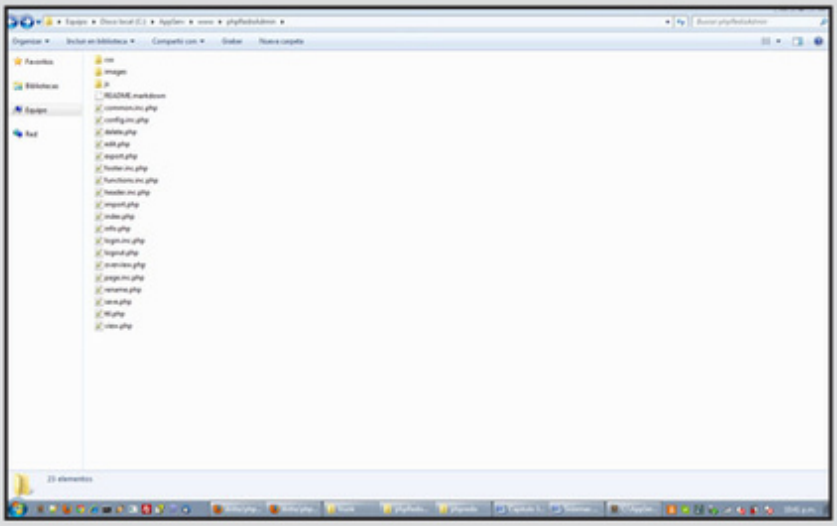
01 Ingrese a **https://github.com/dotte/phpRedisAdmin**. Aquí podrá clonar el proyecto utilizando Git o descargarlo haciendo clic en el botón ZIP.



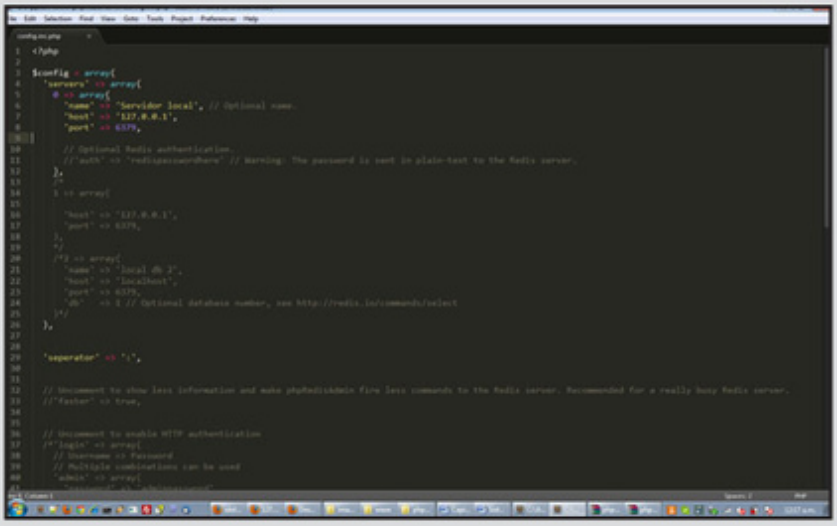
CUÁNTO COBRAR LOS PROYECTOS

La tarea de definir cuánto cobrar por el desarrollo de un proyecto es difícil, ya que por lo general no sabemos cómo medir los gastos ni tampoco el esfuerzo necesario para completarla. Para apoyarnos, el sitio que se encuentra en la dirección **http://freelanceswitch.com** ofrece una calculadora basada en costos personales, que establece un valor sobre la base de los parámetros calculados. Podemos utilizar esta herramienta en: **http://freelanceswitch.com/rates**.

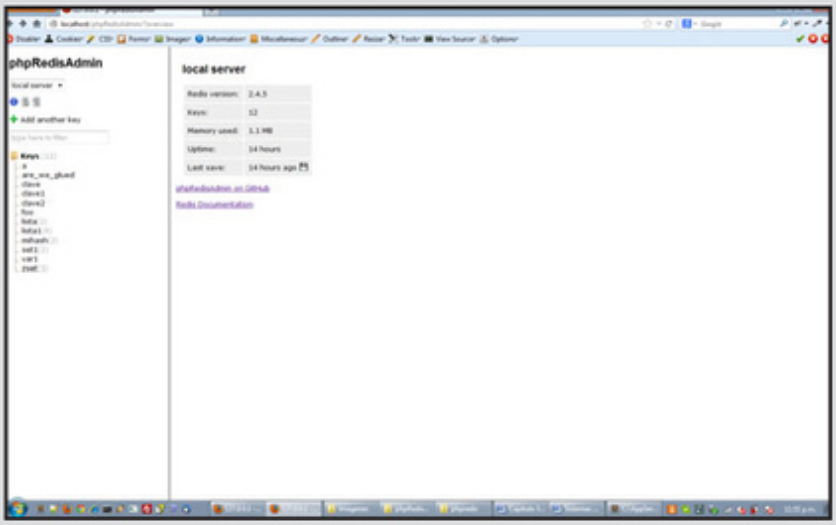
02 Una vez descargado phpRedisAdmin, extraiga todos los archivos en la carpeta pública de su servidor web.



03 Abra el archivo `config.inc.php` con un editor de código y establezca los parámetros `name` (nombre del servidor), `host` (IP del servidor, `127.0.0.1` para una conexión local) y `port` (puerto donde opera Redis, por defecto 6379). Si estableció una clave, deberá descomentar el parámetro `auth` e ingresar su clave.



04 Una vez establecidos los parámetros de conexión, acceda desde su navegador web al directorio público donde descomprimió phpRedisAdmin. Verá una interfaz similar a la siguiente:

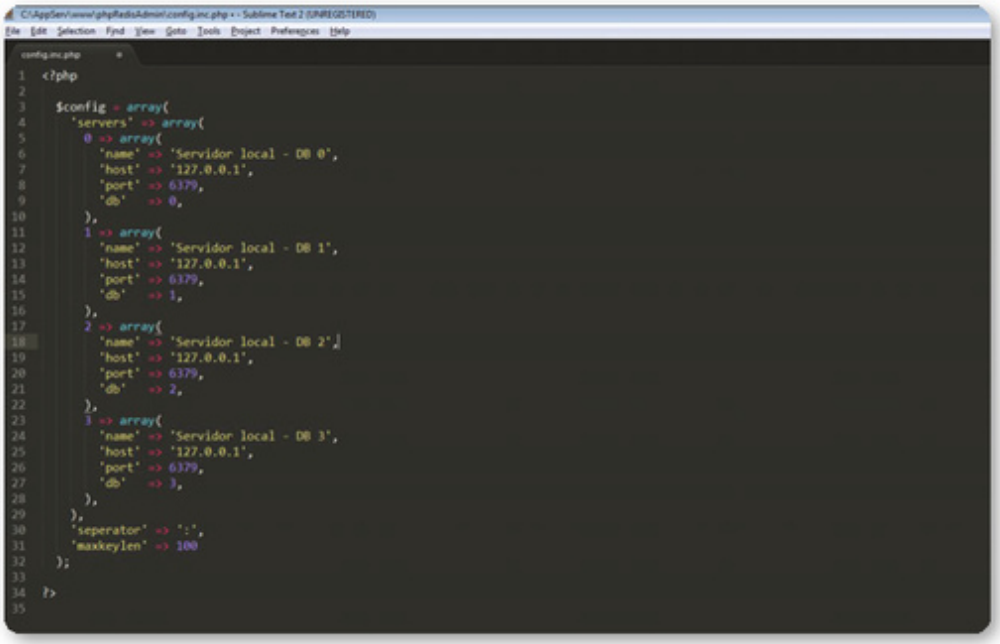


Ahora que tenemos la interfaz de administración podemos ver las claves almacenadas con sus respectivos valores, la información completa del servidor, la cantidad de claves definidas y el espacio en memoria usado, e incluso podemos exportar e importar datos.

Es importante aclarar que, si bien phpRedisAdmin se presenta como una interfaz muy sencilla, posee la capacidad suficiente para administrar varias bases de datos. Solo debemos editar el archivo de configuración y agregar un servidor más, indicándole el número de la base de datos que vamos a administrar.

DIEZ PASOS DEL DISEÑO DE UN SITIO WEB

Según **creativasfera.com**, el desarrollo de un sitio web debe hacerse siguiendo diez pasos: definir los objetivos, definir los contenidos, hacer un modelo en papel, crear un wireframe, crear un diseño en Photoshop, crear el HTML, crear el CSS, verificar la compatibilidad en navegadores, validar el código y subir el sitio al servidor web.



```
<?php

  $config = array(
    'servers' => array(
      0 => array(
        'name' => 'Servidor local - DB 0',
        'host' => '127.0.0.1',
        'port' => 6379,
        'db'   => 0,
      ),
      1 => array(
        'name' => 'Servidor local - DB 1',
        'host' => '127.0.0.1',
        'port' => 6379,
        'db'   => 1,
      ),
      2 => array(
        'name' => 'Servidor local - DB 2',
        'host' => '127.0.0.1',
        'port' => 6379,
        'db'   => 2,
      ),
      3 => array(
        'name' => 'Servidor local - DB 3',
        'host' => '127.0.0.1',
        'port' => 6379,
        'db'   => 3,
      ),
    ),
    'seperator' => ':',
    'maxkeylen' => 100
  );

?>
```

Figura 5. Para administrar más de una base de datos en phpRedisAdmin es necesario agregarla al archivo de configuración.

Ejercicio de prueba con PHP

Vamos a desarrollar un ejercicio donde utilizaremos algunos de los comandos disponibles para cada tipo de dato, para ir familiarizándonos con la utilización de los métodos que ofrece Redis y su uso mediante PHP y phpredis.

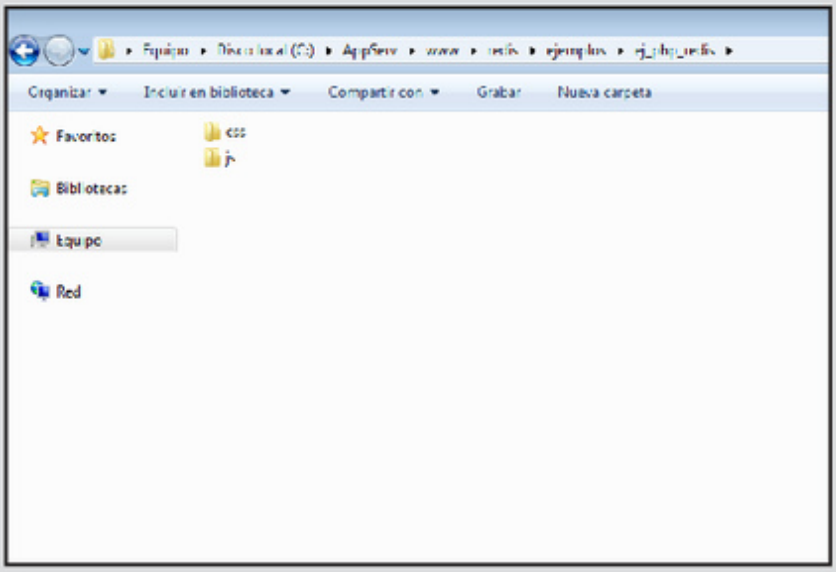
A continuación, definiremos los pasos para crear el entorno de trabajo; para ello, explicaremos en detalle cada una de las acciones que es necesario ejecutar antes de comenzar a trabajar:

HTML5 ROCKS

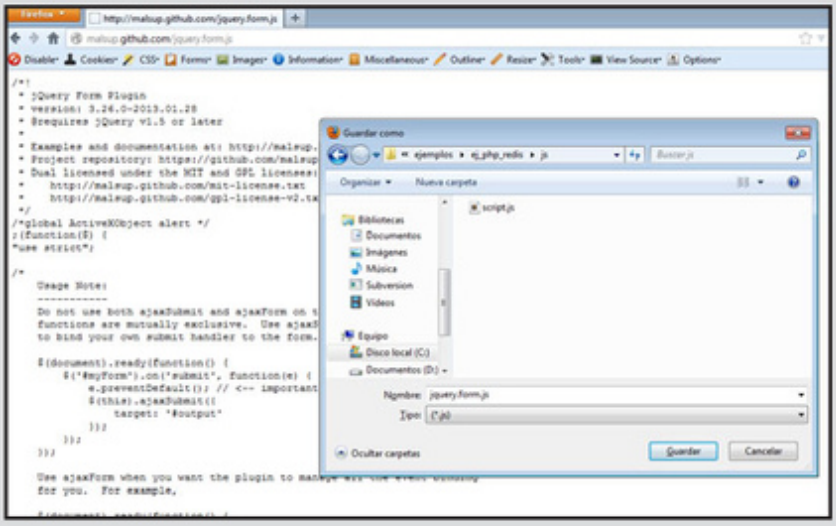
HTML5 ROCKS es uno de los mejores sitios web que existen para aprender la última versión de HTML, ya que cuenta con tutoriales donde podemos diferenciar el formato, artículos, demos, ejemplos y tecnologías como conectividad, semántica y almacenamiento. Para conocer más sobre el tema podemos acceder a la página que se encuentra en la dirección **http://html5rocks.com**.

PAP: CREAR ENTORNO PARA EL EJERCICIO DE PRUEBA

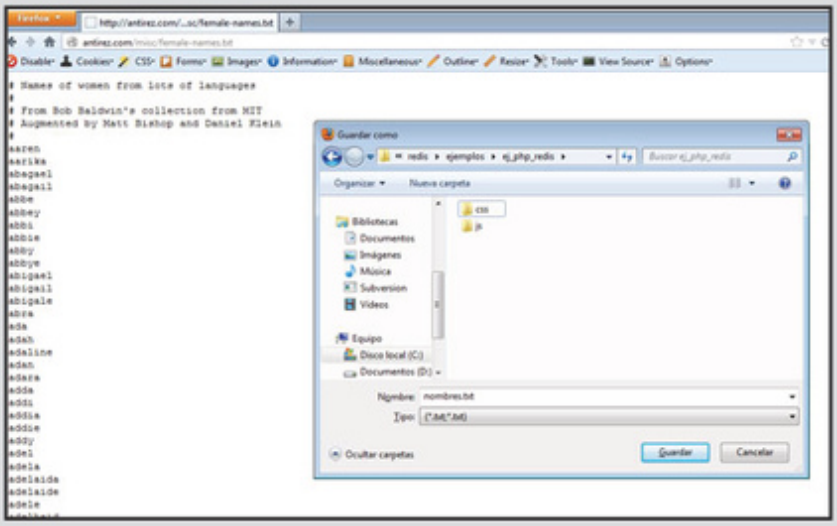
01 Posiciónese en el directorio público de su servidor web y cree una carpeta llamada `ej_php_redis`. Dentro, cree dos carpetas más, una `css` y otra `js`.



02 Diríjase al enlace **http://malsup.github.com/jquery.form.js** y descargue el archivo dentro de la carpeta `js` que ha creado en el paso anterior.



03 Ingrese a **http://antirez.com/misc/female-names.txt** y descargue el archivo en la carpeta `ej_php_redis`, y cambie el nombre por `nombres.txt`.



Una vez que hemos creado la estructura de directorio y hemos descargado los archivos necesarios estamos listos para empezar a codificar. En la **Figura 6** vemos cómo será el aspecto del proyecto.

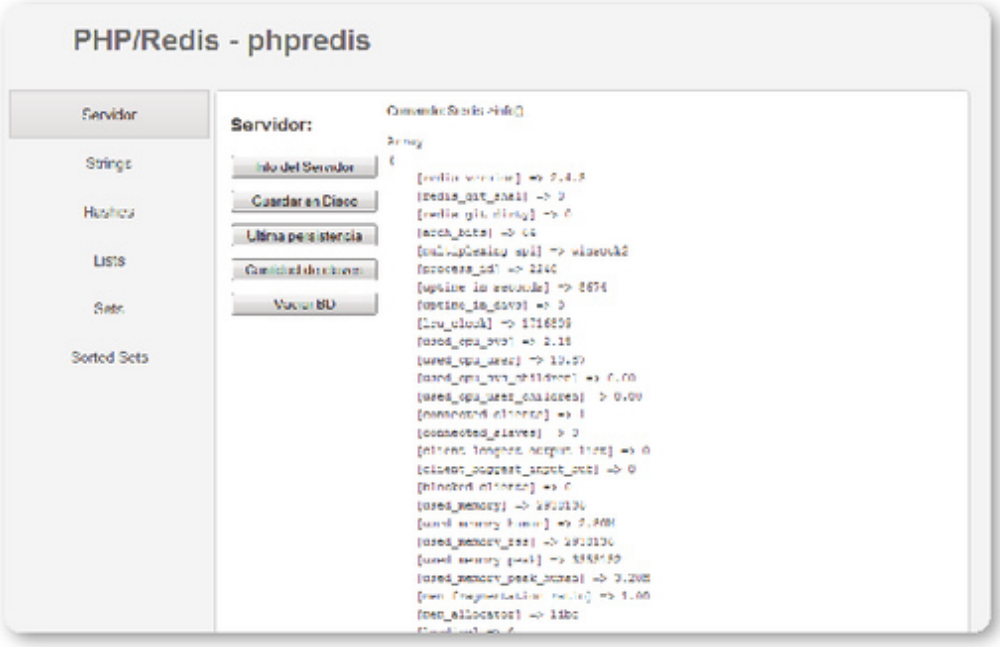


Figura 6. Ejemplo de PHP con Redis. Aquí vemos la información del servidor.

La estructura de directorio deberá quedar como la siguiente (necesitaremos crear cinco archivos):

```
ej_php_redis
        |_ css
        |   |_ style.css
        |
        |_ js
        |   |_ jquery.form.js
        |   |_ script.js
        |
        |_ conexion.php
        |_ funciones.php
        |_ index.html
        |_ nombres.txt
```

Para comenzar vamos a definir cómo conectarnos al servidor. Creamos un archivo llamado **conexion.php**, lo guardamos en el directorio raíz del proyecto y escribimos lo siguiente:

```
(: Instanciamos Redis, nos conectamos al servidor y luego a la base de datos :)
<?php

define('HOST','127.0.0.1');
define('PUERTO',6379);
define('BD',0);
try{
        $redis = new Redis();
        $redis->connect(HOST, PUERTO);
        $redis->select(BD);
}catch(Exception $e){
        die('ERROR '.$e->getCode().': '.$e->getMessage());
}

?>
```

Este código es muy sencillo: hemos definido las constantes de conexión y, mediante un bloque **try/catch**, instanciamos Redis e intentamos conectarnos al servidor. Luego, incluiremos este archivo en otro para hacer uso de los comandos de Redis. Antes de seguir agregando código, es necesario entender el flujo de funcionamiento:

1. Creamos el archivo **index.html**, que tendrá todos los elementos HTML de presentación.
2. Creamos el archivo **script.js**, que tendrá los métodos que responderán a los eventos de **index.html**.
3. Creamos el archivo **funciones.php**, que contendrá los métodos de comunicación con Redis.

La comunicación entre **script.js** y **funciones.php** se realizará mediante las funciones Ajax de **jQuery**. En resumen, cuando hagamos clic en un botón de **index.html** se disparará un evento en **script.js** que, mediante Ajax, se comunicará con **funciones.php**.

Creación del archivo index.html

Para continuar, creamos el archivo **index.html**, agregamos el siguiente código y lo guardamos en el directorio raíz de nuestro ejemplo:

```
<!DOCTYPE html>
<html>
        <head>
                <title>PHP/Redis - phpredis</title>
                <link type="text/css" href="css/style.css" rel="stylesheet" />
                <script type="text/javascript" src="http://ajax.googleapis.com/
ajax/libs/jquery/1.9.0/jquery.min.js"></script>
                <script type="text/javascript" src="js/jquery.form.js"></script>
                <script type="text/javascript" src="js/script.js"></script>

                <!--[if lt IE 9]>
                        <script src="http://html5shiv.googlecode.com/svn/trunk/
html5.js"></script>
                <![endif]-->
        </head>
        <body>
```

```
            <header>
                <h1>PHP/Redis - phpredis</h1>
            </header>
            <nav>
                <ul>
                    <li id="servidor">Servidor</li>
                    <li id="strings">Strings</li>
                    <li id="hashes">Hashes</li>
                    <li id="lists">Lists</li>
                    <li id="sets">Sets</li>
                    <li id="sorted_sets">Sorted Sets</li>
                </ul>
            </nav>
            <section>

            </section>
            <footer>
                <p>PHP-REDIS</p>
            </footer>
        </body>
    </html>
```

Hemos definido la estructura general de nuestro ejemplo; inicialmente, incluimos el archivo de estilos y los de JavaScript, y agregamos una librería para poder utilizar etiquetas de HTML5 en Internet Explorer en versiones anteriores a la versión 9.

Seguidamente, definimos el título y el menú del ejemplo, y más abajo hemos definido la etiqueta **<section>** en donde agregaremos los elementos para manejar las acciones.

HTML5

HTML5 es la quinta revisión importante del lenguaje más utilizado en la World Wide Web: HTML. HTML5 se encarga de especificar dos variantes de sintaxis para HTML: un clásico HTML (text/html), la variante conocida como HTML5, y una variante XHTML conocida como **sintaxis XHTML5**.

A continuación, vamos a agregar los elementos dentro de la etiqueta **<section>**. Inicialmente, vamos a agregar lo que sigue para enviar y recibir información de nuestro servidor de base de datos:

```
<!-- SERVIDOR -->
<article class="servidor">
        <div class="botones">
                <h2>Servidor:</h2>
                <input type="button" id="info" value="Info del Servidor" />
                <input type="button" id="save" value="Guardar en Disco" />
                <input type="button" id="lastSave" value="Ultima persistencia"/>
                <input type="button" id="dbSize" value="Cantidad de claves" />
                <input type="button" id="flush" value="Vaciar BD" />
        </div>
        <div class="resultados" id="rServidor"></div>
</article>
```

Luego, agregamos lo siguiente para operar con el tipo de datos **String** (aquí vamos a desarrollar la funcionalidad de ver e incrementar los valores de una clave y luego vamos a crear un formulario para subir imágenes directamente al servidor):

```
<!-- STRING -->
<article class="strings">
        <div class="botones">
                <h2>Strings:</h2>
                <input type="radio" name="usuario" id="usuario1" value="1"
checked="checked">
                <label for="usuario1">ID:1 Fabi&aacute;n</label>
                <br />
                <input type="radio" name="usuario" id="usuario2" value="2">
                <label for="usuario2">ID:2 Tom&aacute;s</label>
                <br />
                <input type="radio" name="usuario" id="usuario3" value="3">
                <label for="usuario3">ID:3 Eve</label>
                <br />
```

```
    <input type="radio" name="usuario" id="usuario4" value="4">
                    <label for="usuario4">ID:4 Alberto</label>

                    <input type="button" class="sverClave" value="Ver Clave" />
                    <input type="button" class="sIncrDecr" value="+10" />
                    <input type="button" class="sIncrDecr" value="+1" />
                    <input type="button" class="sIncrDecr" value="-1" />
                    <input type="button" class="sIncrDecr" value="-10" />

                    <form enctype="multipart/form-data" id="sSubirImagenForm">
                            <input type="file" name="imagen"/>
                            <input type="submit" id="sSubirImagen"
value="Subir" />
                    </form>
            </div>
            <div class="resultados" id="rStrings"></div>
    </article>
```

A continuación agregamos el siguiente código para interactuar con la base de datos mediante el tipo de datos **Hash** (vamos a utilizar este tipo de datos para buscar, modificar, visitar y eliminar un listado de nombres que han sido importados):

```
    <!-- HASH -->
    <article class="hashes">
            <div class="botones">
                    <h2>Hashes de clientes:</h2>
                    <input type="button" id="aCliente" value="Importar Clientes"
/>
                    <input type="button" id="bCliente" value="Buscar" />
                    <input type="text" id="bClienteId" value="" />

                    <input type="button" id="mCliente" value="Modificar" />
                    <input type="button" id="vCliente" value="Visitar" />
                    <input type="button" id="eCliente" value="Eliminar" />
            </div>
```

```
        <div class="resultados" id="rHashes"></div>
    <div class="resultados" id="rHashesCampos">
                <p id="clienteClave"><b>Cliente:</b></p>
                <label for="clienteNombre">Nombre</label>
                <input type="text" id="clienteNombre" value="" />
                <label for="clienteApellido">Apellido</label>
                <input type="text" id="clienteApellido" value="" />
                <label for="clienteCorreo">Correo</label>
                <input type="text" id="clienteCorreo" value="" />
                <label for="clienteVisitas">Visitas</label>
                <input type="text" id="clienteVisitas" value=""
readonly="readonly"/>
        </div>
    </article>
```

Luego vamos a manejar el tipo de datos **List**. Por lo tanto, a continuación veremos cómo agregar y eliminar elementos a la derecha y a la izquierda de la lista:

```
    <!-- LIST -->
    <article class="lists">
        <div class="botones">
                <h2>Listas:</h2>
                <input type="radio" name="lista" id="lista1" value="1"
checked="checked">
                <label for="lista1">Lista uno</label>
                <br />
                <input type="radio" name="lista" id="lista2" value="2">
                <label for="lista2">Lista dos</label>

                <input type="text" id="lValor" value="" />
                <input type="button" id="lRange" value="Ver Lista" />
                <input type="button" id="lPush" class="push" value="Agregar:
Izquierda" />
                <input type="button" id="lPop" class="pop" value="Eliminar:
Izquierda" />
```

```
    <input type="button" id="rPush" class="push" value="Agregar: Derecha" />
                    <input type="button" id="rPop" class="pop" value="Eliminar:
Derecha" />
            </div>
            <div class="resultados" id="rLists"></div>
    </article>
```

Seguidamente, agregaremos lo necesario para operar con el tipo de datos **Set**. Vamos a simular el funcionamiento de los círculos de **G+**, donde vamos a agregar elementos y luego vamos a hacer la unión e intersección entre los círculos existentes:

```
    <!-- SET -->
    <article class="sets">
            <div class="botones">
                    <h2>Circulos de Alberto:</h2>
                    <input type="radio" name="circulos" id="circuloFamilia"
value="familia" checked="checked">
                    <label for="circuloFamilia">Familia</label>
                    <br />
                    <input type="radio" name="circulos" id="circuloFacultad"
value="facultad">
                    <label for="circuloFacultad">Facultad</label>
                    <br />
                    <input type="radio" name="circulos" id="circuloFutbol"
value="futbol">
                    <label for="circuloFutbol">Futbol</label>

                    <input type="button" id="sMembers" value="Ver Circulo" />
                    <input type="button" data="tom&aacute;s" class="sadd"
value="agregar a Tom&aacute;s" />
                    <input type="button" data="fabi&aacute;n" class="sadd"
value="agregar a Fabi&aacute;n" />
                    <input type="button" data="eve" class="sadd" value="agregar a
Eve" />
```

```
    <input type="button" data="juan" class="sadd" value="agregar a Juan" />

                    <input type="button" id="sUnion" value="Union" />
                    <input type="button" id="sInter" value="Intersecci&oacute;n"
/>
            </div>
            <div class="resultados" id="rSets"></div>
    </article>
```

Finalmente, originaremos los elementos para interactuar con la base de datos con el tipo de datos **Sorted set**. Aquí vamos a importar un listado de nombre, con los cuales vamos a crear un autocompletador:

```
    <!-- SORTED SET -->
    <article class="sorted_sets">
            <div class="botones">
                    <h2>Autocompletado:</h2>
                    <input type="button" id="importarSet" value="Importar Nom-
bres" />
                    <input type="text" id="buscarClaveSet" value="" />
                    <div id="buscarClaveSetLista"></div>
            </div>
            <div class="resultados" id="rSortedSets"></div>
    </article>
```

En la estructura establecida hemos creado los elementos para trabajar con cada tipo de dato que ofrece **Redis**.

ALTERNATIVAS PARA ALMACENAR DATOS

Científicos británicos han logrado almacenar 739 KB de datos en una hebra de ADN, que incluyen más de cien sonetos de Shakespeare, una foto, un archivo PDF y 26 segundos de sonido. El ADN ha demostrado ser muy durable y sin requerimiento de energía.

Creación del archivo style.css

Ahora es necesario modificar el aspecto del proyecto. Para esto, creamos el archivo **style.css** y lo guardamos dentro del directorio **css**.

Inicialmente, en el archivo **style.css** escribimos lo siguiente para definir los elementos de **body**:

```
body{
        background-color: #F2F2F2;
        color: #2F2F2F;
        font: 12px Arial,Helvetica,sans-serif;
        margin: 0;
        padding: 0;
        text-align: center;
        color: #444444;
}
```

Seguidamente definimos el aspecto para el **header**, tal como vemos en el siguiente código:

```
header{
        width: 40%;
        text-align: left;
}
```

A continuación, definiremos como se verá nuestro menú:

```
nav{
        float: left;
        width: 20%;
        padding: 0 5px 0 10px;
}
nav ul{
        list-style: none outside none;
```

```
        padding: 0;
        margin: 0;
    }
    nav ul li{
        border: 1px solid transparent;
        cursor: pointer;
        font-size: 15px;
        height: 100%;
        line-height: 46px;
        padding: 0;
        width: auto;
    }
    nav ul li:hover,
    nav ul .active {
        background-color: #EEEEEE;
        background-image: -moz-linear-gradient(center top , #EEEEEE,
#E0E0E0);
        border: 1px solid #CCCCCC;
        box-shadow: 0 1px 2px rgba(0, 0, 0, 0.1) inset;
        color: #333333;
    }
```

Ahora escribimos el código para estilar los elementos dentro de **section** y **article**:

```
    section{
        float: left;
        padding: 10px;
        width: 73%;
        min-height: 700px;
        text-align: left;
        background-color: #FFFFFF;
        border: 1px solid #CCCCCC;
        border-radius:5px;
        -moz-border-radius:5px;
        -webkit-border-radius: 5px;
```

```
}
article{
        display: none;
        min-height: 160px;
}
article .botones{
        width: 20%;
        margin: 0 5px;
        float: left;
}
article .resultados{
        width: 70%;
        margin: 0 5px;
        float: left;
}
```

Los elementos del **footer** los vamos a estilar de la siguiente manera:

```
footer{
        clear: both;
        height: 80px;
        padding: 10px;
        text-align: center;
}
footer p{
        margin: 0 auto;
        padding: 30px 0;
        width: 300px;
}
```

NUEVOS ELEMENTOS EN HTML5

HTML5 llega con nuevos elementos y atributos que están pensados en el uso de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas ** ** y ****, pero tienen un significado semántico, como por ejemplo **<nav>** (bloque de navegación del sitio web) y **<footer>**.

Finalmente, vamos a estilar los elementos restantes de nuestro ejemplo:

```
input{
        margin: 2px 0;
}
input[type="button"]{
        margin: 5px 0;
        min-width: 146px;
}
label{
        display: inline-block;
        min-width: 75px;
}

.sImagenBD{
        height: 370px;
        width: 370px;
}
#buscarClaveSetLista {
        background-color: #FFFFFF;
        border: 1px solid #ADA9A5;
        margin: -4px 0;
        padding: 0;
        width: 147px;
}
#buscarClaveSetLista .item{
```

DEPENDENCIAS PARA PHP

Composer es una herramienta para administrar dependencias en PHP, que permite declarar las librerías que necesitarán nuestros proyectos y las instala por nosotros. Se encarga de verificar cuál es la versión compatible del paquete requerido y lo descarga directamente a nuestro proyecto. Podemos obtener esta herramienta en: **http://getcomposer.org**.

```
        cursor: pointer;
        padding: 0 5px;
        width: 137px;
}
#buscarClaveSetLista .item:hover{
        background-color: #FFCC80;
}
```

En los bloques de código anterior no hay aspectos relevantes que resaltar, ya que solo nos encargamos de realizar la definición de los estilos que van a tener los elementos del proyecto.

Las imágenes que veremos más adelante nos servirán para ver la forma en que se presentarán las pantallas para cada tipo de dato que ejemplificaremos.

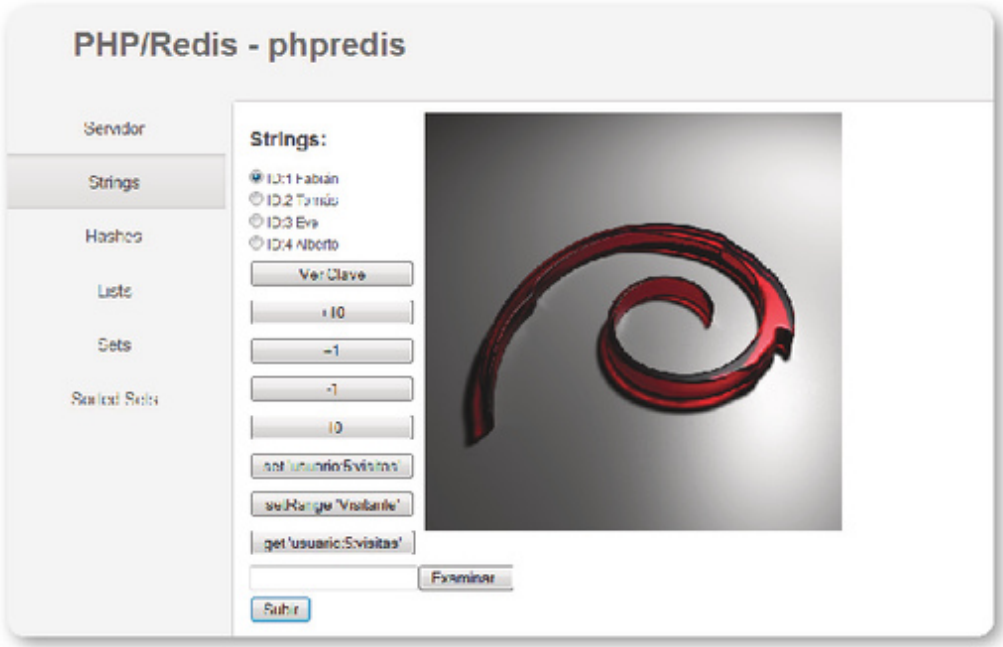


Figura 7. La funcionalidad relevante es poder almacenar imágenes directamente en la base de datos.

En el ejemplo que sigue vamos a realizar la importación de clientes mediante **PHP** de un array definido en el archivo **funciones.php**. Debemos considerar que éstos se guardarán en la base de datos utilizando el tipo de dato hash, que nos permitirá manejar cada uno como si fuera un registro de una base de datos relacional.

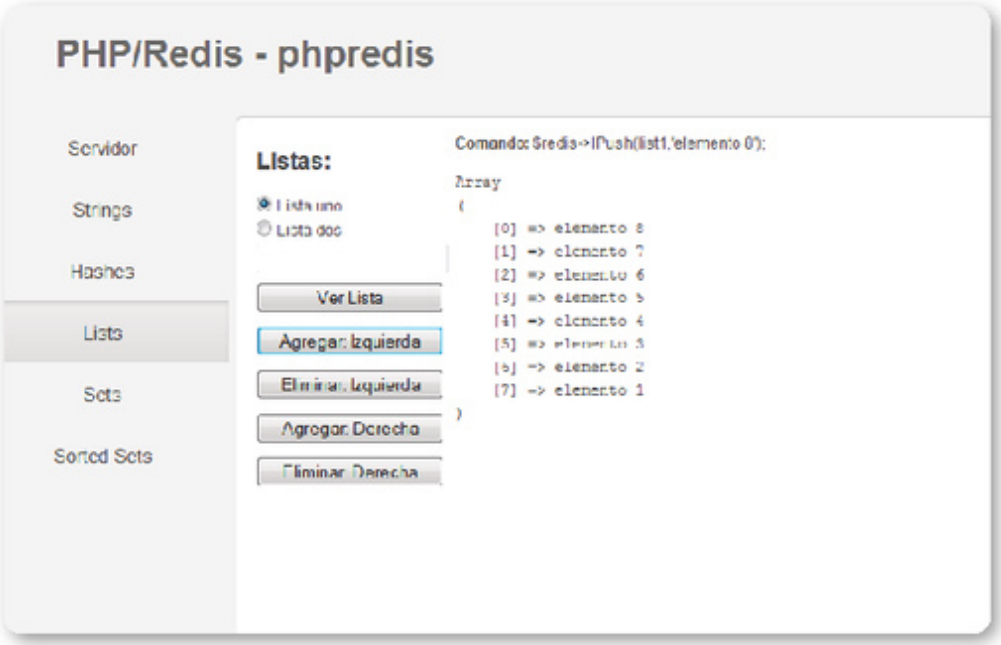


Figura 8. En el ejemplo vemos cómo crear una lista fácilmente. Un modelo de esto es Twitter.

Cuando tenemos la necesidad de almacenar valores que estarán agrupados por algún criterio es útil recurrir al tipo de dato set, en el cual podemos agregar miembros a cada grupo en particular y hacer operaciones como unión o intersección.

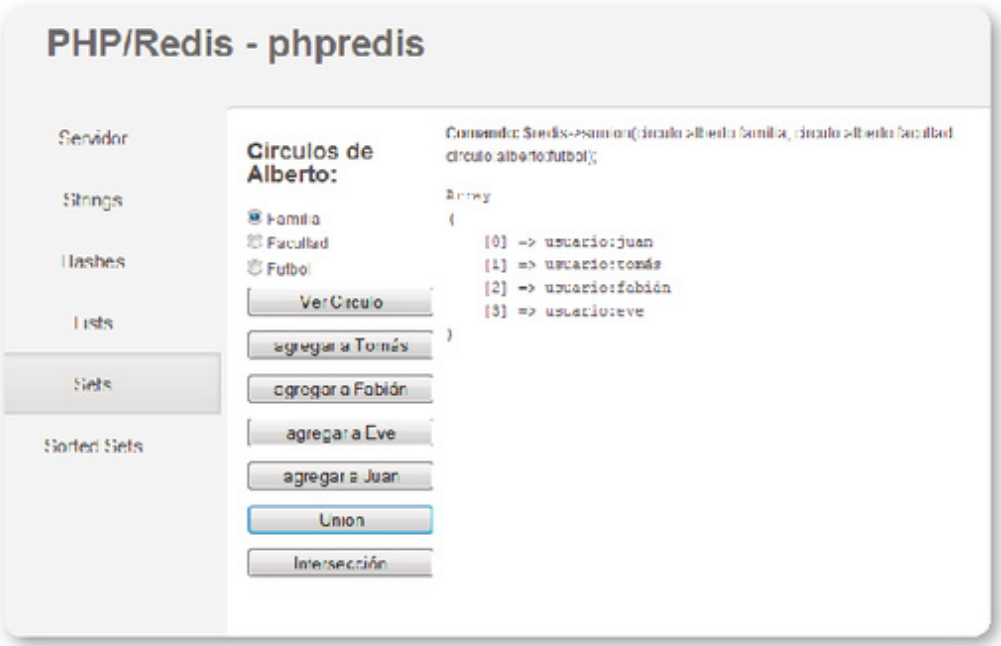


Figura 9. El tipo de dato set es muy útil en Redis, por ejemplo, para dar una funcionalidad similar a los círculos de G+.

Google ha cambiado la implementación del filtrado de palabras cuando se realiza una búsqueda. En este se sugieren las palabras que coinciden con el criterio, posicionándose en la parte superior de la lista aquellas palabras que han sido utilizadas con mayor frecuencia.

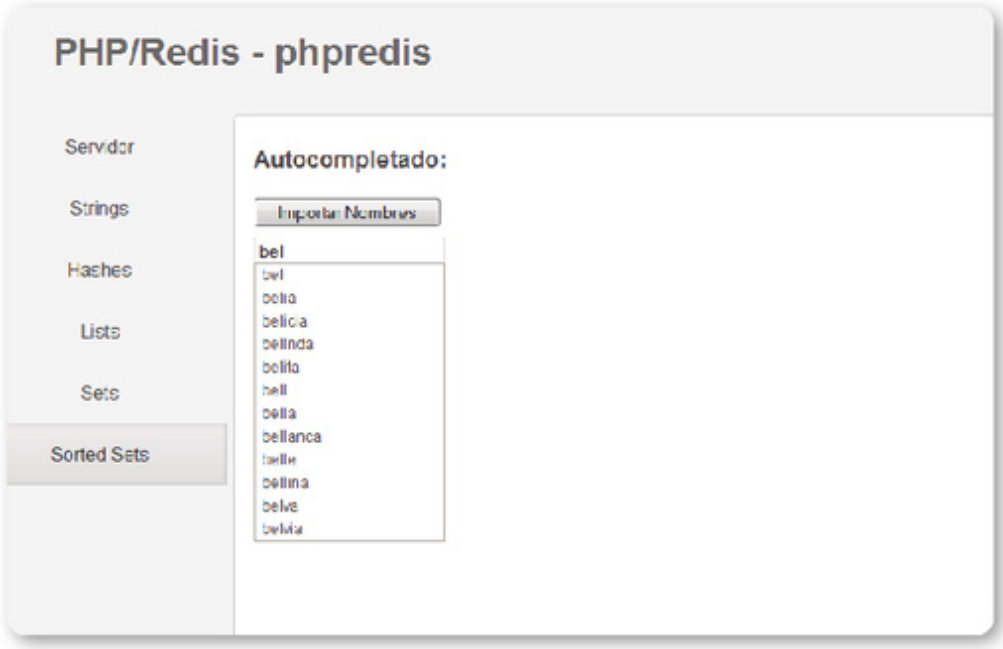


Figura 10. Para crear un filtro podemos usar sortedsets. En los próximos capítulos veremos cómo sugerir resultados mediante el uso de scores.

Creación del archivo script.js

A continuación vamos a crear el archivo **script.js**, que nos servirá para interactuar entre los elementos HTML y la base de datos correspondiente. Para ello, agregaremos el siguiente código y lo guardaremos dentro de la carpeta **js**:

LIBRERÍAS JAVASCRIPT HOSTEADAS

Para los proyectos web siempre es necesario utilizar librerías como **jQuery**. Con este objetivo, Google creó un lugar de almacenamiento, de manera que para utilizar las librerías solo es necesario incluirlas en los documentos donde las necesitemos. Para conocer cuáles son y cómo utilizarlas, podemos acceder al siguiente enlace: **https://developers.google.com/speed/libraries**.

```
$(document).ready(function(){
        // muestra contenido inicial
        mostrarContenido('servidor');

        // muestra contenido dependiendo del item del menu
        $(document).on('click','nav ul li', function(){
                mostrarContenido($(this).attr('id'));
        });

});

function mostrarContenido(id) {
        $('nav ul li').removeClass('active');
        $('#'+id).addClass('active');
        $('article').hide();
        $('.'+id).show();
}
```

En el código anterior hemos definido en el evento **ready** del **document** una llamada a la función **mostrarContenido()** y, más abajo, un evento que captura el clic del menú para mostrar el contenido seleccionado.

Finalmente realizaremos la creación de la función que se encarga de hacer visible el contenido indicado. A continuación, vamos a agregar los métodos para trabajar con el servidor. Observemos que, de ahora en adelante, vamos a utilizar funciones en Ajax para comunicarnos con el archivo **php** que crearemos más adelante.

Cada uno de los eventos debe ser creado dentro del evento **ready** del **document**:

GITHUB

El sitio **http://nodetoolbox.com** es un fantástico repositorio que está conectado a GitHub y nos da la posibilidad de buscar y descargar los proyectos alojados en esta plataforma. Además, cuenta con un agrupamiento por categorías donde se muestran los paquetes para darnos una mejor visión de los proyectos disponibles.

```
// SERVIDOR ---//
// muestra info del servidor Redis
$(document).on('click', '#info', function(){
	$.post("funciones.php","action=getInfo",
		function(data){
			$('#rServidor').html(data);
		}
	);
});
// guardar BD
$(document).on('click', '#save', function(){
	$.post("funciones.php","action=setSave",
		function(data){
			$('#rServidor').html(data);
		}
	);
});
// muestra ultima vez que persistio Redis
$(document).on('click', '#lastSave', function(){
	$.post("funciones.php","action=getLastSave",
		function(data){
			$('#rServidor').html(data);
		}
	);
});
// muestra la cantidad de claves
$(document).on('click', '#dbSize', function(){
	$.post("funciones.php","action=getdbSize",
		function(data){
			$('#rServidor').html(data);
		}
	);
});
// vacia la BD
$(document).on('click', '#flush', function(){
	$.post("funciones.php","action=setFlush",
		function(data){
```

```
                $('#rServidor').html(data);
            }
        );
});
```

Lo siguiente será crear los eventos para manejar el tipo de datos **String**, tal como vemos a continuación:

```
// STRINGS ---//
// Ver clave - Strings
$(document).on('click','.sverClave', function(){
    var IdUsuario = $('input[name="usuario"]:checked').val();
    $.post("funciones.php","action=getString&IdUsuario="+IdUsuario,
        function(data){
            $('#rStrings').html(data);
        }
    );
});
// incrementar/decrementar - Strings
$(document).on('click','.sIncrDecr', function(){
    var IdUsuario = $('input[name="usuario"]:checked').val();
    var valor = $(this).val();
    $.post("funciones.php","action=setValorString&IdUsuario="+IdUsuario+"
&valor="+valor,
        function(data){
            $('#rStrings').html(data);
        }
    );
});
// subir imagen
$(document).on('submit','#sSubirImagenForm', function(e) {
    e.preventDefault();
    $('#sSubirImagen').attr('disabled',''); // disabilitamos el boton de upload
    $(this).ajaxSubmit({
        type: "POST",
```

```
                url: "funciones.php",
                success:  function(data){
                        if (data > 0)
                                $('#rStrings').html('<img class="sImagenBD"
src="funciones.php?action=getImagen&id='+data+'">');
                        else
                                $('#rStrings').html('Ha ocurrido un error :-)');
                        $('#sSubirImagenForm').resetForm();
                        $('#sSubirImagen').removeAttr('disabled'); // habilita-
mos el boton de upload
                }
        });
});
```

A continuación, originamos los eventos para manejar el tipo de datos **Hash**; procedemos tal como mostramos en el siguiente código:

```
// HASHES ---//
// crea un hash
$(document).on('click', '#aCliente', function(){
        $.post("funciones.php","action=aCliente",
                function(data){
                        $('#rHashes').html('<b>Se importaron '+data+' clien-
tes con las siguientes claves:</b> Nombre, Apellido, Correo y Visitas').show().fad-
eOut(9999);
                }
        );
});
// buscar hash
$(document).on('click', '#bCliente', function(){
        $('#rHashesCampos input[type="text"]').val('');
        $.post("funciones.php","action=bCliente"+"&id="+$('#bClienteId').val(),
                function(data){
                        // get json
                        if(data != 0){
```

```
                                        var items = eval('(' + data + ')');
                                        $('#clienteNombre').val(items['nombre']);
                                        $('#clienteApellido').val(items['apellido']);
                                        $('#clienteCorreo').val(items['correo']);
                                        $('#clienteVisitas').val(items['visitas']);
                                        $('#clienteClave').html('<b>Cliente:
'+$('#bClienteId').val()+'</b>');
                                }else{
                                        $('#clienteClave').html('El cliente no existe');
                                }
                        }
                );
        });
        // modificar hash
        $(document).on('click','#mCliente', function(){
                $.post("funciones.php","action=mCliente"+"&id="+$('#bClienteId').val()
        +"&nombre="+$('#clienteNombre').val()+"&apellido="+$('#clienteApellido').
val()+"&correo="+$('#clienteCorreo').val(),
                        function(data){
                                $('#rHashes').html('<b>Cliente '+$('#bClienteId').
val()+' Actualizado</b>').show().fadeOut(9999);
                                $('#rHashesCampos input[type="text"]').val('');
                        }
                );
        });
        // incrementar el contador
        $(document).on('click','#vCliente', function(){
                $.post("funciones.php","action=vCliente"+"&id="+$('#bClienteId').val(),
                        function(data){
                                $('#clienteVisitas').val(data);
                        }
                );
        });
        // eliminar hash
        $(document).on('click','#eCliente', function(){
                $.post("funciones.php","action=eCliente"+"&id="+$('#bClienteId').val(),
                        function(data){
```

```
                $('#rHashes').html('<b>Cliente '+$('#bClienteId').
val()+' Eliminado</b>').show().fadeOut(9999);
                $('#rHashesCampos input[type="text"]').val('');
                $('#clienteClave').html('<b>Cliente:</b>');
            }
        );
    });
```

Ahora vamos a crear los eventos para manejar el tipo de datos **List**, para ello agregamos lo siguiente:

```
    // LISTS ---//
    // ver lista
    $(document).on('click', '#lRange', function(){
        var lista = $('input[name="lista"]:checked').val();
        $.post("funciones.php","action=lRange&lista="+lista,
            function(data){
                $('#rLists').html(data);
            }
        );
        $('#lValor').val('');
    });
    // agrega una clave a la lista
    $(document).on('click', '.push', function(){
        var lista = $('input[name="lista"]:checked').val();
        var valor = $('#lValor').val();
        var side = $(this).attr('id');
        $.post("funciones.php","action=push&lista="+lista+"&valor="+valor+"&s
ide="+side,
            function(data){
                $('#rLists').html(data);
            }
        );
        $('#lValor').val('');
    });
    // elimina una clave de la lista
```

```
$(document).on('click','.pop', function(){
        var lista = $('input[name="lista"]:checked').val();
        var side = $(this).attr('id');
        $.post("funciones.php","action=pop&lista="+lista+"&side="+side,
                function(data){
                        $('#rLists').html(data);
                }
        );
        $('#lValor').val('');
});
```

Seguidamente, vamos a crear los métodos para manejar el tipo de datos **SETS**, para ello escribimos el siguiente código:

```
// SETS ---//
// obtiene los miembros
$(document).on('click','#sMembers', function(){
        var circulo = $('input[name="circulos"]:checked').val();
        $.post("funciones.php","action=sMembers&circulo="+circulo,
                function(data){
                        $('#rSets').html(data);
                }
        );
});
// agrega una clave
$(document).on('click','.sadd', function(){
        var circulo = $('input[name="circulos"]:checked').val();
```

DATOS SETS

Es importante considerar que el tipo de datos SET representa un conjunto de cadenas. De este modo, puede contener uno o más valores, que se eligen de una lista de valores permitidos que se especifican al definir el campo y se separan con comas. En este punto debemos considerar que es posible que contengan un máximo de 64 miembros.

```
        var usuario = $(this).attr('data');
        $.post("funciones.php","action=sAdd&circulo="+circulo+"&usuario="+us
uario,
                function(data){
                        $('#rSets').html(data);
                }
        );
        $('#lValor').val('');
});
// union de sets
$(document).on('click', '#sUnion', function(){
        var circulo = $('input[name="circulos"]:checked').val();
        $.post("funciones.php","action=sUnion",
                function(data){
                        $('#rSets').html(data);
                }
        );
});
// interseccion de sets
$(document).on('click', '#sInter', function(){
        var circulo = $('input[name="circulos"]:checked').val();
        $.post("funciones.php","action=sInter",
                function(data){
                        $('#rSets').html(data);
                }
        );
});
```

Finalmente, nos encargaremos de crear los métodos adecuados para manejar el tipo de datos **SORTED SETS**.

```
// SORTED SETS ---//
// importa los nombres
$(document).on('click', '#importarSet', function(){
        $.post("funciones.php","action=importarSet",
                function(data){
```

```
                        $('#rSortedSets').html(data).show().fadeOut(9999);
                }
        );
});
// busca los nombres coincidentes
$(document).on('keyup', '#buscarClaveSet', function(){
        $('#buscarClaveSetLista').html('');

        $.post("funciones.php","action=buscarClaveSet"+"&prefix="+$(this).
val(),
                function(data){
                        var items = eval('(' + data + ')');
                        for (var i in items)
                                $('#buscarClaveSetLista').append('<div
class="item">'+items[i]+'</div>');
                }
        );
});
// selecciona un nombre de la lista
$(document).on('click', '.item', function(){
        $('#buscarClaveSet').val($(this).html());
        $('#buscarClaveSetLista').html('');
});
```

Creación del archivo funciones.php

A continuación vamos a crear el archivo **funciones.php**, que va a recibir las solicitudes del archivo **script.js** a través de las funciones que hemos creado anteriormente. Este archivo contendrá todos los comandos para interactuar directamente con la base de datos.

```
<?php

include('conexion.php');
$param_action = isset($_REQUEST['action']) ? $_REQUEST['action'] : '';
$param_file           = isset($_FILES['imagen']) ? $_FILES['imagen'] : '';
```

```
    $action                         = !empty($param_file) ? 'setImagen' :
$param_action;

    $nombres                = array('Abbott','Abdiel','Abe','Abel','Abey' ,'Abie' ,'Abi-
jah' ,'Abner','Abraham' ,'Abram');
    $apellidos              = array('Abascal','Agudo','Aguilar','Aja','Lopez','Sainz','
Trapaga','Alarcon','Albertos','Albrecht');

    function mostrarSalida($comando){
            return "<b>Comando: </b>{$comando} <br /><pre>";
    }
    ?>
```

En el código anterior hemos incluido el archivo **conexion.php**, que contiene los parámetros para comunicarnos con **Redis**, y luego capturamos los parámetros y definimos dos **arrays**, que nos servirán de ejemplo en el autocompletado para el tipo de dato **sorted sets**. Luego, definimos una función **mostrarSalida()**, que imprimirá parte de la salida de cada acción.

EN EL ARCHIVO CONEXION.PHP TENEMOS LOS PARÁMETROS PARA CONECTAR A REDIS

Seguidamente, vamos a crear una estructura **switch** para verificar cual será la acción a ejecutar:

```
    switch ($action){

    }
```

DISEÑO ADAPTATIVO

Hoy en día nuestros sistemas no solo deben tener la capacidad de estar disponibles en todo momento, sino también desde cualquier dispositivo: smartphones, tablets, e incluso Smart TV. Para lograr un diseño adaptativo, CSS3 nos ofrece la solución **Media Queries**. En el siguiente enlace podemos ampliar este tema y aplicarlo a nuestros proyectos: **http://css3html5.com.ar/mediaqueries**.

A continuación, vamos a incluir los métodos para trabajar con el servidor; para ello agregamos lo siguiente dentro de la estructura **switch**:

```
// SERVIDOR ---//
case 'getInfo':
        echo mostrarSalida('$redis->info();');
        print_r($redis->info());
        echo '</pre>';
break;
case 'getLastSave':
        echo mostrarSalida('$redis->lastSave();');
        print_r(date('d-m-Y H:i:s',$redis->lastSave()));
        echo '</pre>';
break;
case 'setFlush':
        $redis->flushDB();
        echo mostrarSalida('$redis->flushDB();');
        print_r($redis->info());
        echo '</pre>';
        break;
case 'setSave':
        echo mostrarSalida('$redis->save();');
        print_r($redis->save());
        echo '</pre>';
break;
case 'getdbSize':
        echo mostrarSalida('$redis->dbSize();');
        print_r($redis->dbSize());
        echo '</pre>';
break;
```

En el código anterior ejecutamos los comandos para mostrar información del servidor y otras acciones como guardar en disco rígido la base de datos en memoria.

Ahora nos encargaremos de agregar el código para manejar el tipo de datos **String**:

```
// STRINGS ---//
case 'getString':
        $IdUsuario = $_POST['IdUsuario'];
        $clave = $redis->get('usuario:'.$IdUsuario.':visitas');
        echo mostrarSalida("\$redis->get('usuario:{$IdUsuario}:visitas');");
        echo !empty($clave)?$clave:'(nill)';
        echo '</pre>';
break;
case 'setValorString':
        $IdUsuario = $_POST['IdUsuario'];
        $valor     = $_POST['valor'];

        if ($valor == '+1'){
                $comando   = "\$redis->incr('usuario:{$IdUsuario}:visitas');";
                $resultado = $redis->incr('usuario:'.$IdUsuario.':visitas');
        }elseif ($valor == '+10'){
                $comando   = "\$redis->incrBy('usuario:{$IdUsuario}:visit
as',10);";
                $resultado = $redis->incrBy('usuario:'.$IdUsuario.':visitas', 10);
        }elseif ($valor == '-1'){
                $comando   = "\$redis->decr('usuario:{$IdUsuario}:visitas');";
                $resultado = $redis->decr('usuario:'.$IdUsuario.':visitas');
        }elseif ($valor == '-10'){
                $comando   = "\$redis->decrBy('usuario:{$IdUsuario}:visit
as',10);";
                $resultado = $redis->decrBy('usuario:'.$IdUsuario.':visitas',10);
        }
        echo mostrarSalida($comando);
        echo $resultado;
        echo '</pre>';
break;
```

```
    case 'setImagen':
        if (isset($_FILES['imagen']) && $_FILES['imagen']["error"] == 0){
            $permitidos = array("image/jpg", "image/jpeg", "image/gif",
"image/png");
            $limite_kb = 16384; // max 16Mb
            $id = rand();
            if (in_array($_FILES['imagen']['type'], $permitidos) && $_
FILES['imagen']['size'] <= $limite_kb * 1024){
                $data = file_get_contents($_FILES['imagen']['tmp_
name']);
                $tipo = $_FILES['imagen']['type'];
                $resultado = $redis->mset(array("imagenes:{$id}:imag
en" => base64_encode($data), "imagenes:{$id}:tipo" => $tipo));
            }
        echo ($resultado == 1) ? $id : 0;
        }
    break;
    case 'getImagen':
        if (!empty($id)){
            $imagen = base64_decode($redis-
>get("imagenes:{$id}:imagen"));
            $tipo   = $redis->get("imagenes:{$id}:tipo");

            header("Content-type: {$tipo}");
            echo $imagen;
        }
    break;
```

HTML6

Ya se está hablando de la llegada de HTML6, donde la idea principal es lograr separar la semántica de la presentación. Entre las ventajas propuestas, está la total compatibilidad con versiones anteriores, ya que proveerá una herramienta para la migración. La característica central de la nueva versión del lenguaje es facilitar el uso tanto a los usuarios como a los desarrolladores.

En el código anterior realizamos la tarea de incrementar y decrementar la cantidad de visitas de los usuarios y también nos encargamos de mostrar la cantidad actual. También realizamos la definición de los comandos para poder guardar y mostrar las imágenes que han sido almacenadas directamente en la base de datos.

A continuación veremos la forma en que podemos agregar el código para trabajar con el tipo de datos **Hash**:

EN ESTE PUNTO DEFINIMOS LOS COMANDOS PARA GUARDAR Y MOSTRAR IMÁGENES

```
// HASHES ---//
case 'aCliente':
        $i = 0;
        foreach ($apellidos as $apellido) {
                foreach ($nombres as $nombre) {
                        $i++;
                        $mail = strtolower($nombre.'@'.$apellido.'.com');
                        $redis->hmset('cliente:' . $i, array('nombre' => $nom-
bre, 'apellido' => $apellido, 'correo' => $mail, 'visitas' => 0));
                }
        }
        echo $i;
break;
case 'bCliente':
        $cliente = $redis->hGetAll('cliente:' . $_POST['id']);
        echo !empty($cliente) ? json_encode($cliente) : '0';
break;
case 'mCliente':
        $id = $_POST['id'];
        $nombre = $_POST['nombre'];
        $apellido = $_POST['apellido'];
        $correo = $_POST['correo'];
        $redis->hmset('cliente:' . $id, array('nombre' => $nombre, 'apellido' =>
$apellido, 'correo' => $correo));
break;
case 'vCliente':
```

```
$id = $_POST['id'];
                echo $redis->hIncrBy('cliente:' . $id, 'visitas', 1);
break;
case 'eCliente':
                $id = $_POST['id'];
                $redis->del('cliente:' . $id);
break;
```

Como vimos, en el código anterior trabajamos con los **arrays** que fueron definidos al principio del archivo, luego operamos con ellos realizando una búsqueda por **id**, una modificación, un incremento de las visitas y también el borrado de un registro.

Luego nos encargaremos de realizar la definición de las acciones para trabajar con el tipo de datos **LIST**:

```
// LISTS ---//
case 'lRange':
        $lista = 'list'.$_POST['lista'];
        echo mostrarSalida("\$redis->lRange({$lista}, 0, -1);");
        print_r($redis->lRange($lista,0,-1));
        echo '</pre>';
break;
case 'push':
        $lista = 'list'.$_POST['lista'];
        $valor = $_POST['valor'];
        $side = $_POST['side'];
        $redis->$side($lista,$valor);
```

VIDEOS EN YOUTUBE

Un estudio realizado hace poco tiempo indica que son más de cuatro mil millones los videos que se ven diariamente en el sitio de videos de Google conocido como YouTube. La cantidad de demanda de videos subió cuando Google publico las versiones para Smartphones y Smart TV. Estos dispositivos se cuentan entre los más utilizados para acceder a las redes sociales actuales.

```
        echo mostrarSalida("\$redis->{$side}({$lista},'{$valor}');");
        print_r($redis->lRange($lista,0,-1));
        echo '</pre>';
    break;
    case 'pop':
        $lista = 'list'.$_POST['lista'];
        $side = $_POST['side'];
        $redis->$side($lista);

        echo mostrarSalida("\$redis->{$side}({$lista});");
        print_r($redis->lRange($lista,0,-1));
        echo '</pre>';
    break;
```

En el código definido hemos realizado las acciones **push** y **pop** para agregar y obtener elementos de las listas.

Como podemos ver, el paso que sigue es agregar los comandos para manejar el tipo de datos **Set**:

```
    // SETS ---//
    case 'sMembers':
        $circulo = 'circulo:alberto:'.$_POST['circulo'];
        echo mostrarSalida("\$redis->smembers('{$circulo}');");
        print_r($redis->smembers($circulo));
        echo '</pre>';
    break;
    case 'sAdd':
        $circulo = 'circulo:alberto:'.$_POST['circulo'];
        $usuario = 'usuario:'.$_POST['usuario'];
        $redis->sadd($circulo, $usuario);

        echo mostrarSalida("\$redis->sadd('{$circulo}', '{$usuario}');");
        print_r($redis->smembers($circulo));
        echo '</pre>';
    break;
```

```
    case 'sUnion':
            $cFamilia   = 'circulo:alberto:familia';
            $cFacultad  = 'circulo:alberto:facultad';
            $cFutbol    = 'circulo:alberto:futbol';

            echo mostrarSalida("\$redis->sunion({$cFamilia}, {$cFacultad}, {$cFut-
bol});");
            print_r($redis->sunion($cFamilia, $cFacultad, $cFutbol));
            echo '</pre>';
    break;
    case 'sInter':
            $cFamilia   = 'circulo:alberto:familia';
            $cFacultad  = 'circulo:alberto:facultad';
            $cFutbol    = 'circulo:alberto:futbol';

            echo mostrarSalida("\$redis->sinter({$cFamilia}, {$cFacultad}, {$cFut-
bol});");
            print_r($redis->sinter($cFamilia, $cFacultad, $cFutbol));
            echo '</pre>';
    break;
```

En el código anterior definimos las acciones para agregar y obtener elementos a los círculos **familia**, **facultad** y **futbol**, y luego realizamos la unión e intersección de los tres círculos definidos.

Finalmente vamos a agregar el código que mostramos a continuación, para operar con el tipo de datos **Sorted Set**. Para ello agregamos lo que sigue dentro de la estructura **switch**:

PHP BENCHMARK

En el sitio que se encuentra en la dirección **http://phpbench.com** encontraremos una serie de comparaciones entre las funciones de PHP, que muestran las más óptimas y cómo debemos utilizarlas. Es una buena práctica conocer esta información, ya que puede hacer que nuestros sistemas sean más veloces cambiando algunas costumbres al escribir código.

```
// SORTED SETS ---//
case 'importarSet':
        if (!$redis->exists("nombres")) {
                $file = file("nombres.txt");
                $i = 0;
                foreach ($file as $line) {
                        $line = trim($line);
                        for ($j = 0; $j < strlen($line); $j++) {
                                $prefix = substr($line, 0, $j);
                                $redis->zAdd("nombres", 0, $prefix);
                        }
                        $redis->zAdd("nombres", 0, $line ."*");
                        $i++;
                }
                echo "Se importaron {$i} Nombres";
        }else {
                echo "Los Nombres ya se importaron";
        }
break;
case 'buscarClaveSet':
        $prefix             = $_POST['prefix'];
        $results  = array();
        $count              = 50;
        $rangeLen           = 50;
        $start              = $redis->zRank("nombres", $prefix);

        while(count($results) != $count) {
                $range = $redis->zRange("nombres", $start,
$start+$rangeLen-1);
                $start += $rangeLen;
                if(!$range || count($range) == 0)
                        break;
                foreach($range as $entry) {
                        $minLen = min(strlen($entry), strlen($prefix));
                        if(substr($entry, 0, $minLen) != substr($prefix, 0, $min-
Len))
                                $count = count($results);
```

```
                        if(substr($entry, -1) == "*" && count($results) !=
$count)
                            $results[] = substr($entry, 0, -1);
                    }
            }
            echo json_encode($results);
    break;

    default:
            echo 'PHP/Redis - phpredis';
    break;
```

En el código que mostramos en el bloque anterior hemos realizado la definición de una primera acción, donde pudimos importar nombres del archivo **nombres.txt**. Posteriormente, definimos una búsqueda en donde pudimos mostrar los nombres que coinciden con los caracteres que han sido ingresados por el usuario.

Para terminar, pudimos realizar la definición de una acción por defecto para la estructura **switch**.

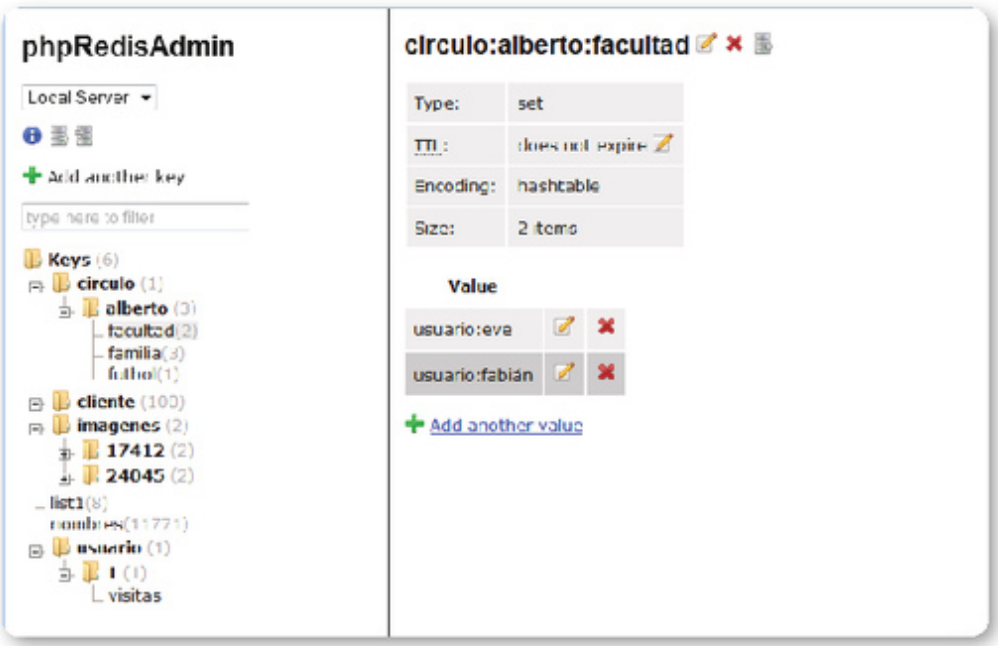


Figura 11. Para verificar cada interacción con la base de datos podemos usar phpRedisAdmin.

Si estamos familiarizados con **jQuery** será sumamente sencillo entender el código. Por ejemplo, para el botón con el id **info**, tenemos el evento **clic**, y allí hacemos una llamada al archivo **funciones.php**. En este archivo capturamos los parámetros y ejecutamos el comando **$redis->info();**, imprimimos la respuesta y luego la mostramos en el contenedor **rServidor**.

RESUMEN

Hemos aprendido a utilizar Redis con PHP mediante un cliente que establece la conexión entre ambos. Para esto, hemos estudiado **Predis** y **phpredis**, centrándonos en este último ya que es fácil de usar y posee todas las características necesarias para el desarrollo de proyectos escalables. Para gestionar las bases de datos utilizamos **phpRedisAdmin**, que es multiplataforma. Finalmente, desarrollamos un ejemplo integral donde incluimos comandos para todos los tipos de datos y casos de aplicación reales, como círculos de amigos y contador de visitas.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Un cliente es un lenguaje de programación? ¿Por qué?
2. ¿Los clientes solo funcionan con PHP?
3. ¿Cuáles son los clientes que funcionan con PHP?
4. ¿Cómo se conecta phpredis a Redis?
5. ¿Se puede instalar phpredis en Linux?
6. ¿phpredis es una librería escrita en C y compilada para PHP?
7. ¿Sin phpredis podemos acceder a los comandos de Redis? ¿Por qué?
8. ¿Qué podemos hacer con phpRedisAdmin?
9. ¿Podemos configurar varios servidores en phpRedisAdmin?
10. ¿Es posible administrar varias bases de datos en phpRedisAdmin?

EJERCICIOS PRÁCTICOS

1. Cree una estructura utilizando el tipo de dato string y almacene la información del navegador.
2. En la estructura incremente un contador por cada vez que se visite la página.
3. Utilizando el tipo de dato hash genere una estructura que contenga los elementos **provincia**, **idProvincia**, **localidad**, **idLocalidad** y **cantidad de habitantes**.
4. Cree una estructura utilizando el tipo de dato que mejor se ajuste para generar grupos de interés, y guarde la información de cada integrante.
5. Utilizando el tipo de dato list genere un historial de accesos.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*04

Administración de Redis

Veremos la configuración de servidores esclavos, las opciones de persistencia de datos y el modelo de seguridad de Redis. Además, conoceremos la utilidad redis-benchmark, que nos permite realizar pruebas de rendimiento con diferentes configuraciones. Por último, desarrollaremos un ejemplo similar a Twitter, en el que aplicaremos algunos conceptos vistos hasta el momento.

- Configuración de Redis 126
- Replicación 129
- Tipos de persistencia 132
- Seguridad 139
- Rendimiento de Redis (benchmarks) 141
- Ejemplo de prueba: Mini-twt 142
- Resumen 169
- Actividades 170

Configuración de Redis

Como vimos en los **Capítulos 2** y **3**, Redis es capaz de funcionar sin directivas extras mediante una configuración por defecto, pero trabajar de esta manera es únicamente recomendable para entornos de desarrollo.

La aplicación posee un archivo de configuración que contiene las directivas y una explicación en inglés de la función de cada una. El archivo se denomina **redis.conf** y se encuentra dentro del directorio de instalación. Las directivas tienen un formato como el siguiente:

```
port 6379
```

En la línea de arriba se establece el puerto por el cual se comunicará Redis empleando la directiva **port**. Para argumentos que contengan espacios es necesario utilizar comillas dobles.

Pasaje de parámetros por la línea de comandos

A partir de la versión 2.6 es posible pasar parámetros de configuración utilizando la línea de comandos. Esta característica es muy útil para realizar pruebas de comportamiento, donde el formato de los argumentos debe ir con el prefijo **--** (doble guion).

Ahora veremos cómo crear una instancia de Redis usando el puerto **6380** como esclavo de otra instancia que opera en el puerto **6379**:

```
./redis-server --port 6380 --slaveof 127.0.0.1 6379
```

SPONSORS DE REDIS

Desde mayo del 2013 Redis está siendo sponsoreado por **Pivotal** (**www.gopivotal.com**); hasta entonces, la compañía que sponsoreaba el proyecto era **VMware**. En el pasado han aportado donaciones económicas empresas como **Engine Yard**, **Hirmeister**, **Citrusbyte**, **Slicehost** y **Linode**.

Reconfigurar Redis en tiempo de ejecución

Redis ofrece la posibilidad de reconfiguración sin detener ni reiniciar el servicio, mediante los comandos **CONFIG GET** y **CONFIG SET**. No todas las directivas pueden funcionar de esta manera, por lo que, para saber cuáles admiten la reconfiguración, debemos ejecutar el comando **CONFIG GET** *.

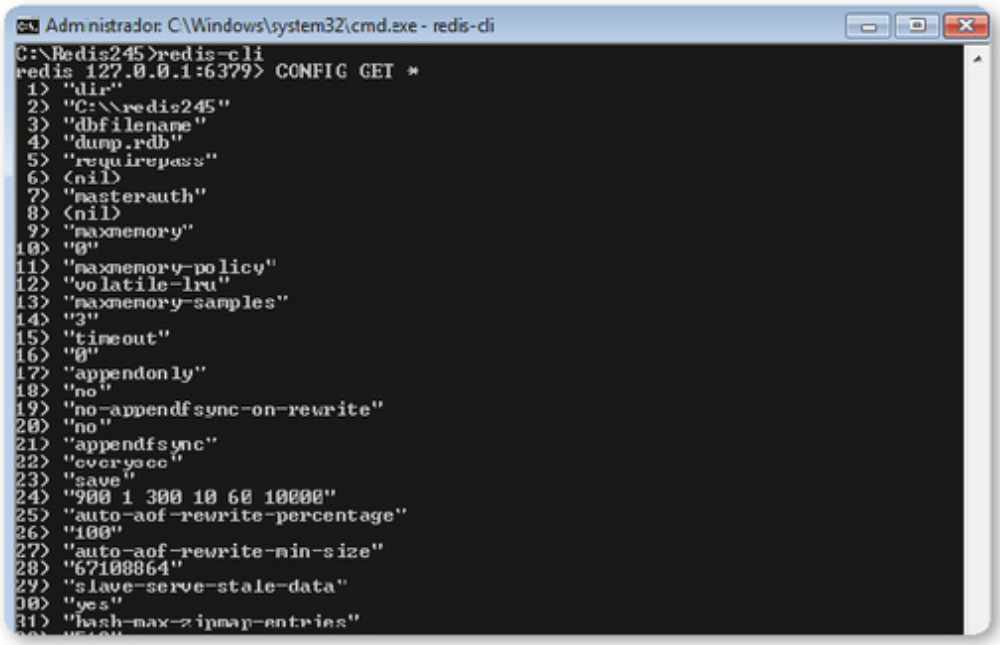


Figura 1. Lista de parámetros y sus respectivos valores, que podemos reconfigurar mediante **CONFIG SET**.

Cuando se ejecuta **CONFIG SET**, los parámetros establecidos son automáticamente cargados a Redis y tendrán efecto en el siguiente comando. Sin embargo, estas reconfiguraciones no afectan al archivo de configuración, de manera que, cuando Redis sea reiniciado, cargará los parámetros que están definidos en **redis.conf** sin ningún cambio.

HEADER FIJO

Una característica propia del diseño de las redes sociales es que mantienen la cabecera del sitio en una posición fija y siempre visible. No importa en qué parte del contenido se encuentre el usuario para poder acceder al menú de navegación. Esto mejora la experiencia cuando se tiene gran cantidad de contenidos.

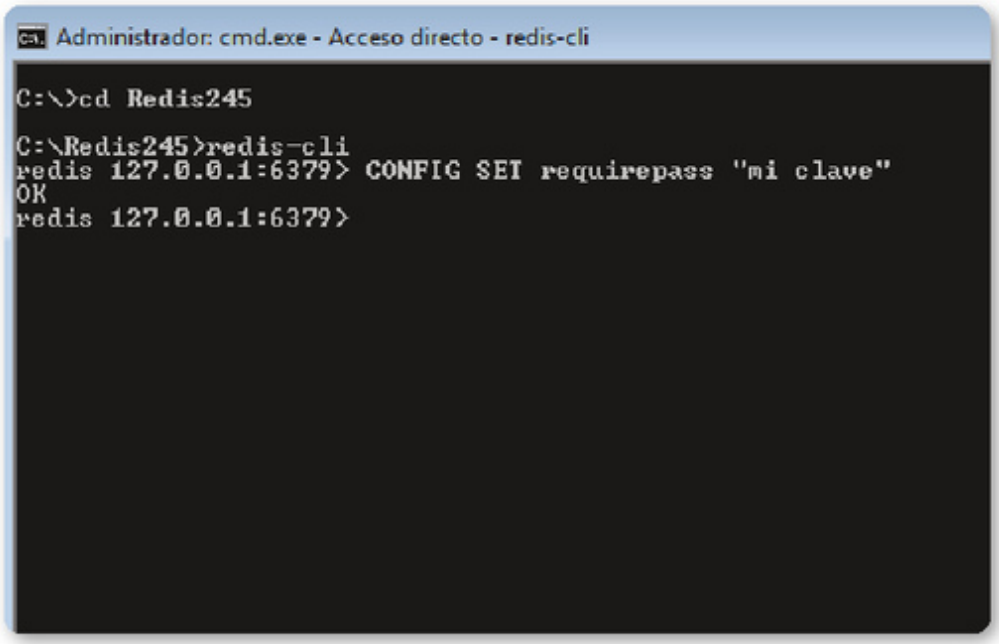


Figura 2. En la imagen se muestra cómo cambiar la clave de autenticación en tiempo de ejecución.

Configurar Redis como caché

En algunas situaciones puede ser útil combinar una base de datos relacional con Redis, ya que podremos aprovechar la velocidad que ofrece y utilizarla como mecanismo de caché, donde cada clave tendrá una expiración establecida o una configuración similar.

Por ejemplo, veamos lo siguiente:

```
maxmemory 2mb
maxmemory-policy allkeys-lru
```

Como definimos un tamaño máximo de 2 MB de memoria, todas las claves serán eliminadas cuando se aproximen al tamaño máximo establecido. Este mecanismo es más eficiente que tener una configuración de expiración, porque definir un tiempo de vida para cada clave demanda memoria adicional.

En caso de que un proyecto necesite utilizar Redis como caché y además requiera almacenar datos, es recomendable crear dos instancias de Redis, una como caché configurada de este modo y otra con los requerimientos necesarios.

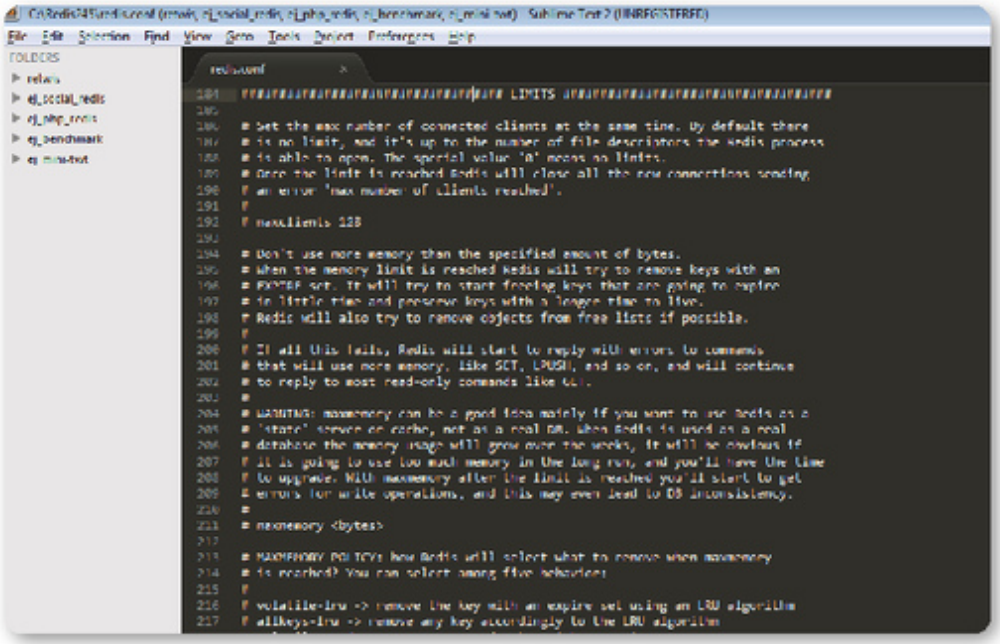


Figura 3. En el archivo **`redis.conf`** podemos definir las reglas de expiración, en la sección **`LIMITS`**.

Replicación

En Redis la replicación permite usar una arquitectura maestro-esclavo, donde los servidores esclavos son una copia exacta de los servidores maestros. Esto parece complejo pero en realidad es muy fácil de configurar y utilizar. A continuación, detallamos los factores más importantes de esta arquitectura:

> UN MAESTRO PUEDE TENER VARIOS ESCLAVOS, LOS QUE TAMBIÉN PUEDEN ACEPTAR ESCLAVOS

- Un maestro puede tener múltiples esclavos.
- Los esclavos son capaces de admitir otros esclavos, en un nivel jerárquico.
- La replicación no se bloquea del lado del maestro: el maestro seguirá sirviendo consultas mientras uno o más esclavos realizan la primera sincronización.
- La replicación no se bloquea del lado del esclavo: mientras el esclavo está intentando realizar la primera sincronización puede atender las consultas utilizando la versión actual del conjunto de datos.

- Es posible utilizar la replicación para evitar el proceso de persistencia del lado del maestro, donde solo es necesario comentar todas las directivas **save** en el archivo **redis.conf** del maestro y luego conectar el esclavo correspondiente.

Funcionamiento de la replicación de Redis

Cuando se configura un esclavo, este se conecta y envía el comando **SYNC** para sincronizarse; por su parte, el maestro inicia un proceso de guardado asincrónico y almacena todos los comandos que han modificado los datos durante el proceso. Cuando finaliza el guardado, el maestro transfiere la base de datos al esclavo, quien la almacena en el disco y la carga en su memoria. Luego, el maestro envía al esclavo todos los comandos almacenados junto con los nuevos comandos recibidos por los clientes que han modificado la base de datos.

Los esclavos son capaces de reconectarse automáticamente cuando se pierde la conexión con el maestro y, si el maestro recibe solicitudes de sincronización concurrentes, realiza un solo guardado con el fin de optimizar el proceso y poder servir a todas las solicitudes.

Configuración de un esclavo

Para configurar un servidor esclavo de otro, solo será necesario que ingresemos la directiva **slaveof** y, como argumentos, escribamos la dirección IP y el puerto del servidor maestro:

```
slaveof 192.168.1.1 6379
```

HOSTING REDIS

El sitio **http://redis4you.com** ofrece un sistema de almacenamiento de Redis muy interesante para implementar en nuestros proyectos. Además del almacenamiento de las bases de datos, cuenta con clientes para diferentes lenguajes de programación y ejemplos de implementación. Posee un plan gratuito y dos pagos. Sin duda, es una muy buena alternativa.

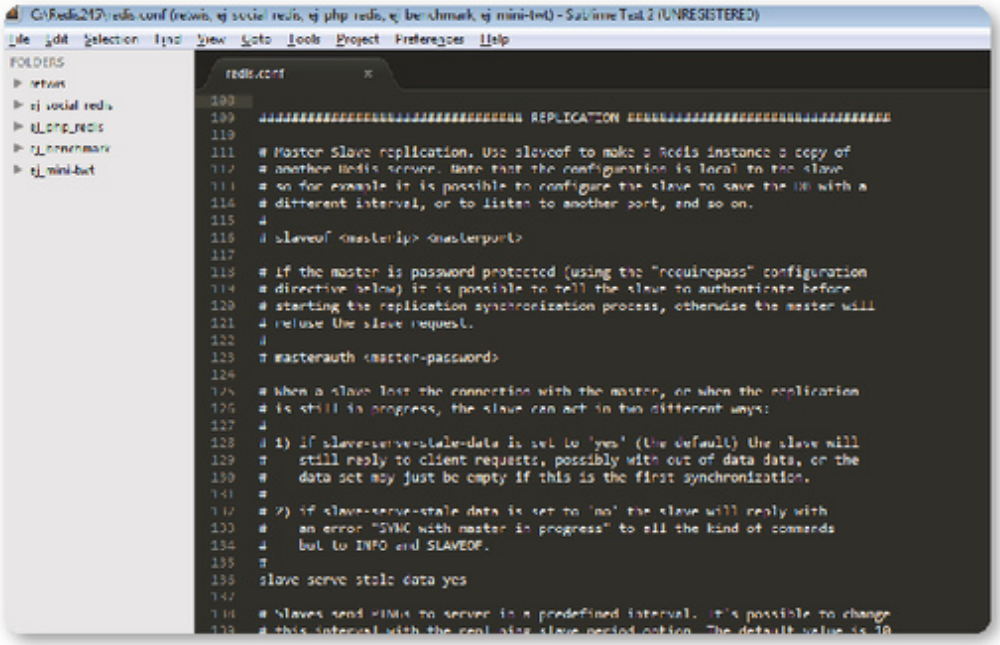


Figura 4. En el archivo **redis.conf** podemos configurar una instancia como esclava de otra, en la sección **REPLICATION**.

Configuración de un esclavo de solo lectura

Debemos considerar que, desde la versión 2.6 de Redis, los esclavos son capaces de soportar el modo de solo lectura, que es habilitado por defecto y puede ser configurado en el archivo **redis.conf** mediante la directiva **slave-read-only**. Esta configuración también puede ser modificada en tiempo de ejecución utilizando el comando **CONFIG SET**.

> CONFIG.SET NOS PERMITE CONFIGURAR EL MODO DE SOLO LECTURA EN LOS ESCLAVOS

Es importante tener en cuenta que configurar esclavos donde se permitan comandos de escritura puede resultar un problema, ya que puede ocurrir que mientras se estén actualizando datos se pierda la conexión con el maestro, lo que ejecutaría el proceso de resincronización, quedando sin efecto las escrituras realizadas previamente. También recordemos que al reiniciar un esclavo automáticamente se sincroniza con el maestro que corresponde, y de esta manera se pierden todos los cambios que hemos realizado en la base de datos conectada.

Configuración de autenticación de esclavo a maestro

Si el maestro tiene establecida una clave de autenticación es necesario configurar los esclavos para que puedan realizar las operaciones de sincronización. Podemos hacerlo agregando la siguiente línea en el archivo de configuración:

```
masterauth <password>
```

También es posible establecer la clave en tiempo de ejecución, utilizando esta sentencia:

```
config set masterauth <password>
```

Tipos de persistencia

Redis dispone de dos mecanismos para realizar la persistencia de datos desde la memoria al disco rígido: **RDB** (**Redis Data Base**) y **AOF** (**Append Only File**). Algunas de sus características son:

- El tipo de persistencia **RDB** se encarga de guardar los datos en intervalos específicos de tiempo.
- El tipo de persistencia **AOF** registra cada operación de escritura recibida por el servidor. Cuando el servidor inicia, se restaura y reconstruye la base de datos original. También registra los comandos usando el formato del protocolo de Redis en modo de solo lectura.

BIG DATA

El termino **Big Data** existe desde hace bastante tiempo y se refiere al problema de procesamiento de datos en el cual se incluyen problemáticas como capacidad de almacenamiento, capacidad de almacenar gran variedad de datos y velocidad en el movimiento de la información. Hoy en día, gracias a las bases de datos NoSQL, es mucho más fácil referirse al tema y proponer una solución real.

- Es posible deshabilitar todas las opciones de persistencia de modo que los datos necesarios solo estén disponibles mientras el servidor se encuentre funcionando.
- Redis ofrece la posibilidad de combinar los dos mecanismos de persistencia en la misma instancia.

Persistencia mediante RDB

Por defecto, Redis utiliza el mecanismo RDB para persistir, que guarda los datos en intervalos de tiempo en un archivo compacto llamado **dump.rdb**. Este mecanismo es muy útil para realizar copias de seguridad y nos permite restaurar fácilmente diferentes versiones en caso de que ocurra algún problema. Debemos considerar que, al ser un archivo, puede ser transferido a una unidad de almacenamiento externo de manera sencilla.

> EL ARCHIVO DUMP.RDB GUARDA LOS DATOS EN INTERVALOS DE TIEMPO

RDB nos permite configurar la frecuencia del almacenamiento de datos según la cantidad de cambios; por ejemplo, cada 60 segundos si se han realizado al menos 1000 cambios. Para esto debemos editar el archivo **redis.conf** en la sección **SNAPSHOTTING** y agregar lo siguiente:

```
save <segundos> <cambios>
```

Reemplazamos **<segundos>** y **<cambios>** por los valores que queremos establecer para el guardado automático. A esta estrategia se la conoce como **SNAPSHOTTING**. Para cambiar las secuencias de guardado debemos editar el archivo **redis.conf** modificando la directiva **save**.

Una característica importante de RDB es que permite reiniciar de manera rápida una instancia con un conjunto de datos de gran tamaño. Sin embargo, para implementaciones donde es necesario minimizar las posibilidades de pérdida de datos no es recomendable utilizar solo este mecanismo. Por ejemplo, si tenemos configurado RDB para persistir cada cinco minutos en cien modificaciones de la base de datos, nos podemos encontrar con que, antes de llegar a la cantidad establecida, por alguna razón, se apague el servidor provocando la pérdida irrecuperable de datos de los últimos minutos.

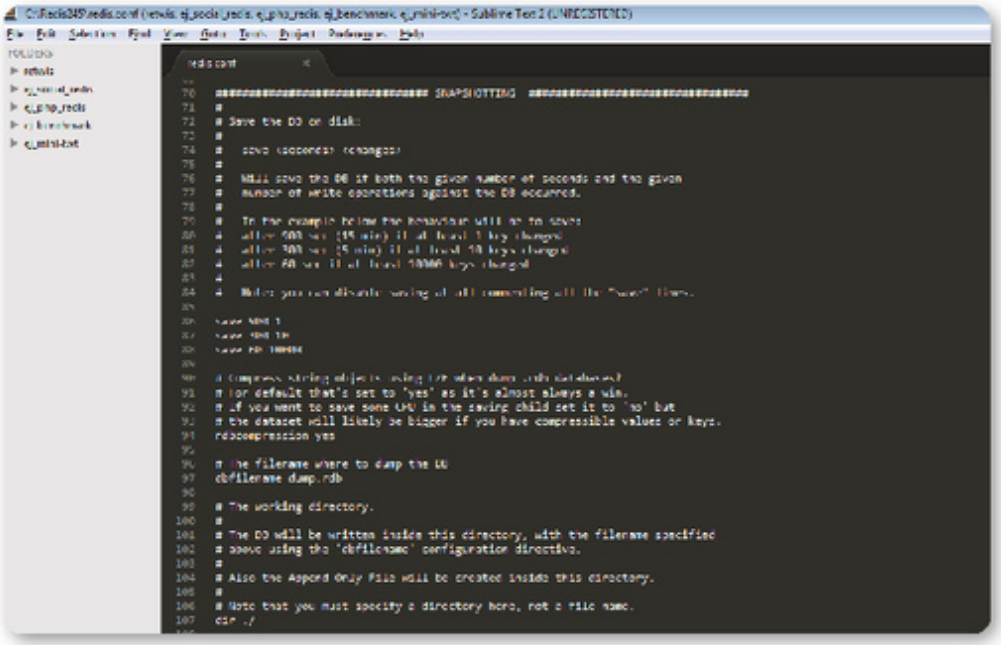


Figura 5. En la imagen vemos los valores por defecto de la directiva **save**.

Persistencia mediante AOF

Antes de conocer las características que ofrece AOF, definiremos brevemente un concepto: **fsync** es una función que se utiliza para copiar todas las partes de un archivo que está en memoria al disco rígido, y que espera a que el dispositivo responda indicando que todas las partes fueron almacenadas con éxito.

La persistencia AOF permite establecer diferentes configuraciones utilizando **fsync**, como por ejemplo persistir cada segundo, en cada actualización o no persistir.

AOF utiliza un archivo de solo lectura en el que se van agregando los comandos, entonces, en caso de que el servidor se pare inesperadamente, el archivo no quedará inconsistente.

DISEÑO ADAPTATIVO

Los dispositivos móviles están cada vez más cerca de los usuarios y, como desarrolladores, debemos ser capaces de diseñar sistemas que se adapten a la mayor cantidad de dispositivos. Con la técnica de **responsive design** nuestros sistemas se podrán ver en pantallas de diferentes resoluciones.

Dado que el archivo AOF contiene todas las operaciones llevadas a cabo, una detrás de la otra y en un formato fácil de entender, es posible editarlo con sencillez. Por ejemplo, si hemos eliminado todas las claves por error y no se han registrado comandos nuevos, podemos recuperar los datos con solo detener el servidor, quitar el último comando del archivo y reiniciar Redis.

En comparación con RDB, los archivos AOF son más grandes para la misma cantidad de datos y suelen ser más lentos dependiendo de la configuración. Por lo general, con una configuración de persistencia cada segundo, el rendimiento sigue siendo muy bueno.

Para activar el mecanismo AOF editamos el archivo **redis.conf**, buscamos la sección **APPEND ONLY MODE** y configuramos la directiva **appendonly**.

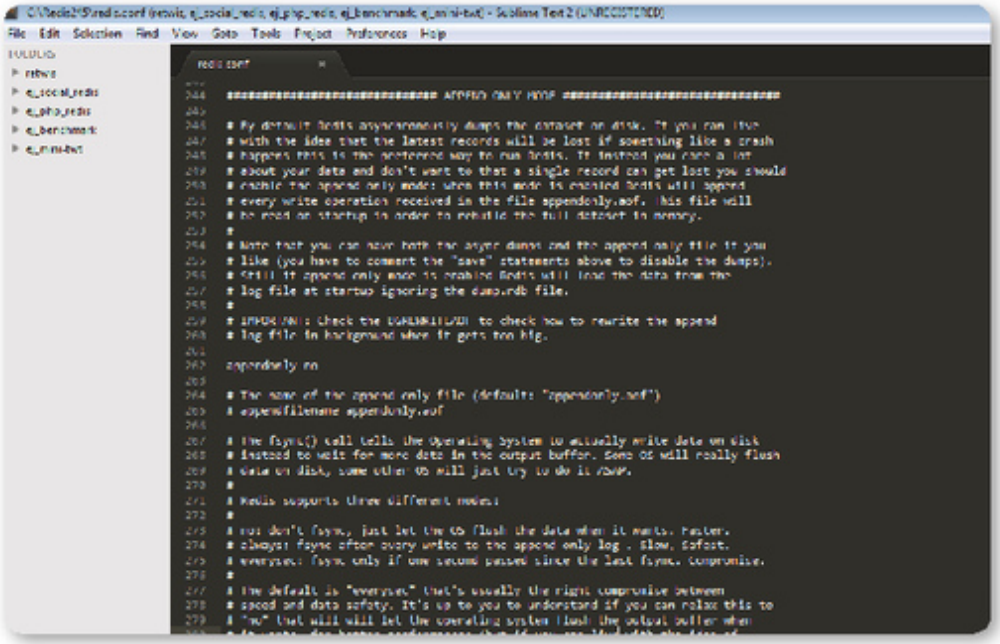


Figura 6. En la configuración por defecto, el mecanismo AOF se encuentra deshabilitado.

Las tres opciones posibles de **fsync** son:

- **appendfsync always**: cada comando de escritura se agrega al archivo AOF. Es muy seguro pero lento.
- **appendfsync everysec**: es muy rápido, similar a **SNAPSHOTTING**, donde en caso de fallas solo se pierde un segundo.
- **appendfsync no**: el almacenamiento de los datos depende del sistema operativo. Es rápido pero menos seguro.

Recomendación de persistencia

Es recomendable utilizar los métodos RDB y AOF de manera conjunta, ya que de este modo les proporcionaremos a nuestros datos un grado de seguridad comparable con otros motores de gran prestigio.

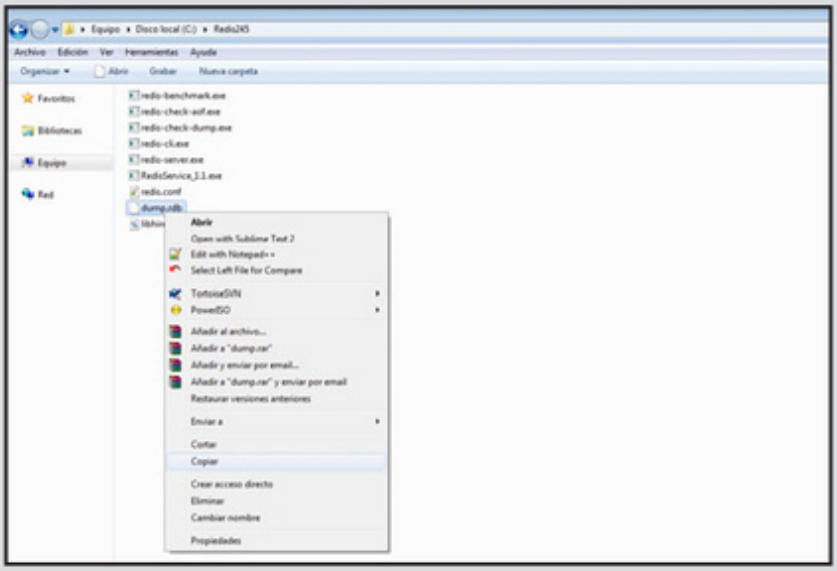
Para sistemas donde la seguridad es muy importante pero es posible seguir trabajando con algunos minutos de pérdida, en caso de fallas o desastres únicamente se puede usar RDB. El uso de AOF como único método de persistencia es desaconsejable, porque RDB persistiendo cada cierto tiempo nos permite tener copias de seguridad, un reinicio más rápido y un respaldo en caso de que el motor AOF falle.

Habilitar AOF mientras se utiliza RDB

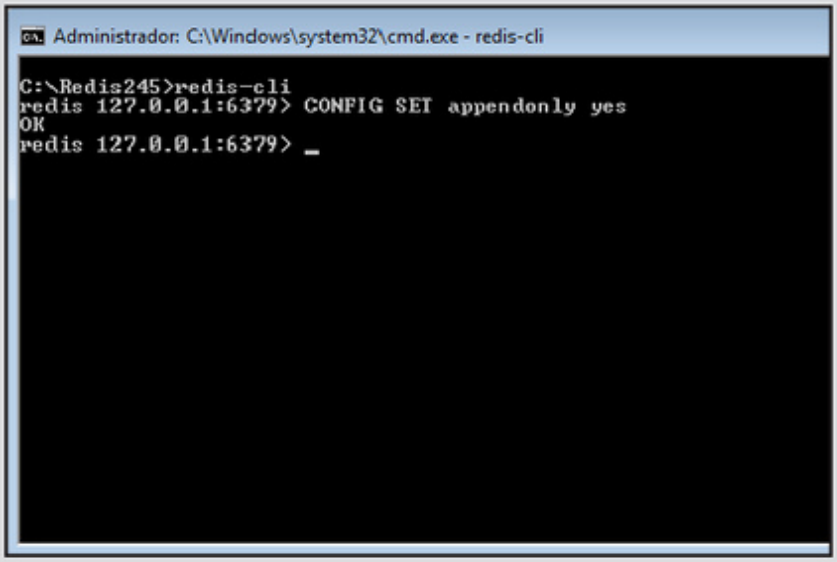
A partir de Redis en su versión 2.2 es posible habilitar el mecanismo AOF para persistir sin necesidad de reiniciar el servidor.
A continuación, veremos cómo hacerlo.

PAP: INICIO DE LA PERSISTENCIA MEDIANTE AOF

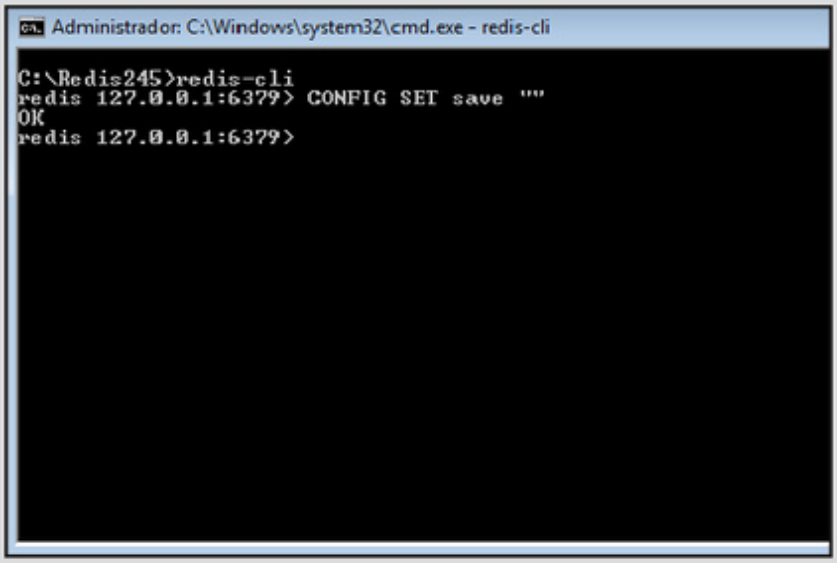
01 Haga una copia del archivo `dump.rdb` y guárdelo en un lugar seguro. Es recomendable hacerlo en un almacenamiento externo al sistema.



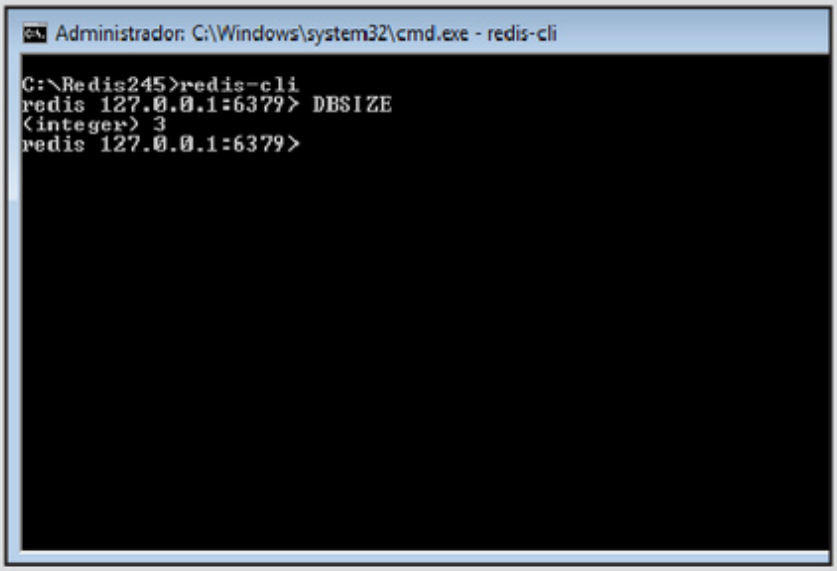
02 Abra una terminal y ejecute la utilidad `redis-cli`, luego habilite la persistencia AOF con el siguiente comando: `config set appendonly yes`.



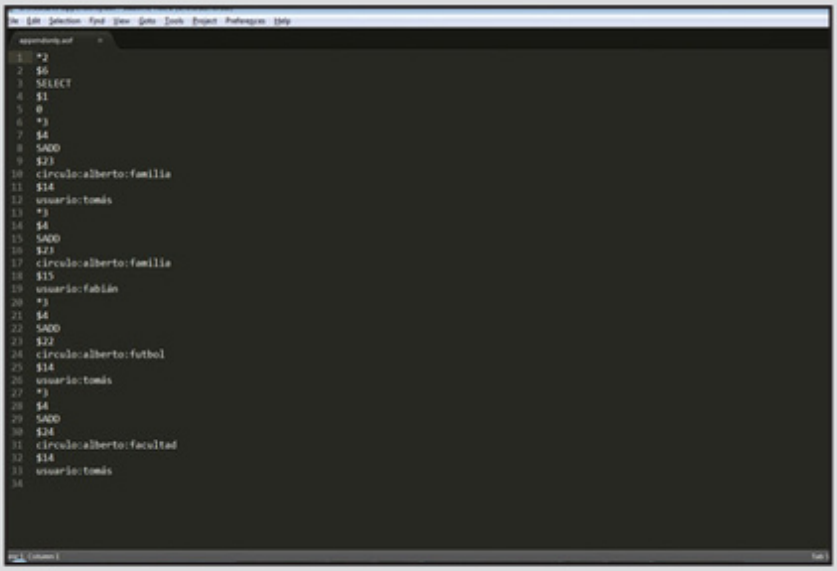
03 Puede deshabilitar la persistencia RDB con el comando `config set save ""`, pero no es recomendable ya que RDB puede realizar backups periódicas.



04 Verifique que la base de datos contenga la misma cantidad de claves que tenía anteriormente, mediante el comando DBSIZE.



05 Finalmente, abra el archivo `appendonly.aof` y verifique que se hayan agregado correctamente las claves.



Es importante editar el archivo **redis.conf** y habilitar la persistencia con AOF. De otra manera, cuando reiniciemos el servidor no estarán disponibles los datos guardados en el archivo **appendonly.aof**, ya que, como vimos anteriormente, los cambios realizados en tiempo de ejecución se pierden al reiniciar Redis.

Seguridad

Redis ha sido diseñado para ser accedido desde entornos confiables, por lo que no es recomendable exponer una instancia directamente en Internet o en entornos donde usuarios no autorizados puedan acceder a los puertos o sockets. Es necesario crear una capa de seguridad para validar el acceso, los datos que proporcionan y las operaciones que pueden realizar los usuarios en la base de datos.

Seguridad de acceso

Redis ofrece una directiva para indicar las direcciones IP que pueden acceder a la instancia. Para incorporar una dirección solo es necesario agregar, en el archivo **redis.conf**, lo siguiente:

```
bind <Dirección IP>
```

Para permitir el acceso únicamente desde el servidor local, debemos configurar la directiva del siguiente modo:

```
bind 127.0.0.1
```

NORMALIZE.CSS

Normalize.css es, básicamente, un archivo de estilos CSS que establece valores iniciales para todas las etiquetas HTML. Su uso es muy recomendado, ya que existen elementos que se muestran de manera diferente dependiendo del navegador. Para usarlo, solo lo descargamos y lo agregamos al sitio antes de los CSS propios. Podemos obtenerlo en **http://necolas.github.com/normalize.css**.

Un aspecto importante es definir políticas de acceso a los puertos mediante una firewall; de esta manera, podremos prevenir la ejecución de ataques de visitantes externos.

Autenticación

Redis también provee una directiva de seguridad en la cual se puede establecer una clave de autenticación. Así, el servidor aceptará únicamente las consultas de los usuarios que se hayan autenticado. Para establecer la clave debemos asignar un valor a la directiva **requirepass** en el archivo **redis.conf**:

```
requirepass "123 Mi Nueva Clave 123"
```

Debido a que la clave es establecida en el archivo de configuración, recomendamos utilizar una de gran longitud para afrontar posibles ataques por fuerza bruta.

Deshabilitar comandos

La configuración por defecto de Redis permite a los clientes utilizar todos los comandos disponibles, pero esto puede resultar problemático, por lo que existe la posibilidad de deshabilitar o renombrar comandos para que sea imposible ejecutarlos.

Por ejemplo, si ofrecemos un servicio de almacenamiento compartido, los usuarios no deberían poder ejecutar el comando **CONFIG**, ya que podrían modificar la configuración de nuestro servidor. Para ello necesitaremos editar el archivo **redis.conf** y agregar lo siguiente:

```
rename-command CONFIG 0f52f59e41c299c02d5240be4456c
```

IMPACTO DE NOSQL

Según un estudio realizado por el Instituto de Analíticas Avanzadas de la Universidad de Carolina del Norte se ha determinado que menos del 2,5% de los programadores tienen algún conocimiento de bases de datos NoSQL. En Europa existe una tendencia clara de aplicación de este tipo de bases de datos.

En el ejemplo, hemos renombrado el comando **CONFIG** para hacerlo indescifrable. También es posible deshabilitar este o cualquier otro comando con solo asignarle una cadena vacía:

```
rename-command CONFIG ""
```

Rendimiento de Redis (benchmarks)

Redis ofrece una utilidad llamada **redis-benchmark**, que simula operaciones de lectura y escritura en la base de datos. Es posible pasarle parámetros para configurar, entre otras cosas, la cantidad de clientes y de consultas que serán ejecutadas en forma concurrente.

Para ejecutarla abrimos una consola, vamos al directorio donde tenemos instalado Redis e ingresamos el comando **redis-benchmark**.

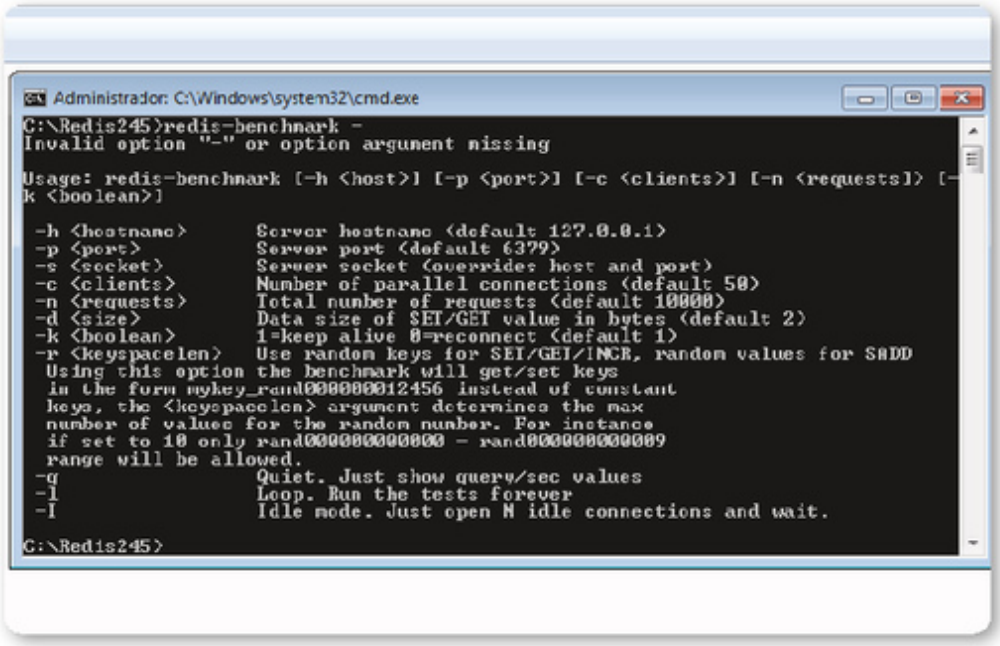


Figura 7. Podemos ver los parámetros que dispone la utilidad ejecutando **redis-benchmark**.

En el ejemplo que presentamos a continuación veremos cómo ejecutar la utilidad. Le pasaremos el parámetro **-q** para indicar que

solo mostraremos las consultas por segundo, el parámetro **-n** para establecer la cantidad de solicitudes y el parámetro **-c** para indicar la cantidad de clientes que se van a simular.

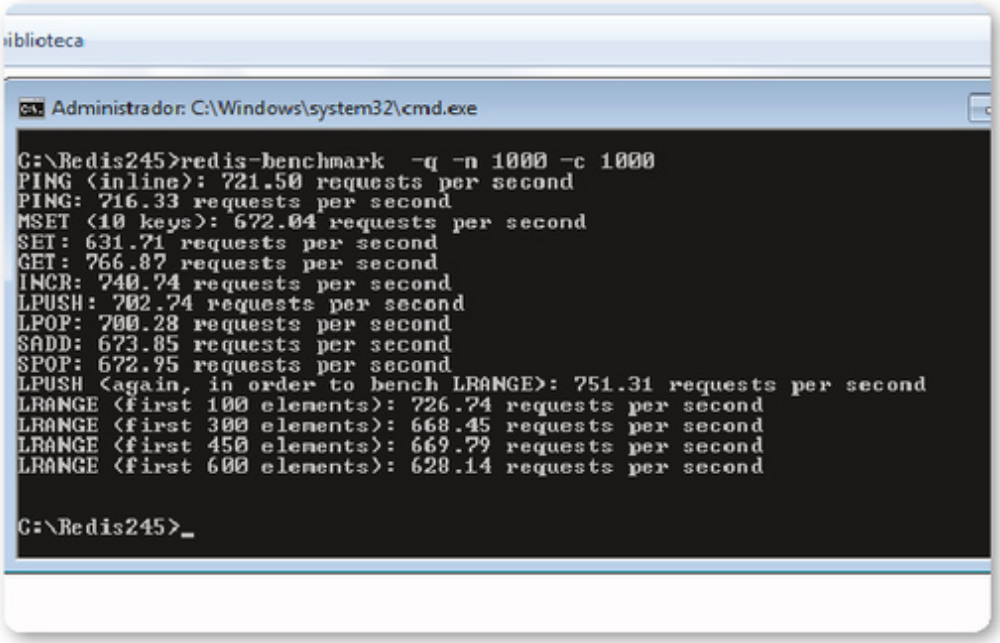


Figura 8. De esta manera podemos hacer una prueba de rendimiento con la utilidad de Redis.

Ejemplo de prueba: Mini-twt

El desarrollo del siguiente ejemplo, al que llamaremos **Mini-twt**, está inspirado en el funcionamiento de la red social Twitter. En él, los usuarios se podrán registrar, publicar posts, seguir a otros usuarios y tener sus propios seguidores. Realizaremos la implementación utilizando los lenguajes PHP y JavaScript.

VERIFICAR NUESTRO DISEÑO

Los desarrolladores de **Responsinator** han creado una herramienta muy útil que permite probar cómo se verán nuestros sistemas en diferentes dispositivos, como **smartphones** y **tablets**. Debemos acceder a **www.responsinator.com** e ingresar la dirección de nuestro sitio.

El comportamiento de refresco automático en el timeline no lo desarrollaremos en esta instancia porque es en los siguientes capítulos donde estudiaremos Node.js, que hace posible esta característica.

Antes de empezar a escribir el código vamos a ver en la **Figura 9** cómo quedará nuestro ejemplo.

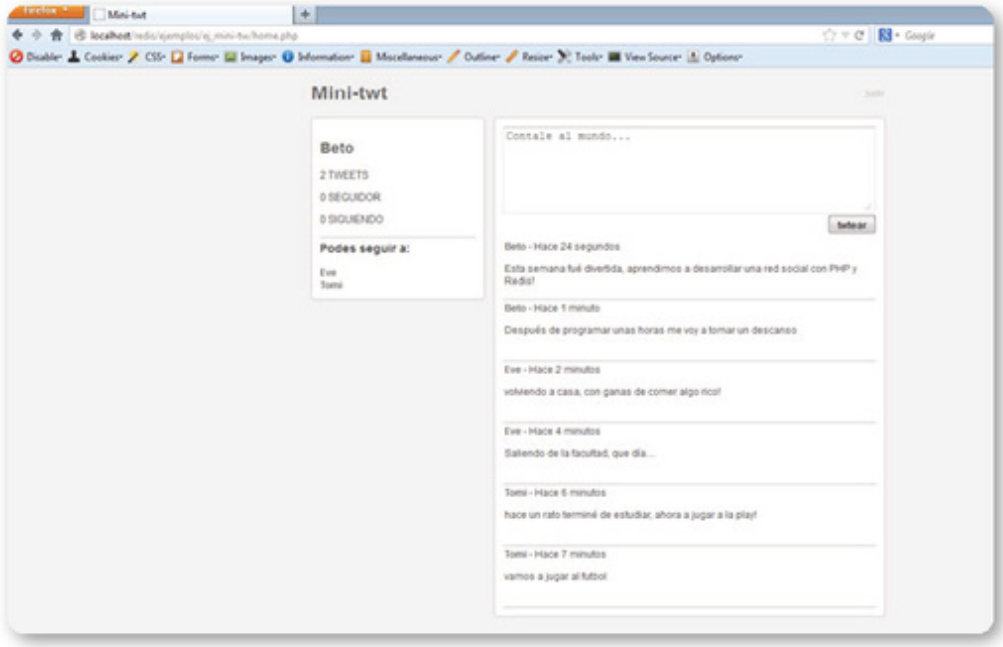


Figura 9. En el timeline general se pueden ver todos los posts que han escrito los usuarios del sistema.

Para comenzar, vamos a definir la estructura del ejemplo. Dentro del directorio público de nuestro servidor web creamos una carpeta con el nombre **ej_mini_twt** y dentro de ella otras dos, denominadas **css** y **js**. Estas últimas contendrán los archivos de estilos y JavaScript correspondientes. Todos los archivos PHP irán a la misma altura que CSS y JS, es decir, en el directorio raíz del proyecto.

RETINA DISPLAY

Los dispositivos móviles como laptops, teléfonos y tabletas han evolucionado de tal manera que es necesario tener en cuenta cómo se verán nuestros sitios en las diferentes resoluciones existentes. **Retina display** es una tecnología que va mas allá de estas resoluciones e implementa pantallas con el doble de densidad de píxeles de la generación tradicional.

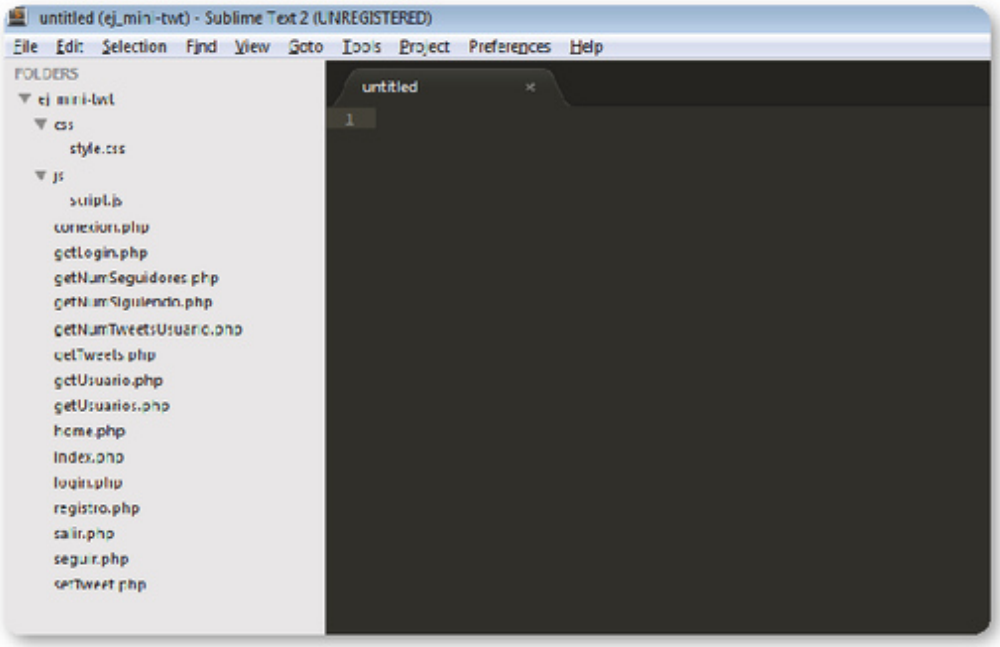


Figura 10. Así quedará la estructura de nuestro ejemplo.

Ahora determinaremos el aspecto gráfico. Para esto, creamos un archivo, donde ingresamos el código que detallamos a continuación, y lo guardamos en el directorio **css** con el nombre **style.css**.

```
body{
        background-color: #F2F2F2;
        color: #444444;
        font: 12px Arial,Helvetica,sans-serif;
        margin: 0;
        padding: 0;
}
header{
        color: #666666;
        font-weight: bold;
        height: 60px;
        margin: 0 auto;
        text-shadow: 1px 1px 0 #FFFFFF;
        width: 700px;
}
section{
        background-color: #FFFFFF;
```

```
border: 1px solid #CCCCCC;
        border-radius: 5px;
        -moz-border-radius:5px;
        -webkit-border-radius: 5px;
        min-height: 160px;
        padding: 10px;
}
header h1{
        float: left;
}
header .salir{
        color: #CCCCCC;
        float: right;
        margin-top: 25px;
}
a{
        color: #666666;
        font-weight: bold;
        text-decoration: none;
        text-transform: capitalize;
}
a:hover{
        color: #444444;
}
h3{
        border-top: 1px solid #CCCCCC;
        padding-top: 5px;
}
textarea{
        display: block;
        height: 100px;
        width: 448px;
}
input[type="text"],
input[type="password"]{
        display: block;
}
```

```
.wrapper{
        clear: both;
        margin: 0 auto;
        width: 700px;
}
.info{
        float: left;
        width: 190px;
}
.home{
        float: right;
        width: 455px;
}
#login{
        border-right: 1px solid #CCCCCC;
        float: left;
        padding: 0 80px;
}
#registro{
        float: left;
        padding: 0 80px;
}
.result{
        height: 20px;
}
#post_btn{
        float: right;
}
#usuario, .usuarios{
        display: block;
}
.usuarios:hover{
        background-color: #F2F2F2;
}
#timeline{
        display: block;
        margin: 30px 0 0;
```

```
        min-height: 420px;
        }
.post{
        border-bottom: 1px solid #CCCCCC;
        min-height: 70px;
        padding: 2px;
}
```

No es necesario explicar este código, ya que solo se definen los estilos gráficos. A continuación, creamos el archivo **conexion.php**, en el cual generamos una instancia de Redis, nos conectamos al servidor y seleccionamos una base de datos para nuestro proyecto.

```
<?php
        define('HOST','127.0.0.1');
        define('PUERTO',6379);
        define('BD',0);
        try{
                // instanciamos Redis
                $redis = new Redis();
                // nos conectamos al servidor
                $redis->connect(HOST, PUERTO);
                // seleccionamos la base de datos
                $redis->select(BD);
        }catch(Exception $e){
                die('ERROR '.$e->getCode().': '.$e->getMessage());
        }
?>
```

FASTICE

FastIce es un framework php y motor de plantillas, que opera de manera muy veloz utilizando como motor de persistencia a Redis. Entre sus características se destacan las siguientes: provee estructura html, sistema de caché avanzado y es posible utilizar js, metas y CSS. Podemos conocerlo mejor en: **http://fastice.tk**.

El paso siguiente es crear la primera interfaz con las acciones de ingreso al sistema y registro de un usuario nuevo.

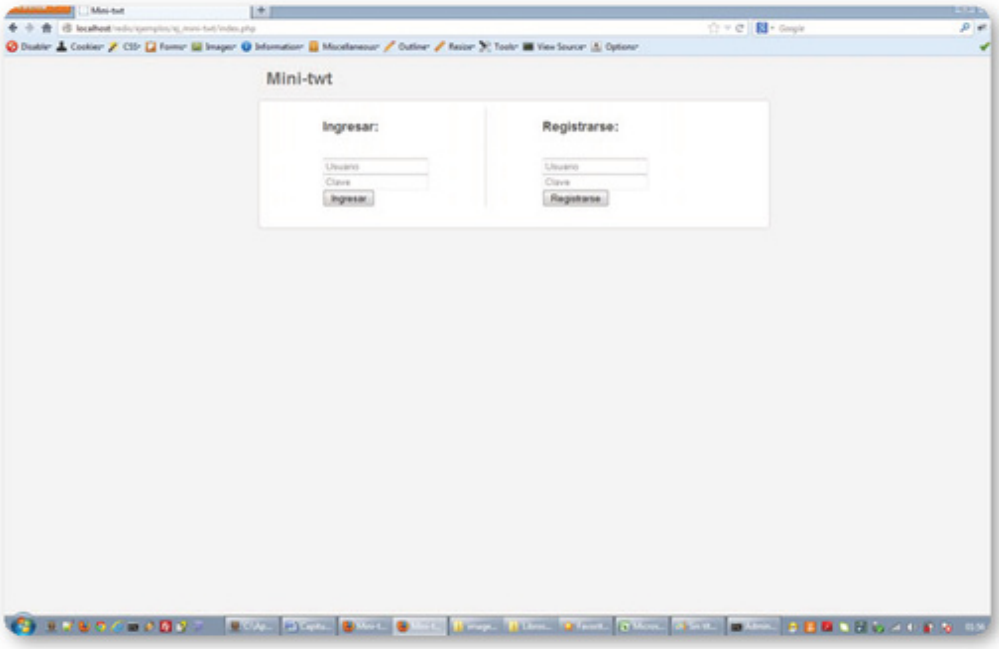


Figura 11. Las primeras acciones que desarrollaremos son de registro e ingreso al sistema.

Primero creamos un archivo **index.php** con el siguiente código:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Mini-twt</title>
                <link type="text/css" href="css/style.css" rel="stylesheet" />
                <script type="text/javascript" src="http://ajax.googleapis.com/
ajax/libs/jquery/1.9.0/jquery.min.js"></script>
                <script type="text/javascript" src="js/script.js"></script>
                <!--[if lt IE 9]><script src="http://html5shiv.googlecode.com/svn/
trunk/html5.js"></script><![endif]-->
        </head>
        <body>
                <header>
                        <h1><a class="titulo" href="index.php">Mini-twt</
```

```
    a></h1>
    </header>
                    <section class="wrapper">
                            <article>
                                    <div id="login">
                                            <h2>Ingresar:</h2>
                                            <div class="result" id="log_re-
sult"></div>
                                            <form id="log_form" action="">
                                                    <input type="text"
id="log_usuario" placeholder="Usuario"/>
                                                    <input type="password"
id="log_clave" placeholder="Clave"/>
                                                    <input type="button"
id="log_btn" value="Ingresar" />
                                            </form>
                                    </div>
                                    <div id="registro">
                                            <h2>Registrarse:</h2>
                                            <div class="result" id="reg_re-
sult"></div>
                                            <form id="reg_form" action="">
                                                    <input type="text"
id="reg_usuario" placeholder="Usuario"/>
                                                    <input type="password"
id="reg_clave" placeholder="Clave"/>
                                                    <input type="button"
id="reg_btn" value="Registrarse" />
                                            </form>
                                    </div>
                            </article>
                    </section>
                    <footer>
                    </footer>
            </body>
    </html>
```

En el código anterior establecimos la estructura HTML donde referenciamos los archivos JavaScript y CSS del proyecto, además de la librería jQuery. También definimos un formulario para ingresar al sistema y otro para el registro de nuevos usuarios. Para ambos formularios solo usamos los campos **usuario** y **clave**.

Para darles funcionalidad crearemos el archivo **script.js** y lo guardaremos en el directorio **js**. Dentro de este archivo escribiremos el siguiente código:

```
$(document).ready(function(){
        // set usuario
        $(document).on('click', '#reg_btn', function(){
                setUsuario();
        });
        // login
        $(document).on('click', '#log_btn', function(){
                login();
        });
});
function setUsuario() {
        if ($('#reg_usuario').val() != '' && $('#reg_clave').val() != '') {
                $.post("registro.php","usuario="+$('#reg_usuario').
val()+"&clave="+$('#reg_clave').val(),
                        function(data){
                                if (data == 1)
                                        window.location = "home.php";
                                else
                                        $('#reg_result').html(msg);
                        }
                );
        }
}
function login() {
        if ($('#log_usuario').val() != '' && $('#log_clave').val() != '') {
                $.post("login.php","usuario="+$('#log_usuario').
val()+"&clave="+$('#log_clave').val(),
                        function(data){
```

```
        if (data == 1)
                                        window.location = "home.php";
                                else
                                        $('#log_result').html('Usuario o
clave incorrectos');
                        }
                );
        }
}
```

Hemos definido los eventos para cuando hacemos clic en los botones **Ingresar** y **Registrarse**, que llaman a las funciones **login()** y **setUsuario()**, respectivamente. Es necesario tener en cuenta que definiremos todas las acciones dentro del evento **ready** del **document** y, las funciones, fuera.

Primero analizaremos la función **setUsuario()**. Mediante una llamada Ajax, le pasamos el usuario y la clave al archivo **registro.php** que creamos con el siguiente código:

```
<?php
        include('conexion.php');
        $result = 0;
        $uid = $redis->incr('global:proximoUid');
        $auth = md5($_POST['usuario'].$_POST['clave']);

        $redis->set('usuario:'.$_POST['usuario'].':uid', $uid);
        $redis->set('uid:'.$uid.':usuario', $_POST['usuario']);
        $redis->set('uid:'.$uid.':clave', md5($_POST['clave']));

        $redis->set('uid:'.$uid.':auth', $auth);
        $redis->sadd('usuarios',$uid);
        if ($redis->set('auth:'.$auth,$uid)) {
                setcookie("auth",$auth,time()+3600*24*365);
                $result = 1;
        }
        echo $result;
?>
```

MEDIANTE UNA COOKIE Y UNA CLAVE AUTH SABREMOS CUÁNDO UN USUARIO LEE O ACTUALIZA

En este código realizamos varias acciones. Primero, incluimos el archivo **conexion.php**; luego, incrementamos la clave **global:proximoUid**, que nos sirve como contador para asignar un identificador de usuario simulando un contador autoincrementar. Posteriormente, mediante el uso de **set** generamos una clave con el usuario y otra con la clave encriptada en **md5**.

Por otra parte, creamos una **cookie** y una clave **auth** en Redis, que nos servirá para identificar cuándo un usuario intenta realizar operaciones de lectura y actualización. Además, creamos una clave **usuarios** donde iremos almacenando los **id's** de cada usuario nuevo, que usaremos luego para obtener a los que vamos a seguir.

Debemos tener en cuenta que, una vez que hacemos el registro, nos encargamos de retornar el valor **1** a la función **login()**, y procedemos a efectuar la redirección a **home.php**. En este punto, nuestro sistema ya cuenta con la funcionalidad de registrar usuarios. Para verificar que todo va bien podemos usar la herramienta conocida como **phpRedisAdmin** y desde ella comprobar que todas las claves han sido creadas correctamente.

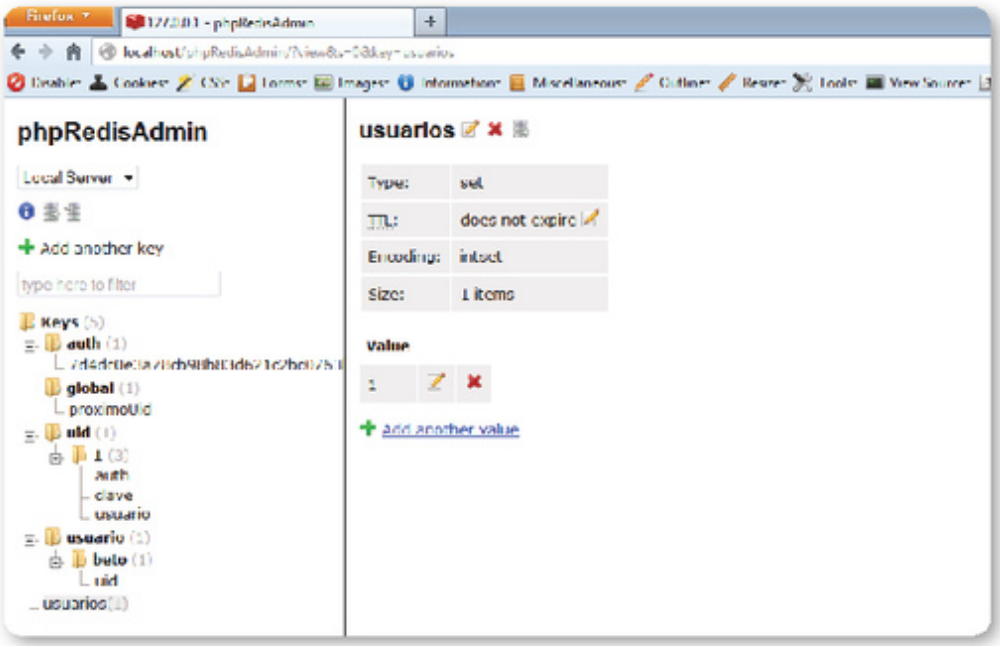


Figura 12. Con **phpRedisAdmin** verificamos las claves que creamos cuando registramos un usuario.

Volviendo a JavaScript, la función **login()**, también mediante una llamada Ajax, enviará las variables **usuario** y **clave** al archivo **login.php**. En este archivo escribimos el siguiente código:

```
<?php
        include('conexion.php');
        $result = 0;
        $uid = $redis->get('usuario:'.$_POST['usuario'].':uid');
        if ($uid) {
                $clave = $redis->get('uid:'.$uid.':clave');
                if ($clave == md5($_POST['clave'])){
                        setcookie("auth",$redis->get('uid:'.$uid.':auth'),ti
me()+3600*24*365);
                        $result = 1;
                }
        }
        echo $result;
?>
```

En el código, primero incluimos el archivo de conexión a la base de datos; luego, obtenemos el **uid** mediante el nombre de usuario y, con él, la clave para comparar con la enviada por **POST**. En caso de que la validación sea correcta, devolveremos el valor **1** a la función **login()**, donde haremos la redirección al archivo **home.php**.

Cuando un usuario se registra o ingresa mediante el login, se crea una cookie para validar el acceso a **home.php**. Esta validación la llevaremos a cabo mediante un archivo llamado **getLogin.php**, que contendrá el código que sigue:

W3C

Cuando necesitamos implementar atributos, definir reglas CSS, usar etiquetas html5 o saber cuáles son los estándares de la web, es recomendable visitar la web oficial de la **World Wide Web Consortium**. Aquí encontraremos todo lo relacionado a diseño, arquitectura, semántica, términos XML, web services, navegadores y más. El sitio se encuentra en la dirección **www.w3.org**.

```
<?php
        include('conexion.php');
        $result = 0;
        if (isset($_COOKIE['auth'])) {
                $uid = $redis->get('auth:'.$_COOKIE['auth']);
                if ($_COOKIE['auth'] == $redis->get('uid:'.$uid.':auth'))
                        $result = 1;
        }
        if ($result == 0)
                header("Location: index.php");
?>
```

En el código incluimos el archivo de conexión a la base de datos; luego, verificamos que la cookie **auth** esté establecida para poder obtener el **uid** del usuario al que pertenece. A continuación, con este **uid** obtenemos el valor de la clave **auth** almacenada en Redis y la comparamos con la cookie.

Ahora crearemos el archivo **home.php**:

```
<?
        include('getLogin.php');
?>
<!DOCTYPE html>
<html>
        <head>
                <title>Mini-twt</title>
                <link type="text/css" href="css/style.css" rel="stylesheet" />
                <script type="text/javascript" src="http://ajax.googleapis.com/
ajax/libs/jquery/1.9.0/jquery.min.js"></script>
                <script type="text/javascript" src="js/script.js"></script>
                <!--[if lt IE 9]><script src="http://html5shiv.googlecode.com/svn/
trunk/html5.js"></script><![endif]-->
        </head>
        <body>
                <header>
                        <h1><a class="titulo" href="home.php">Mini-twt</
```

```
    a></h1>
    <a class="salir" href="salir.php">Salir</a>
                    </header>
                    <div class="wrapper">
                            <section class="info">
                                    <h2 id="usuario"></h2>
                                    <p id="tweets"></p>
                                    <p id="seguidores"></p>
                                    <p id="siguiendo"></p>
                                    <h3>Podes seguir a:</h3>
                                    <div id="usuarios"></div>
                            </section>
                            <section class="home">
                                    <article>
                                            <form id="post_form" action="">
                                                    <textarea id="post"
placeholder="Contale al mundo..."></textarea>
                                                    <input type="button"
id="post_btn" value="twtear" />
                                            </form>
                                            <div id="timeline"></div>
                                    </article>
                            </section>
                    </div>
                    <footer>
                    </footer>
            </body>
    </html>
```

En este código hemos incluido el archivo **getLogin.php** para validar el acceso y, luego, hemos creado los elementos HTML que vamos a utilizar. Las acciones que realizaremos son: obtener el nombre del usuario, la cantidad de posts publicados, la cantidad de seguidores y la cantidad de personas a quienes sigue el usuario. También realizamos la inclusión de una lista de usuarios a quienes seguir y, por otra parte, un formulario donde se escribirá cada post.

En el timeline se pueden ver los posts de todos los usuarios, de uno en particular, y los propios. Para obtener el nombre de usuario, agregamos lo siguiente en el archivo **script.js**, dentro del evento **ready** del **document**:

```
// get info usuario
if ($('#usuario').size()) {
        getUsuario();
};
```

A continuación, agregamos la función **getUsuario()** al archivo **js** para obtener el nombre del usuario actual. Es importante tener en cuenta que todas las funciones deben estar fuera del evento **ready** del **document**.

```
function getUsuario() {
        $.post("getUsuario.php","",
                function(data){
                        var id,usuario;
                        var items = eval('(' + data + ')');
                        for (var i in items) {
                                for (var j in items[i]) {
                                        uid     = items[i][j]['uid'];
                                        usuario = items[i][j]['usuario'];
                                        $('#usuario').append('<a
class="usuarioTl" uid="'+uid+'" href="#">'+usuario+'</a>');
                                }
```

REPOSITORIO DE PROYECTOS

Gracias al desarrollo colaborativo, hoy en día disponemos de sistemas de código abierto, inclusive muy evolucionados como las distribuciones de Linux. Una buena idea es publicar nuestros proyectos en GitHub, con lo cual obtendremos la ventaja de usar un sistema de control de versiones y una comunidad infinita para proponer mejoras. Si todavía no lo hemos probado, podemos empezar accediendo al sitio que encontramos en la siguiente dirección web: **https://github.com**.

```
                        }
                    }
            );
    }
```

En este código, obtenemos mediante el uso de Ajax los datos del usuario y posteriormente los procesamos para mostrarlos en el contenedor. Ahora, creamos el archivo **getUsuario.php**:

```
<?php
        include('conexion.php');
        $uid = $redis->get('auth:'.$_COOKIE['auth']);
        $usuario = json_encode(array('uid' => $uid, 'usuario' => $redis-
>get('uid:'.$uid.':usuario')));
        echo '{"content" : ['.$usuario.']}';
?>
```

Aquí obtenemos la información almacenada en Redis, la codificamos en el formato **JSON** y la devolvemos a la función **getUsuario()**, que será la encargada de mostrarlo en el contenedor indicado.

A continuación veremos la forma en que el usuario puede escribir un post. Para realizar esta tarea debemos agregar lo siguiente en el archivo **js**, dentro del evento **ready** del **document**:

```
// twitear
$(document).on('click', '#post_btn', function(){
        setTweet();
});
```

En el código anterior invocamos a la función **setTweet()** cuando se presiona el botón **twitear**. Por eso necesitamos crear la función con el siguiente código en el mismo archivo **js**:

```
function setTweet(){
        $.post("setTweet.php","post="+$('#post').val(),
                function(data){
                        $('#post').val('');
                        getTweets('');
                        getNumTweetsUsuario();
                });
}
```

Aquí pasamos el valor del elemento **#post** a **setTweet.php** y llamamos a las funciones **getTweets('')** y **getNumTweetsUsuario()**, que nos permiten mostrar el post en el timeline y la cantidad de posts del usuario.

Lo que sigue es crear el archivo **setTweet.php**:

```
<?php
        include('conexion.php');
        $uid = $redis->get('auth:'.$_COOKIE['auth']);
        $pid = $redis->incr("global:proximoPid");

        $post = $uid."|".time()."|".htmlentities($_POST['post']);
        $redis->set('post:'.$pid, $post);
        $redis->incr('uid:'.$uid.':nposts');

        $seguidores = $redis->sMembers("uid:".$uid.":seguidores");
        $seguidores[] = $uid;
        foreach($seguidores as $sid)
                $redis->lpush('uid:'.$sid.':posts',$pid);
        $redis->lpush("global:timeline",$pid);
        $redis->ltrim("global:timeline",0,20);
?>
```

Primero incluimos el archivo de conexión y, luego, obtenemos el **uid** del usuario e incrementamos el contador de posts para saber qué id va a identificar al nuevo post. Después, concatenamos el **uid** con la hora y la variable **post**, que la pasamos por parámetro. Guardamos esto con la instrucción **$redis->set** e incrementamos el contador de la clave **nposts**.

Obtenemos la lista de seguidores y la almacenamos en **$seguidores**. Agregamos el **uid** del usuario actual para, después, con el **foreach**, agregarlo al timeline del usuario.

Finalmente, agregamos el id del post a la clave **global:timeline** y con **ltrim** mantenemos solo los últimos veinte posts. Este proceso no devuelve ninguna salida, por lo que vamos a seguir con la llamada **getTweets('')** del archivo de JavaScript. Para eso, crearemos la función con el siguiente código:

```
function getTweets(lista){
        // en caso de que mostremos los posts de un usuario en particular
        if (lista != '' && lista !='default')
                $('#timeline').html('');
        $.post("getTweets.php",'lista='+lista,
                function(data){
                        var id,usuario;
                        var items = eval('(' + data + ')');
                        for (var i in items) {
                                for (var j in items[i]) {
                                        uid     = items[i][j]['uid'];
                                        usuario = items[i][j]['usuario'];
                                        post    = items[i][j]['post'];
                                        hora    = items[i][j]['hora'];
                                        html    ='<div class="post"><a
class="usuarioTl" uid="'+uid+'" href="#">'+usuario+'</a> - <span>Hace
'+hora+'</span> <p>'+post+'</p></div>';
                                        if (lista != '')
                                                $('#timeline').
append(html);
                                        else
                                                $('#timeline').
prepend(html);
                                }
                        }
                }
        );
}
```

Llamamos, mediante Ajax, al archivo **PHP** que nos devolverá la información de cada post en formato **JSON** y que mostraremos en el timeline. Debemos crear el archivo **getTweets.php**, con el siguiente código:

```
<?php
        include('conexion.php');
        $uid    = $redis->get('auth:'.$_COOKIE['auth']);
        $lista      = $_POST['lista'];
        $cant   = 10;
        if ($_POST['lista'] == '') {
                $clave = 'uid:'.$uid.':posts';
                $cant = 0;
        }elseif($lista == 'default') {
                $clave = 'global:timeline';
        }else{
                $clave = 'uid:'.$lista.':posts';
        }
        $posts      = $redis->lRange($clave,0,$cant);
        foreach($posts as $pid) {
                $aux        = explode("|", $redis->get('post:'.$pid));
                $id         = $aux[0];
                $hora       = horaConFormato($aux[1]);
                $post       = html_entity_decode($aux[2]);
                $usuario = $redis->get('uid:'.$id.':usuario');
                $timeline .= json_encode(array('uid'=>$id, 'usuario'=>$usuario,
'post'=>$post, 'hora'=>$hora)).',';
        }
        echo '{ "content" : [ '.substr ($timeline, 0, -1).']}';
```

JSON

JSON es el acrónimo de JavaScript Object Notation, se trata de un formato ligero para el intercambio de datos. Es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML. Su simplicidad ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. Una de sus ventajas sobre XML es que se presenta como mejor formato de intercambio de datos.

```
        function horaConFormato($hora) {
                $d = time() - $hora;
                if ($d < 60) return "$d segundos";
                if ($d < 3600) {
                        $m = (int)($d/60);
                        return "$m minuto".($m > 1 ? "s" : "");
                }
                if ($d < 3600*24) {
                        $h = (int)($d/3600);
                        return "$h hora".($h > 1 ? "s" : "");
                }
                $d = (int)($d/(3600*24));
                        return "$d dia".($d > 1 ? "s" : "");
        }
?>
```

Primero, incluimos el archivo de conexión; luego, obtenemos el **uid** y la lista pasada por parámetro, y establecemos la cantidad de posts. Si la lista es igual a vacío, devolvemos los posts del usuario actual; si es igual a default, devolvemos la lista global y, en caso contrario, devolvemos la lista del usuario que vino a través del parámetro.

A continuación, obtenemos un rango de la lista y la cantidad especificada, y luego con el **foreach** armamos el **json** de salida. Finalmente, devolvemos el **json**. La función **horaConFormato($hora)** es para formatear hace cuánto tiempo se publicó cada post.

Retornando a la función **setTweet()** de JavaScript, hacemos la llamada a la función **getNumTweetsUsuario()** para mostrar la cantidad de posts del usuario actual. La vamos a crear con el siguiente código:

```
function getNumTweetsUsuario(){
        $.post("getNumTweetsUsuario.php","",
                function(data){
                        $('#tweets').html(data+' TWEETS');
                }
        );
}
```

En el código anterior, hacemos una llamada Ajax al archivo **getNumTweetsUsuario.php**:

```
<?php
        include('conexion.php');
        $uid = $redis->get('auth:'.$_COOKIE['auth']);
        echo $redis->get('uid:'.$uid.':nposts');
?>
```

Primero incluimos el archivo de conexión y luego obtenemos el **uid** del usuario que utilizaremos para obtener la cantidad de posts almacenados en la clave **uid:{uid de usuario}:nposts**.

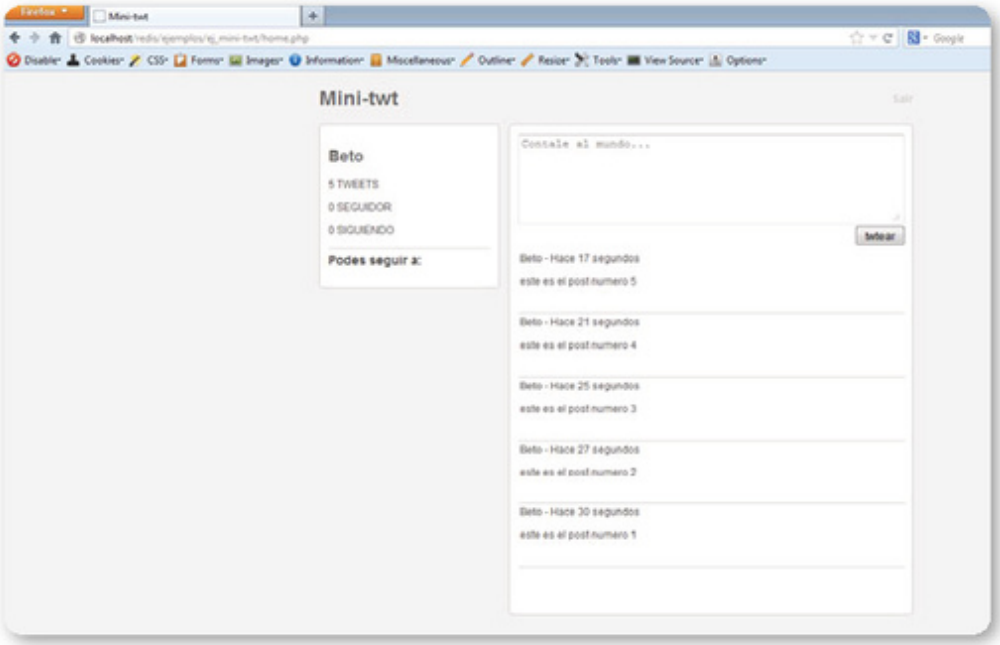


Figura 13. Aquí vemos el timeline del usuario actual con la cantidad de posts.

IONCUBE PHP ACELERATOR

IonCube PHP Acelerator es una extensión para PHP que provee un caché y es capaz de brindar un cambio sustancial en la velocidad de ejecución de los scripts. El uso de esta extensión no requiere ninguna modificación del código fuente. Podemos conocer más en: **www.php-accelerator.co.uk**.

A continuación, creamos el archivo **getUsuarios.php**:

```
<?php
        include('conexion.php');
        $result = 0;
        $uid    = $redis->get('auth:'.$_COOKIE['auth']);
        $uids   = $redis->sDiff('usuarios', 'uid:'.$uid.':siguiendo');
        foreach ($uids as $id){
                if ($uid != $id)
                        $usuario .= json_encode(array('uid' => $id, 'usuario' =>
$redis->get('uid:'.$id.':usuario'))).',';
        }
        echo '{ "content" : [ '.substr ($usuario, 0, -1).']}';
?>
```

Aquí obtenemos el **uid** del usuario actual y hacemos una diferencia entre las listas de usuarios registrados y de los usuarios a los que está siguiendo; luego, con el **foreach**, recorremos la lista resultante y vamos codificando en formato **JSON** los resultados. Finalmente, devolvemos el contenido a la función **getUsuarios()**. En este momento podemos crear usuarios visibles para poder seguirlos.

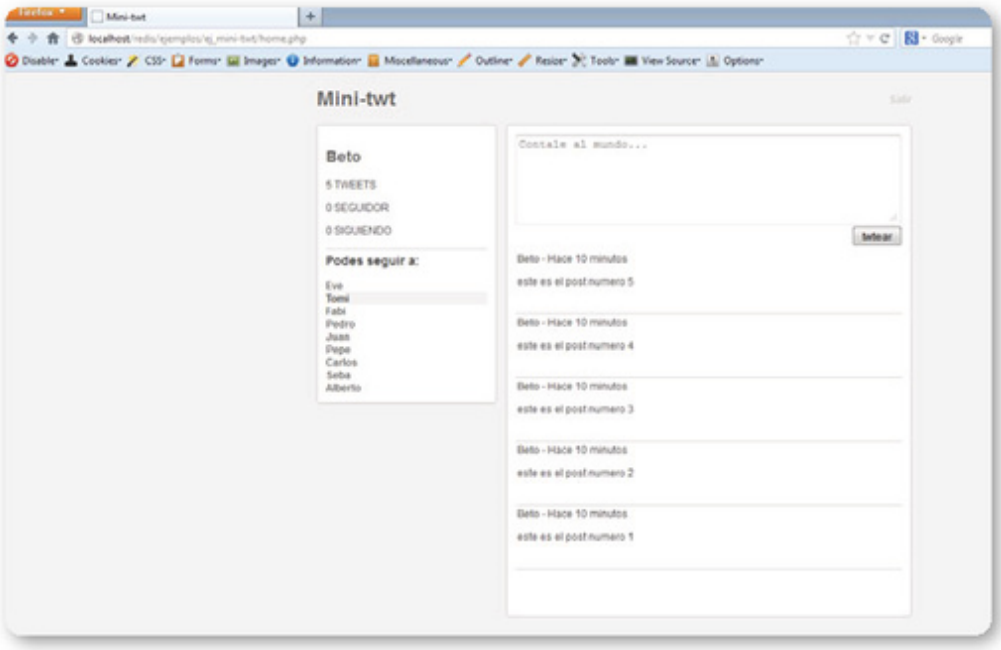


Figura 14. En el menú izquierdo vemos a los usuarios que podemos seguir.

Para realizar el seguimiento de un usuario, debemos agregar el siguiente código dentro del evento **ready** del **document**, donde capturamos el evento **click** de la lista de usuarios:

```
// seguir a
$(document).on('click', '.usuarios', function(e){
        e.preventDefault();
        seguir($(this));
});
```

Hemos hecho una llamada a la función **seguir()**, que tendremos que definir también en el archivo **script.js**:

```
function seguir(element) {
        $.post("seguir.php","usuario="+element.attr('uid'),
                function(){
                        getNumSiguiendo();
                });
        element.remove();
}
```

Dentro de la función anterior hicimos una llamada Ajax al archivo **seguir.php** y le pasamos el **uid** del usuario como parámetro. Creamos este archivo con código que vemos a continuación:

BOILERPLATE

Boilerplate se presenta como una herramienta que nos sirve para simplificar el proceso de construcción de un sitio web en HTML5. Para esto, solo es necesario descargar la plantilla base que contiene la estructura normalizada para todos los navegadores con estilos CSS para IE, y los archivos JavaScript basados en capacidades del navegador, entre otros. Podemos descargarlo si visitamos la página web que se encuentra en la dirección **http://html5boilerplate.com**.

```
<?php
        include('conexion.php');
        if (isset($_POST['usuario'])) {
                $uid = $redis->get('auth:'.$_COOKIE['auth']);
                $redis->sadd('uid:'.$uid.':siguiendo',$_POST['usuario']);
                $redis->sadd('uid:'.$_POST['usuario'].':seguidores',$uid);
                echo 1;
        }
?>
```

En el código, obtenemos el usuario a quien se seguirá y, luego, el **uid** del usuario actual. A continuación, agregamos la clave **siguiendo** al usuario actual con el usuario que pasamos por parámetro, y una clave **seguidores** al usuario que estamos siguiendo con el **uid** actual.

Volviendo a la función **seguir()** de **script.js**, cuando retornamos de la llamada Ajax, ejecutamos la función **getNumSiguiendo()**, que definimos en el mismo archivo de la siguiente manera:

```
function getNumSiguiendo(){
        $.post("getNumSiguiendo.php","",
                function(data){
                        $('#siguiendo').html(data+' SIGUIENDO');
                }
        );
}
```

WEBMASTER DE GOOGLE

Google se encarga de ofrecernos un conjunto de herramientas muy útiles para que podamos enfrentar el desarrollo de sitios web, que brindan información como el estado del servidor, cantidad de clics, cantidad de enlaces, tráfico obtenido, URLs bloqueadas y estadísticas de rastreo, entre otros. Si lo deseamos, podemos utilizar este servicio a través de la página que se encuentra en la siguiente dirección: **https://google.com/webmasters/tools/home?hl=es**.

Mediante el uso de Ajax podemos obtener la cantidad de usuarios a los que estamos siguiendo. Lo hacemos creando el archivo **getNumSiguiendo.php**, con el siguiente código:

```
<?php
        include('conexion.php');
        $uid = $redis->get('auth:'.$_COOKIE['auth']);
        echo $redis->sCard('uid:'.$uid.':siguiendo');
?>
```

Aquí obtenemos el número de elementos de la clave **siguiendo** y lo devolvemos a la función JavaScript. Luego, para mostrar la cantidad de seguidores que tiene el usuario actual, creamos la función **getNumSeguidores()** en el archivo **script.js**:

```
function getNumSeguidores(){
        $.post("getNumSeguidores.php","",
                function(data){
                        $('#seguidores').html(data>1?data+'
SEGUIDORES':data+' SEGUIDOR');
                }
        );
}
```

En el código anterior, hacemos una llamada Ajax al archivo **getNumSeguidores.php**:

DISEÑO ELÁSTICO

Como dijimos, es indispensable desarrollar sitios web con diseños que se puedan adaptar a todos los dispositivos, tanto móviles como de escritorio. La primera medida que debemos tomar es definir un diseño elástico, es decir, establecer medidas relativas tipo **em** para los elementos, de manera que se ajusten automáticamente a cualquier resolución.

```
<?php
        include('conexion.php');
        $uid = $redis->get('auth:'.$_COOKIE['auth']);
        echo $redis->sCard('uid:'.$uid.':seguidores');
?>
```

Aquí obtenemos el número de elementos de la clave **seguidores** y lo devolvemos a la función JavaScript.

Para mostrar la cantidad de usuarios a los que seguimos y la cantidad de los que nos siguen, debemos agregar las llamadas de las funciones **getNumSeguidores()** y **getNumSiguiendo()** al condicional **if ($('#usuario').size())**, de la siguiente manera:

```
if ($('#usuario').size()) {
        getUsuario();
        getTweets('default');
        getNumTweetsUsuario();
        getUsuarios();
        getNumSeguidores();
        getNumSiguiendo();
};
```

Ahora, creamos una funcionalidad para poder hacer clic en el autor de un post y ver su timeline. Solo necesitamos escribir lo siguiente, dentro del evento **ready** del **document**:

```
// timeline de Usuario
$(document).on('click', '.usuarioTl', function(e){
        e.preventDefault();
        getTweets($(this).attr('uid'));
});
```

Hemos capturado el evento **click** del nombre del autor en el timeline y hecho una llamada a la función **getTweets()**, definida anteriormente, con la particularidad de que le pasamos el **uid** del usuario seleccionado.

Por último, nos encargaremos de crear la funcionalidad del enlace **salir**. Para realizar esto es necesario crear un archivo **PHP**, que denominaremos **salir.php**, con este código:

```
<?php
        if(setcookie("auth","")
                header('Location: index.php');
?>
```

En el código anterior simplemente hemos vaciado la cookie y direccionamos al usuario al archivo **index.php**.

RESUMEN

Hemos terminado de conocer a Redis. Vimos la replicación y los tipos de persistencia, que hacen de esta herramienta una de las bases de datos favoritas para el desarrollo de sistemas de alto rendimiento. Aprendimos que mediante la utilidad **redis-benchmark** es posible adecuar la configuración a nuestras necesidades. Y, por último, a través de un ejemplo práctico, nos encargamos de realizar el análisis sobre cómo implementar Redis en sistemas que, por su naturaleza, son escalables y deben estar preparados para un incremento exponencial del volumen de datos.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Es obligatorio definir una clave de acceso a Redis?
2. ¿Qué es la replicación?
3. ¿Un maestro puede requerir autenticación a los esclavos?
4. ¿Qué diferencia existe entre los tipos de persistencia AOF y RDB?
5. ¿Es posible configurar RDB para persistir cada diez segundos?
6. ¿Cuál es la mejor configuración de persistencia para **fsync**?
7. ¿En Redis es posible limitar a los usuarios que se conectan mediante su dirección IP?
8. ¿De qué manera es posible deshabilitar un comando en Redis?

EJERCICIOS PRÁCTICOS

1. Defina una clave de autenticación de acceso a su instancia de Redis en tiempo de ejecución.
2. Si dispone de otra máquina, instale Redis e inicie una instancia como esclavo de la actual e intente replicar la base de datos.
3. Configure Redis con el tipo de persistencia RDB y AOF en paralelo.
4. Agregue la funcionalidad **Retwittear** posts al ejemplo desarrollado en este capítulo.
5. Agregue una función para poder subir una foto de cada usuario al ejemplo desarrollado.
6. Cree una función para dejar de seguir a un usuario en el ejemplo.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*05

Programación orientada a eventos

Aprenderemos qué es un evento y cuáles son los diferentes paradigmas en el desarrollo de sistemas; luego, nos enfocaremos en las características de la programación orientada a eventos. Esto nos servirá de introducción para comprender el funcionamiento de la tecnología Node.js.

- **Paradigmas en el desarrollo de sistemas 172**
  - Programación orientada a objetos 172
  - Programación imperativa 173
  - Programación declarativa 174
  - Programación estructurada 175
  - Programación orientada a aspectos 176
  - Programación orientada a eventos 178
- **Resumen 179**
- **Actividades 180**

Paradigmas en el desarrollo de sistemas

Para comenzar, vamos a definir **paradigma de programación** como la técnica de desarrollo de software utilizada por una comunidad de programadores, que intentan resolver los problemas de una manera óptima. Un paradigma debe ser capaz de ofrecer una solución significativa, sin afectar la calidad del producto.

A continuación, conoceremos algunos de los paradigmas más utilizados en la actualidad para el desarrollo de sistemas.

Programación orientada a objetos

La **programación orientada a objetos**, comúnmente llamada POO, ofrece una solución al problema de la división de datos y procesos. Es decir, mediante la definición de un objeto podemos contener datos y métodos que pueden operar con ellos, así como también interactuar con otros objetos, encapsular y heredar métodos de otros. Este paradigma se popularizó en los años noventa y hoy es uno de los más utilizados para el desarrollo de software.

Los objetos son entidades que contienen datos definidos mediante atributos, a los cuales se les asignan valores concretos. Además, tienen métodos que sirven para manejar los datos propios o de otros objetos. Por último, cada objeto tiene una identidad que permite diferenciarlo del resto, implementada también por un atributo.

Este paradigma está representado por el lenguaje de programación **Smalltalk**, orientado a objetos, pero existen también innumerables

CLOUD 9

Cloud 9 es un entorno de desarrollo en línea con características muy interesantes: integración con GitHub, un entorno para JavaScript y más de 25 lenguajes. Además, brinda un sistema de desarrollo compartido con chat interno, un entorno para instalar herramientas y librerías y una consola para ejecutar comandos de Linux. Podemos conocerlo en el enlace **https://c9.io**.

lenguajes que lo implementan de manera parcial o total. Un ejemplo de implementación se halla en PHP, cuyo modelo de objetos se ha reescrito a partir de la versión 5. De esta manera, se encarga de brindar un mejor rendimiento, con más características.

Figura 1. En el sitio oficial de PHP, **`http://php.net/manual/es/language.oop5.php`**, podemos ver cómo implementar el paradigma POO.

Programación imperativa

Debemos considerar que este paradigma define estados de un programa y sentencias encapsuladas en procedimientos que efectúan cambios de dichos estados. Se trata del más utilizado para realizar desarrollos convencionales, porque los programas imperativos le indican a la computadora el modo en que puede realizar cada tarea mediante un conjunto de instrucciones que se ejecutan de manera secuencial.

Un ejemplo de implementación de este paradigma a bajo nivel es el funcionamiento de las computadoras, cuyas sentencias son instrucciones en lenguaje máquina y los datos son los contenidos que están en la memoria.

> LA PROGRAMACIÓN IMPERATIVA DEFINE ESTADOS Y SENTENCIAS ENCAPSULADAS

Debemos considerar que existen lenguajes de programación imperativos de alto nivel que usan variables y conjuntos de sentencias más complejas, pero que siguen el mismo paradigma; entre los más conocidos se encuentran C, Perl, PHP, Java y Python.

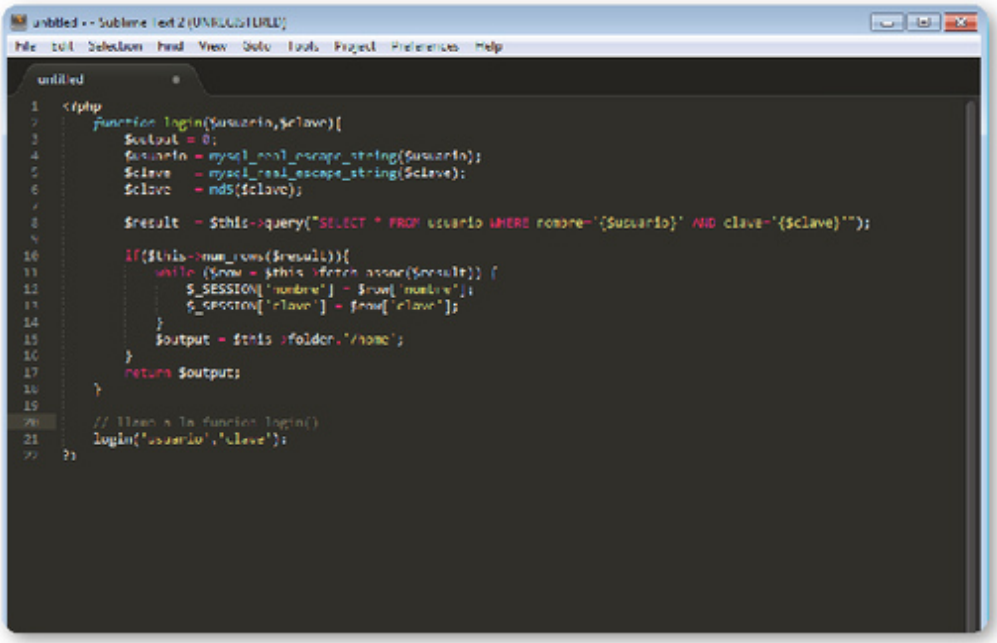


Figura 2. Una simple función, que ejemplifica el paradigma de programación imperativa mediante PHP.

Programación declarativa

Se trata de un paradigma que está basado en la declaración de un conjunto de condiciones, proposiciones, afirmaciones y también restricciones que definen un problema y describen su solución mediante el uso de mecanismos de control, que indican qué es lo que se quiere obtener y no cómo hacerlo.

GOOGLE WEB FONTS

Muchas veces nos hemos preguntado cómo implementar una determinada fuente tipográfica que no está instalada en todos los sistemas operativos. Hoy en día existe una posibilidad muy recomendable, llamada **Google Web Fonts**, mediante la cual podemos utilizar alrededor de 620 fuentes. Para saber cómo implementarlas, podemos acceder a: **http://google.com/webfonts**.

La ventaja que tienen los lenguajes declarativos es que pueden ser manejados matemáticamente, dando la posibilidad de optimizar el rendimiento de los programas.

Consideremos que en el mundo del desarrollo web el lenguaje declarativo más conocido es SQL, implementado para interactuar con bases de datos relacionales como MySQL.

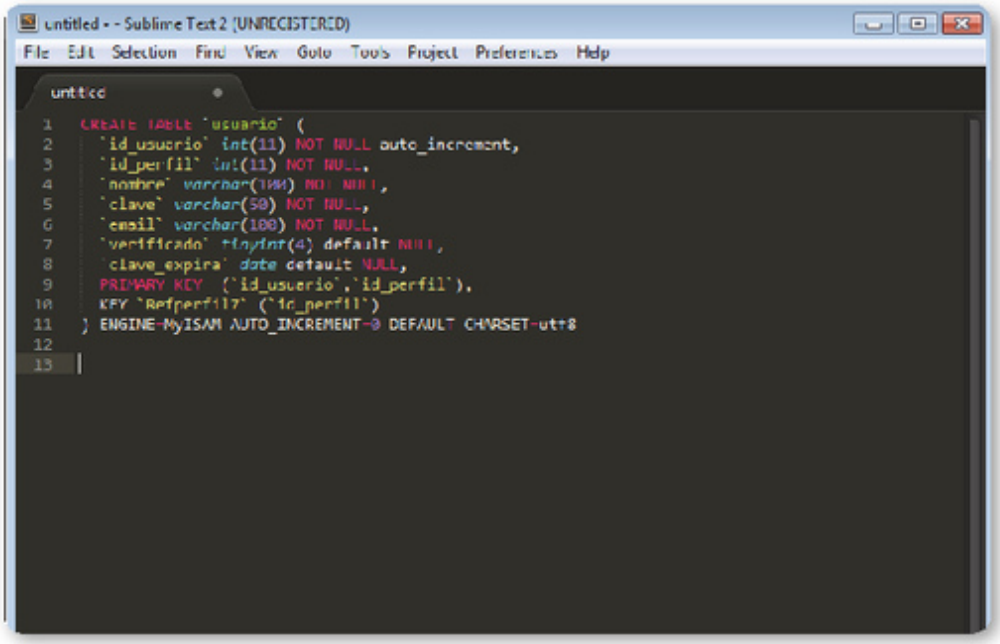


Figura 3. Una implementación del paradigma declarativo SQL para crear la tabla **`Usuario`**.

Programación estructurada

Paradigma creado para dar más claridad en el proceso de desarrollo, con mayor calidad y en menor tiempo, mediante el uso, únicamente, de subrutinas de manera secuencial y estructuras de control de selección e iteración. Tengamos en cuenta que puede ser implementado en cualquier lenguaje de desarrollo de sistemas.

La programación estructurada considera innecesario y negativo el uso de saltos directos a otras rutinas (por ejemplo, el uso de **`GOTO`**) porque esto genera un código confuso y difícil de depurar.

Las ventajas que ofrece este paradigma son: la creación de sistemas más fáciles de entender (porque el seguimiento es secuencial y no hay necesidad de hacer saltos de líneas), la minimización del esfuerzo en la fase de prueba y depuración del código (ya que la estructura es más

sencilla y comprensible) y la reducción de los costos de mantenimiento en dinero y tiempo. Como consecuencia, debido a que los programas son más sencillos, ofrece un mejor rendimiento.

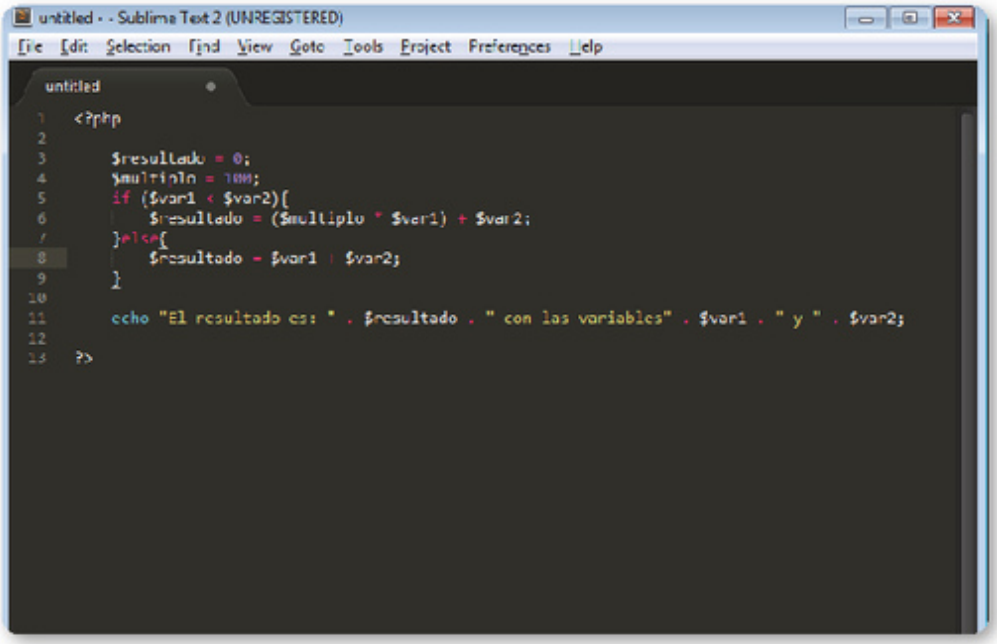


Figura 4. Implementación del paradigma estructurado con el lenguaje de programación PHP.

Programación orientada a aspectos

Este paradigma es relativamente nuevo y consiste en modularizar las aplicaciones, posibilitando separar funciones comunes para varios objetos. Es decir, para la mayoría de los sistemas es necesario tener un controlador de errores, pero como los errores están diseminados por todas las clases definidas en el sistema, esto provoca que cada clase

CAPTCHA

Probablemente, la mayoría de los programadores sabe para qué se utiliza el sistema **CAPTCHA**, pero pocos saben por qué existen dos palabras, una de las cuales se encuentra escrita de manera casi inentendible para los usuarios. Esto se debe a que a nivel mundial se están digitalizando libros antiguos y hay términos que no están claros, por lo que han decidido incluir estas palabras y, sobre la base de las entradas de usuarios, se selecciona la que más veces se ingresó.

tenga que ocuparse de tareas que no son propias de su definición. Cuando se implementa este paradigma, se separan las funcionalidades propias de cada módulo y las funcionalidades comunes utilizadas a lo largo del sistema. Cada una se encapsula en una sola entidad.

> FLOW3 ES UN ENTORNO DE DESARROLLO MVC QUE PODEMOS USAR PARA PHP

Existen muchos módulos, extensiones y entornos de desarrollo que permiten utilizar este paradigma en diferentes lenguajes. Para PHP podemos usar **FLOW3**, que es un entorno de desarrollo **MVC** (modelo-vista-controlador) que incluye un módulo para realizar programación orientada a aspectos en desarrollos nuevos.

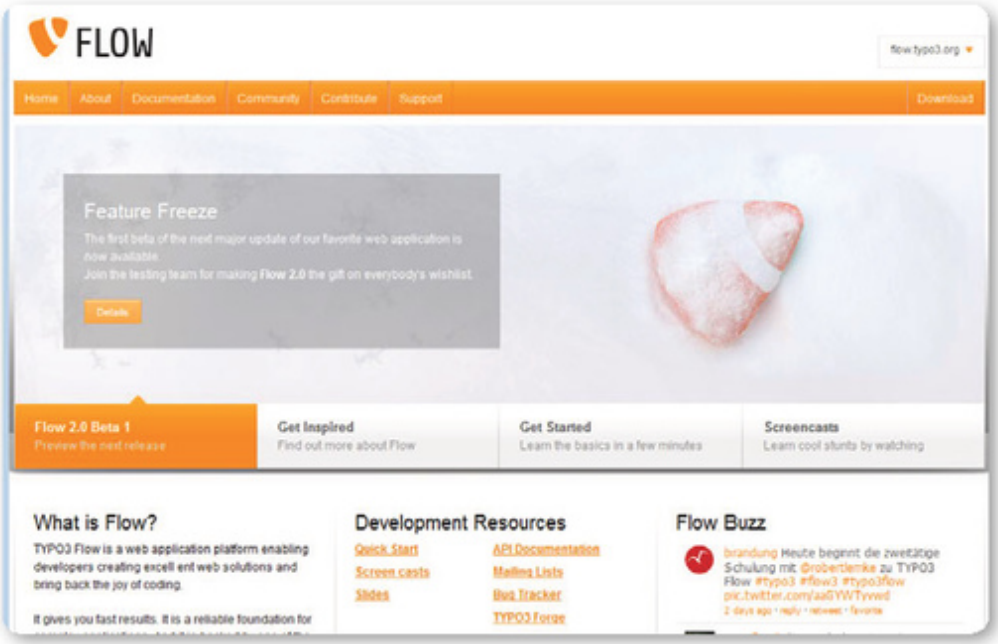


Figura 5. Desde **`http://flow.typo3.org`** podemos descargar **FLOW3** para implementar el paradigma orientado a aspectos en PHP.

DUMMY DATA

Cuando vamos a mostrar una funcionalidad o una iteración del proyecto, es necesario utilizar datos de prueba para que el cliente o quienes deben hacer el proceso de testing puedan visualizar los cambios. Para ello se recurre a la técnica **Dummy Data**, más conocida como **Lorem Ipsum**. Para utilizarla accedemos a: **http://chuckipsum.com**.

Programación orientada a eventos

Hasta aquí, hemos visto diferentes paradigmas de desarrollo donde cada uno define la manera de estructurar un sistema. Ahora vamos a desarrollar el **paradigma orientado a eventos**, en el que tanto la estructura como la ejecución están determinadas por los sucesos provocados por los usuarios o por comportamientos desencadenados que ocurren en el sistema.

> EN LA POE EL COMPORTAMIENTO DEL USUARIO DETERMINA EL FLUJO DEL SISTEMA

En la programación orientada a eventos el comportamiento del usuario determina el flujo del sistema, por lo que los programadores debemos definir los eventos que manejará el desarrollo y las acciones que se llevarán a cabo cuando esto suceda. Esto debe ser implementado mediante un manejador de eventos.

Cuando se inicia el sistema, el manejador de eventos es inicializado y queda a la espera de que ocurra algún evento para poder atenderlo y realizar las acciones correspondientes.

Comúnmente, esta programación está basada en la interacción del usuario con la interfaz del sistema, aunque también pueden emplearse componentes que se comuniquen entre sí, lo que genera hilos de ejecución concurrentes que ejecutan rutinas independientes. A los eventos producidos por los usuarios como, por ejemplo, la introducción de texto, se los denomina **externos**. A los eventos producidos por el sistema se los denomina **internos**, como por ejemplo el vencimiento de un temporizador.

Entre los lenguajes que manejan eventos se encuentra JavaScript, que será el objeto de nuestro estudio. En los siguientes capítulos de esta obra nos encargaremos de mencionar el uso de la tecnología

INGLÉS, EL IDIOMA PARA TODO PROGRAMADOR

En la actualidad es casi imposible no tener que recurrir a documentación de tecnología en inglés y más aún para lenguajes de programación, ya que están escritos en este idioma. Una buena idea es recurrir a **Duolingo**, una plataforma de aprendizaje interactiva totalmente gratuita, donde se combinan texto, imágenes, audio y voz. Para conocer más y empezar a practicar podemos acceder a: **http://duolingo.com/es**.

Node.js, que nos permitirá no solo proceder a realizar el manejo de eventos sino también interactuar con los usuarios y componentes del sistema en tiempo real.

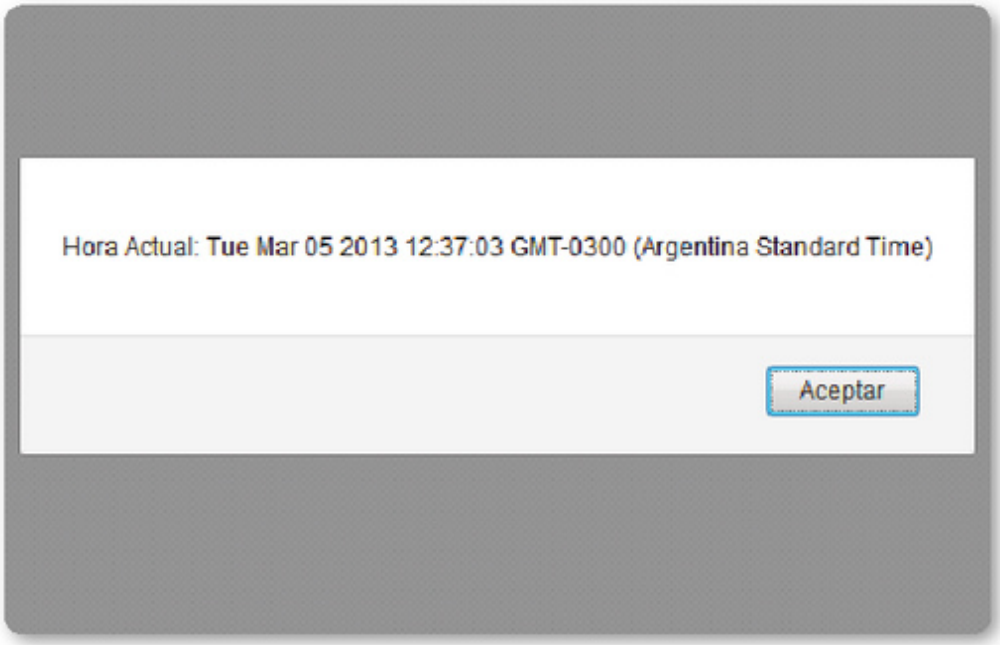


Figura 6. Gracias a JavaScript es posible manejar eventos que ocurren en el navegador respondiendo a acciones de los usuarios.

RESUMEN

Hemos aprendido algunas de los paradigmas de desarrollo de sistemas más utilizados, los cuales tienen ventajas y desventajas frente a otros, si bien cada uno debe emplearse para una situación en particular. El paradigma por excelencia es el de **programación orientada a objetos**, que brinda un marco de desarrollo ordenado y eficiente para desarrollos grandes. Para terminar, vimos una pequeña introducción al **paradigma orientado a eventos**, que nos servirá para entender conceptos que veremos luego.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Un paradigma de desarrollo es un lenguaje de programación?
2. ¿Qué significa el paradigma POO?
3. ¿Podemos utilizar PHP con el paradigma de programación imperativa?
4. ¿Cómo utilizan las bases de datos relacionales el paradigma declarativo?
5. ¿Es posible implementar la programación estructurada en PHP?
6. ¿Qué es FLOW3? ¿Está relacionado a PHP y al paradigma orientado a aspectos?
7. ¿JavaScript es un lenguaje orientado a eventos?
8. ¿Qué determina el flujo del sistema?
9. ¿Qué es el manejador de eventos de un sistema?
10. ¿Con el paradigma orientado a eventos es posible implementar un sistema de tiempo real?

EJERCICIOS PRÁCTICOS

1. Cree una clase llamada "persona" en PHP.
2. Investigue el significado de MVC y cómo lo implementa FLOW3.
3. Implemente un manejador de eventos que capture la entrada de texto.
4. En el manejador implemente una llamada Ajax para mostrar en un contenedor.
5. Investigue de qué manera jQuery implementa el manejador de eventos.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*06

Node.js

Conoceremos Node.js, las soluciones que ofrece y cuándo implementarlo. Analizaremos sus características principales para entender las diferencias que existen con servidores web como Apache. También veremos las ventajas de usarlo en lugar de Ajax. Por último, aprenderemos a instalarlo y a crear nuestro primer sistema en Node.js.

- Node.js 182
- Diferencias entre Node y Apache 185
- Ajax versus Comet 186
- Cuándo usar Node 188
- Cuándo no usar Node 189
- Empresas que utilizan Node .. 190
- Instalación de Node en Linux, Mac y Windows 195
- Primeros pasos con Node 202
- Resumen 207
- Actividades 208

Node.js

La mayoría de los desarrolladores web conocemos JavaScript y las ventajas que ofrece para el desarrollo de sistemas. Con implementaciones como Ajax y librerías como jQuery, JavaScript se transformó en el lenguaje por excelencia para interactuar del lado del cliente con los elementos HTML. Si, hasta ahora, lo hemos visto siempre funcionar dentro del contexto de un navegador, de aquí en más vamos a introducir un nuevo concepto que lleva su potencial a otro nivel.

Node.js es una plataforma de programación orientada a eventos, que se ejecuta del lado del servidor y está implementada en **V8** (el motor de JavaScript creado por Google e implementado en su navegador Chrome). Fue creada por **Ryan Dahl** en 2009, mientras buscaba la manera de implementar un lenguaje orientado a eventos en la Web.

Node.js permite crear aplicaciones distribuidas altamente escalables, como servidores web que requieren un muy buen rendimiento o sistemas de tiempo real que precisan manejar miles de solicitudes simultáneas en un solo servidor de manera asincrónica.

En adelante, por una cuestión de comodidad, nos referiremos a Node.js solo con el término Node.

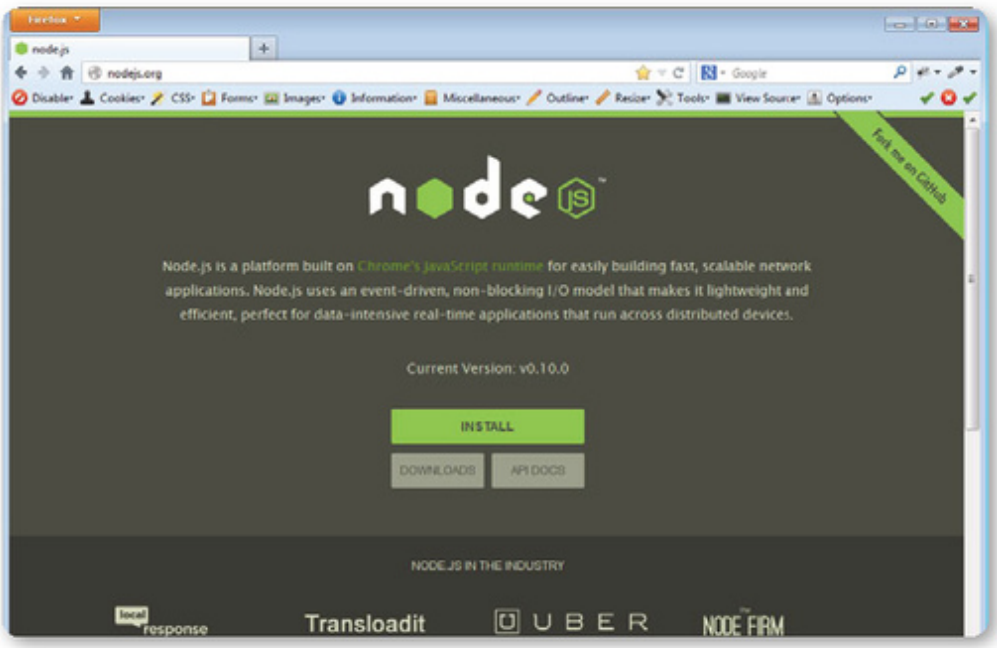


Figura 1. Podemos acceder al sitio oficial de Node.js mediante el siguiente enlace: **`http://nodejs.org`**.

Soluciones

Como sabemos, uno de los principales factores que influyen en el rendimiento de los sistemas web es la disponibilidad del hardware.

El gran problema de los sistemas desarrollados con PHP o Java es que por cada conexión con el servidor se genera un nuevo hilo que tiene asignado un espacio de memoria, lo que presenta un límite máximo de conexiones que depende en forma directa de la cantidad de memoria instalada.

En estos sistemas, para responder a todas las solicitudes de manera óptima, es necesario agregar más memoria al servidor o replicarlo horizontalmente a medida que aumenta el número de usuarios. Esto genera costos a nivel operativo y de disponibilidad, ya que los recursos compartidos deben estar replicados en todos los servidores.

> LA DISPONIBILIDAD DE HARDWARE INFLUYE EN EL RENDIMIENTO DE LOS SISTEMAS WEB

Node resuelve este problema cambiando la manera en que se realizan las conexiones con el servidor, porque cada conexión dispara un evento dentro de su proceso central. Y, por definición, no permite bloqueos de entrada/salida al tiempo que afirma que en un servidor se pueden ejecutar millones de conexiones de manera simultánea.

Características

- Utiliza un ciclo de eventos en lugar de hilos.
- Un servidor Node puede escalar a millones de conexiones simultáneas.
- Las operaciones de entrada/salida son asincrónicas, permitiendo que el servidor pueda seguir atendiendo solicitudes de entrada mientras otras operaciones de entrada/salida se están llevando a cabo.
- Es multiplataforma: actualmente se puede instalar en **Mac OS**, **Linux**, **Windows** y **SunOS**.
- Soporta los protocolos **TCP**, **DNS**, **HTTP**, **TLS** y **SSL**. Además, soporta **SPDY**, que ha sido desarrollado principalmente por Google e intenta modernizar el protocolo HTTP aportando un rendimiento superior de aproximadamente 60%.
- Puede mantener tantas conexiones como el número máximo de sockets soportados por el sistema. En Unix, este límite ronda las 65.000.
- Posee un rendimiento excelente.

Funcionamiento

V8 es el motor JavaScript creado por Google para su navegador Chrome, el cual interpreta código JavaScript y lo ejecuta. Este motor está escrito en C++ y se caracteriza por su velocidad extrema.

Cuando descargamos y posteriormente utilizamos Node, estamos ejecutando el motor V8 en un contexto diferente al navegador, que nos permite incorporarlo a cualquier sistema que desarrollemos y manejarlo a través de eventos.

A continuación, vemos un ejemplo de cómo manejar eventos del lado del cliente mediante el uso de la librería jQuery:

```
$("#btnEnviar").click(function(){
    if($("#clave").val() != $("#claveConfirm").val())
        alert("Atención: La clave y su confirmación deben ser iguales");
});
```

Pensar en eventos del lado del servidor puede resultar un poco extraño, ya que en este contexto no ejecutamos eventos **onClick** en un botón o **MouseOver** en un enlace, sino que somos los programadores quienes debemos definir cuáles son los eventos y las acciones que se ejecutarán cuando este ocurra.

En un servidor Node se ve cada proceso como un único hilo que responde a todas las peticiones que recibe. En este contexto, todo está contenido en un solo ciclo de eventos y no es necesario que nos preocupemos por la cantidad de memoria disponible. Debemos tener en cuenta que esto disminuye considerablemente el tiempo de desarrollo y de resolución de bugs.

HERRAMIENTA JAVASCRIPT EN LÍNEA

JS Bin (**http://jsbin.com**) es una herramienta para crear código HTML, CSS y JS y probarlo en la misma interfaz. Es posible visualizar una consola para depurar código de manera fácil y dispone de una sección para ver el resultado. Brinda la posibilidad de agregar librerías como **jQuery** y pre procesadores CSS, como **Stylus**, entre otros. Permite compartir el código con otros desarrolladores.

Diferencias entre Node y Apache

En servidores web como Apache, por cada conexión se genera un nuevo hilo. Esto es útil si la cantidad de conexiones no crece demasiado, pero con la creación de nuevos hilos y el cambio de contexto se genera un costo de rendimiento asociado. En la **Figura 2** vemos una comparación de rendimiento entre Apache y Node con conexiones simultáneas. Podemos observar la diferencia a partir de cuatrocientas conexiones concurrentes: por un lado, Apache tarda considerablemente en atender las solicitudes, mientras que Node se mantiene casi sin cambios.

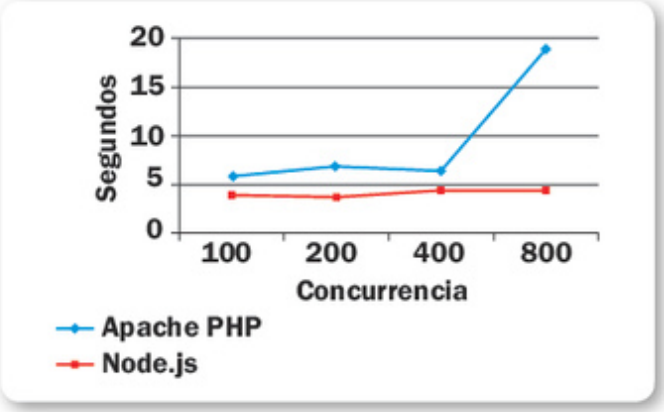


Figura 2. El gráfico muestra la diferencia que existe entre Node y Apache para el manejo de conexiones múltiples.

Un aspecto importante de Node es que posee la capacidad de mantener múltiples conexiones abiertas y en espera, a diferencia de Apache que establece un máximo por defecto de 256 conexiones mediante el parámetro **MaxClients**. Este parámetro puede ser modificado para incrementar su número, pero para sistemas web dinámicos (por ejemplo con PHP) es probable que al establecer un valor alto el servidor no pueda atender a todas las solicitudes y se bloquee.

> NODE ES CAPAZ DE MANTENER VARIAS CONEXIONES ABIERTAS Y EN ESPERA

Debido a que un sistema Node utiliza un solo hilo, en caso de que exista alguna operación bloqueante el sistema creará de manera

automática otro hilo en segundo plano, de manera tal que ninguna operación detenga el flujo normal de ejecución.

En la actualidad, conocemos lenguajes como PHP que, para funcionar, dependen de un servidor web como Apache. Es aquí donde Node ofrece una gran diferencia en comparación con otras alternativas, ya que permite crear un servidor web casi para cualquier puerto que le indiquemos y sin depender de ningún otro servicio.

Ajax versus Comet

Hace unos años el funcionamiento de las páginas web era básico: cuando el navegador realizaba una petición, se conectaba por HTTP al servidor, que realizaba alguna operación, devolvía un resultado y finalizaba la conexión. La desventaja de este procedimiento estaba en que, para actualizar los datos del cliente, era necesario recargar la página generando una nueva solicitud al servidor (que generaba consumo de ancho de banda y desperdicio de tiempo).

AJAX PERMITE AL NAVEGADOR SOLICITAR UN FRAGMENTO DE INFORMACIÓN

Recientemente se introdujo la técnica Ajax (que significa **JavaScript Asincrónico y XML**), que permite al navegador solicitar solo un fragmento de información al servidor, reduciendo de manera significativa tanto el tiempo de respuesta como el ancho de banda empleado.

Si bien Ajax introdujo un gran avance en el desarrollo de sistemas web, existen algunos casos en los cuales su uso no es una buena práctica, como por ejemplo, en sistemas de mensajes

35 TIPS PARA EL DISEÑO DE LOGOS

Si hemos tenido la idea de fundar una empresa, crear algún producto o diseñar una imagen corporativa para nuestros clientes, puede que nos hayamos encontrado con la difícil tarea de diseñar el logo. El sitio **Creative Bloq** ha publicado 35 tips para tener en cuenta al trabajar sobre la imagen que necesitamos crear. Podemos verlos en: **http://creativebloq.com/graphic-design/pro-guide-logo-design-21221**.

instantáneos o chat. En estos, para actualizar los mensajes o el estado de un usuario, es necesario que el navegador envíe al servidor una solicitud nueva cada cierto tiempo, para pedir los cambios que hayan ocurrido, generando un consumo elevado de ancho de banda y tiempos de ocio entre evento y solicitud.

Comparación de rendimiento

En Ajax el cliente realiza una petición cada cierto tiempo y el servidor responde. Puede ocurrir que justo después de que el servidor haya enviado una respuesta vacía (porque no hubo cambios) se produzca un evento y el servidor esté a la espera de la próxima petición del cliente para devolver este cambio. Este sistema es muy deficiente para modelos de tiempo real, ya que existen períodos ociosos y elevado ancho de banda con posibilidad de respuestas vacías.

En el modelo Comet con **long-polling**, el cliente realiza una petición y deja abierta la conexión hasta que se produzca un evento en el servidor. Cuando esto ocurre el servidor envía de forma inmediata la respuesta y cierra la conexión. Luego, cuando transcurre el tiempo que ha sido establecido para que el cliente realice la siguiente petición, se vuelve a repetir el proceso correspondiente.

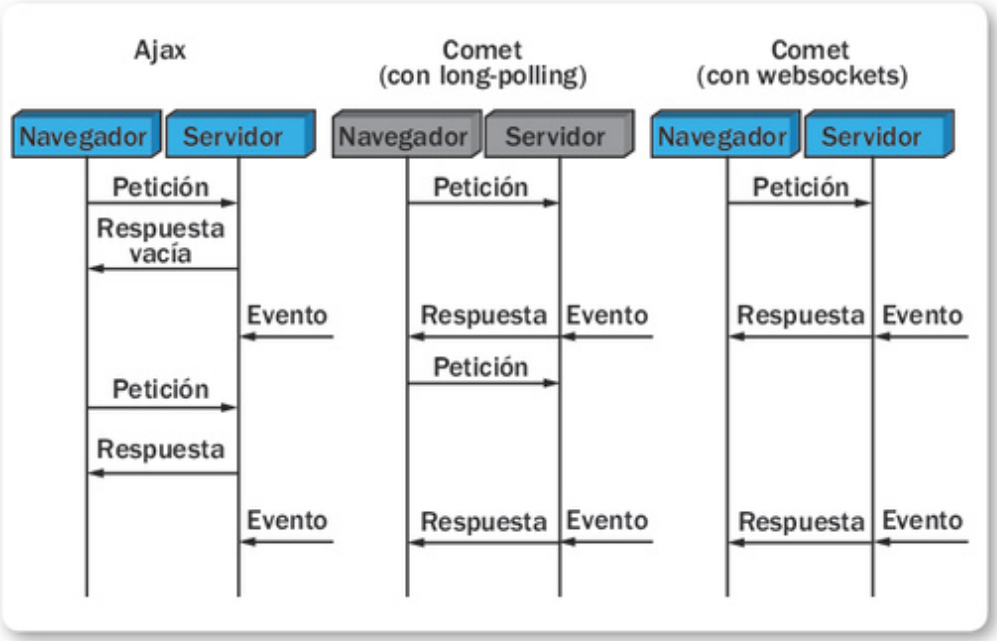


Figura 3. `Ajax` resuelve la mayoría de los problemas de interacción cliente/servidor pero, para ciertos casos, `Comet` es más apropiado.

Por último, en el modelo Comet con **Websockets**, el cliente abre una conexión y realiza una petición al servidor. Del lado del servidor, a medida que ocurren los eventos, los envía al cliente, ya que la conexión permanece abierta. Es el modelo óptimo para sistemas de tiempo real.

Es importante aclarar que Node usa conexiones de tipo Comet, o sea persistentes, que permiten la comunicación bidireccional entre el cliente y el servidor. Así, el servidor puede comunicarse asincrónicamente con el cliente sin crear nuevas conexiones.

Cuándo usar Node

Node es excelente para el desarrollo de sistemas en tiempo real y de juegos en línea. Cuando necesitamos crear un cliente de correo que nos mantenga constantemente actualizados (por ejemplo, con mensajes entrantes en la bandeja de entrada), es la mejor alternativa. Es muy útil para el desarrollo de herramientas de colaboración.

Otra aplicación muy usada es la construcción de redes sociales, debido a que en estos sistemas existe una gran demanda de interacción cliente/servidor. Es ideal para sistemas de traducción en tiempo real.

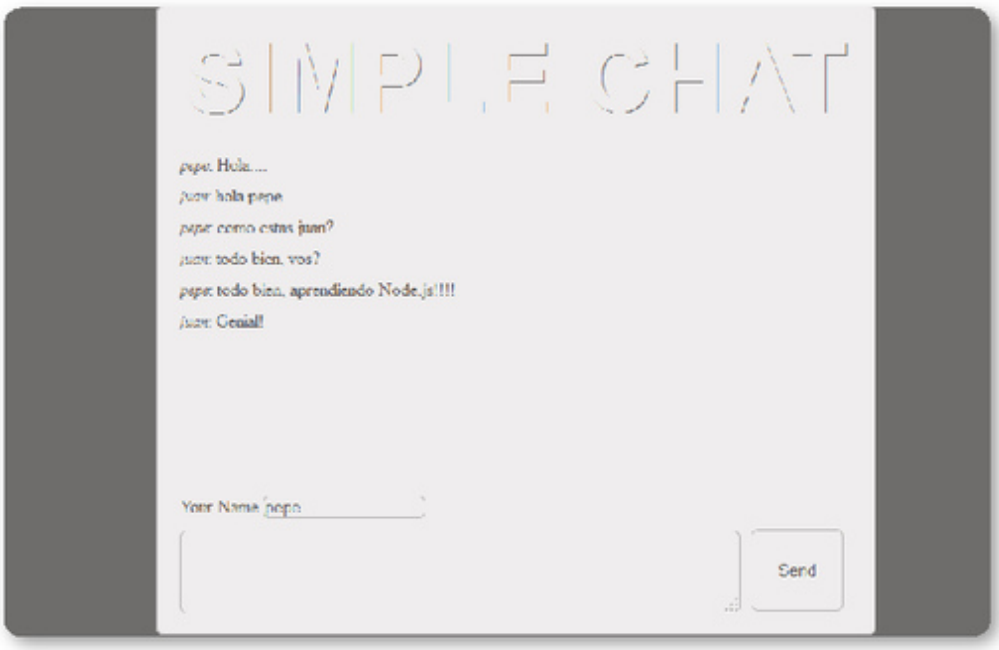


Figura 4. Un uso muy común de Node se encuentra en los sistemas de chat, ya que debe manejar los mensajes en tiempo real.

Cuándo no usar Node

Node posee un potencial extremo para desarrollos de sistemas que requieren un gran rendimiento y podemos decir que casi no tiene puntos en contra, ya que no existen condiciones reales que lo hagan poco recomendable para algún tipo de desarrollo. Sin embargo, siempre debemos usar la herramienta que mejor se adapta a nuestras necesidades. Por ejemplo, es una buena práctica, antes de implementar Node, evaluar las siguientes condiciones:

- Tiempos de respuesta
- Cantidad de usuarios concurrentes que habrá
- Revisar el nivel de interacción de tiempo real que se va a necesitar en el proyecto
- Magnitud del sistema
- Relación costo-beneficio de otras tecnologías similares, si existen

> DEBEMOS EVALUAR ALGUNAS CONDICIONES ANTES DE IMPLEMENTAR NODE

Una vez que hayamos evaluado las cuestiones anteriores estaremos en condiciones de definir si Node es la mejor solución a implementar.

Ventajas del uso de Node

Las ventajas de Node que veremos a continuación se encuentran a nivel desarrollador ya que, por lo analizado hasta el momento, son claros las beneficios para los usuarios finales en cuanto al rendimiento.

- La utilización de Node es comparable a trabajar con Apache sin tener módulos extra; para cada requerimiento podemos instalar solo los módulos necesarios y sin agregados que nunca usaremos.
- Un aspecto que muchos consideran ventaja es que Node nos permite entender la forma de manejar las solicitudes HTTP y de diferentes protocolos, a diferencia de Apache, que realiza esta acción de manera invisible para el programador.

- Debido a que existen cientos de módulos ya desarrollados, estamos en contacto de manera continua con código de otros programadores, de los cuales aprendemos a escribir código ordenado y estandarizado.
- Node abre el espectro acerca del manejo de eventos del lado del servidor, de la misma manera en que estamos acostumbrados al manejo del lado del cliente.
- Node es JavaScript. Esto es sumamente importante, ya que la mayoría de los programadores web conocen este lenguaje y la curva de aprendizaje es mucho menor que al tener que aprender un lenguaje nuevo.
- Como desarrolladores tenemos que usar un solo lenguaje, tanto para el **front-end** como para el **back-end**. Es decir, un sistema podría estar desarrollado íntegramente con HTML5, jQuery (JavaScript), CSS3 o Node (JavaScript), y utilizar a Node para la comunicación con la base de datos, la gestión de archivos y la administración de plantillas.

Empresas que utilizan Node

Tengamos en cuenta que cada vez son más las empresas que utilizan Node para diferentes implementaciones. Muchas han reemplazado funcionalidades en sus sitios de producción y otras todavía están en el proceso de cambio. Una de las que primero han optado por Node es **LinkedIn**, la plataforma de contacto profesional que más ha evolucionado en estos últimos años.

CHROME DEVELOPER TOOL

La mejor herramienta para depuración de código HTML, CSS y JS es **Chrome Developer Tool**, que ofrece toda la información necesaria para depurar y observar el comportamiento de un sitio: frames, datos almacenados en local store, variables de sesión, cookies y caché. Para acceder a ella, presionamos **F12** en el navegador **Chrome**. Para más información podemos ingresar al sitio que se encuentra en la dirección web **https://developers.google.com/chrome-developer-tools**.

Figura 5. En **`http://venturebeat.com/2011/08/16/linkedin-node`** encontramos una nota sobre cómo LinkedIn implementó Node para ofrecer un mejor servicio.

Microsoft es otra de las empresas que ofrecen una implementación de desarrollo utilizando Node, mediante su plataforma de servicios en la nube llamada **Windows Azure**.

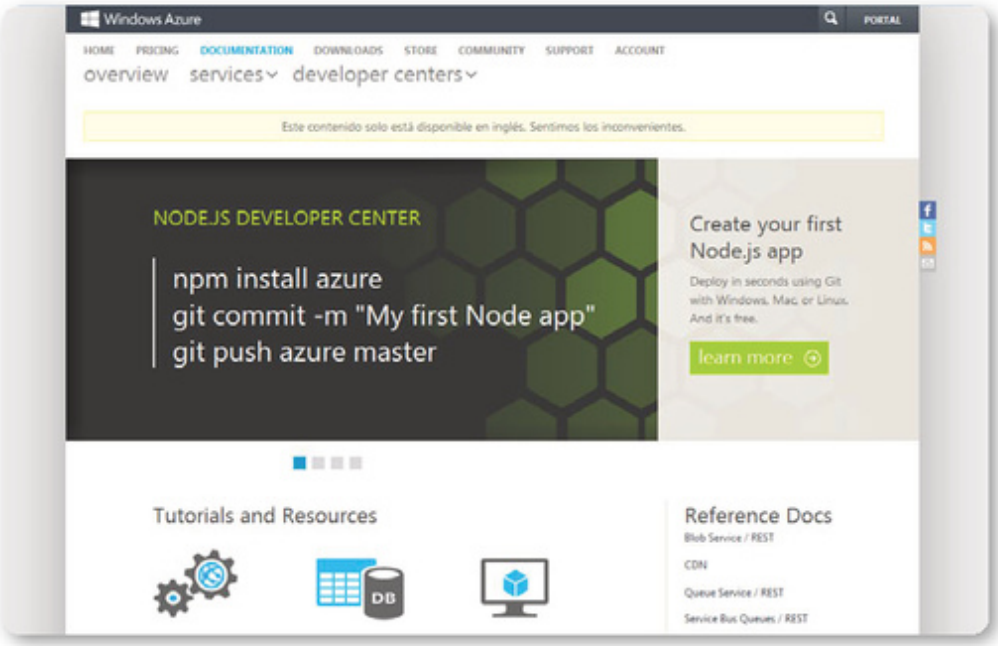


Figura 6. Podemos probar Windows Azure en **`https://windowsazure.com/en-us/develop/nodejs`**.

Yahoo ha estado trabajando intensamente en la implementación de tecnologías tales como HTML5, Node, CSS3 y también JavaScript, entre otras, para crear diferentes productos. Entre estos se encuentran **Mojito**, un framework para aplicaciones web en JavaScript, y **Manhattan**, un entorno de alojamiento para Mojito desarrollado con JavaScript del lado del servidor.

Figura 7. Podemos conocer más de los proyectos de Yahoo en: **http://developer.yahoo.com/blogs/ydn/posts/2011/11/yahoo-announces-cocktails-%E2%80%93-shaken-not-stirred**.

A fines del año 2011, en el blog de la empresa **eBay** anunciaron los cambios que habían realizado para mejorar su disponibilidad. Entre ellos, informaron que habían elegido Node después de haber considerado los factores que mencionamos a continuación:

MD5 EN JAVASCRIPT

Los sistemas web utilizan algoritmos de encriptación como **SHA1** o **MD5** para mantener las contraseñas en la base de datos, y este proceso se realiza del lado del servidor. Una buena idea es encriptar la clave del lado del cliente antes de enviarla, para agregar un nivel de seguridad a los sistemas. Para implementar **MD5** podemos usar la librería creada por **Paul Johnston** (**http://pajhome.org.uk/crypt/md5**).

- Es una alternativa escalable y ofrece un increíble rendimiento para hacer frente a grandes cargas de entrada/salida.
- Su operatividad permite realizar la detección de fallas y posteriormente ofrecer soluciones rápidas cuando se producen errores.
- La demanda de memoria es muy baja, teniendo en cuenta la gran cantidad de conexiones simultáneas que se manejan.
- Su ecosistema, que se encuentra basado en el lenguaje JavaScript, provee las herramientas que necesitamos para construir sistemas que presentan una gran complejidad, como por ejemplo un sitio de ventas en línea.

> NODE ES ESCALABLE Y POSEE UN EFICIENTE SISTEMA DE DETECCIÓN DE FALLAS

Luego de realizar algunos análisis de rendimiento sobre JavaScript y Node, teniendo en cuenta las características mencionadas, la empresa eBay se encargó de destacar la siguiente información: "Se ha configurado un servidor con Ubuntu, el cual maneja más de 120.000 conexiones activas por proceso, donde cada uno de estos procesos solo consume alrededor de 2Kb de memoria".

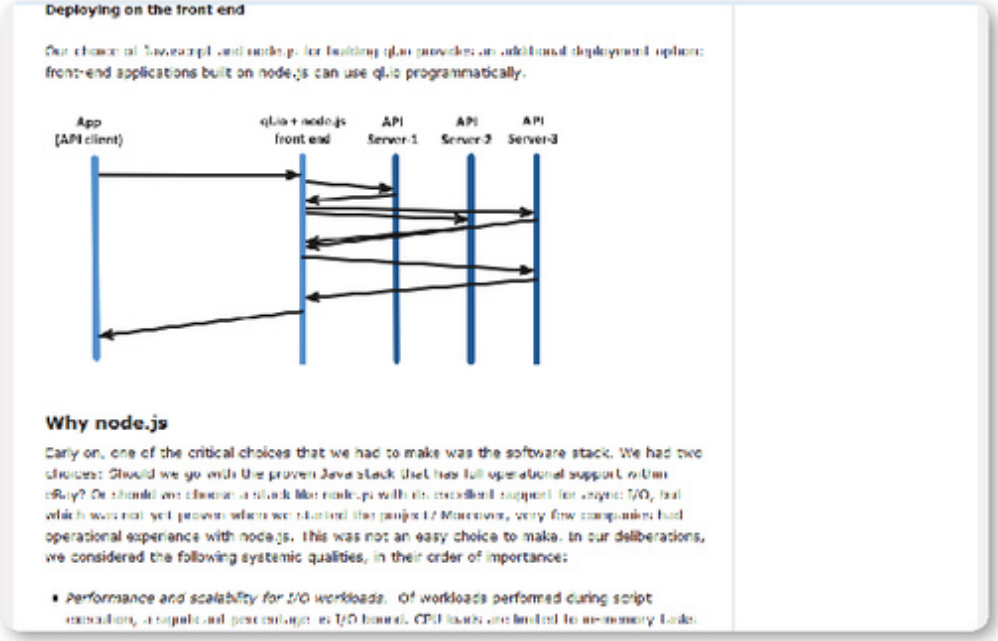


Figura 8. En el blog de eBay podemos conocer más sobre la implementación de Node en su plataforma: **http://ebaytechblog.com/2011/11/30/announcing-ql-io**.

Otra empresa que utiliza Node, pero de una manera fuera de lo común, es **Nodejitsu**, una plataforma de alojamiento de proyectos del tipo **PaaS** (*Platform as a Service* o **plataforma como servicio**) que alberga sistemas desarrollados con Node y permite probarlos en línea. Entre sus características más importantes se encuentran:

- Es 100% de código abierto.
- Se integra con **Github**, soporta **websockets** y módulos extras.
- Implementa Node en todas sus operaciones.
- Es posible realizar la instalación de las dependencias utilizando el **gestor de paquetes** de Node.
- Provee acceso al repositorio de módulos propios en GitHub.
- No es necesario hacer ningún cambio en los desarrollos para subir los archivos al servidor.
- Provee soporte y ofrece MongoDB, Redis y CouchDB, para que podamos utilizar en nuestros proyectos.

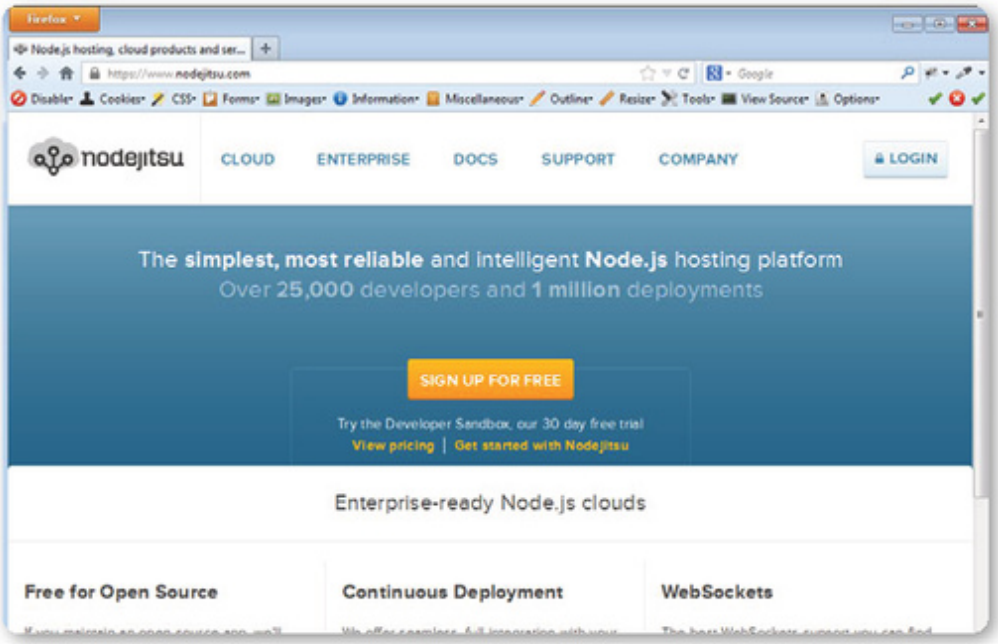


Figura 9. **Nodejitsu** provee una versión gratuita de la plataforma, por 30 días. Podemos acceder a ella desde **www.nodejitsu.com**.

Por último, en el sitio web de Node, podemos observar una sección que detalla algunas de las empresas que utilizan esta herramienta, como **Local response**, **UBER**, **The Node Frim**, **Iris Couch**, **Storify** y **Transloadit**, entre las más importantes.

Instalación de Node en Linux, Mac y Windows

La instalación de Node es una tarea muy sencilla. Esta aplicación se encuentra disponible para sistemas Windows, Linux y Mac y, además, es posible instalarla en sistemas operativos **POSIX**, como **Solaris** y **BSD**.

Podemos obtenerla desde dos fuentes principales: el sitio oficial, **http://nodejs.org**, o su repositorio en GitHub, **http://github.com/joyent/node**. A continuación, vamos a describir cómo instalarla en los tres sistemas operativos más comunes.

Instalación en Linux (distribución Ubuntu)

La manera más sencilla, pero no recomendada, de instalar Node en Ubuntu es ejecutar el siguiente comando en una terminal:

```
sudo apt-get install nodejs
```

Decimos que este es un procedimiento no recomendado porque el centro de software no siempre está actualizado con la última versión.

A continuación, instalaremos en Ubuntu la versión **0.10.0** de Node, de la manera recomendada. Para comenzar, ejecutamos el siguiente comando para instalar los prerrequisitos:

```
sudo apt-get install python-software-properties python g++ make
```

PARSER JSON

El sitio **Json Parser Online** ofrece una herramienta muy útil para generar estructuras JSON.

Para utilizarlo, solo debemos acceder al enlace **http://json.parser.online.fr** y, en el cuadro de la izquierda, escribir o pegar nuestra estructura.

Automáticamente mostrará la estructura de una manera más amigable.

Luego, agregamos el repositorio que contiene la última versión mediante el siguiente comando:

```
sudo add-apt-repository ppa:chris-lea/node.js
```

Seguidamente, actualizamos el sistema:

```
sudo apt-get update
```

Por último, instalamos Node:

```
sudo apt-get install nodejs
```

Una vez que tenemos instalado Node, podemos verificar que todo haya salido bien ejecutando el siguiente comando, que nos devolverá la versión instalada:

```
node -v
```

El comando anterior debería devolvernos **v0.10.0**.

Instalación en Mac

En Mac el proceso de instalación también es muy sencillo. Antes de comenzar necesitamos tener instalado **Xcode** y **Git**. Luego, en una terminal, ejecutamos los siguientes comandos:

```
git clone git://github.com/ry/node.git
cd node
./configure
make
sudo make install
```

Con este procedimiento hemos clonado el repositorio Node de GitHub para después instalarlo. Una alternativa es instalarlo a través del paquete de instalación propio de Mac, que puede descargarse desde el enlace **http://nodejs.org/download**.

Una vez que tenemos instalado Node, podemos verificar la instalación ejecutando el siguiente comando:

```
node -v
```

Con el comando anterior veremos la versión que hemos instalado.

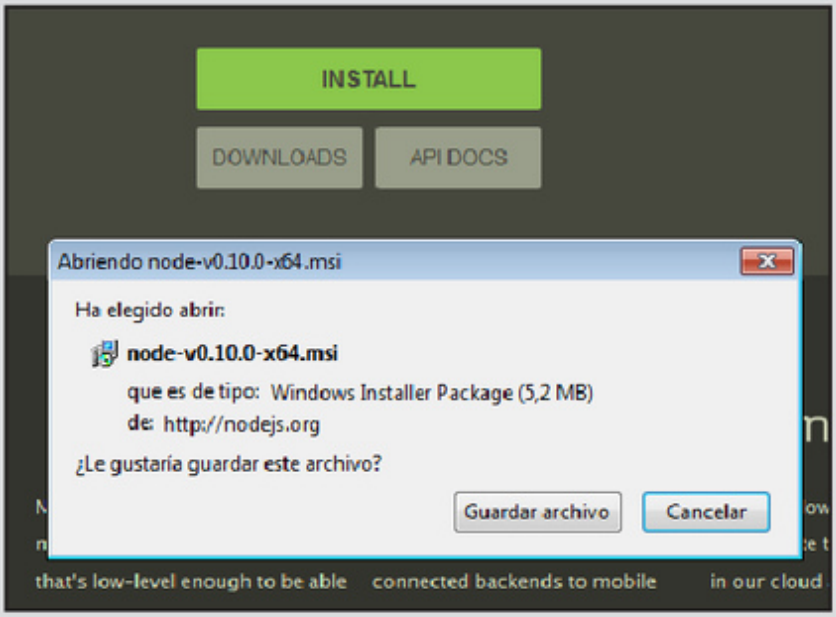
> NODE -V NOS PERMITE REALIZAR LA VERIFICACIÓN DE LA INSTALACIÓN DE NODE

Instalación en Windows

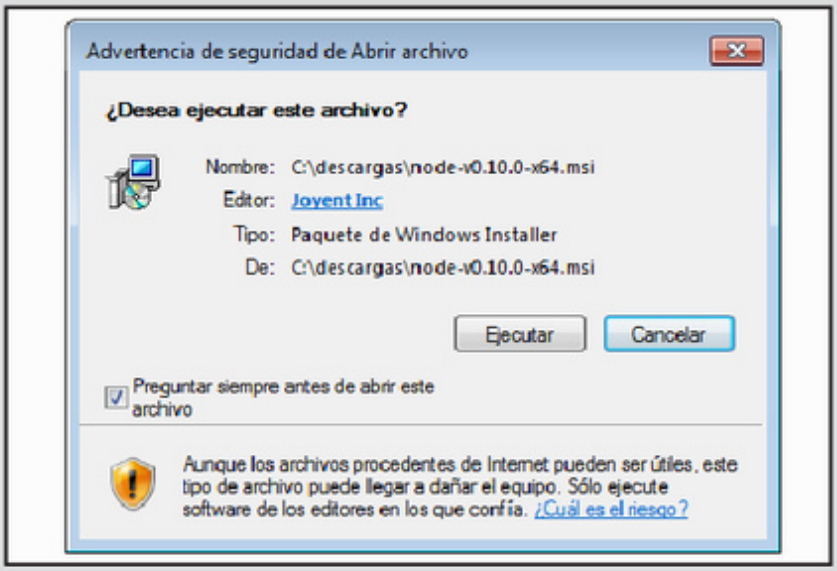
Vamos a describir cómo instalar Node en Windows de la manera tradicional, es decir, mediante el paquete de instalación específico para este sistema operativo.

PAP: INSTALACIÓN DE NODE EN WINDOWS

01 Diríjase a **http://nodejs.org** y haga clic en el enlace INSTALL. Se abrirá una ventana para descargar el instalador correspondiente a su arquitectura, de 32 o 64 bits. Luego, pulse el botón Guardar archivo.

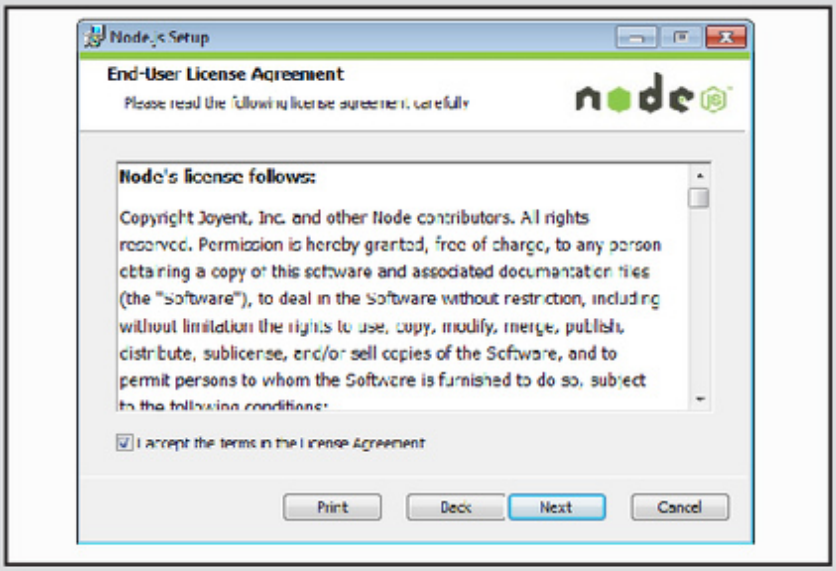


02 Una vez que haya descargado el instalador, diríjase al directorio y ejecute el archivo. Si aparece una ventana de seguridad, haga clic en `Ejecutar`.

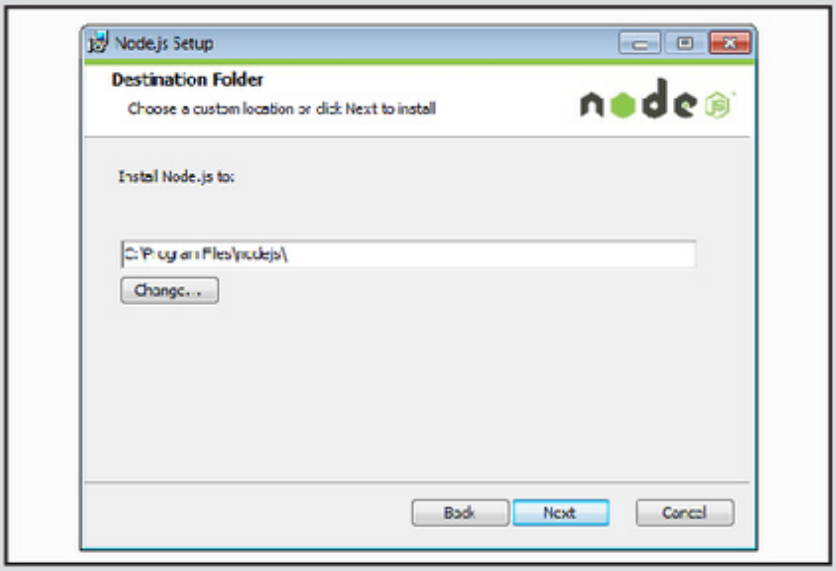


03 Al ejecutar el instalador, la primera ventana que verá es la de bienvenida. Continúe con el próximo paso de la instalación haciendo clic en `Next`.

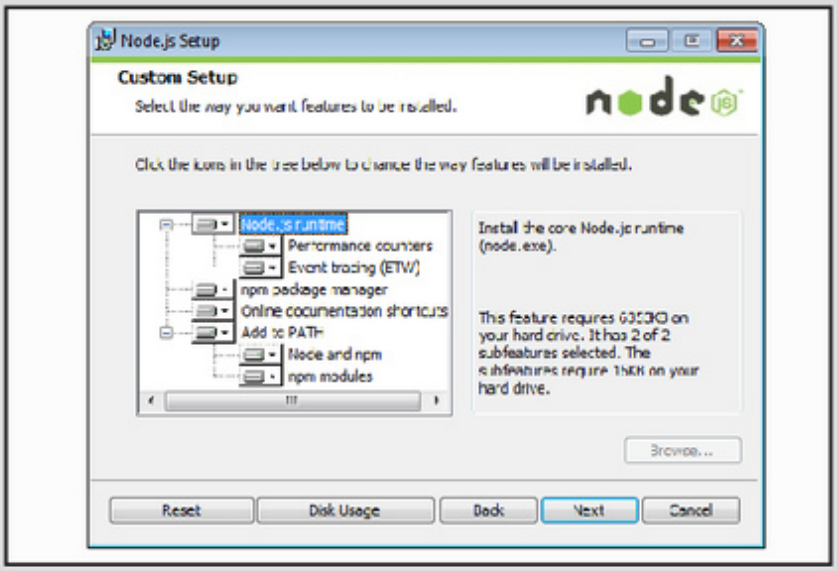
04 A continuación aparecerá la ventana de licencia. Haga clic en la casilla de verificación para aceptar las condiciones y presione el botón Next.



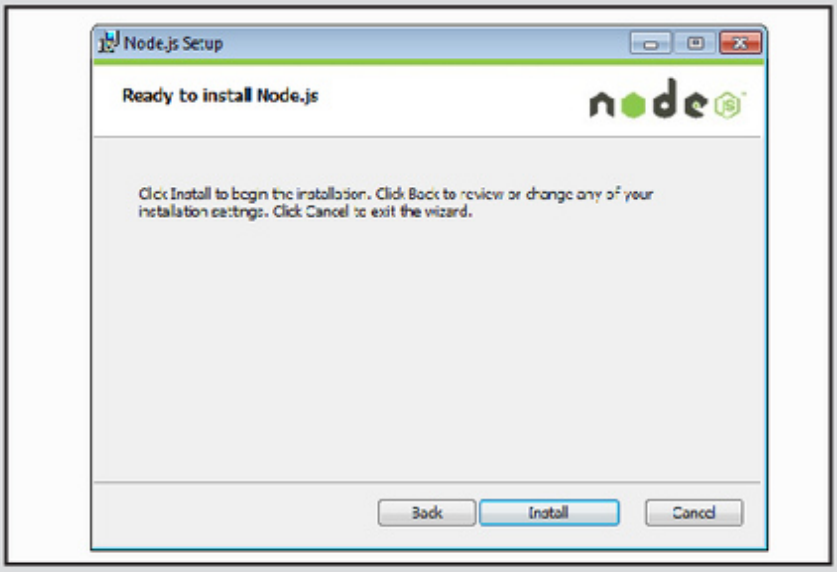
05 Luego verá la dirección por defecto donde se instalará Node. Puede cambiar de directorio haciendo clic en el botón Change. Luego presione el botón Next.



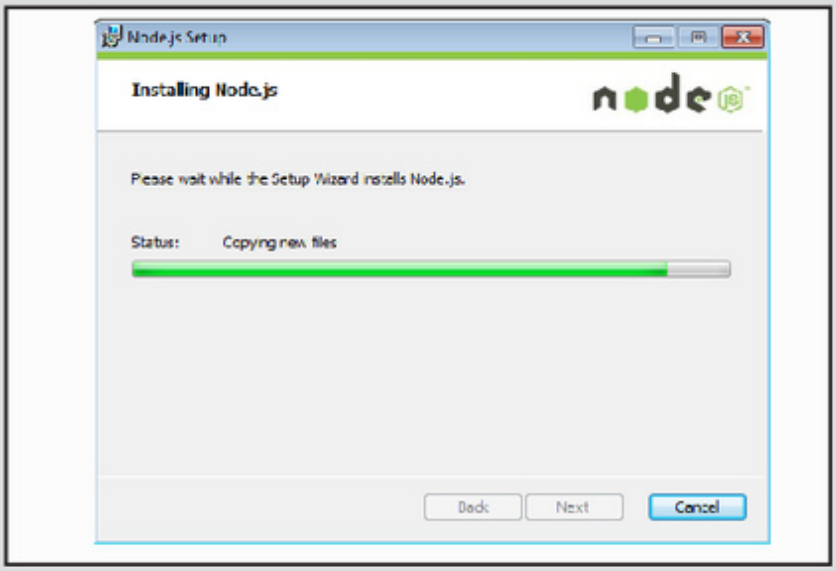
06 Seleccione las opciones de instalación. Es recomendable que active todas las que aparecen disponibles. Haga clic en el botón `Next`.



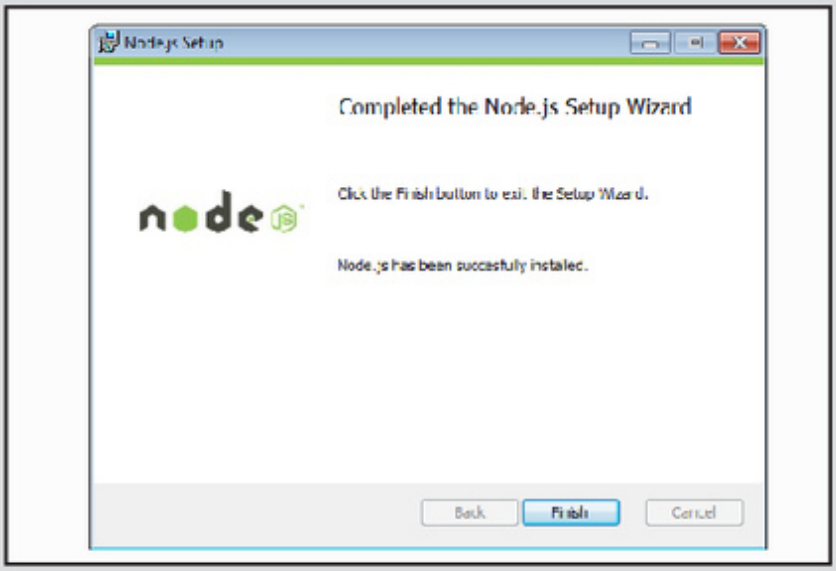
07 En la siguiente pantalla iniciará el proceso de instalación. Para continuar con el procedimiento haga clic en el botón `Install`.



08 Verá una pantalla que indicará el estado de la instalación mediante una barra de progreso. Espere a que finalice el proceso.



09 Por último, verá la pantalla que indica que Node se ha instalado de manera satisfactoria. Haga clic en el botón `Finish` para cerrar la pantalla.



Ahora que tenemos instalado Node, podemos verificar su versión abriendo una ventana de comandos y escribiendo **node -v** o el comando **node -h** para ver la lista de opciones disponibles.

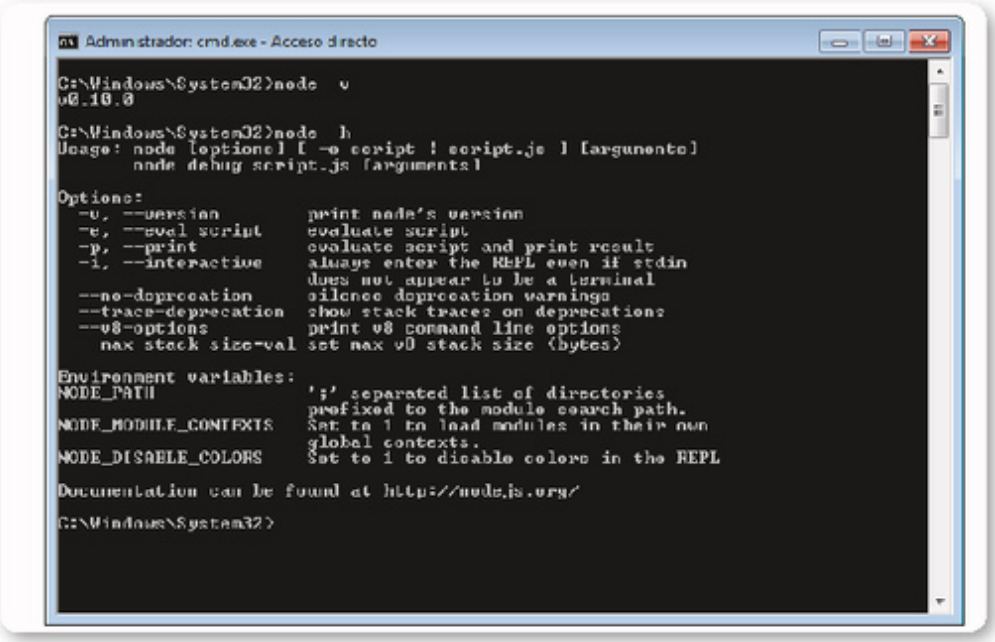


Figura 10. Mediante la herramienta **cmd.exe** de Windows podemos ejecutar cualquier comando de Node.

Primeros pasos con Node

Como hemos visto, Node nos permite ejecutar instrucciones como cualquier otro entorno de desarrollo. Directamente desde la consola, podemos ejecutar código utilizando Node **REPL (Read-Evaluate-Print-Loop)**, que lee, evalúa e imprime un resultado.

```
> .help
.break  Sometimes you get stuck, this gets you out
.clear  Alias for .break
.exit   Exit the repl
.help   Show repl options
.load   Load JS from a file into the REPL session
.save   Save all evaluated commands in this REPL session to a file
```

En la **Tabla 1** vemos las opciones que dispone REPL:

OPCIONES DE REPL

COMANDO	DESCRIPCIÓN
.break	Sirve para salir del bloque actual
.clear	Es un alias para **.break**
.exit	Sale de REPL
.help	Muestra las opciones que ofrece REPL
.load	Carga un archivo JavaScript a la sesión REPL
.save	Guarda en un archivo todos los comandos evaluados en la sesión actual de REPL

Tabla 1. Comandos disponibles en **REPL**.

A continuación vamos a realizar algunos ejemplos para ir familiarizándonos con Node y su entorno. Es necesario tener en cuenta que, para ejecutar estos ejemplos, trabajaremos directamente en una consola. En el caso de Linux o Mac tenemos que abrir una terminal y, en el caso de Windows, debemos abrir una ventana de comandos.

Ahora vamos a ejecutar todos nuestros ejemplos en el sistema operativo Windows. El primero consiste simplemente en crear varias condiciones que Node evaluará, a partir de lo cual generará un resultado. Para esto escribimos lo siguiente:

COMANDO CONSOLE CON ESTILOS CSS

Muchas veces es necesario imprimir valores para depurar el código JavaScript mediante el comando **console.log()**. Una curiosidad interesante es la posibilidad de poder formatear el mensaje impreso: por ejemplo, podemos ejecutar **console.log('%mensaje','color:red')**, que va a imprimir el texto en color rojo. También es posible asignarle un color de fondo y cambiar el tamaño de la fuente.

```
> 100 > 99
> true
> true == 1
> true
> true === 1
> false
```

PODEMOS CREAR VARIABLES PARA LUEGO UTILIZARLAS EN EL CÓDIGO

En el ejemplo que vimos anteriormente, primero nos encargamos de evaluar si **100** es mayor que **99**; luego, evaluamos si **true** es igual a **1** y, por último, comparamos si **true** es igual y del mismo tipo que **1**.

En el próximo ejemplo realizaremos la creación de la variable denominada **usuario**, luego le asignaremos un valor determinado y, para terminar, lo mostraremos en la consola de comandos junto a un mensaje de bienvenida:

```
> usuario = 'Carlos Alberto'
'Carlos Alberto'
> console.log('Bienvenido ' + usuario)
Bienvenido Carlos Alberto
```

Como vemos, es posible realizar la creación de variables y posteriormente utilizarlas en el desarrollo del código.

Ahora veremos un caso un poco más complejo que los anteriores, pero que aun así no deja de ser muy simple. Primero, definimos

LA NUBE TRANSFORMARÁ EL MERCADO

Cada vez más las compañías están apostando a la suscripción de software y al almacenamiento en la nube. **IDC** (International Data Corporation) estima que, para el año 2016, el 60% del mercado de trabajo se albergará en la nube mediante el software como servicio **SaaS**. Esto implicará un rediseño de los tipos de productos y de la manera en que se ofrecen.

una variable llamada **usuario**, que es un objeto que contiene los atributos **nombre** y **apellido**. Luego, establecemos una función llamada **saludo**, que va a tener como parámetro **usr**, que contendrá el objeto creado anteriormente. En el cuerpo de la función nos encargamos de concatenar una cadena con los atributos del usuario. Por último, ejecutamos **saludo(usuario)** para ver el resultado.

```
> usuario = {nombre:'Alberto', apellido:'Benitez'}
{ nombre: 'Alberto', apellido: 'Benitez' }
> saludo = function(usr){
..... return 'Bienvenido ' + usr.nombre + ',' + usr.apellido;
..... }
[Function]
> saludo(usuario)
'Bienvenido Alberto, Benitez'
```

En el próximo ejemplo crearemos un servidor web simple, que estará escrito en Node y responderá "Hola mundo!, Node es genial!" a todas las solicitudes. Para ejecutar el servidor web, primero generamos una carpeta llamada **Node** en la unidad **C**. Luego, nos encargamos de crear un archivo JavaScript con el siguiente código y lo guardamos con el nombre **servidor.js** dentro de la carpeta que habíamos creado:

PARA GENERAR UN SERVIDOR WEB SIMPLE CREAMOS EL ARCHIVO SERVIDOR.JS

```
var http = require('http');

http.createServer(function (request, response) {
        response.writeHead(200, {'Content-Type': 'text/plain'});
        response.end('Hola mundo!, Node es genial!');
}).listen(3000,'127.0.0.1');

console.log('Servidor web iniciado en http://127.0.0.1:3000/');
```

Por ahora no vamos a explicar qué hace cada línea de este código, ya que en el **Capítulo 7** lo estudiaremos en profundidad.

Para ejecutar el servidor, abrimos una consola y nos dirigimos a la carpeta donde se encuentra el archivo. En nuestro caso será **C:\Node**.

Una vez que estamos allí ejecutamos el servidor mediante el comando que mostramos a continuación:

```
node servidor.js
```

Tengamos en cuenta que al ejecutar este archivo, veremos un mensaje que informa que el servidor web ha iniciado y está funcionando en la dirección **127.0.0.1** en el puerto **3000**.

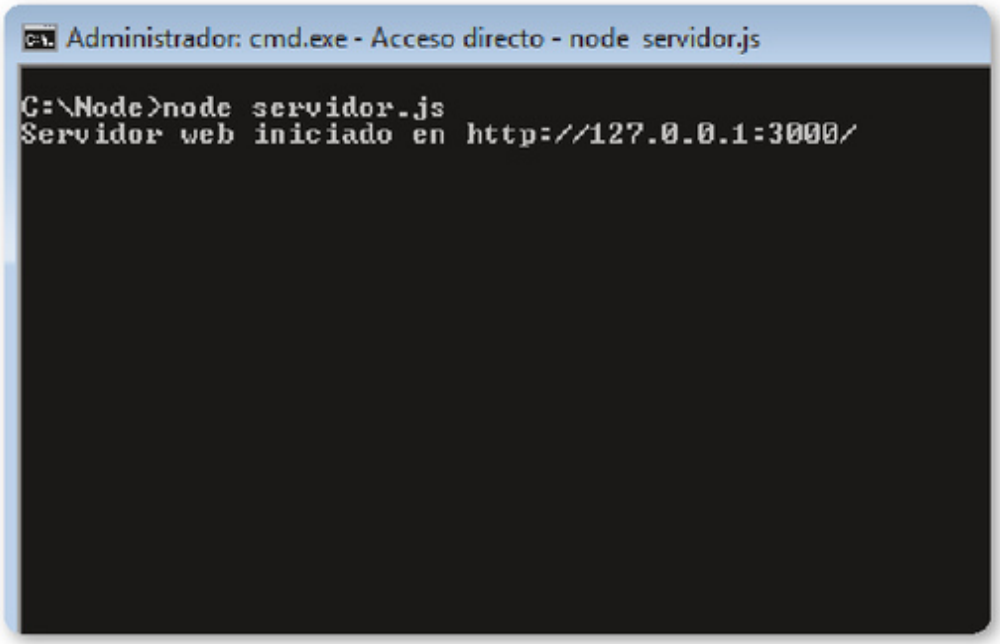


Figura 11. El servidor web creado con Node está ejecutándose en la dirección **127.0.0.1:3000**.

MÓDULO DE APACHE PARA OPTIMIZACIÓN

Para optimizar los tiempos de carga de nuestros sitios es recomendable usar un módulo para Apache llamado **mod_pagespeed**. Es de código abierto e implementa las mejores prácticas de rendimiento web sin necesidad de modificar nuestro código. Lo podemos descargar desde el siguiente enlace: **https://developers.google.com/speed/pagespeed/mod**.

Si es posible ver el mensaje, entonces el servidor está ejecutándose y podemos probarlo. Para esto, debemos abrir un navegador web y luego escribir la dirección **http://127.0.0.1:3000**.

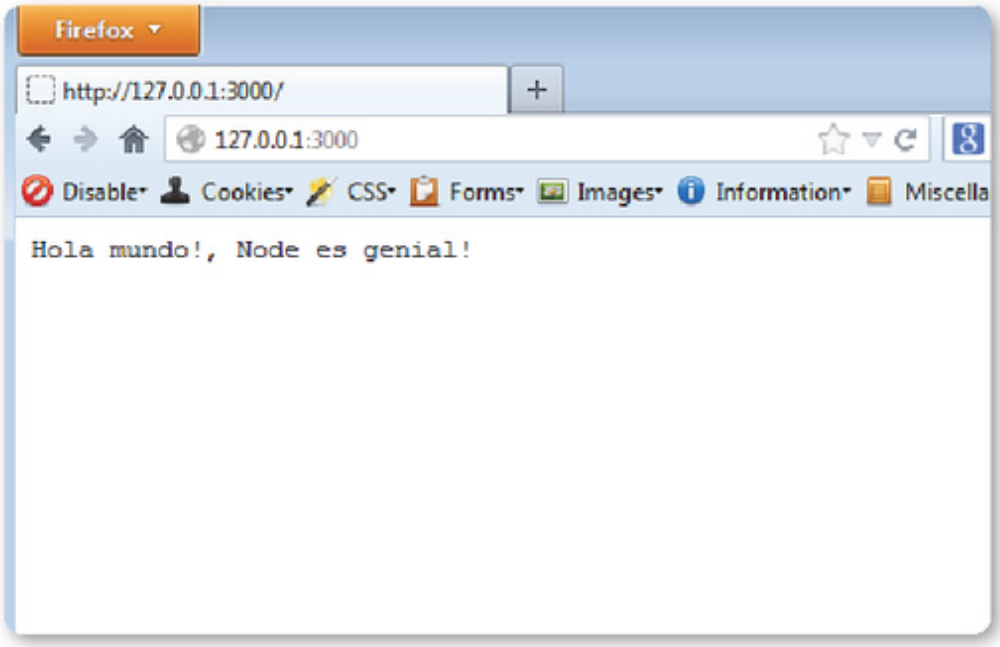


Figura 12. Se muestra el clásico mensaje **`Hola mundo`**, que indica que el servidor web está funcionando.

Hasta aquí, hemos desarrollado varios ejemplos que ejecutan código JavaScript fuera del navegador para obtener resultados mediante Node.

RESUMEN

Empezamos a conocer Node y algo de lo que podemos hacer con él. Conocimos las características más importantes para entender su funcionamiento y, luego, algunas de las diferencias que tiene con servidores web como Apache y otras tecnologías como Ajax. También vimos que Node es una excelente solución para el desarrollo de sistemas de tiempo real, y nombramos algunas de las empresas que lo utilizan. Por último, aprendimos a instalar Node en los tres sistemas operativos más populares y dimos nuestros primeros pasos con su entorno de desarrollo, REPL.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Node es un framework? Justificar.
2. ¿Cuál es el lenguaje que interpreta Node?
3. ¿Qué es el motor V8?
4. ¿Con Node podemos crear un servidor web?
5. ¿Node es sincrónico o asincrónico?
6. ¿Es recomendable utilizar Node para crear un sistema colaborativo?
7. ¿Node es aplicable al desarrollo de sistemas en tiempo real?
8. ¿Qué empresas utilizan Node?
9. ¿Es posible instalar Node en otra distribución de Linux que no sea Ubuntu?
10. ¿Es posible utilizar Node únicamente desde la consola?

EJERCICIOS PRÁCTICOS

1. En el ejemplo del servidor, modifique el código para devolver etiquetas HTML.
2. En el ejemplo del servidor, antes de la línea **response.end()** agregue **response.write('Respuesta!')** y observe qué respuesta se muestra en el navegador.
3. Utilice REPL para crear una función que calcule el valor factorial de un número.
4. Intente cambiar el puerto del ejemplo del servidor y ejecutarlo en el 3001.
5. Duplique el archivo **servidor.js** y modifique el puerto. Verifique si es posible tener dos instancias de un servidor web.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*07

Manejo de peticiones en Node

Aprenderemos a desarrollar un sistema en Node, donde veremos cómo estructurar el código para que sea más robusto y consistente. Continuaremos el ejemplo del Capítulo 6, en el que creamos un servidor web, pero le agregaremos más funcionalidad, convirtiéndolo en un módulo. Por último, crearemos un mecanismo para manejar las variables enviadas desde un formulario.

- Estructuración de código 210
- Convertir el servidor en un módulo 213
- Ruteo 215
- Manejadores 219
- Registro de accesos 229
- Servidor de documentos HTML 231
- Ruteo por defecto 234
- Manejo de peticiones POST 238
- Resumen 243
- Actividades 244

Estructuración de código

Cuando desarrollamos código en Node podemos establecer qué tipo de estructura utilizar en los proyectos. Una manera de hacerlo es definir todas las funciones en un solo archivo, pero no es recomendable ya que, a medida que se van agregando funcionalidades, el código se vuelve más difícil de entender y genera una complejidad extra. En principio, la manera más ordenada y clara es utilizar una arquitectura similar al patrón **MVC (Modelo-Vista-Controlador)**, mediante el cual se definen varios archivos o módulos.

A continuación veremos un ejemplo donde crearemos varios archivos para clarificar la estructura del sistema; en el **Capítulo 9**, usaremos un marco de trabajo para trabajar mediante el patrón **MVC**.

Servidor web

En el **Capítulo 6** armamos un servidor web simple utilizando Node; en este caso, vamos a crear un servidor muy similar pero con algunas características nuevas. Para comenzar, generamos una carpeta llamada **Node2** en la unidad **C** y, luego, un archivo JavaScript con el nombre **servidor.js**; dentro de este archivo agregamos el siguiente código:

```
var http = require('http');

function arrancarServidor(request, response) {
        console.log("Se ha conectado un usuario.");
        response.writeHead(200,{"Content-Type":"text/html"});
        response.write("<h1>Bienvenido. Servidor web de Node.js</h1>");
        response.end();
}

http.createServer(arrancarServidor);
http.listen(3000,'127.0.0.1');
console.log('Servidor web iniciado en http://127.0.0.1:3000/');
```

Analicemos cada línea. En la primera, mediante el comando **require**, incluimos el módulo **http**, que nos servirá para hacer uso de todos los métodos del protocolo, como por ejemplo, **createServer()** y **listen()**.

Luego, creamos la función llamada **arrancarServidor()**, que se ejecuta cada vez que un usuario accede al servidor y recibe los parámetros **request** (que contendrán toda la información de solicitud) y **response** (que contendrán el mensaje de salida de la función). Dentro de la función, lo primero que hacemos es enviar un mensaje a la consola para informar que un usuario acaba de conectarse.

> EL MÉTODO WRITEHEAD() NOS PERMITE ESCRIBIR EL ENCABEZADO DEL MENSAJE

En la siguiente línea, con el método **writeHead()** del objeto **response**, escribimos el encabezado del mensaje, que tiene un primer parámetro que indica el código de estado **200**: este valor determina que la solicitud será respondida con éxito. El segundo parámetro indica el tipo de dato que enviaremos al cliente; en este caso, usaremos **text/html**, para informar que enviaremos etiquetas HTML. Para continuar, escribimos el cuerpo del mensaje mediante el método **write** del objeto **response**. Podemos utilizar cualquier etiqueta HTML válida. En nuestro ejemplo, simplemente usaremos la etiqueta de título y un breve texto. Mediante el método **end()**, finalizamos el cuerpo de la respuesta.

Luego, nos encargaremos de utilizar el método **createServer()** del objeto **http** para realizar la creación del servidor web, y aquí pasamos el nombre de la función que hemos definido antes como parámetro. Mediante el método **listen** le indicamos al servidor que se quede escuchando en el puerto **3000** del servidor local.

Por último, enviamos un mensaje a la consola para indicar que el servidor web se ha iniciado.

CONTRASEÑAS INSEGURAS

Aunque la mayoría de los sitios que requieren autenticación recomiendan utilizar contraseñas difíciles de descifrar, la mayor parte de los usuarios prefieren utilizar palabras simples como **fútbol**, **ninja**, **123456**, **123abc**, **123456789**, etcétera. Lo más recomendable es utilizar contraseñas de ocho o más caracteres con espacios, como por ejemplo una frase de cuatro palabras.

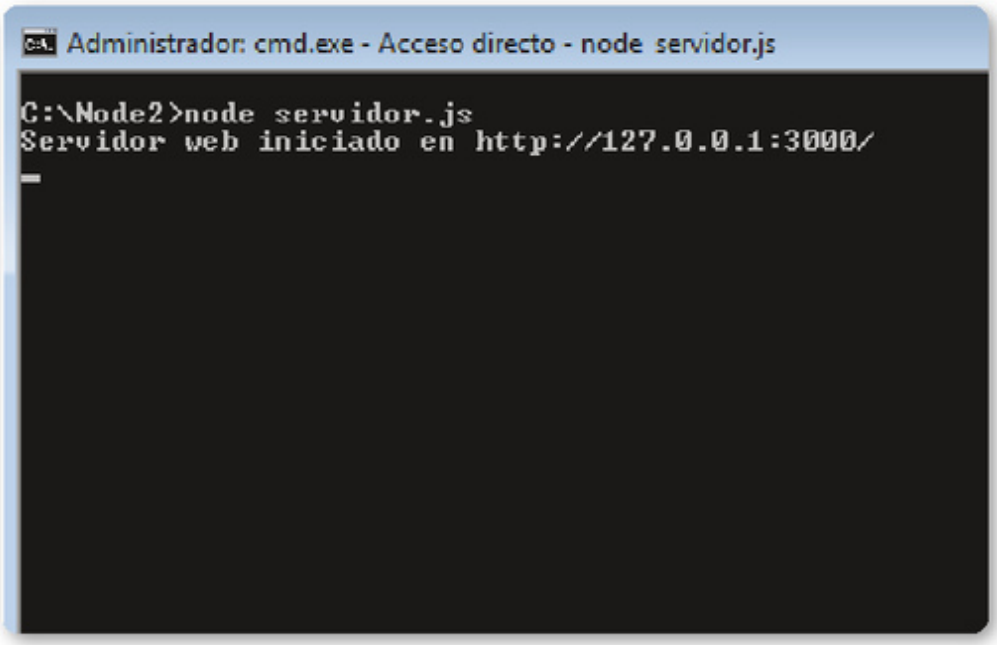


Figura 1. Al iniciar el servidor vemos el mensaje en la consola que indica la dirección y el puerto que está escuchando.

Para ejecutar el servidor abrimos una consola, nos situamos en el directorio **C:\Node2** y ejecutamos el comando **node servidor.js**.

Luego, abrimos un navegador e ingresamos a la dirección donde se encuentra el servidor; debemos ver el mensaje de bienvenida.

Figura 2. Al ingresar al servidor local en el puerto indicado vemos que Node presenta un documento HTML con un título.

Si regresamos a la consola veremos que tenemos un nuevo mensaje, que nos informa que un usuario nuevo se ha conectado. Esto es útil para obtener información de los usuarios y su dirección IP.

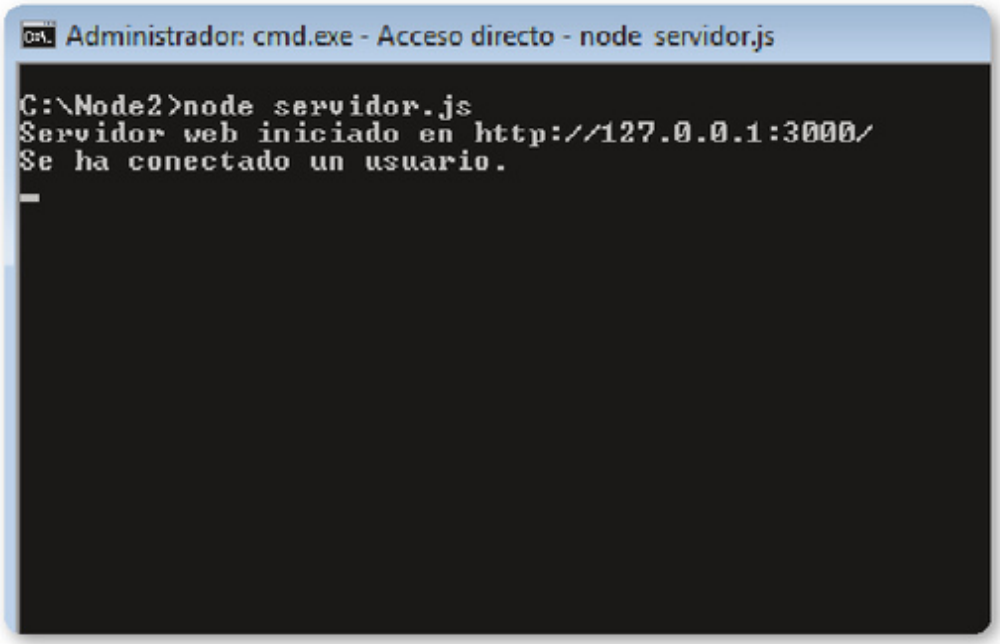


Figura 3. En la consola veremos un mensaje indicando que un nuevo usuario se ha conectado.

Convertir el servidor en un módulo

Ahora haremos unos cambios en el servidor para hacerlo más flexible. La idea es convertir el objeto servidor en un módulo, para que lo podamos llamar desde un archivo externo y pasarle parámetros adicionales, generando un comportamiento más dinámico y adaptable.

Para convertir el servidor en un módulo, primero debemos encapsular la función **arrancarServidor()** dentro de otra.

```
var http = require('http');

function iniciar() {
```

```
    function arrancarServidor(request, response) {
                    console.log("Se ha conectado un usuario.");
                    response.writeHead(200,{"Content-Type":"text/html"});
                    response.write("<h1>Bienvenido. Servidor web de Node.js</
h1>");
                    response.end();
            }

            http.createServer(arrancarServidor).listen(3000,'127.0.0.1');
            console.log('Servidor web iniciado en http://127.0.0.1:3000/');
    }

    exports.iniciar = iniciar;
```

Analizando el código que presentamos arriba, vemos que hemos definido la función **iniciar()**, que inicialmente carece de parámetros y contiene la función que crea el servidor.

Una cuestión a tener en cuenta es que, de ahora en adelante, vamos a trabajar con más de un archivo y es necesario especificar la comunicación entre ellos. Usaremos **exports**, que hace posible que las funciones sean accesibles para cualquier archivo que lo requiera.

Ahora crearemos el archivo principal en el mismo directorio de **servidor.js**, al que llamaremos **app.js** y contendrá el siguiente código:

```
var servidor = require('./servidor');

servidor.iniciar();
```

PRIMERA PROGRAMADORA DEL MUNDO

Ada Lovelace (1815-1852) es considerada la primera programadora informática del mundo, ya que fue la creadora de un programa informático que calculaba los **números de Bernoulli** mediante un algoritmo que contenía bucles. En homenaje a esta programadora, en los años 80, el Departamento de Defensa de Estado Unidos desarrolló un lenguaje de programación orientado a objetos llamado **ADA**.

En el código anterior, en primer lugar, creamos una variable y, mediante **require**, incluimos el archivo **servidor.js**; con **./** indicamos que el archivo se encuentra en el mismo directorio que **app.js**. Como Node trabaja con JavaScript, no es necesario que agreguemos la extensión, ya que por defecto se asume que se trata de un archivo de esta clase. Luego, utilizamos la variable **servidor** e invocamos al método **iniciar()**, que es la función que exportamos antes.

En este punto, para iniciar el servidor ya no debemos invocar a **servidor.js** sino al archivo **app.js**, que se encargará de iniciar el servidor. Para comprobar que todo funciona como esperamos, abrimos una consola, ejecutamos el comando **node app.js** y verificamos en el navegador como lo hicimos antes.

Mediante estas modificaciones hemos convertido el archivo **servidor.js**, que era el principal, a un módulo que utilizamos en **app.js**. En principio no obtuvimos ninguna ventaja de este procedimiento, pero preparamos una estructura que nos servirá para poder pasarle parámetros al servidor y personalizar su comportamiento.

Ruteo

Como nuestro objetivo es crear un servidor web, debemos manejar los comportamientos del servidor especificando diferentes direcciones en la URL. Hasta ahora, el servidor tiene un comportamiento estático, es decir que responde de la misma manera si accedemos a **127.0.0.1:3000** o a **127.0.0.1:3000/inicio**. Haremos que el comportamiento dependa de la ruta HTTP solicitada.

Para empezar, creamos un nuevo archivo en el mismo directorio del archivo principal con el nombre **enrutador.js** y luego agregamos el código que sigue:

```
function enrutar(ruta){
        console.log('Rutear a: ' + ruta);
}

exports.enrutar = enrutar;
```

ENRUTAR() ESPERA UNA RUTA COMO PARÁMETRO PARA IMPRIMIR EN CONSOLA

En el código anterior, inicialmente definimos la función **enrutar()**, que espera como parámetro una ruta que luego vamos a imprimir en la consola. Luego, exportamos la función para que sea accesible desde otro archivo.

A continuación, realizaremos algunos cambios en el archivo principal llamado **app.js** para poder usar el enrutador. Lo que haremos es incluir el archivo **enrutador.js** y, posteriormente, pasaremos la variable creada como parámetro al método **iniciar()**. Para ello agregamos el siguiente código:

```
var servidor  = require('./servidor');
var enrutador = require('./enrutador');

servidor.iniciar(enrutador.enrutar);
```

Ahora trabajaremos en el archivo **servidor.js**. Primero vamos a requerir un nuevo módulo de sistema, llamado **url**, que es el encargado de registrar la dirección de la ruta solicitada para que luego podamos definir un comportamiento personalizado para cada petición. Incluimos el módulo de la siguiente manera:

```
var url  = require('url');
```

En la función **iniciar()** capturamos el parámetro que agregamos antes en **app.js**. Luego, dentro de **iniciar()**, vamos a capturar la URL que se especificó en el navegador y la asignaremos a una variable. Para continuar, invocamos el método **enrutar()** y le pasamos como parámetro la variable que acabamos de crear:

```
var ruta = url.parse(request.url).pathname;
enrutar(ruta);
```

Ahora veremos cómo debe quedar el código completo de **servidor.js**.

```
var http = require('http');
var url  = require('url');

function iniciar(enrutar) {
        function arrancarServidor(request, response) {
                console.log("Se ha conectado un usuario.");

                var ruta = url.parse(request.url).pathname;
                enrutar(ruta);

                response.writeHead(200,{"Content-Type":"text/html"});
                response.write("<h1>Bienvenido. Servidor web de Node.js</
h1>");
                response.end();
        }

        http.createServer(arrancarServidor).listen(3000,'127.0.0.1');
        console.log('Servidor web iniciado en http://127.0.0.1:3000/');
}

exports.iniciar = iniciar;
```

Abrimos una consola, nos situamos en el directorio del servidor y ejecutamos **node app.js**. Si vemos **Servidor web iniciado en http://127.0.0.1:3000/** no se ha producido ningún error y podemos acceder al navegador para probarlo.

Si en el navegador vamos a **http://127.0.0.1:3000** y luego regresamos a la consola, encontraremos dos mensajes, uno que informa que un usuario se ha conectado y otro muestra la URL a la que se ha ruteado. Es probable que luego tengamos otro mensaje que indica que un usuario se conectó solicitando **/favicon.ico**; esto porque la mayoría de los navegadores realizan dos peticiones por cada solicitud: una por contenido y otra por el icono del documento.

> LA MAYORÍA DE LOS NAVEGADORES REALIZAN DOS PETICIONES POR CADA SOLICITUD

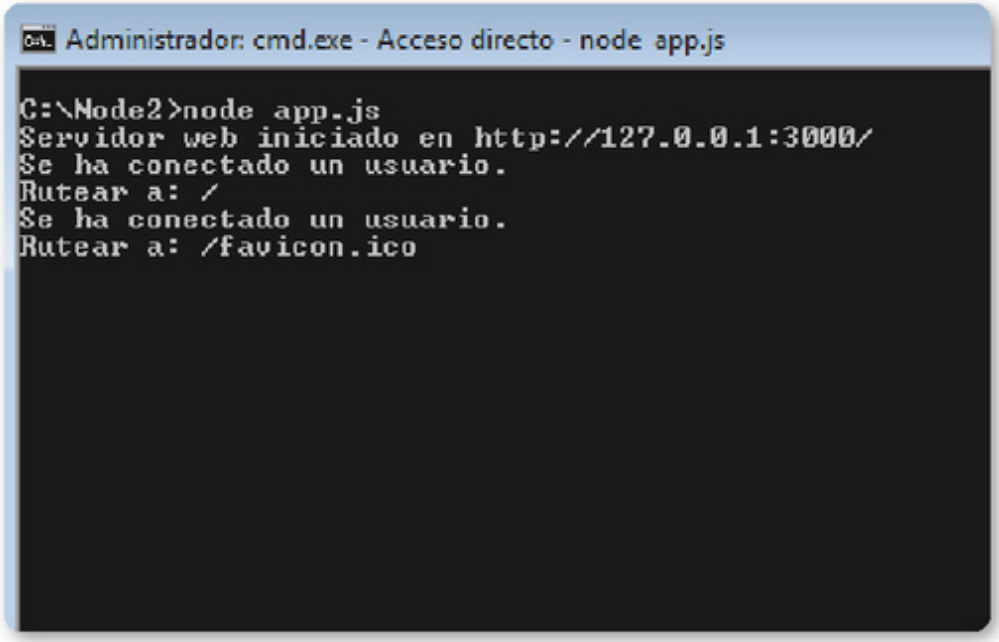


Figura 4. En la consola vemos el mensaje que indica la ruta solicitada desde el navegador.

A continuación, si escribimos la URL **http://127.0.0.1:3000/pagina1**, el servidor se encargará de recibir esta solicitud y posteriormente escribirá en la consola lo que corresponde.

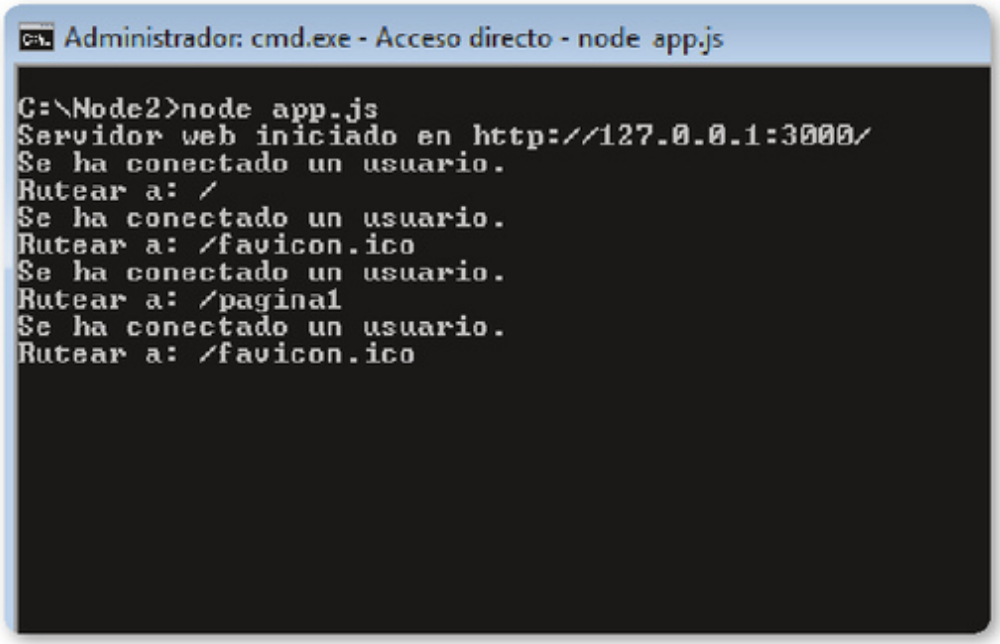


Figura 5. Para la solicitud de **`/pagina1`** el servidor nuevamente escribirá la ruta en la consola y recibirá una solicitud para **`/favicon.ico`**.

Manejadores

Anteriormente, capturamos la ruta que viene en la URL. Ahora debemos definir qué comportamiento tendrá el servidor para responder a cada petición.

Para comenzar, vamos a crear un nuevo archivo con el nombre **manejador.js** en el mismo directorio del servidor. Aquí vamos a generar una función para cada petición que necesitemos atender y que devolverá un contenido al navegador y escribirá un mensaje en la consola.

A modo de ejemplo, vamos a establecer los comportamientos para **inicio**, **paginal** y **salida**; además, es necesario definir una función para la petición de **favicon**. Si no definimos esta última función, se producirá un error cada vez que accedamos a una dirección del servidor.

Dentro de **manejador.js** escribimos el siguiente código:

```
function inicio(response) {
        console.log('Solicitud para: inicio');
        response.writeHead(200, {'Content-Type': 'text/html'});
        response.write('<h1>Contenido de inicio</h1>');
        response.end();
}

function paginal(response) {
        console.log('Solicitud para: paginal');
        response.writeHead(200, {'Content-Type': 'text/html'});
        response.write('<h1>Contenido de paginal</h1>');
        response.end();
}

function salida(response) {
        console.log('Solicitud para: salida');
        response.writeHead(200, {'Content-Type': 'text/html'});
        response.write('<h1>Contenido de salida</h1>');
        response.end();
}
function favicon(response) {
```

```
        console.log('Solicitud para: favicon');
        response.writeHead(200, {'Content-Type': 'text/html'});
        response.write('favicon');
        response.end();
}

exports.inicio  = inicio;
exports.paginal = paginal;
exports.salida  = salida;
exports.favicon = favicon;
```

En el código vemos que, en cada función, se devuelve un mensaje a la consola; luego, devolvemos el código de estado y el contenido al navegador, y finalizamos el mensaje. Cada función debe ser exportada para que pueda ser utilizada desde otro archivo.

> PARA QUE PUEDA SER USADA POR OTRO ARCHIVO, CADA FUNCIÓN DEBE SER EXPORTADA

Un aspecto importante a tener en cuenta es que en cada función definimos la respuesta para el navegador; de esta manera, prevenimos códigos bloqueantes. Es decir, si por algún motivo una función presentara un error o un bucle infinito, se bloquearía el servidor. En cambio, como los manejadores son independientes, el servidor seguirá atendiendo solicitudes sin que sea afectado por procesos que demoren mucho tiempo.

Ahora modificaremos el archivo principal. Para esto, abrimos **app.js** e incluimos el archivo **manejador.js**; luego, creamos un **array** que contendrá las rutas para las cuales vamos a devolver un contenido. Aunque parezca un poco extraño, vamos a almacenar cada

GOOGLE DE LIMPIEZA

Hace un tiempo, Google informó la lista de servicios que ya no estarán disponibles, debido a que centrarán sus esfuerzos en su red social **G+**. Algunos de los servicios que se han dejado atrás son: Desktop, la Api de Google Maps para Flash, Google Pack, Notebook, Sidewiki y Google Web Security, entre otros.

función del manejador en un índice del array **peticion**, y luego, cuando necesitemos invocar cada método, lo haremos desde esta variable.

Por último, nos encargaremos de pasarle como parámetro el array al método **iniciar()** del servidor que corresponde.

```
var servidor  = require('./servidor');
var enrutador = require('./enrutador');
var manejador = require('./manejador');
var peticion  = {};

peticion['/inicio']      = manejador.inicio;
peticion['/pagina1']     = manejador.pagina1;
peticion['/salida']      = manejador.salida;
peticion['/favicon.ico'] = manejador.favicon;

servidor.iniciar(enrutador.enrutar, peticion);
```

Lo siguiente es modificar el archivo **servidor.js**. En primer lugar, agregamos el parámetro **peticion** a la función **iniciar()**; luego, pasamos las variables **peticion** y **response** como parámetros al método **enrutar()**. Recordemos que la variable **peticion** es un array que contiene en cada índice una función que devuelve un contenido al navegador, y **response** es el objeto por el cual se capturan las respuestas.

Algo adicional, pero no indispensable, es comentar las líneas de este archivo que devuelven las respuestas al navegador, ya que en cada función del manejador definimos una respuesta personalizada.

El archivo **servidor.js** debe quedar de la siguiente manera:

RECIBIR VARIABLES EN PHP

Debemos tener en cuenta que muchas veces en PHP es necesario pasar variables desde un lugar a otro. Para recuperar las variables existen dos posibilidades, mediante el uso de **$_POST** o **$_GET**, pero si no sabemos por cuál método vienen es posible utilizar **$_REQUEST**, lo que nos garantiza que las variables se recuperarán sin importar el método utilizado.

```
var http = require('http');
var url  = require('url');

function iniciar(enrutar, peticion) {
        function arrancarServidor(request, response) {
                console.log("Se ha conectado un usuario.");

                var ruta = url.parse(request.url).pathname;
                enrutar(ruta, peticion, response);

                //response.writeHead(200,{"Content-Type":"text/html"});
                //response.write("<h1>Bienvenido. Servidor web de Node.js</
h1>");
                //response.end();
        }

        http.createServer(arrancarServidor).listen(3000,'127.0.0.1');
        console.log('Servidor web iniciado en http://127.0.0.1:3000/');
}

exports.iniciar = iniciar;
```

Por último, vamos a modificar el archivo **enrutador.js**. En primer lugar, debemos incluir los parámetros en la función **enrutar()** y, dentro de esta, devolver el contenido personalizado dependiente de la ruta establecida. El archivo deberá quedar como sigue:

EL SIGNIFICADO DE GNU

GNU es un sistema creado por **Richard Stallman**, que tiene como objetivo promulgar el espíritu de cooperación para el desarrollo de sistemas informáticos basado en cuatro libertades. GNU es un acrónimo recursivo que significa **GNU No es Unix** o **GNU is Not Unix**. Si deseamos obtener datos adicionales e información interesante sobre este sistema podemos visitar la página que se encuentra en la dirección web: **http://gnu.org/home.es.html**.

```
function enrutar(ruta, peticion, response){
        console.log('Rutear a: ' + ruta);

        return peticion[ruta](response);
}

exports.enrutar = enrutar;
```

La función **enrutar()** va a retornar un contenido mediante el índice del array **peticion**, que contiene un método definido en **manejador.js**.

Para probar el funcionamiento del manejador, debemos abrir una consola y situarnos en el directorio **C:\Node2**; luego, debemos iniciar el servidor con el comando **node app.js**. Si todo se encuentra bien, veremos un mensaje indicando que el servidor web está iniciado.

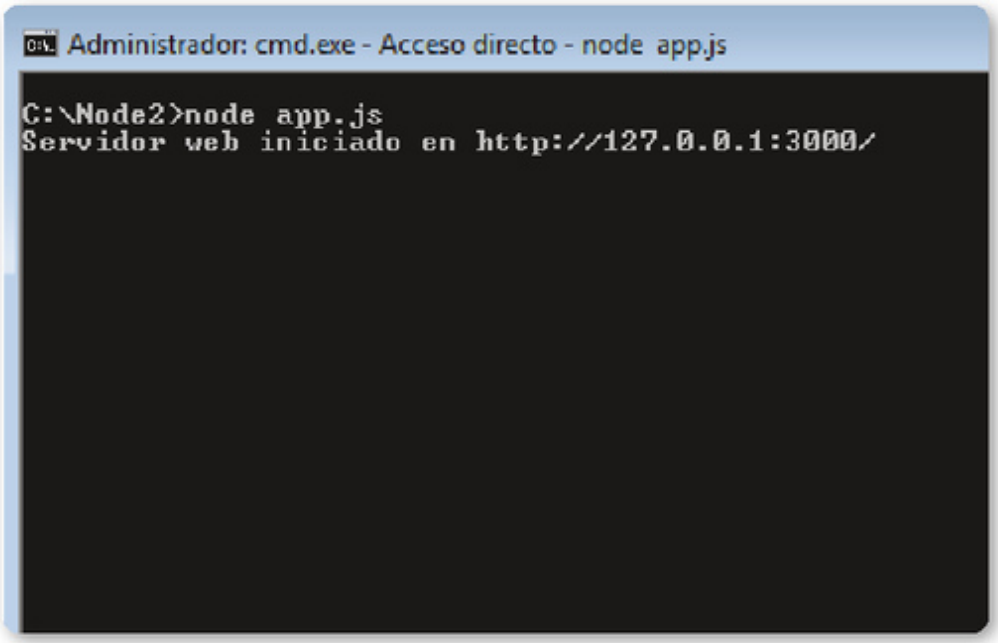


Figura 6. Estamos en condiciones de utilizar el servidor web si vemos el mensaje que informa que ha iniciado.

A continuación tendremos que abrir un navegador web y, en la barra de direcciones correspondiente, escribimos lo siguiente: **127.0.0.1:3000/inicio**. Si todo resulta bien podremos ver un título que muestra el siguiente texto: **Contenido de inicio**.

Figura 7. Ingresando la ruta **`/inicio`** en un navegador, el servidor devuelve el contenido correspondiente.

Si regresamos a la consola, además del mensaje que informa que el servidor ha iniciado de manera correcta, se presentarán los mensajes que corresponden a las solicitudes que hemos enviado al servidor cuando intentamos acceder a **/inicio**.

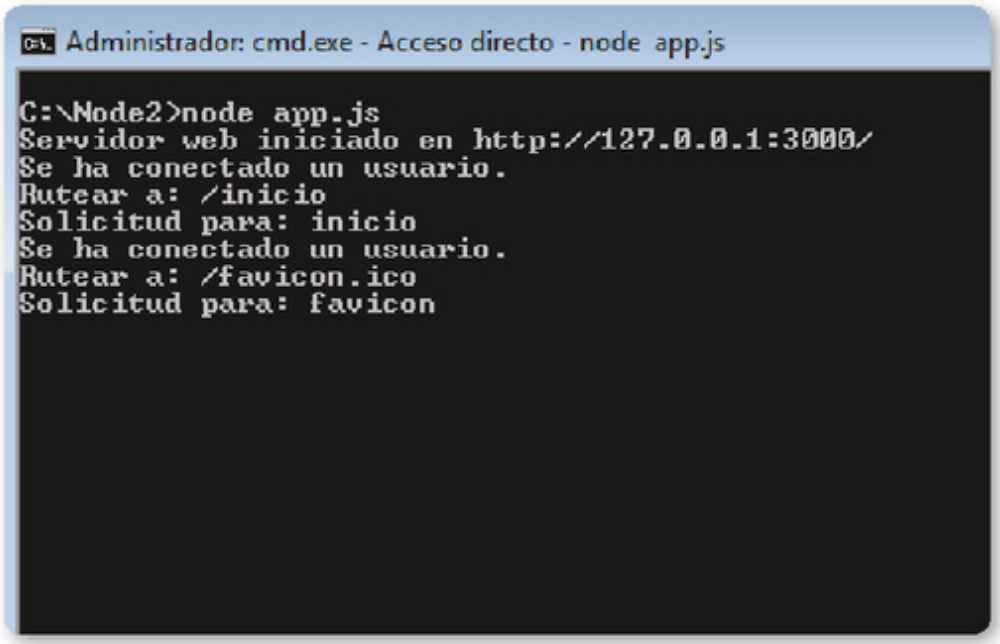


Figura 8. Por cada solicitud vemos que también se solicita el **`favicon`**. Lo controlamos mediante la función **`favicon()`** en **`manejador.js`**.

Ahora, si accedemos a las otras rutas que hemos definido en el manejador, por ejemplo **/pagina1** y **/salida**, el servidor mostrará el contenido correspondiente y, en la consola, veremos las peticiones que se han realizado para cada solicitud.

De este modo podremos controlar en todo momento las solicitudes que se han realizado al servidor.

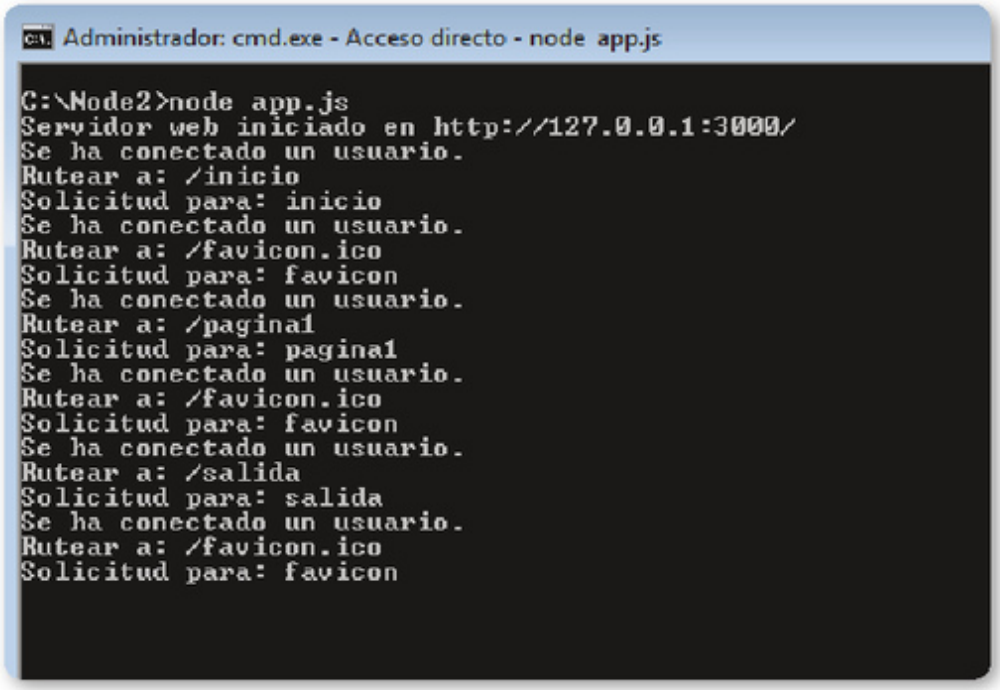


Figura 9. En la consola vemos que se han solicitado las rutas **`/inicio`**, **`/pagina1`** y **`/salida`**; además, el navegador solicita el **`favicon`** de modo automático.

Vamos a intentar acceder a una ruta que no hayamos definido, por ejemplo **/pagina2**. Aquí, veremos que se ha producido un error y no hemos podido establecer la conexión con el servidor.

NGINX

Nginx es un servidor web muy potente de alto rendimiento. Como una de sus principales características, ofrece un proxy para los protocolos de correo electrónico. Este servidor se distribuye como software libre y de código abierto, es multiplataforma y es utilizado por sitios como Wordpress, GitHyb y SourceForge entre otros. Podemos conocerlo mejor en el siguiente enlace: **http://nginx.org**.

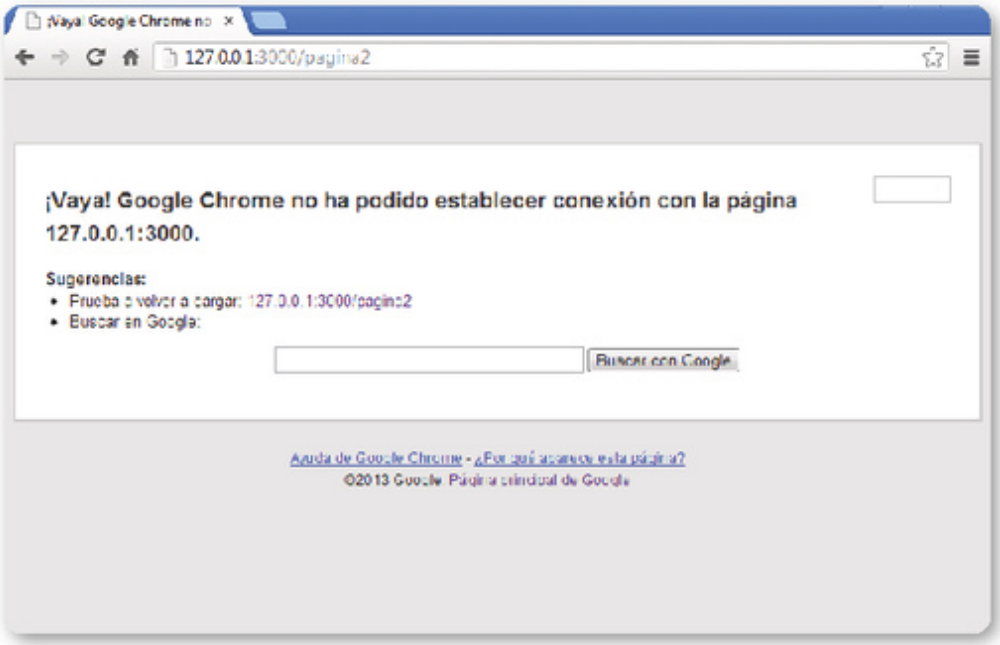


Figura 10. El servidor no encontró la ruta solicitada y se perdió la comunicación.

Para conocer la causa del error, debemos ver qué es lo que nos informa la consola. En ella podemos encontrar información que nos permitirá identificar el tipo de problema al cual nos enfrentamos. Un ejemplo de esto lo vemos en la siguiente imagen:

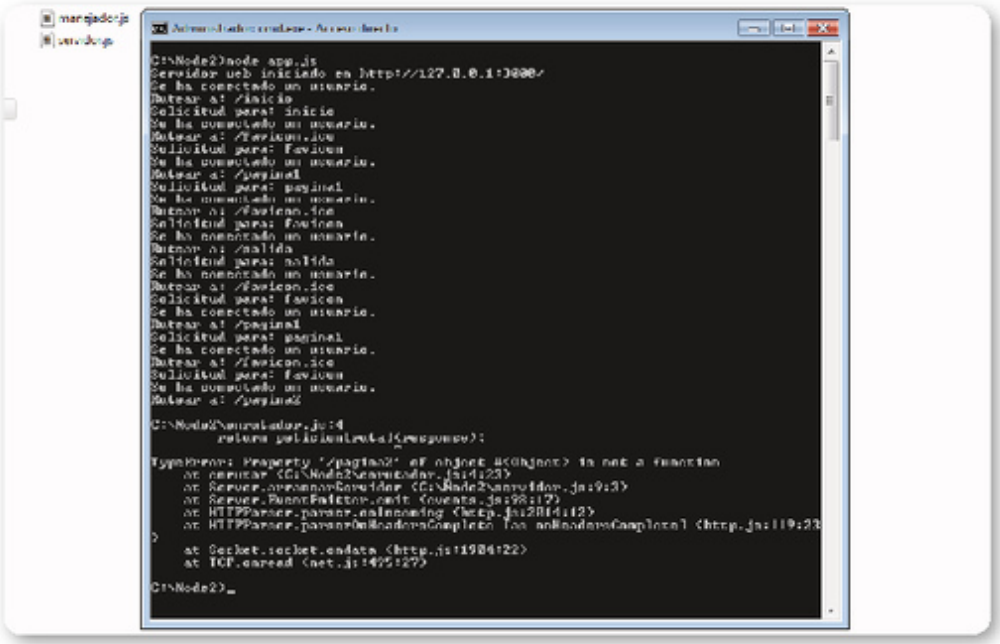


Figura 11. El servidor se ha detenido porque no pudo encontrar ninguna función que maneje la ruta solicitada.

Debemos considerar que para prevenir los errores de rutas inexistentes, vamos a trabajar en el archivo **enrutador.js**. Lo modificamos tal como mostramos en el siguiente código:

```
function enrutar(ruta, peticion, response){
    console.log('Rutear a: ' + ruta);

    if (typeof peticion[ruta] === 'function'){
        return peticion[ruta](response);
    }else{
        console.log('No se ha definido una función para ' + ruta);
        response.writeHead(404, {"Content-Type": "text/html"});
        response.write('<h1>404 - No se ha encontrado: ' + ruta + '</h1>');
        response.end();
    }
}

exports.enrutar = enrutar;
```

En el código hemos ingresado un condicional antes de devolver el resultado, que comprueba que **peticion[ruta]** sea una función. Consideremos que, en el caso contrario, devuelve una típica respuesta de página no encontrada con un código de error **404**.

Para verificar si resolvimos el problema de solicitudes para páginas no encontradas, podemos acceder al navegador e ingresar, por ejemplo, a **127.0.0.1:3000/pagina2**. Podemos ver que se presenta una imagen similar a la que mostramos a continuación.

¿QUE ES WEB SEMÁNTICA?

Desde hace algún tiempo hemos venido escuchando el término web semántica. La web semántica se puede resumir como una web extendida, donde cada usuario puede encontrar respuestas a sus preguntas de forma simple y rápida, gracias a que el principal objeto de la web es la información contenida en una infraestructura común. Podemos saber más del tema en la guía básica de la w3C: **www.w3c.es/Divulgacion/GuiasBreves/WebSemantica**.

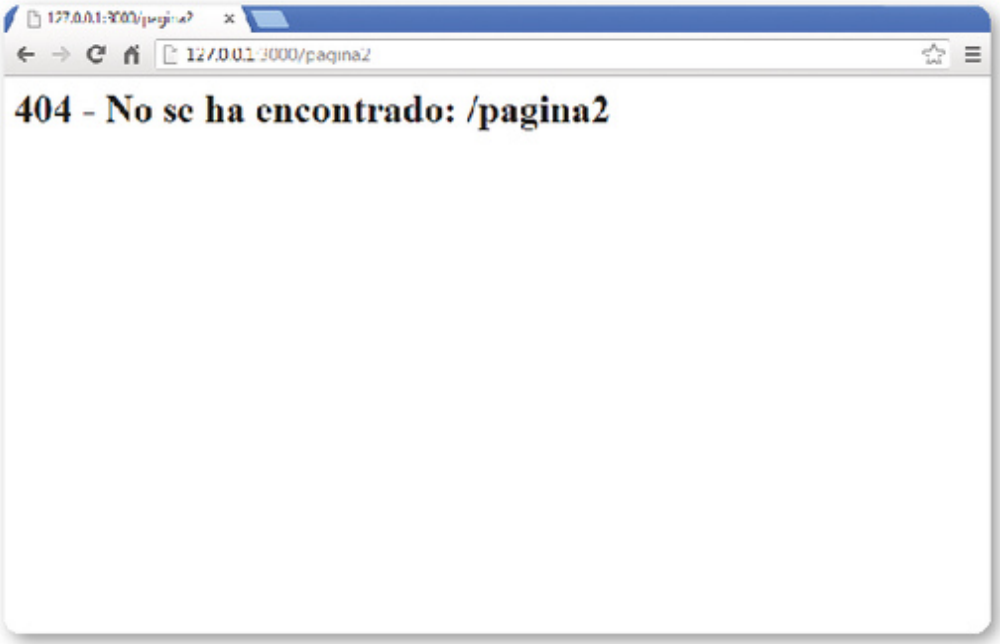


Figura 12. El servidor devolverá un mensaje de error de página no encontrada cuando no pueda enrutar la solicitud.

Si vamos a la consola, observamos que muestra un detalle de error de enrutamiento. Con esta última modificación podemos controlar cualquier solicitud que no pueda resolver el servidor.

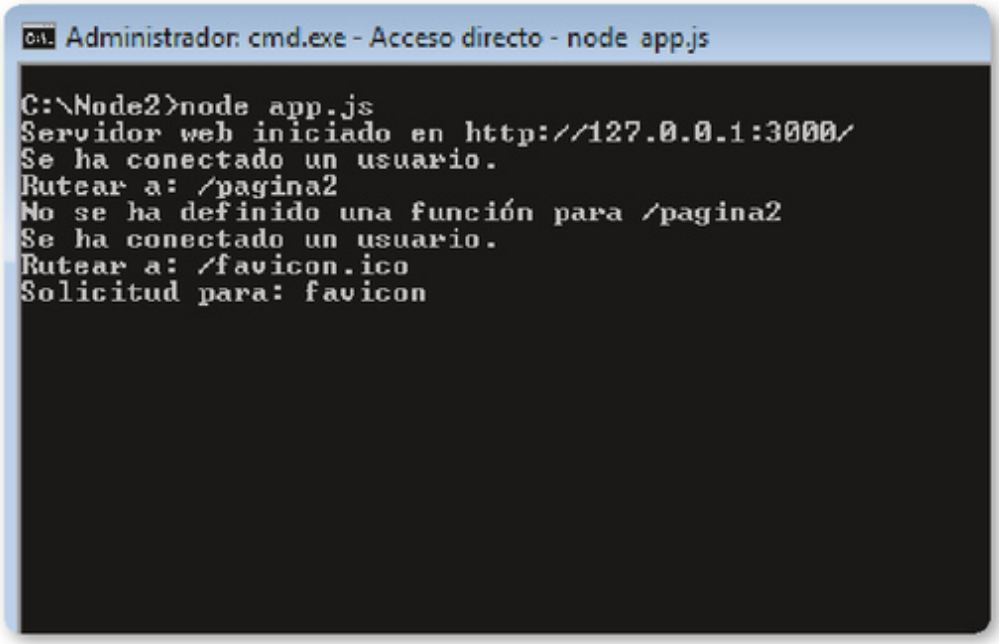


Figura 13. Como no hemos definido un comportamiento para **/pagina2**, al intentar rutear esta petición se informa en la consola.

Registro de accesos

Hasta ahora hemos creado un mecanismo para que Node nos informe, a través de la consola, qué está sucediendo con el servidor web. Pero existe un problema: si por alguna razón necesitamos reiniciar el servidor, la consola perderá todos los registros obtenidos hasta el momento y no podremos recuperar la información.

Para solucionar esta deficiencia, nos encargaremos de crear un sistema para registrar de manera permanente todos los accesos. Utilizaremos el módulo **fs** de Node, que nos permitirá leer y escribir archivos en el servidor.

> EL MÓDULO FS DE NODE NOS PERMITE LEER Y ESCRIBIR ARCHIVOS EN EL SERVIDOR

Lo único que necesitamos es abrir el archivo **servidor.js** y modificarlo con el siguiente código:

```
var http = require('http');
var url  = require('url');
var fs   = require('fs');

function iniciar(enrutar, peticion) {
        function arrancarServidor(request, response) {
                console.log("Se ha conectado un usuario.");

                var ruta = url.parse(request.url).pathname;
                enrutar(ruta, peticion, response);

                var archivo_accesos = fs.createWriteStream('accesos.
txt',{'flags':'a'});
                archivo_accesos.write(ruta + '\n')

                //response.writeHead(200,{"Content-Type":"text/html"});
                //response.write("<h1>Bienvenido. Servidor web de Node.js</
h1>");
                //response.end();
        }
```

```
        http.createServer(arrancarServidor).listen(3000,'127.0.0.1');
        console.log('Servidor web iniciado en http://127.0.0.1:3000/');
}

exports.iniciar = iniciar;
```

En el código, incluimos el módulo **fs** y luego creamos una variable llamada **archivo_accesos**, a la cual le asignamos el resultado del método **createWriteStream()** (donde el primer parámetro es el nombre del archivo y el segundo el indicador para que el contenido se agregue al final). Escribimos la ruta y un salto de línea para la siguiente escritura.

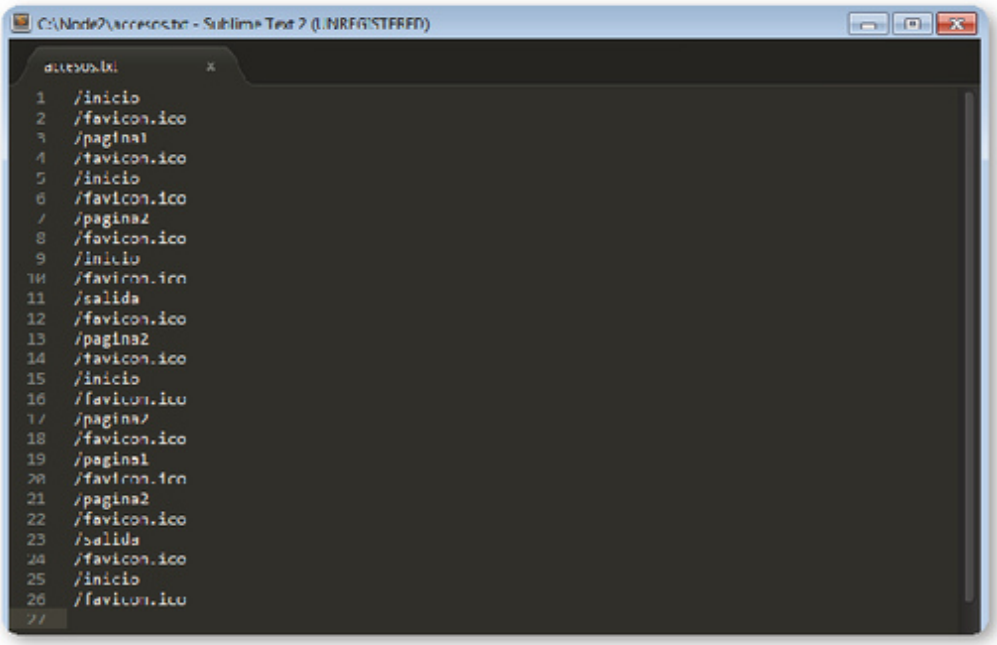


Figura 14. Si abrimos el archivo **accesos.txt** veremos todas las peticiones que ha recibido el servidor.

GOOGLE FLIGHT

Un servicio muy útil para los viajeros es el que Google ha llamado **Flight**. Con este sistema es posible buscar destinos, escalas, compañías aéreas, precios, duración y horarios, entre otros datos. Además, este servicio permite encontrar los días y horarios en que están disponibles los vuelos más baratos.

Servidor de documentos HTML

Node puede ser utilizado tanto como servidor de procesos como de documentos. Hemos creado un manejador para cada petición que devuelve una estructura HTML básica, pero no es la manera más adecuada, ya que resulta bastante dificultoso escribir una estructura HTML compleja en una función JavaScript. Ahora vamos a modificar el comportamiento del servidor para devolver archivos HTML reales.

> NODE PUEDE SER USADO COMO SERVIDOR DE PROCESOS Y TAMBIÉN DE DOCUMENTOS

En primer lugar, dentro del servidor, creamos una carpeta llamada **www** y, dentro, tres archivos, denominados **inicio.html**, **pagina1.html** y **salida.html**. Dentro de estos archivos vamos a escribir código.

En **inicio.html**:

```
<h1>Página inicio.html</h1>
<p>Documento HTML<p>
```

Luego, en **pagina1.html**:

```
<h1>Página inicio.html</h1>
<p>Documento HTML<p>
```

Por último, en el archivo **salida.html**:

```
<h1>Página salida.html</h1>
<p>Documento HTML<p>
```

Necesitamos responder a cada solicitud con el contenido de los archivos que hemos creado antes. Para esto, trabajaremos otra vez con el módulo **fs** en el archivo **servidor.js**:

```
var http = require('http');
var url  = require('url');
var fs   = require('fs');

function iniciar(enrutar, peticion) {
        function arrancarServidor(request, response) {
                console.log("Se ha conectado un usuario.");

                var ruta = url.parse(request.url).pathname;
                //enrutar(ruta, peticion, response);
                var html = fs.readFileSync('www/' + ruta);

                var archivo_accesos = fs.createWriteStream('accesos.
txt',{'flags':'a'});
                archivo_accesos.write(ruta + '\n')

                response.writeHead(200,{"Content-Type":"text/html"});
                response.write(html);
                response.end();
        }

        http.createServer(arrancarServidor).listen(3000,'127.0.0.1');
        console.log('Servidor web iniciado en http://127.0.0.1:3000/');
}

exports.iniciar = iniciar;
```

Lo que hicimos primero fue crear la variable **html**, que contendrá el resultado devuelto por el método **readFileSync()**. Este leerá el archivo que se ha pedido mediante la variable **ruta** dentro de la carpeta **www**.

Luego, descomentamos el bloque que devuelve contenido al navegador y le asignamos el contenido de la variable **html**. Como ahora la respuesta al navegador la damos de manera directa desde **servidor.js**, el método **enrutar()** ya no será necesario, por lo tanto lo comentamos.

Para probar el procedimiento, reiniciamos el servidor y, en el navegador, ingresamos **127.0.0.1:3000/inicio.html**. Debemos ver lo que se muestra en la **Figura 15**.

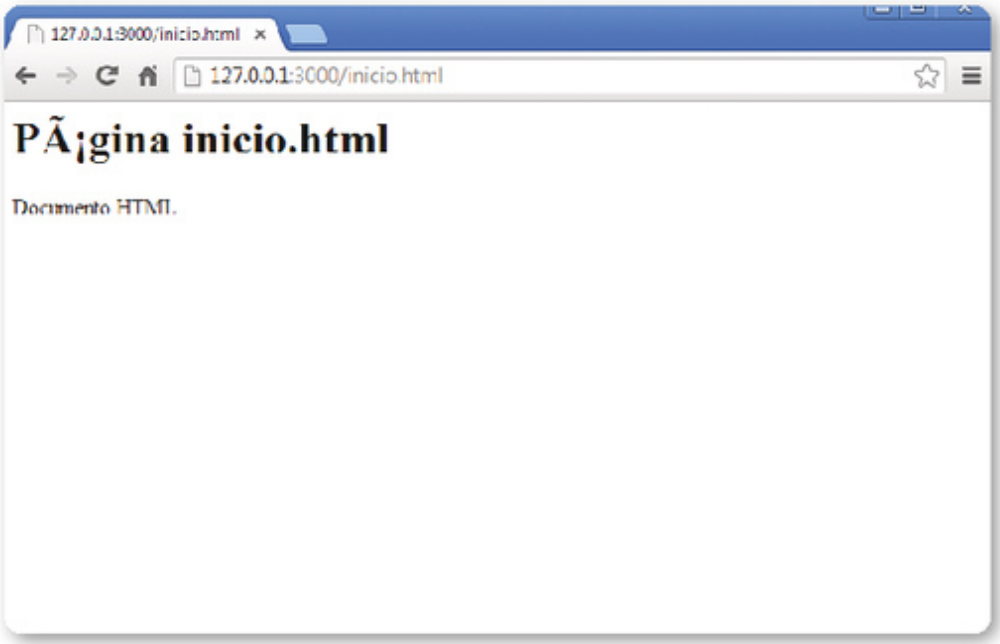


Figura 15. El servidor responde a las solicitudes que coinciden con los documentos HTML que hemos creado.

Si regresamos a la consola, veremos dos veces el mensaje **Se ha conectado un usuario**. Primero, porque se ha solicitado una página, y segundo, porque también se pidió el **favicon**. Más abajo vemos que el módulo **fs** ha generado un error intentando obtener el archivo **favicon.ico**.

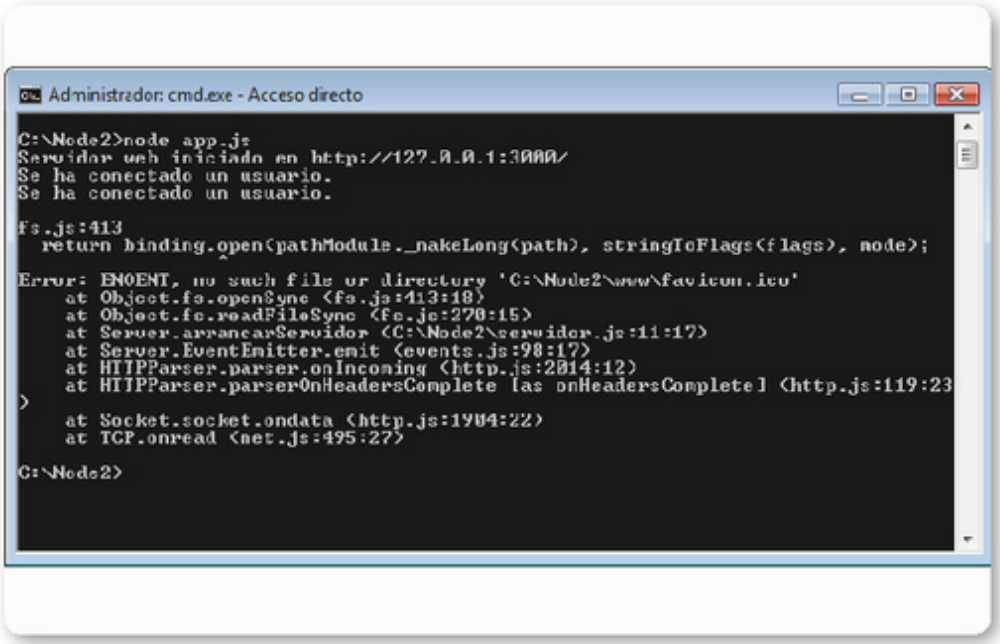


Figura 16. El error devuelto por Node indica que no es posible encontrar el archivo `favicon.ico`.

Para solucionar este error debemos tener un archivo con el nombre **favicon.ico** dentro del directorio **www**, de manera que el servidor pueda responder a la petición que realiza el navegador con cada solicitud.

Ruteo por defecto

La mayoría de los servidores, como Apache, responden a un archivo por defecto cuando se accede a la raíz del sitio. Sin embargo, si en nuestro servidor accedemos a la dirección **127.0.0.1:3000**, se producirá un error debido a que no hemos definido un comportamiento para esto y Node no sabe qué archivo debe devolver.

Para solucionar este error vamos a modificar el archivo **servidor.js**, para que, cuando la ruta sea igual a */*, devuelva el archivo **inicio.html**:

```
var http = require('http');
var url  = require('url');
var fs   = require('fs');

function iniciar(enrutar, peticion) {
        function arrancarServidor(request, response) {
                console.log("Se ha conectado un usuario.");

                var ruta = url.parse(request.url).pathname;
                //enrutar(ruta, peticion, response);

                if(ruta == '/')
```

MENSAJES EN CONSOLA

El objeto **console** posee diferentes métodos para mostrar mensajes. Por ejemplo, con **log()** es posible enviar un mensaje convencional, con **warm()** se muestra el mensaje precedido de un símbolo de advertencia, con **error()** se presenta el mensaje en color rojo con un símbolo de error y con **info()** aparece el mensaje con un símbolo de información.

```
                    ruta = 'inicio.html';

                var html = fs.readFileSync('www/' + ruta);

                var archivo_accesos = fs.createWriteStream('accesos.
txt',{'flags':'a'});
                archivo_accesos.write(ruta + '\n')

                response.writeHead(200,{"Content-Type":"text/html"});
                response.write(html);
                response.end();
        }

        http.createServer(arrancarServidor).listen(3000,'127.0.0.1');
        console.log('Servidor web iniciado en http://127.0.0.1:3000/');
}

exports.iniciar = iniciar;
```

Debemos reiniciar el servidor y comprobar accediendo a **127.0.0.1:3000** y luego a **127.0.0.1:3000/inicio.html**. Veremos **inicio.html** y, en la consola, notaremos que no se han producido errores.

Cambio de la codificación

Como hemos visto en la **Figura 15**, en el navegador, cuando accedemos a **inicio.html**, los caracteres con tilde o la letra ñ no se muestran de manera correcta. Esto se debe a que no hemos definido la codificación que utilizará el documento.

Para solucionar este error, en cada documento HTML debemos agregar una etiqueta para informarle al navegador que se devolverá un contenido **UTF-8**. Esta codificación hace posible la representación de cualquier caracter **Unicode**. La etiqueta de codificación es la siguiente:

```
<meta charset="UTF-8">
```

Para utilizarla, vamos a agregarla dentro de una estructura HTML5 básica, donde **inicio.html** quedará de la siguiente manera:

```
<!doctype html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Inicio</title>
</head>
<body>
        <h1>Página inicio.html</h1>
        <p>Documento HTML<p>
</body>
</html>
```

En **pagina1.html** también realizamos el cambio, y lo dejamos de la siguiente manera:

```
<!doctype html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Pagina1</title>
</head>
<body>
        <h1>Página pagina1.html</h1>
        <p>Documento HTML<p>
</body>
</html>
```

Por último, en el archivo **salida.html** también realizamos este cambio:

```
<!doctype html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
        <title>Salida</title>
</head>
<body>
        <h1>Página salida.html</h1>
        <p>Documento HTML<p>
</body>
</html>
```

Debido a que las modificaciones realizadas en los archivos HTML no tienen influencia en el servidor, no es necesario que lo reiniciemos; con solo recargar la página deberemos ver de manera correcta las tildes.

Figura 17. Como hemos definido la codificación del documento, la tilde se muestra perfectamente.

DIFERENCIAS ENTRE C Y C++

El lenguaje **C** fue creado en 1972 como evolución del lenguaje **B**; mientras que **C++** fue creado a mediados de los años 80, con la intención de extender el lenguaje **C** con mecanismos que permitían el manejo de objetos. Podemos decir que **C** es el lenguaje original y **C++** su ampliación.

Manejo de peticiones POST

A continuación veremos cómo manejar las peticiones de tipo POST provenientes del cliente. Para esto, crearemos un formulario en la página de inicio, y luego procesaremos los parámetros en el servidor y los mostraremos en la página de salida.

Para comenzar, debemos modificar el archivo **inicio.html**, agregando un formulario de sugerencia:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Inicio</title>
  <style type="text/css">
    body{
      background-color: #F2F2F2;
      color: #444444;
      font: 12px Arial;
      margin: 0;
      padding: 0;
    }
    input,
    label{
      display: block;
      width: 340px;
    }
  </style>
</head>
<body>
  <h1>Página inicio.html</h1>

  <h2>Formulario de sugerencias</h2>

  <form action="/salida.html" method="post">
    <label for="nombre">Nombre:</label>
```

```
    <input type="text" name="nombre" />

        <label for="apellido">Apellido:</label>
        <input type="text" name="apellido" />

        <label for="correo">Correo electrónico:</label>
        <input type="text" name="correo" />

        <label for="sugerencia">Sugerencia:</label>
        <textarea name="sugerencia" rows="10" cols="40"></textarea>

        <input type="submit" name="enviar" value="enviar" />
    </form>
</body>
</html>
```

En el código anterior, primero agregamos estilos **css** para que la página sea más amigable; luego, creamos un formulario con cuatro campos: **nombre**, **apellido**, **correo** y **sugerencia**. Estas variables se enviarán mediante el método POST a la ruta **/salida.html**.

Figura 18. En **`inicio.html`** definimos un simple formulario de sugerencia para que el servidor procese los valores ingresados.

Lo siguiente que debemos hacer es preparar el archivo denominado **salida.html** para que reciba los parámetros que le enviaremos:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Salida</title>
    <style type="text/css">
        body{
            background-color: #F2F2F2;
            color: #444444;
            font: 12px Arial;
            margin: 0;
            padding: 0;
        }
        input,
        label{
            display: block;
            width: 340px;
        }
    </style>
</head>
<body>
    <h1>Página salida.html</h1>

    <h2>Sugerencia recibida</h2>

    <p><b>Tu nombre: </b> {nombre}, {apellido}</p>
    <p><b>Tu correo: </b> {correo}</p>
    <p><b>Tu sugerencia: </b> {sugerencia}</p>
    <br />
    <a href="/">Regresar al Inicio</a>
</body>
</html>
```

En el código que vimos anteriormente, al igual que para **inicio.html**, incorporamos estilos **css** para mejorar la presentación del ejemplo; luego, agregamos tres párrafos donde definimos los parámetros que esperamos recibir: **{nombre}**, **{apellido}**, **{correo}** y **{sugerencia}**. El contenido de estas etiquetas será reemplazado, después, con los valores que contengan las variables del formulario.

Ahora vamos a trabajar en el archivo **servidor.js**, que potenciará nuestro ejemplo. Escribimos lo siguiente:

```
var http        = require('http');
var url         = require('url');
var fs          = require('fs');
var querystring = require("querystring");

function iniciar(enrutar, peticion) {
   function arrancarServidor(request, response) {
      console.log("Se ha conectado un usuario.");

      var ruta = url.parse(request.url).pathname;
      //enrutar(ruta, peticion, response);

      if(ruta == '/')
         ruta = 'inicio.html';

      var html = fs.readFileSync('www/' + ruta);

      var archivo_accesos = fs.createWriteStream('accesos.txt',{'flags':'a'});
      archivo_accesos.write(ruta + '\n')

      if(request.method == 'POST') {
         var posData = null;
         request.setEncoding('utf8');

         request.on("data", function(data) {
            posData = querystring.parse(data);

            html = html.toString();
```

```
            html = html.replace('{nombre}', posData.nombre);
            html = html.replace('{apellido}', posData.apellido);
            html = html.replace('{correo}', posData.correo);
            html = html.replace('{sugerencia}', posData.sugerencia);

            response.writeHead(200,{"Content-Type":"text/html"});
            response.write(html);
            response.end();
        });
    }else{
        response.writeHead(200,{"Content-Type":"text/html"});
        response.write(html);
        response.end();
    }
}

http.createServer(arrancarServidor).listen(3000,'127.0.0.1');
console.log('Servidor web iniciado en http://127.0.0.1:3000/');
}

exports.iniciar = iniciar;
```

En el código anterior nos encargamos de realizar varios cambios, que detallamos a continuación:

- Incluimos el módulo **querystring** y lo asignamos a una variable con el mismo nombre. Este módulo nos permitirá obtener las variables que serán pasadas por POST.
- Dentro de la función **arrancarServidor()**, agregamos un condicional para verificar mediante el método **request.method** si se ha producido una solicitud del tipo POST. En caso de que esto ocurra, primero definimos una variable llamada **posData** que va a contener las variables provenientes del formulario y, como segundo paso, codificamos con **utf8** los datos provenientes de la solicitud.
- Creamos un **listener on()**, que no es más que un manejador que se ejecutará cada vez que ocurra el evento **data**, es decir, cuando se produzca el envío de datos.

- Utilizamos el método **querystring.parse()** para dejar de serializar la cadena que contiene las variables provenientes del formulario, y la asignamos a la variable **posData**.
- Usamos el método **toString()** para decodificar y devolver el código html del archivo **salida.html**. Una vez que tenemos el html, reemplazamos las etiquetas **{nombre}**, **{apellido}**, **{correo}** y **{sugerencia}** con los elementos correspondientes del objeto **posData**.
- Por último, devolvemos el HTML resultante al navegador.

Figura 19. Una vez que nuestro servidor creado con Node procesó la información, se muestra el contenido en `salida.html`.

RESUMEN

Creamos un sistema que maneja peticiones y rutas y que, además, es un servidor web. Esto quiere decir que con Node podemos desarrollar un sistema de manera integral sin tener un servidor externo como Apache. También aprendimos que Node dispone de módulos internos que contienen las herramientas para cada necesidad. Sin dudas, esto es una gran ventaja, ya que cada sistema es independiente y solo debe incluir los módulos que utilizará sin incorporar archivos innecesarios.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Es posible utilizar el patrón de diseño **MVC** en Node?
2. ¿Qué son los módulos internos de Node?
3. ¿Los módulos **HTTP** y **FS** pueden ser utilizados desde cualquier archivo de un sistema creado con Node?
4. ¿Qué determinan los métodos **writeHead()**, **write()** y **end()**?
5. ¿Cuál es el objetivo de exportar un módulo y cuál es la manera de hacerlo?
6. ¿Cómo es posible obtener la ruta definida en el navegador?
7. ¿Se puede crear un comportamiento para cada ruta? ¿Cómo?
8. ¿Node puede devolver solo archivos HTML o también de texto plano?
9. ¿Para qué se utiliza el módulo **querystring**?
10. Cuando se solicita una página ¿intervienen los métodos POST o GET?

EJERCICIOS PRÁCTICOS

1. Cree enlaces en cada página del ejemplo para navegar a través del sistema.
2. Acceda a una URL que no esté definida en el sistema de ejemplo y cree un mecanismo para resolver este error.
3. Guarde en el archivo **accesos.txt** la fecha y la hora de acceso y el navegador utilizado por el cliente.
4. Cree una nueva página que contenga un formulario de contacto.
5. Cree un archivo para guardar las sugerencias recibidas.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*08

Gestión de módulos en Node

Para comenzar, aprenderemos cómo se estructuran los módulos en Node y cómo es el método de carga. Crearemos un módulo paso a paso y definiremos su funcionalidad de manera estándar. Por último, conoceremos el gestor de paquetes de Node y lo utilizaremos para crear un módulo que publicaremos en el repositorio público de npm.

- **Módulos 246**
  - Módulos propios de Node..... 247
  - Módulo File 247
  - Incluir módulos desde el directorio node_modules..... 248
  - Cacheo de módulos 248
- **Estructuración y creación de módulos..... 249**
- **Gestor de paquetes npm..... 258**
  - Comandos de npm..... 258
  - README.md 265
  - package.json 267
- **Resumen..... 273**
- **Actividades..... 274**

Módulos

Node posee un sistema muy simple y eficaz para cargar módulos, donde los archivos hacen referencia a los módulos mediante su nombre. Por ejemplo: en nuestro sistema vamos a utilizar un módulo para calcular el área de un cubo; para esto, primero creamos el archivo **cubo.js** que será nuestro módulo.

```
function area(lado) {
   return lado * lado;
};

function longitud(lado) {
   return lado * 4;
};

exports.area     = area;
exports.longitud = longitud;
```

En el código anterior, definimos dos funciones que calculan el área y la longitud para un valor establecido. Luego, exportamos las funciones para que puedan ser utilizadas desde cualquier lugar.

A continuación, necesitamos crear el archivo que utilizará nuestro módulo. Generamos un archivo con el nombre **app.js**, en el debemos escribir lo que sigue:

```
var cubo = require('./cubo');

console.log( 'El área de un cubo de 5 cm de lado es: ' + cubo.area(5));

console.log( 'La longitud de un cubo de 5 cm de lado es: ' + cubo.longitud(5));
```

En el código anterior, mediante **require()** incluimos el archivo **cubo.js**, que de ahora en adelante se presenta como un módulo, del cual utilizamos los métodos **area()** y **longitud()**.

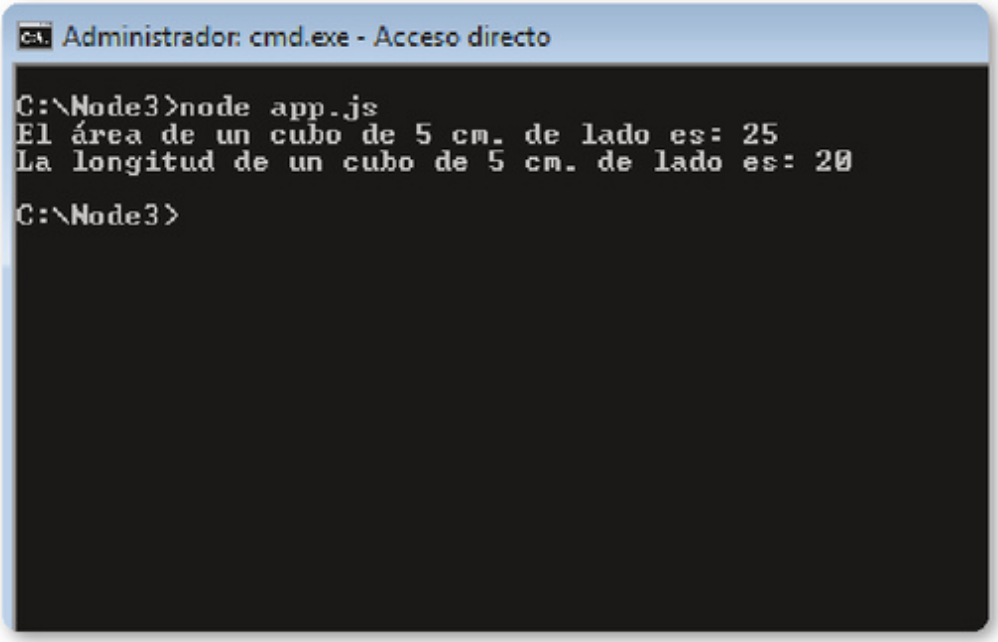


Figura 1. Al ejecutar el archivo **app.js** vemos el resultado de los métodos exportados de **cubo.js**.

Módulos propios de Node

Node contiene algunos módulos propios, que están compilados en binario y se encuentran dentro del mismo directorio de instalación de Node, en la carpeta **lib/**. Estos módulos tienen una prioridad mayor de carga si son incluidos con **require()**; es decir, si en nuestro sistema definimos **require('http')**, Node devolverá el módulo **HTTP**, incluso si hay un archivo con este nombre.

Módulo File

Cuando intentamos cargar un módulo a través de **require()** y el archivo con el nombre exacto no es encontrado, Node intentará primero cargar el archivo con la extensión **.js**; luego, el archivo con la extensión **.json** y por último, el archivo con la extensión **.node**. En este caso, **.js** es interpretado como un archivo de texto JavaScript, **.json** como un archivo con formato JSON, y **.node** como una extensión compilada de un módulo que es cargado con **dlopen**.

Por otro lado, si especificamos el prefijo **/** en el nombre del módulo que queremos incluir, nos estaremos refiriendo a una ruta absoluta. Es decir, **require('/proyectos/node/modulo.js')** cargará el archivo en **/proyectos/**

node/modulo. En cambio, si anteponemos el prefijo **./** indicaremos que el módulo se encuentra en una ruta relativa desde donde fue incluido. Por último, si omitimos tanto **/** como **./**, indicaremos que se trata de un módulo básico o que se cargará desde el directorio **node_modules**.

Incluir módulos desde el directorio node_modules

Si el nombre del módulo que necesitamos incluir a través de **require()** no es un módulo propio ni está precedido de los caracteres **/**, **./** o **../** (este último indica acceso al directorio en un nivel superior), Node iniciará el proceso de carga desde el directorio principal del módulo actual añadiendo **/node_modules**. En caso de que el módulo no sea encontrado, Node se dirigirá a un directorio superior al actual, y así sucesivamente hasta llegar a la raíz.

Como ejemplo, supongamos que en el archivo **/proyectos/node/ejemplo1/app.js** incluimos un módulo con **require('modulo.js')**. Entonces, Node buscará este módulo en las siguientes ubicaciones:

- **/proyectos/node/ejemplo1/node_modules/modulo.js**
- **/proyectos/node/node_modules/modulo.js**
- **/proyectos/node_modules/modulo.js**
- **/node_modules/modulo.js**

Como podemos ver, Node permite desarrollar programas de manera que estos encuentren sus dependencias sin que se generen conflictos por las rutas de los módulos.

Cacheo de módulos

Node provee un sistema de cacheo para los módulos, es decir que, después de que un módulo es cargado por primera vez, es almacenado en caché y, entonces, devolverá el mismo objeto en futuras llamadas mediante el método **require()**. Por ejemplo, diferentes llamadas a **require('modulo')** no serán ejecutadas múltiples veces. Si quisiéramos este comportamiento, deberíamos exportar las funciones del módulo y ejecutarlas tantas veces como lo necesitemos.

Estructuración y creación de módulos

Vamos a conocer cómo se estructuran los módulos que se utilizan en Node, lo que no es un estándar sino una convención. Como información relevante, debemos tener en cuenta que la siguiente estructura está presente en aproximadamente el 80% de los módulos disponibles en Node, por lo que es una buena idea basarnos en ella para nuestros ejemplos:

```
Nombre del módulo
        |_ lib/
        | |_ nombre del módulo.js
        |
        |_ bin/
        |  |_ archivo binario
        |
        |_ examples/
        |  |_ example.js
        |
        |_ test/
        |  |_ test.js
        |
        |_ index.js
        |_ package.json
        |_ README.md
```

RAZONES PARA UTILIZAR HTML5

Actualmente, HTML se encuentra en la versión 5 y ofrece muchos beneficios: soporte para audio y video, un código más limpio y marcado semánticamente, soporte para el uso de bases de datos locales y compatibilidad con los navegadores actuales. Algunos navegadores antiguos (como IE6), junto con los navegadores móviles, se han adaptado a esta tecnología.

TODA LA LÓGICA DEL MÓDULO LA ENCONTRAMOS EN EL ARCHIVO JS, DENTRO DE LIB/

Al analizar la estructura, vemos que en el directorio **lib/** hay un archivo **js** con el nombre del módulo, donde se encontrará toda su lógica. Por supuesto, es posible tener más archivos, que serán necesarios para el funcionamiento del módulo.

En el directorio **bin/** encontramos un archivo binario que permite la ejecución del módulo desde la línea de comandos sin utilizar Node. Por ahora no vamos a utilizarlo.

Las carpetas **examples/** y **test/** son autodescriptivas y contienen un ejemplo de uso y un test del módulo, respectivamente.

En el directorio principal tenemos el archivo **index.js**, que servirá como punto de acceso al módulo; es decir, cuando incluyamos el módulo en una aplicación se invocará a este archivo, y es aquí donde vamos a exportar la funcionalidad del módulo propiamente dicho (lo entenderemos mejor cuando desarrollemos un ejemplo).

Por último, tenemos dos archivos más: **package.json** y **README.md**; por el momento los vamos a crear sin contenido, ya que en el siguiente tema explicaremos para qué sirven y cómo utilizarlos.

Ahora desarrollaremos un ejemplo que nos ayudará a entender mejor cómo funcionan los módulos en Node. Debemos considerar que el objetivo será traducir los días de la semana y los meses del año a los idiomas inglés y portugués.

En primer lugar debemos crear la estructura necesaria. Para realizar esta tarea, nos situamos en cualquier directorio (por ejemplo, en **C:**) y generamos una carpeta denominada **Node5**. Luego, dentro de ésta, creamos otra con el nombre **traductor**.

POSTMAN

Google Chrome posee una extensión llamada **Postman** que sirve para auditar APIS. Además, tiene soporte de autenticación, posee una interfaz compacta, permite visualizar las respuestas en formato JSON y XML, posee una vista previa y un historial, y es posible editar los parámetros. Podemos conocer más visitando la página que está en la dirección web **https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm**.

A continuación, generamos un archivo llamado **index.js** y las carpetas **bin/**, **examples/**, **lib/** y **test/**. Por último, dentro de **lib/**, creamos un archivo llamado **traductor.js**. Nuestro módulo deberá quedar con el código que mostramos a continuación:

```
traductor
        |_ lib/
        | |_ traductor.js
        |
        |_ bin/
        |
        |_ examples/
        |
        |_ test/
        |
        |_ index.js
        |_ package.json
        |_ README.md
```

Una vez que tenemos la estructura armada vamos a darle funcionalidad. Para realizar esta tarea debemos abrir el archivo **traductor.js** y escribir el código que sigue:

```
var salida = null;

var terminos = {
   "dias": {
      "en": {
         "lunes"    : "Monday",
         "martes"   : "Tuesday",
         "miercoles" : "Wednesday",
         "jueves"   : "Thursday",
         "viernes"  : "Friday",
         "sabado"   : "Saturday",
         "domingo"  : "Sunday",
      },
```

```
    "br": {
      "lunes"    : "segunda-feira",
      "martes"   : "terça-feira",
      "miercoles" : "quarta-feira",
      "jueves"   : "quinta-feira",
      "viernes"  : "sexta-feira",
      "sabado"   : "sábado",
      "domingo"  : "domingo",
    }
  },
  "meses": {
    "en": {
      "enero"    : "January",
      "febrero"  : "February",
      "marzo"    : "March",
      "abril"    : "April",
      "mayo"     : "May",
      "junio"    : "June",
      "julio"    : "July",
      "agosto"   : "August",
      "septiembre": "September",
      "octubre"  : "October",
      "noviembre" : "November",
      "diciembre" : "December"
    },
    "br": {
      "enero"    : "Janeiro",
      "febrero"  : "Fevereiro",
      "marzo"    : "Março",
      "abril"    : "Abril",
      "mayo"     : "Maio",
      "junio"    : "Junho",
      "julio"    : "Julho",
      "agosto"   : "Agosto",
      "septiembre": "Setembro",
      "octubre"  : "Outubro",
      "noviembre" : "Novembro",
```

```
            "diciembre" : "Dezembro"
        }
    }
};

var traducir_dia = function (dia, idioma) {
    dia    = dia.toLowerCase();
    idioma = idioma.toLowerCase();

    if (terminos['dias'][idioma] == null)
        salida = 'El Idioma: "' + idioma + '" No existe.';
    else{
        if (terminos['dias'][idioma][dia] == null)
            salida = 'El Día: "' + dia  + '" No existe para el idioma "' + idioma + '"';
        else
            salida = dia + ' en ' + idioma +' es: ' + terminos['dias'][idioma][dia];
    }
    return salida;
};

var traducir_mes = function (mes, idioma) {
    mes    = mes.toLowerCase();
    idioma = idioma.toLowerCase();

    if (terminos['meses'][idioma] == null)
        salida = 'El Idioma: "' + idioma + '" No existe.';
    else{
        if (terminos['meses'][idioma][mes] == null)
```

DATOS AL SOLICITAR UN JSON

Es importante saber que, cuando se devuelve un objeto JSON desde el servidor, es necesario cambiar el tipo de contenido de la respuesta. Por ejemplo, desde PHP se debe agregar **Content-Type: application/json**. De esta manera, el navegador entenderá de qué se trata la respuesta.

```
            salida = 'El Mes: \'' + mes  + '\' No existe para el idioma \'' + idioma + '\'';
        else

            salida =  mes + ' en ' + idioma +' es: ' + terminos['meses'][idioma][mes];
    }
    return salida;
};

exports.traducir_dia = traducir_dia;
exports.traducir_mes = traducir_mes;
```

En el código anterior, primero definimos una variable llamada **terminos** –que contiene un JSON con las claves **dias** y **meses**– y, dentro de cada una, nos encargamos de especificar el idioma que contendrá la traducción correspondiente a cada palabra.

LA FUNCIÓN TRADUCIR_DIA() TRABAJARÁ CON DOS PARÁMETROS: DÍA E IDIOMA

Luego definimos la función **traducir_dia()** que recibirá dos parámetros: el **día** y el **idioma** al cual necesitamos traducir.

En las siguientes líneas, dentro de la función, nos encargamos de convertir a minúsculas los parámetros para asegurarnos de que coincidan con las claves de la variable **terminos**.

A continuación verificamos el idioma dentro de la clave **dias** del JSON. Si no existe se devuelve un mensaje y, si existe, entonces pasa a verificar la existencia del día. Como en el caso anterior, devuelve un mensaje si no se encuentra el día y, si se encuentra, muestra otro con el día en el idioma solicitado. Como vemos, esta función es muy sencilla: simplemente verificamos la existencia de las claves y devolvemos el valor correspondiente a cada solicitud.

Luego definimos una nueva función, prácticamente igual a **traducir_dia()**, a la que llamamos **traducir_mes()**. En ella tendremos que hacer la traducción del mes de manera similar a la anterior, pero en esta ocasión cambiamos las claves que necesitamos traducir.

Como podemos suponer, tener dos funciones que realizan casi el mismo procedimiento no es una buena idea pero, en este caso, lo hacemos así solo para poder demostrar el uso de varias funciones

en un mismo módulo. Por último, exportamos las dos funciones para que puedan ser accedidas desde otro lugar de nuestro módulo.

Un aspecto importante es que tanto la variable **salida** como **terminos** quedan encapsuladas dentro del módulo, ya que no hemos exportado ninguna de ellas. Esto nos permite definir variables y métodos que son únicamente accesibles desde dentro del módulo.

Con las funciones definidas, debemos invocarlas desde el archivo principal. Para esto abrimos **index.js** y escribimos lo siguiente:

```
var traductor = require('./lib/traductor');

var dia = traductor.traducir_dia("Lunes", "BR");
var mes = traductor.traducir_mes("enero", "en");

console.log(dia);
console.log(mes);
```

En el código anterior, primero incluimos el archivo **traductor.js** y luego llamamos a los métodos para traducir el día y el mes en diferentes idiomas; por último, imprimimos en consola cada resultado.

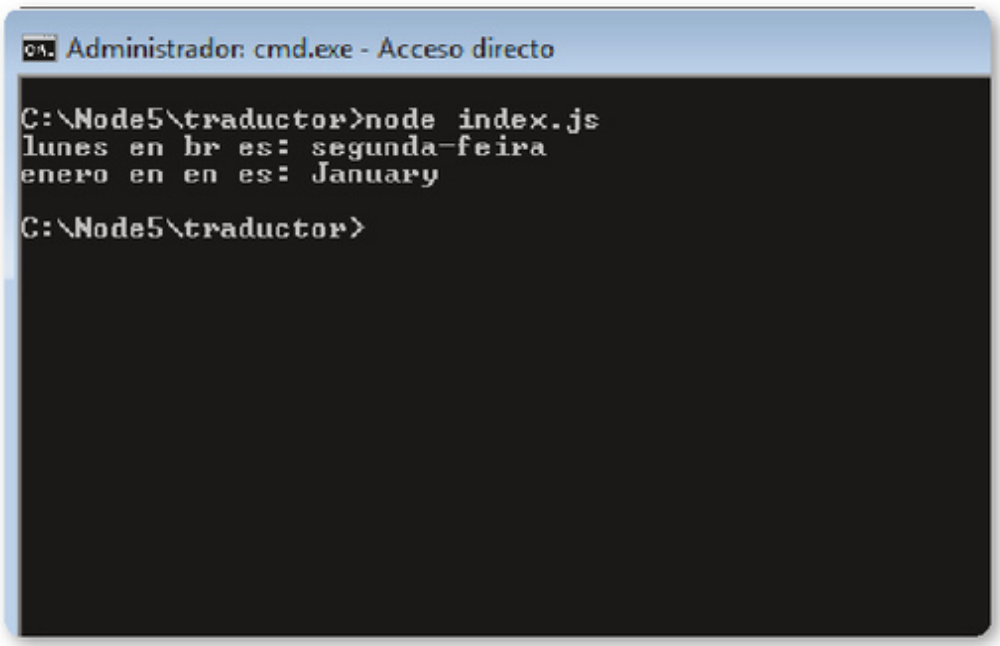


Figura 2. Para probar el traductor nos situamos en el directorio **Node5/traductor/** y ejecutamos el comando **node index.js**.

Hasta ahora nos encargamos de realizar la creación de un módulo, pero todavía no es lo suficientemente robusto como para funcionar de manera aislada en cualquier aplicación.

En el archivo **index.js** incluimos el módulo para probar la funcionalidad, pero la función de este archivo es otra. Lo que debemos hacer es crear un mecanismo de entrada al módulo de manera que esté accesible desde cualquier lugar. Para hacerlo reemplazamos todo el contenido de **index.js** con el siguiente código:

```
module.exports = require('./lib/traductor');
```

De esta manera, exportamos el archivo **traductor.js** que habíamos creado en la carpeta **lib/**. De ahora en más, cuando necesitemos incluir el módulo en una aplicación, simplemente hacemos un **require** de **traductor**.

Para verificar que el módulo funciona como tal, dentro del directorio de nuestro ejemplo (**Node5**) creamos una carpeta llamada **node_modules** y movemos todo el módulo allí. La estructura deberá quedar tal como presentamos en el siguiente bloque:

```
Node5
  |_ node_modules/
     |_ traductor/
                |_ lib/
                | |_ traductor.js
                |
                |_ bin/
                |
                |_ examples/
                |
                |_ test/
                |
                |_ index.js
                |_ package.json
                |_ README.md
```

En la raíz del proyecto, creamos un archivo llamado **app.js** y dentro de éste agregamos el siguiente código:

```
var traductor = require('traductor');

var dia = traductor.traducir_dia("Lunes", "BR");
var mes = traductor.traducir_mes("enero", "en");

console.log(dia);
console.log(mes);
```

Ahora **app.js** es el archivo principal de la aplicación. En la primera línea incluimos el módulo **traductor**, y como no le hemos especificado los caracteres **'/'**, **'./'** o **'../'**, Node inicia el proceso de carga desde el directorio **node_modules/**, que a su vez ejecuta el archivo **index.js**.

Continuando con el archivo **app.js**, en las siguientes líneas ejecutamos los métodos **traducir_dia()** y **traducir_mes()**, respectivamente, asignamos los resultados en las variables, y las mostramos en la consola.

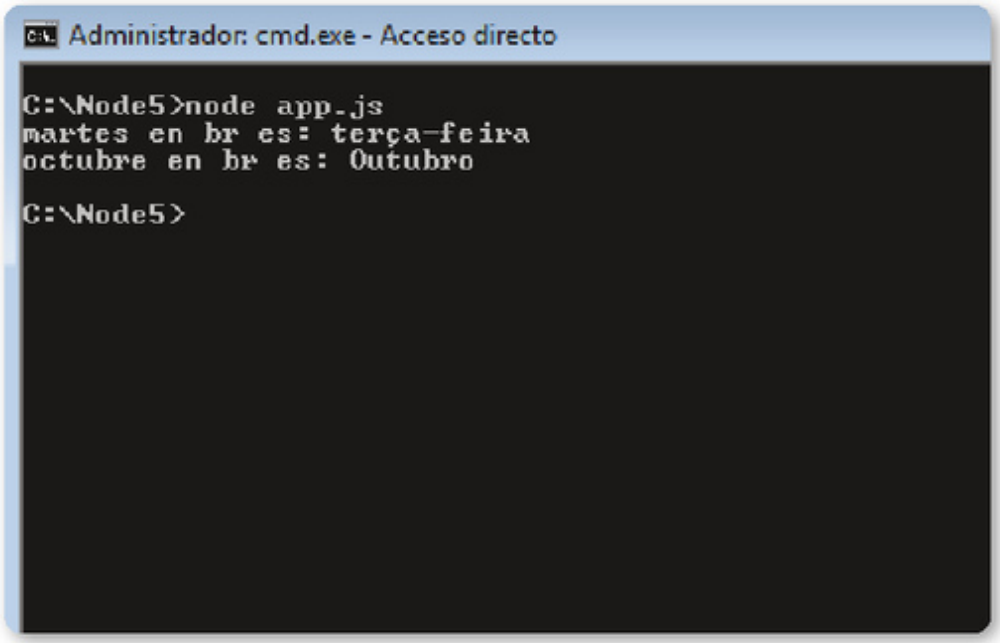


Figura 3. Ahora **app.js** utiliza los métodos del traductor simplemente con incluir el módulo.

Con estas modificaciones, creamos un módulo que puede ser usado en cualquier aplicación al copiar el contenido de la carpeta **traductor/** dentro de la carpeta **node_modules/** e incluir el módulo. Luego, haremos que el módulo esté disponible para cualquier aplicación.

Gestor de paquetes npm

Una de las utilidades de Node es el gestor de paquetes **npm**, llamado **Node Package Manager** (que en realidad significa **Non-Parametric Mapping**). Este gestor de paquetes es la principal herramienta para manejar dependencias o módulos en Node, porque provee una manera sencilla para distribuir código a través de un repositorio global.

Como npm viene integrado a Node desde la versión **0.6.3**, es probable que ya lo tengamos instalado. Para verificar que efectivamente sea así, abrimos una ventana de comandos y escribimos lo siguiente:

```
npm version
```

En la **Figura 4** podemos la información que nos devuelve la ejecución del comando anterior.

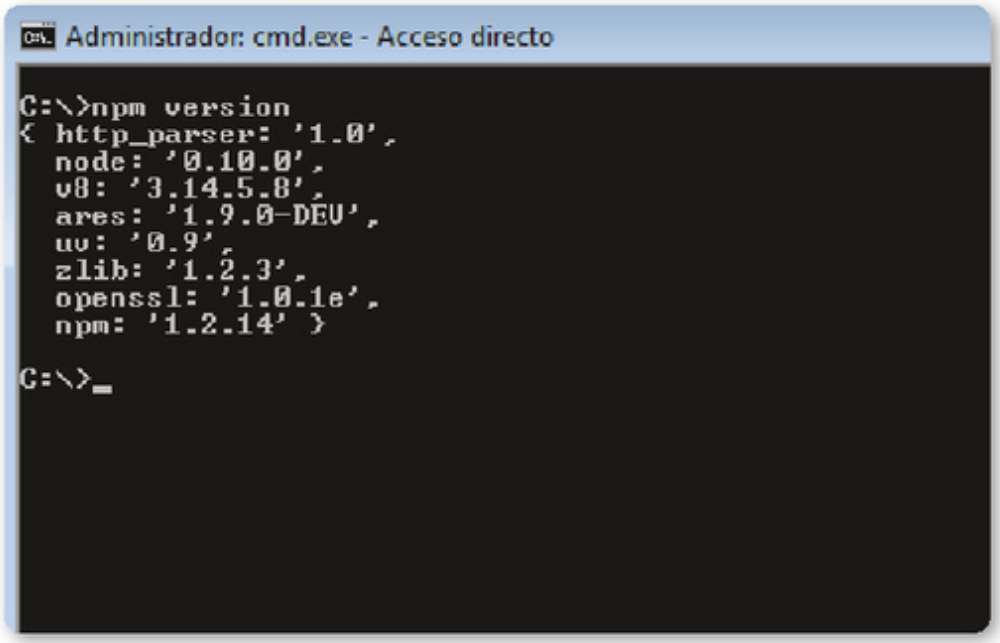


Figura 4. Al ejecutar el comando **`npm version`** vemos las versiones de Node y del gestor de paquetes que tenemos instaladas.

Comandos de npm

El gestor npm es muy fácil de usar y dispone de varios comandos; los más comunes permiten instalar, buscar y publicar paquetes.

A continuación, en la **Tabla 1**, conoceremos los más importantes para luego utilizarlos en nuestro módulo.

COMANDOS DE NPM

▼ COMANDO	▼ DESCRIPCIÓN
npm adduser	Crea o verifica un usuario en el registro de npm y guarda las credenciales en un archivo **.npmrc**. Este comando solicita nombre de usuario, clave y correo electrónico.
npm init	Este comando es quizás el más importante de todos, porque nos permite crear, modificar y generar un archivo **package.json**, que contiene toda la información de un módulo en particular.
npm search <módulo| palabra clave>	Permite buscar en el registro de npm algún módulo coincidente con el nombre o una palabra clave.
npm info <módulo>	Muestra información de un módulo en formato json.
npm ls	Lista todos los paquetes instalados en un directorio en formato de árbol.
npm outdated	Verifica si existen versiones nuevas de los módulos instalados en una aplicación.
npm install <módulo>	Permite instalar cualquier módulo y sus dependencias desde el repositorio de npm. Con el parámetro **-g** instala el módulo de manera global.
npm publish <módulo>	Publica un módulo en el repositorio público de npm. El módulo debe contener el archivo **package.json**.
npm uninstall <módulo>	Desinstala el módulo indicado.
npm unpublish <módulo>	Elimina el módulo del repositorio público de npm.
npm home <módulo>	Muestra el sitio web del módulo.
npm help folders	Presenta la explicación de cómo funciona la instalación de módulos.
npm view <módulo> [<version>] [<campo>]	Brinda información del módulo especificado; además, es posible especificar la versión y un campo en particular.
npm help npm	Explica cómo funciona npm.
npm whoami	Muestra la configuración del usuario en caso de que se haya agregado.

Tabla 1. Algunos de los principales comandos de **npm** que vamos a utilizar en los módulos.

Si bien recién estamos conociendo npm, ya podemos darnos cuenta de que es una herramienta muy útil para instalar módulos y publicar los propios sin mayor dificultad.

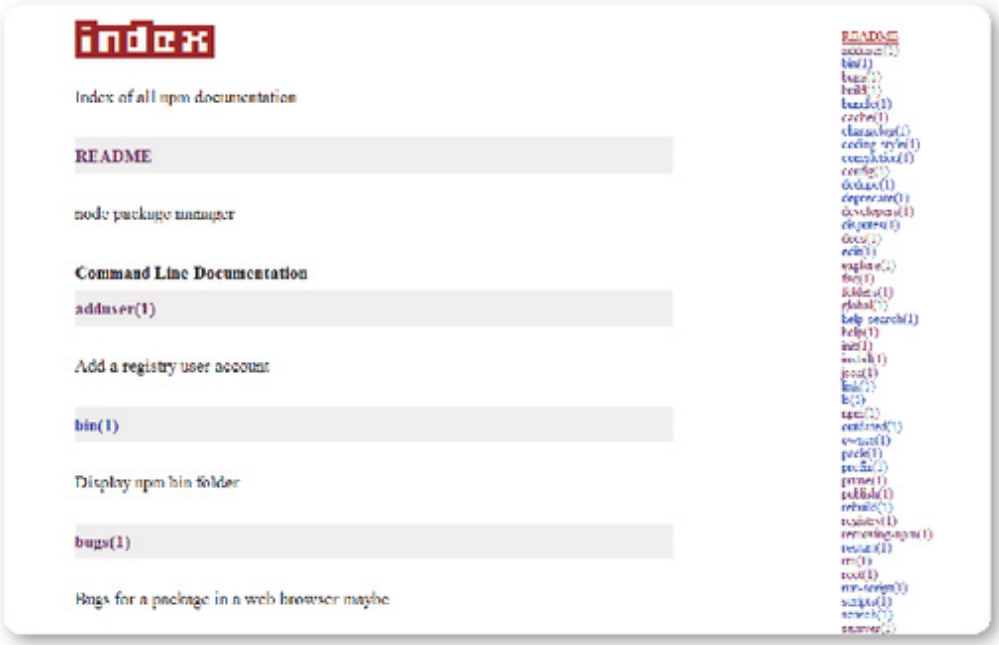


Figura 5. Para ver la lista completa de los comandos de **npm** podemos acceder al siguiente enlace: **`https://npmjs.org/doc`**.

Para utilizar npm en nuestro ejemplo, debemos instalar el módulo **colors**, que nos servirá para colorear la salida en consola y hacerla más legible. En primer lugar, abrimos una consola y nos situamos en la carpeta del proyecto (**Node5/**); luego ejecutamos el comando **npm install colors**. Este procedimiento instalará el módulo dentro de la carpeta **node_modules/**.

Vemos que el módulo se ha instalado a la misma altura que **traductor/**. En este punto debemos considerar que esto quiere decir que estará disponible para cualquier lugar de la aplicación.

CONNECT

Connect es un conjunto de herramientas para el desarrollo de aplicaciones en Node. Es muy útil porque posee componentes que manejan un sistema de acceso, compresión, autenticación y un parseador de JSON y, además, contiene un mecanismo para manejar sesiones, directorios, favicon y errores, entre otros. Podemos conocer más en el siguiente enlace: **http://senchalabs.org/connect**.

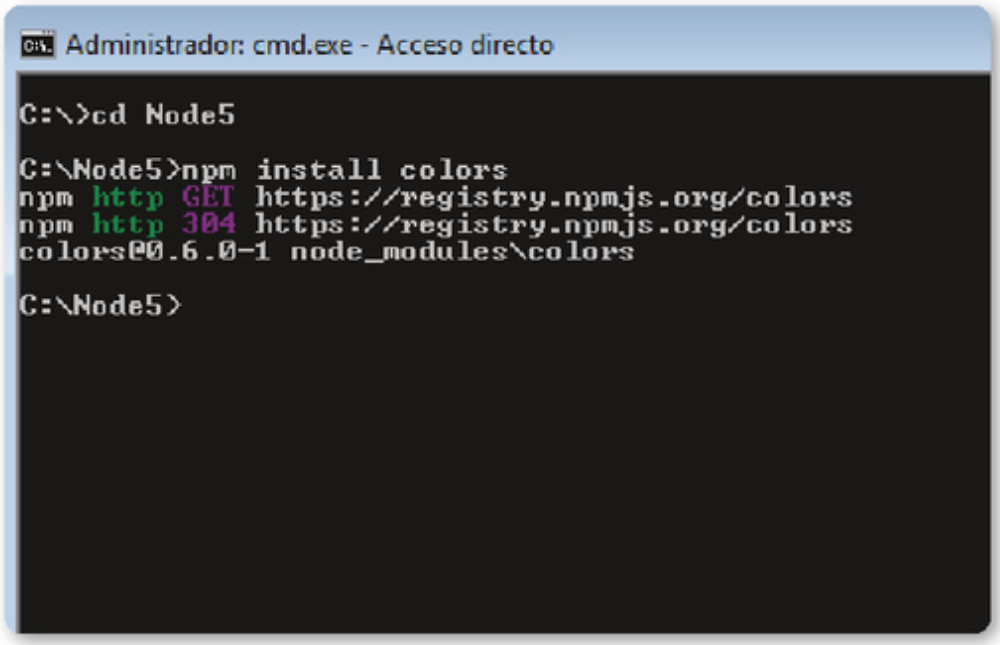


Figura 6. Con solo ejecutar el comando **`npm install colors`** en la consola, tenemos disponible el módulo en el directorio actual.

Para utilizar **colors** en nuestro módulo, debemos incluirlo en el archivo **lib/traductor.js** y, luego, agregar los colores a las diferentes salidas que imprimimos en la consola. El archivo **traductor.js** quedará de la siguiente manera:

```
var colors = require('colors');
var salida = null;

var terminos = {
  "dias": {
    "en": {
      "lunes"    : "Monday",
      "martes"   : "Tuesday",
      "miercoles" : "Wednesday",
      "jueves"   : "Thursday",
      "viernes"  : "Friday",
      "sabado"   : "Saturday",
      "domingo"  : "Sunday",
    },
    "br": {
```

```
        "lunes"    : "segunda-feira",
        "martes"   : "terça-feira",
        "miercoles" : "quarta-feira",
        "jueves"   : "quinta-feira",
        "viernes"  : "sexta-feira",
        "sabado"   : "sábado",
        "domingo"  : "domingo",
      }
    },
    "meses": {
      "en": {
        "enero"    : "January",
        "febrero"  : "February",
        "marzo"    : "March",
        "abril"    : "April",
        "mayo"     : "May",
        "junio"    : "June",
        "julio"    : "July",
        "agosto"   : "August",
        "septiembre": "September",
        "octubre"  : "October",
        "noviembre" : "November",
        "diciembre" : "December"
      },
      "br": {
        "enero"    : "Janeiro",
        "febrero"  : "Fevereiro",
        "marzo"    : "Março",
```

NPM

Es interesante tener en cuenta que el repositorio público de módulos **NPM** (Node Packaged Modules) posee una cantidad impresionante de módulos disponibles para utilizar: hasta el momento se encuentran registrados más de treinta mil. Podemos buscar, instalar o publicar módulos en cualquier momento, mediante el acceso al siguiente enlace: **https://npmjs.org**.

```
            "abril"    : "Abril",
            "mayo"     : "Maio",
            "junio"    : "Junho",
            "julio"    : "Julho",
            "agosto"   : "Agosto",
            "septiembre": "Setembro",
            "octubre"  : "Outubro",
            "noviembre" : "Novembro",
            "diciembre" : "Dezembro"
        }
    }
};

var traducir_dia = function (dia, idioma) {
    dia    = dia.toLowerCase();
    idioma = idioma.toLowerCase();

    if (terminos['dias'][idioma] == null)
        salida = ('El Idioma: \'' + idioma + '\' No existe.').red;
    else{
        if (terminos['dias'][idioma][dia] == null)
            salida = ('El Día: \'' + dia  + '\' No existe para el idioma \'' + idioma + '\'').
red;
        else
            salida = dia + ' en ' + idioma.bold +' es: ' + terminos['dias'][idioma][dia].
magenta;
    }
```

NODESTER

Nodester es un servicio de alojamiento para aplicaciones creadas en Node. Es de código abierto (open source) y provee un servicio llamado **plataforma como servicio**, desarrollado también con Node a través de la API **RESTful**. Para obtener más información podemos acceder a la página que se encuentra en la dirección web **http://nodester.com**.

```
    return salida;
};

var traducir_mes = function (mes, idioma) {
    mes    = mes.toLowerCase();
    idioma = idioma.toLowerCase();

    if (terminos['meses'][idioma] == null)
        salida = ('El Idioma: \'' + idioma + '\' No existe.').red;
    else{
        if (terminos['meses'][idioma][mes] == null)
            salida = ('El Mes: \'' + mes  + '\' No existe para el idioma \'' + idioma +
'\'').red;
        else
            salida =  mes + ' en ' + idioma.bold +' es: ' + terminos['meses'][idioma]
[mes].magenta;
    }
    return salida;
};

exports.traducir_dia = traducir_dia;
exports.traducir_mes = traducir_mes;
```

En el código anterior definimos que los errores se muestren en color rojo, el idioma en negrita y los días y meses, en magenta. Para verificar los cambios, nos situamos en el directorio de nuestro ejemplo y ejecutamos **node app.js**.

NODE Y LOS MÚLTIPLES PROCESADORES

Debemos tener en cuenta que Node no necesita la existencia de múltiples procesadores para escalar, sin embargo, es posible crear nuevos hilos de ejecución que funcionarán en paralelo, haciendo que la ejecución de tareas sea más eficiente. También se puede utilizar el módulo **cluster** para balancear la carga de las conexiones entrantes a través de múltiples procesos.

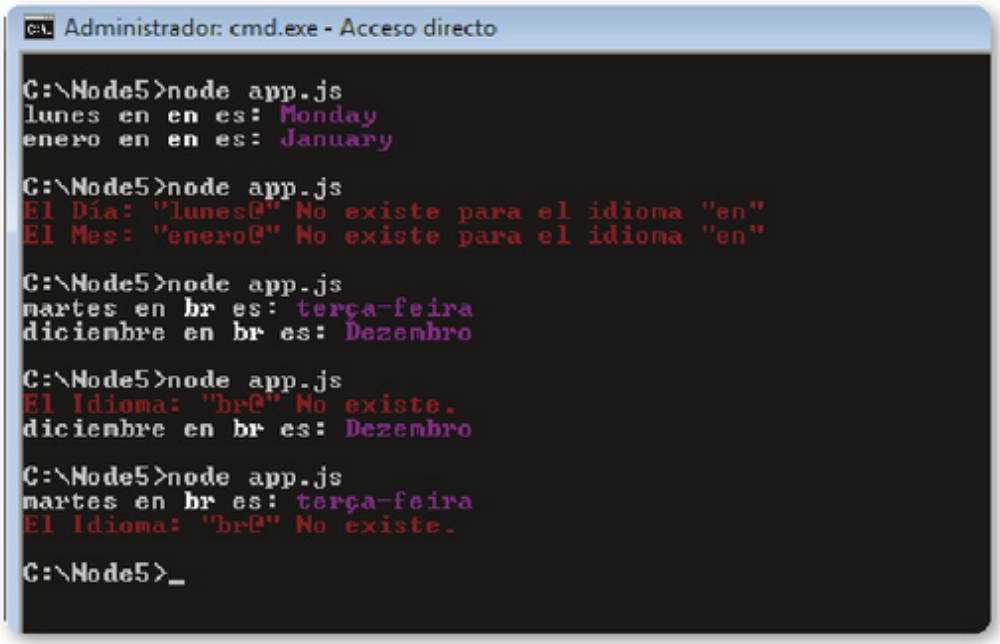


Figura 7. Con el módulo **colors** instalado podemos mostrar mensajes con color en la consola; esto genera una presentación más amigable.

En este punto ya tenemos instalado un módulo del cual depende el nuestro para funcionar, es decir que **traductor** necesitará de **colors** para mostrar los mensajes en la consola.

Siguiendo con nuestro ejemplo, llegó el momento de trabajar con los archivos **README.md** y **package.json**.

README.md

El archivo **README.md** es de suma importancia, porque constituye la presentación de nuestro módulo ante el mundo. Esto significa que debe contener toda la información que creamos relevante acerca del módulo. En el archivo **README.md** de nuestro ejemplo escribimos lo siguiente:

```
# Traductor

## Este módulo es útil para traducir días y meses del idioma español al inglés y portugués.

## Instalación
```

```
    npm install traductor

## Uso

```js
  var traductor = require('traductor');

  var dia = traductor.traducir_dia("Martes", "en");
  var mes = traductor.traducir_mes("diciembre", "br");

  console.log(dia);
  console.log(mes);
```

Dependencias

  colors

Licencia

  BDS
```

A continuación, mencionaremos algunos consejos importantes para escribir correctamente el archivo **README.md**:

- Debe contener un título y una breve descripción del módulo.
- Se debe indicar el comando de instalación.
- Se debe incluir un ejemplo de uso, si es aplicable.
- Se deben detallar los colaboradores, en caso de que existan.
- Se debe indicar el tipo de licencia.
- El archivo debe llamarse **README.md**, no **ReadMe.md** ni **README**.

Hasta este momento hemos preparado nuestro módulo para que tenga una presentación cuando lo compartamos en el directorio público de npm, pero aún no hemos realizado la definición de cómo se compone el módulo. Para esto, necesitamos editar el archivo **package.json**. A continuación veremos cómo hacerlo.

Figura 8. En **`http://blog.nodejitsu.com/how-to-write-a-readme`** podemos ver un artículo acerca de cómo escribir un **Readme**.

package.json

El archivo **package.json** es imprescindible para npm porque en él se definen todas las características propias de un módulo.

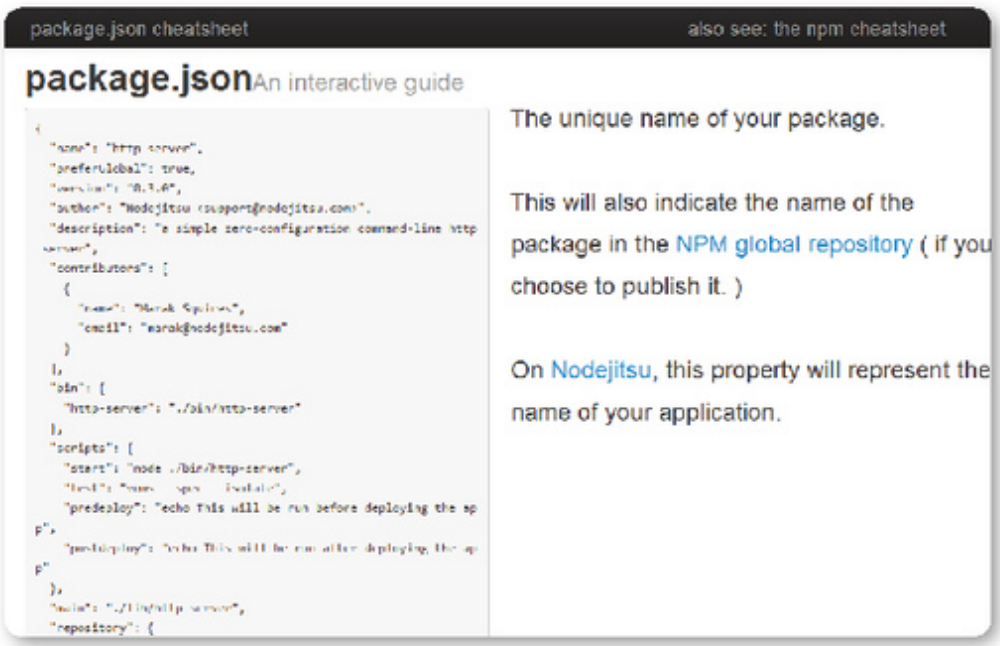


Figura 9. El sitio **`http://package.json.nodejitsu.com`** posee una guía que muestra la estructura del archivo **`package.json`**.

EL COMANDO NPM INIT PERMITE AGREGAR LA INFORMACIÓN DEL MÓDULO AL ARCHIVO

Para agregar la información del módulo en este archivo, abrimos una consola y nos situamos dentro del directorio del módulo (es decir en **Node5\node_modules\traductor**) y posteriormente ejecutamos el comando **npm init**.

Debemos considerar que una vez que hayamos ejecutado el comando mencionado veremos una serie de preguntas que debemos ir completando para que se pueda completar la configuración del archivo **json**. La utilidad nos ofrece un valor sugerido para algunos parámetros; en caso de queramos usarlo, solo debemos pulsar la tecla **ENTER**.

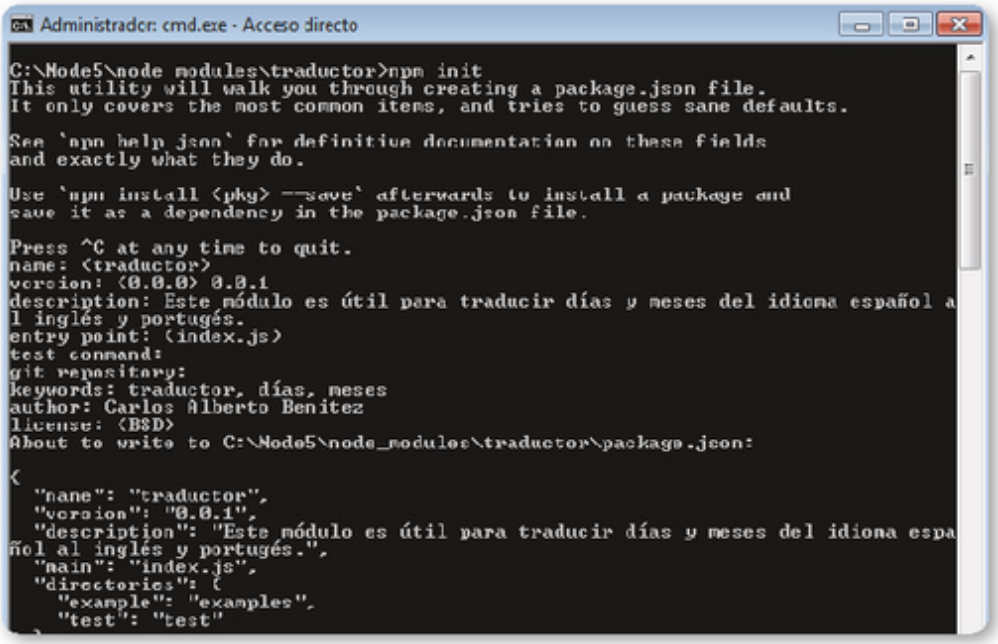


Figura 10. Al momento de ejecutar el comando **npm init** debemos ingresar los parámetros para crear **package.json**.

COMUNIDAD NODE

Como sabemos, Node posee una comunidad muy grande de desarrolladores y entusiastas repartidos por todo el mundo. En la página oficial de este sistema hay una sección donde podemos encontrar documentación, listas de correos, conferencias y canales de IRC, entre otros. Podemos ser parte de ella accediendo al siguiente enlace: **http://nodejs.org/community**.

Tengamos en cuenta que el archivo **package.json** debería tener un contenido similar al que presentamos a continuación:

```
{
  "name": "traductor",
  "version": "0.0.1",
  "description": "Este módulo es útil para traducir días y meses del idioma español al inglés y portugés.",
  "main": "index.js",
  "directories": {
    "example": "examples",
    "test": "test"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": "",
  "keywords": [
    "traductor",
    "días",
    "meses"
  ],
  "author": "Carlos Alberto Benitez",
  "license": "BSD",
  "readmeFilename": "README.md"
}
```

En este momento ya tenemos el módulo casi listo para poder publicarlo en el repositorio de npm, solo nos falta agregar el módulo **colors** como dependencia. Debemos editar el archivo **package.json** y agregar lo siguiente, justo antes de la clave **author**:

```
"dependencies": {
  "colors": "*"
},
```

Recordemos que el archivo **package.json** debería verse parecido a lo que presentamos en el siguiente código:

```
{
  "name": "traductor",
  "version": "0.0.1",
  "description": "Este módulo es útil para traducir días y meses del idioma español al inglés y portugés.",

  "main": "index.js",
  "directories": {
    "example": "examples",
    "test": "test"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": "",
  "keywords": [
    "traductor",
    "días",
    "meses"
  ],
  "dependencies": {
    "colors": "*"
  },
  "author": "Carlos Alberto Benitez",
  "license": "BSD",
  "readmeFilename": "README.md"
}
```

En este punto estamos listos para realizar la publicación de nuestro módulo en el repositorio público correspondiente. Lo primero que debemos hacer es crear nuestro usuario, ejecutando en la consola el comando que mostramos a continuación:

```
npm adduser
```

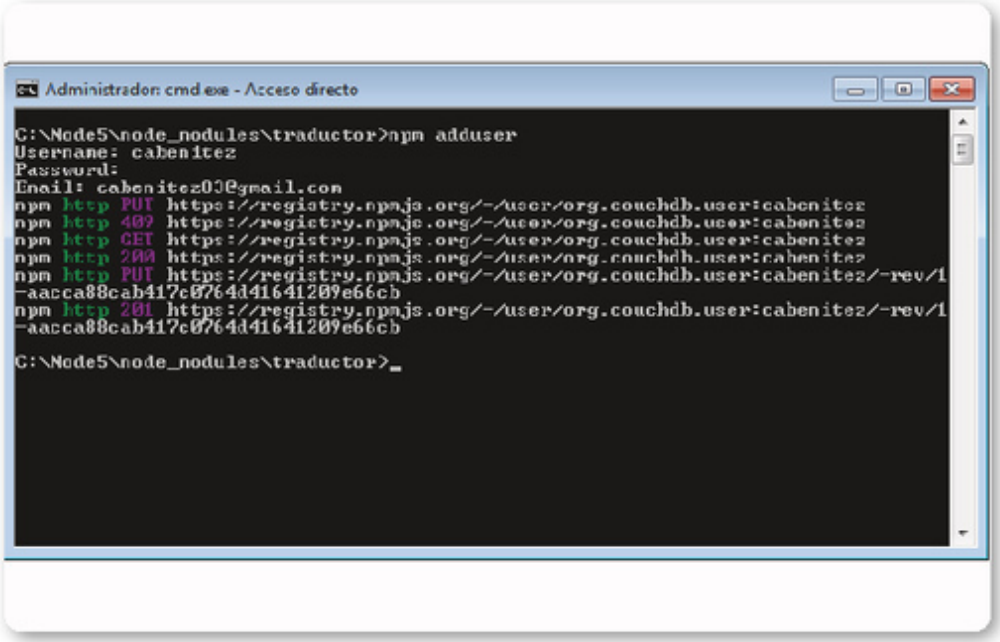


Figura 11. Cuando ejecutamos el comando **npm adduser** debemos ingresar un nombre de usuario, una clave y un correo electrónico.

Por último, vamos a publicar el módulo. Para esto, en la consola nos situamos en el directorio del módulo (es decir, en **Node5\node_modules\traductor**) y ejecutamos el siguiente comando:

```
npm publish
```

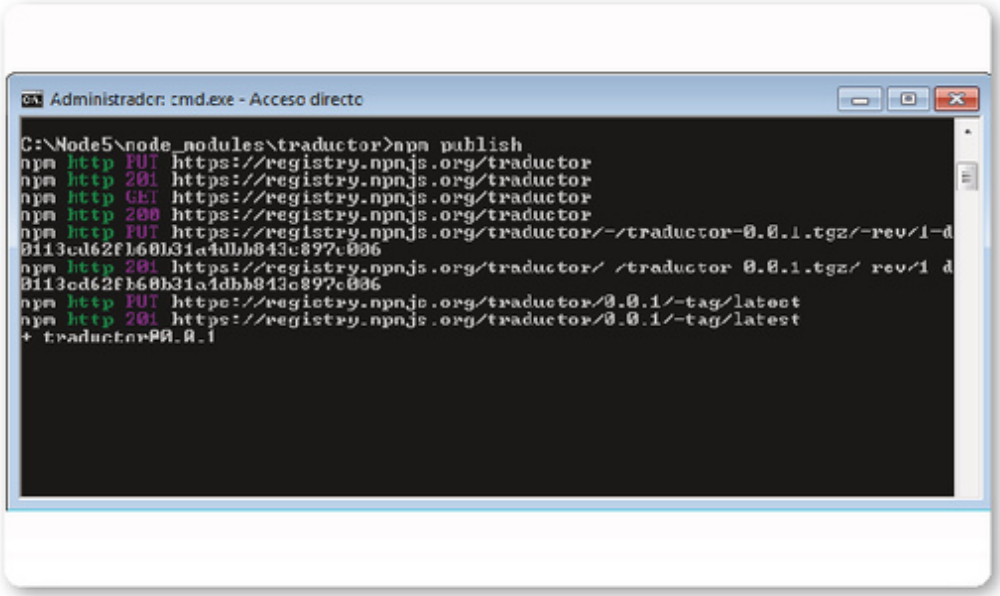


Figura 12. El comando **npm publish** automáticamente sube el módulo al repositorio público de **npm**.

Si deseamos verificar que efectivamente hemos realizado la publicación del módulo en el repositorio público de npm, podemos acceder a la página que se encuentra en la dirección **https://npmjs.org/package/traductor**. Otra opción interesante y eficiente es realizar una búsqueda accediendo al sitio web que se encuentra en la dirección **https://npmjs.org**.

Figura 13. En el sitio de npm podemos buscar el módulo y ver cómo se presenta ante el mundo.

Por otra parte, si queremos realizar la verificación de que el módulo pueda ser instalado de manera correcta, será necesario que creemos un directorio nuevo y, dentro de éste, un archivo JavaScript; por ejemplo, con el nombre **app.js**. El archivo que hemos creado deberá contener el código que mostramos a continuación:

FRAMEWORK PARA DESARROLLOS EN HTML5

iio Engine es un magnífico framework para el desarrollo de aplicaciones en HTML5. Está bien documentado y crear funcionalidades complejas no requiere más que unas pocas líneas. Además, incluye un sistema de depuración avanzado, ocupa muy poco espacio y es de código abierto. Podemos conocer más en el siguiente enlace: **http://iioengine.com**.

```
var traductor = require('traductor');

var dia = traductor.traducir_dia("Lunes", "br");
var mes = traductor.traducir_mes("enero", "br");

console.log(dia);
console.log(mes);
```

Después, en la consola, nos situamos en el directorio recién creado y ejecutamos el comando **npm install traductor**. Finalizada la descarga, ejecutamos la aplicación con el comando **node app.js**, que deberá mostrarnos el día y el mes traducido en la consola.

RESUMEN

En esta sección aprendimos cómo se estructuran los módulos y conocimos los métodos de carga que utiliza Node para el sistema de paquetes. También descubrimos una de las herramientas más poderosas de Node, **npm**, que nos permite instalar cualquier módulo desde el repositorio público a nuestro entorno local y posibilita la publicación de nuestros propios módulos para que estén accesibles para cualquier persona. Además, hemos conocido los módulos externos, que hacen de Node una tecnología muy ventajosa frente a otras, ya que son fáciles de utilizar y, al tener una estructura uniforme y encapsulada, permiten ser usados con total libertad.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Node posee módulos propios?
2. ¿Qué es el comando **exports** y cuándo se debe utilizar?
3. ¿Es correcto utilizar **requiere()** para agregar un módulo? Justificar.
4. ¿Los módulos son instalados automáticamente en el directorio **node_modules**?
5. ¿Los módulos son cacheados o se cargan cada vez que son incluidos?
6. ¿Existe un modo estándar de estructurar los módulos?
7. ¿Qué orden implementa Node para cargar un módulo?
8. ¿npm sirve para crear el archivo **package.json**? Explicar.
9. ¿Para qué se utilizan los comandos **npm install** y **npm publish**?
10. ¿Un módulo puede ser una dependencia de otro módulo?
11. ¿Es necesario definir el archivo **package.json** para publicar un módulo? Justificar.

EJERCICIOS PRÁCTICOS

1. Cree la estructura necesaria para publicar el primer ejemplo del capítulo en el repositorio público de npm.
2. Cree un método para traducir las estaciones en los idiomas inglés y portugués.
3. Realice cambios, modifique el color de la salida e intente publicar los cambios.
4. Cree un módulo para imprimir en consola texto generado al azar.
5. Publique el módulo anterior y vuelva a instalarlo en una nueva aplicación.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*09

Módulos más importantes de Node

Nos enfocaremos en algunos de los módulos más importantes de Node. Conoceremos los frameworks MVC y, en particular, Express. Aprenderemos a realizar aplicaciones en tiempo real y a enviar correos electrónicos. Además, veremos cómo interactuar con Redis desde Node.

- Frameworks MVC 276
- Express 277
- Socket.io 291
- Nodemailer 298
- node_redis 302
- Redis Commander 305
- Nodemon 306
- Resumen 309
- Actividades 310

Frameworks MVC

Podemos afirmar que un **framework** es un entorno de trabajo que nos entrega algunas herramientas de uso común, por ejemplo una estructura estándar que implementa conceptos y prácticas optimizadas. Su uso nos ofrece un marco de trabajo que podemos tomar como base de cualquier desarrollo.

Por otro lado, cuando hablamos de **MVC** (**Modelo-Vista-Controlador**), estamos haciendo referencia a un patrón de diseño de estructuración de archivos, donde se separan los datos, la lógica de negocio y la presentación de sistema.

Ahora que tenemos una mínima referencia de qué es un framework y también qué significa MVC, vamos a entender por qué es importante optar por un framework en Node.

En el **Capítulo 8** hemos desarrollado un ejemplo simple, donde no necesitábamos más que imprimir los resultados en la consola; pero, a la hora de crear un sistema web completo, es una buena práctica contar con un marco de referencia para manejar plantillas HTML, ruteo de solicitudes, cookies, sesiones y persistencia de datos, entre otras capacidades propias de cualquier sistema.

En la **Tabla 1** vemos algunos de los frameworks más conocidos.

FRAMEWORKS MVC

NOMBRE	DESCRIPCIÓN
Express	Es el framework más conocido de Node, está basado en Sinatra y contiene un conjunto de herramientas muy útiles. Es robusto, rápido y, además, es muy sencillo utilizarlo.
Meteor	Plataforma que permite construir sistemas web de manera rápida. La sintaxis que utiliza es JavaScript y tiene un método de sincronización muy útil que permite que, cuando los datos cambian, automáticamente se actualicen las plantillas.
Sails.js	Permite crear sistemas escalables a nivel corporativo y con arquitectura orientada a servicios. Es ideal para crear sistemas en tiempo real o juegos multiusuario, provee un sistema de acceso a datos muy simple y crea automáticamente las acciones básicas, como ABM de entidades.

CompoundJS	Está basado en Express y nos permite crear aplicaciones web de una manera estructurada. Posee un ORM llamado JugglingDB. Soporta internacionalización, testing y debugging.
Geddy	Es un framework muy simple y estructurado, lo cual lo hace fácil de usar. Está basado en enrutamiento. Soporta Jade como motor de plantillas, MongoDB y cookies. Integra una API para aplicaciones de tiempo real.

Tabla 1. Características de los diferentes frameworks que podemos utilizar en Node.

Express

Como mencionamos antes, Express es el framework más conocido de Node. Su éxito se debe a la simplicidad de uso y a la flexibilidad que proporciona para organizar la estructura de diversos tipos de sistemas.

En síntesis, debemos tener en cuenta que brinda una capa de abstracción que nos simplifica el trabajo con plantillas HTML, ruteo de solicitudes y manejo de cookies y sesiones. Posee un conjunto de herramientas denominado **middleware** que provee lo siguiente:

- **basicAuth()**: esta herramienta se encarga de implementar un sistema de autenticación de usuarios básico, gracias al cual realizamos estas tareas en forma sencilla.
- **bodyParser()**: decodifica los datos que se reciben en las solicitudes de los clientes; soporta los formatos **JSON**, **urlencoded** y **multipart**.
- **cookieParser()**: esta herramienta nos permite parsear las cookies que han sido enviadas por el navegador web que usamos.
- **cookieSession()**: provee un sistema de cookies basadas en sesiones.
- ***directory()**: posibilita utilizar los archivos de un directorio definido.
- ***query()**: parsea un **query string**.
- ***favicon()**: maneja archivos **favicon**.
- ***session()**: brinda un sistema para manejar sesiones.

> EL CONJUNTO DE HERRAMIENTAS MIDDLEWARE POSEE DIVERSAS OPCIONES IMPORTANTES

Instalación

En este momento, en que ya conocemos un poco mejor a Express, podemos empezar a trabajar con él. Para instalarlo necesitamos contar con Node y npm en nuestro sistema. Para comenzar, abrimos una consola y escribimos lo siguiente:

```
npm install -g express
```

Mediante el uso de este comando hemos instalado Express en modo global, de manera que se encontrará disponible en todo el sistema desde el gestor de paquetes npm.

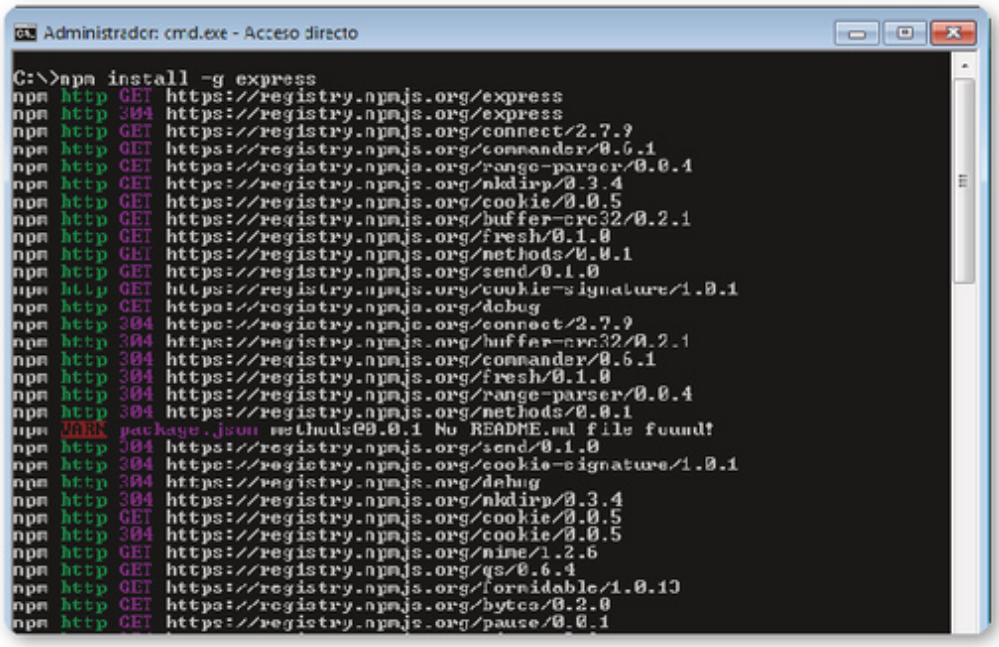


Figura 1. Al instalar Express de manera global podremos crear una aplicación en cualquier directorio.

UTILIZAR EJS EN LUGAR DE JADE EN EXPRESS

Es necesario considerar que una alternativa al motor de vistas por defecto de Express es utilizar **EJS** (**Embedded JavaScript**). La ventaja que posee este motor es que nos ofrece una gran similitud a la escritura de HTML, por esta razón no demanda tiempo para aprender a usarlo, a diferencia de Jade que utiliza una sintaxis basada en la indentación. Podemos conocerlo mejor si visitamos el sitio web que encontramos en la dirección: **http://embeddedjs.com**.

Creación de una aplicación

Una vez que tenemos instalado Express, ya podemos crear nuestra aplicación. Para esto, nos situamos en cualquier directorio de la consola (por ejemplo **C:**) y escribimos lo siguiente:

```
express EjemploApp
```

Mediante este comando, Express creará una estructura totalmente funcional con los archivos necesarios para empezar a trabajar. La estructura generada es la siguiente:

> MEDIANTE UNA CONSOLA DE COMANDOS PODEMOS CREAR UNA APLICACIÓN EXPRESS

```
EjemploApp
        |_ public/
        |          |_ images/
        |          |_ javascripts/
        |          |_ stylesheets/
        |                     |_ style.css
        |
        |_ routes/
        |          |_ index.js
        |          |_ user.js
        |
        |_ views/
        |          |_ index.jade
        |          |_ layout.jade
        |
        |_ app.js
        |_ package.JSON
```

Como podemos ver en el bloque anterior, la estructura de nuestra aplicación es bastante sencilla, e incluso ya tenemos creado el archivo denominado **package.JSON**, el cual se encarga de contener todas las dependencias que nuestra aplicación necesite.

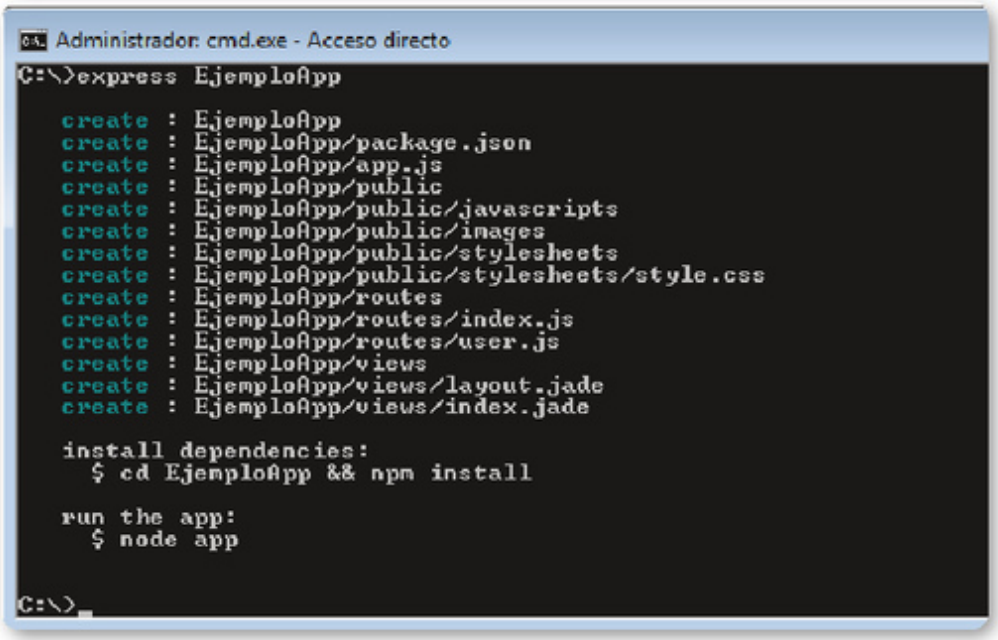


Figura 2. Al ingresar el comando, **EjemploApp** ya tiene la estructura básica para funcionar.

Instalación de dependencias

Una vez que hemos realizado las acciones necesarias para crear la aplicación, necesitamos instalar las dependencias. Si observamos el mensaje generado por Express luego de la creación de **EjemploApp**, notaremos que muestra lo siguiente:

```
cd EjemploAPP && npm install
```

Con estos comandos nos hemos situado dentro de la aplicación y, mediante npm, hemos instalado las dependencias necesarias dentro del

ALTERNATIVA A NODE

Es interesante considerar que existe una alternativa bastante eficiente a Node, **Vert.x**. Se presenta como un proyecto que soporta **JavaScript**, **Coffee Script**, **Ruby**, **Python**, **Groovy**, **Java** y que incluso brinda la posibilidad de combinar estos lenguajes. Vert.x permite crear sistemas escalables y de tiempo real de manera muy fácil. Podemos encontrar información adicional visitando el sitio web que se encuentra en la siguiente dirección: **http://vertx.io**.

directorio denominado **node_modules/**. En este punto podemos observar que también se incluye el módulo llamado **express** en la aplicación: es necesario considerar que esto es necesario ya que, desde ahora, **EjemploApp** se encargará de utilizarlo para poder funcionar.

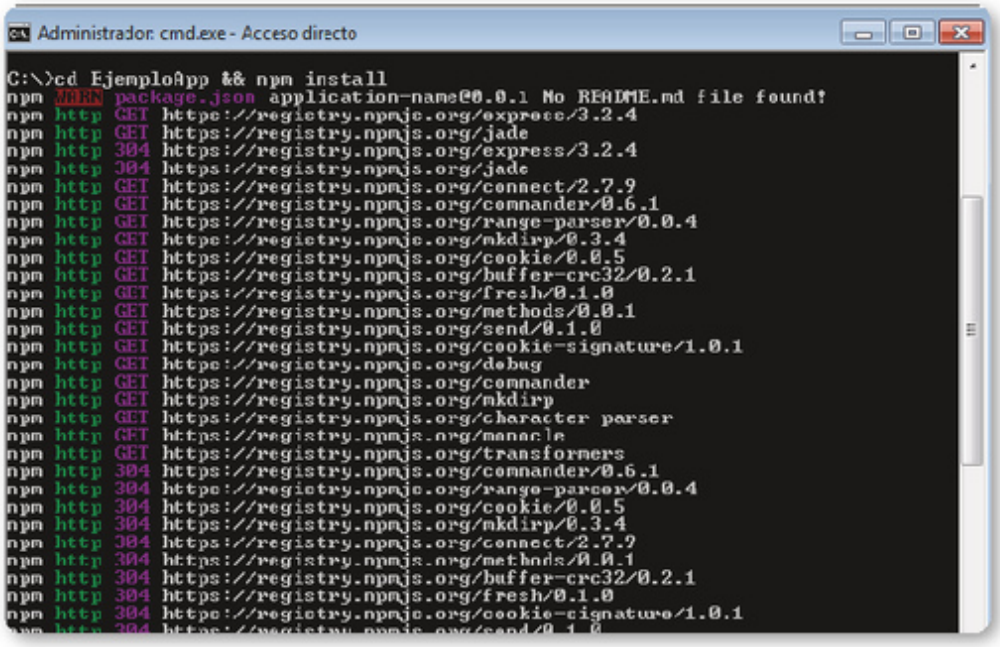


Figura 3. La instalación de las dependencias es necesaria para que las aplicaciones funcionen correctamente.

Prueba de la aplicación

Después de efectuar la correcta instalación de las dependencias, podemos proceder a realizar la ejecución de la aplicación. Para ejecutar el programa solo será necesario acceder a una consola y escribir el comando que mostramos a continuación:

```
node app.js
```

MÓDULOS PARA NODE

Para buscar módulos de Node, además de acceder a **https://npmjs.org**, también es posible utilizar los servicios de **http://nodetoolbox.com**, **http://npmdoc.nodejitsu.com** y **http://eirikb.github.com/nipster**. Cada uno de estos sitios ofrece una presentación particular, sin perder la eficacia.

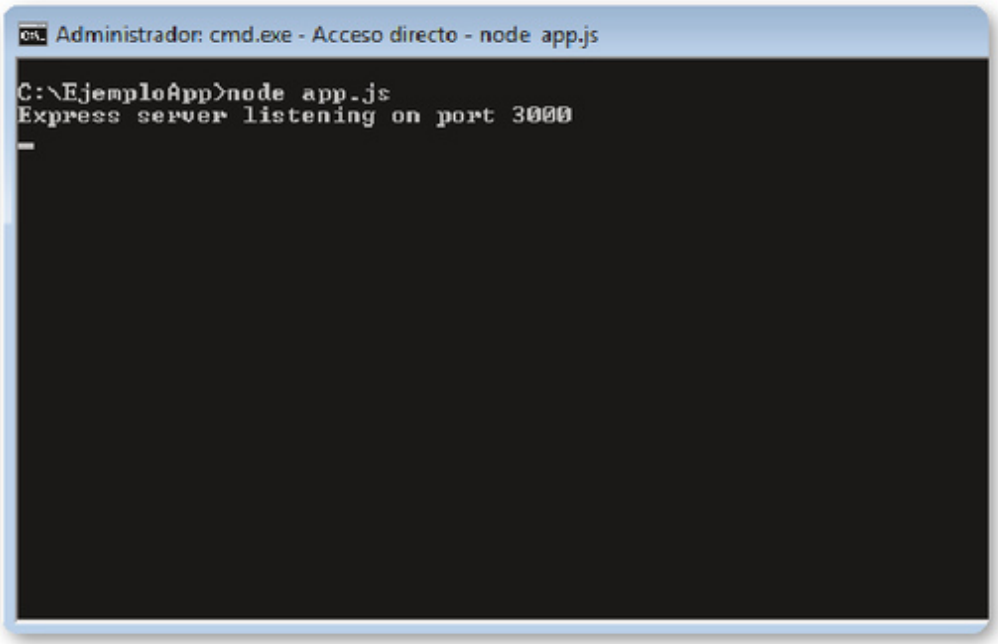


Figura 4. Cuando ejecutamos la aplicación, Express nos informa que está escuchando en el puerto **3000**.

Ahora, para verificar que efectivamente la aplicación está ejecutándose, tenemos que abrir el navegador y acceder a la siguiente dirección: **http://localhost:3000**.

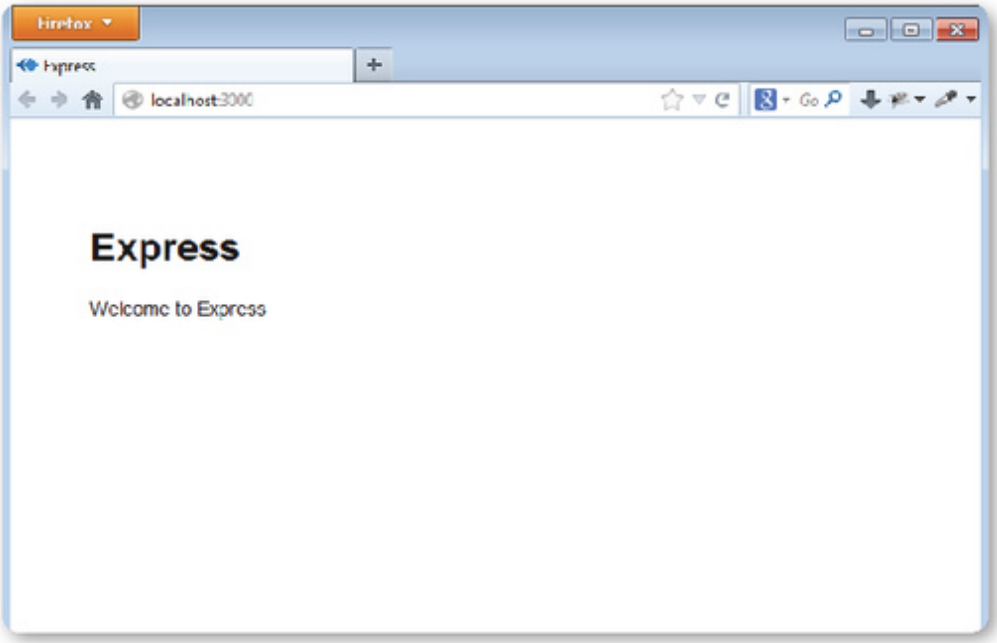


Figura 5. Ya podemos empezar a trabajar en la aplicación porque hemos verificado que funciona correctamente.

A continuación, vamos a analizar cada uno de los componentes de la estructura de la aplicación.

El archivo app.js

El archivo **app.js** es la interfaz por la cual se ejecuta la aplicación, y tiene una estructura similar a la siguiente:

```
/**
 * Module dependencies.
 */

var express = require('express')
  , routes = require('./routes')
  , user = require('./routes/user')
  , http = require('http')
  , path = require('path');

var app = express();

// all environments
app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'jade');
app.use(express.favicon());
app.use(express.logger('dev'));
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));

// development only
if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}
app.get('/', routes.index);
```

```
app.get('/users', user.list);

http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

En el código anterior, primero se importan los módulos que necesitará utilizar la aplicación, entre los cuales se encuentran **express**, **path** y **http**. A continuación, se crea y asigna una instancia de **express** en la variable **app**.

Luego se utiliza el método **set** del objeto **app** para establecer variables que estarán disponibles en todo el sistema (es decir, en las rutas y las vistas). Las variables que se definen por defecto son el puerto, el directorio donde estarán las vistas y el motor de plantillas.

A continuación, se utiliza el método **use** del objeto **app**, que recibe dos parámetros: el primero es opcional y define la ruta a la que se aplicará el segundo parámetro, que es una función del middleware. Cuando el primer parámetro es omitido se establece el valor por defecto **"/"**.

Por último, se define el uso del middleware **errorHandler()** solo para el entorno **development**. Es importante aclarar que es posible configurar las variables y habilitar herramientas para diferentes entornos.

Ruteo

La característica más importante de Express es su capacidad de manejar rutas y acciones correspondientes para cada una de ellas, sin las cuales sería imposible generar una interfaz para cada petición del cliente. Para manejar las rutas recurrimos al siguiente código:

```
app.VERBO(path, [callback], callback);
```

MÓDULOS AGRUPADOS POR CATEGORÍA

En la cuenta de Joyent en Github es posible ver una lista extensa de módulos para Node agrupados por categoría, entre las cuales se encuentran: frameworks web, clientes para bases de datos, administradores de plantillas, CMS, parseadores y manejadores de gráficos.

En esta línea, vemos que **VERBO** puede ser alguno de los métodos del objeto **http**, como **get**, **post**, **put** y **delete**, entre otros. Luego, como parámetros, tenemos **path** (que es la URL a la que queremos responder), **[callback]** (que es opcional y una herramienta que se va a ejecutar antes del segundo **callback**) y finalmente un tercer parámetro obligatorio, que es la acción que se llevará a cabo para atender la petición.

Como ejemplo agregamos el código al archivo **app.js**, de manera que, cuando se acceda a la dirección **http://localhost:3000/info**, se devuelva la información de la ruta, el host y el protocolo solicitado en **formato JSON**.

```
app.get('/info', function(req, res){
   res.JSON({'Ruta'     : req.path,
             'Host'     : req.host,
             'Protocolo' : req.protocol
            });
});
```

Debemos ubicar este código antes del comando **http.createServer()** del archivo **app.js**. Para probar nuestro ejemplo, abrimos una consola, nos situamos en el directorio y ejecutamos la aplicación con **node app.js**.

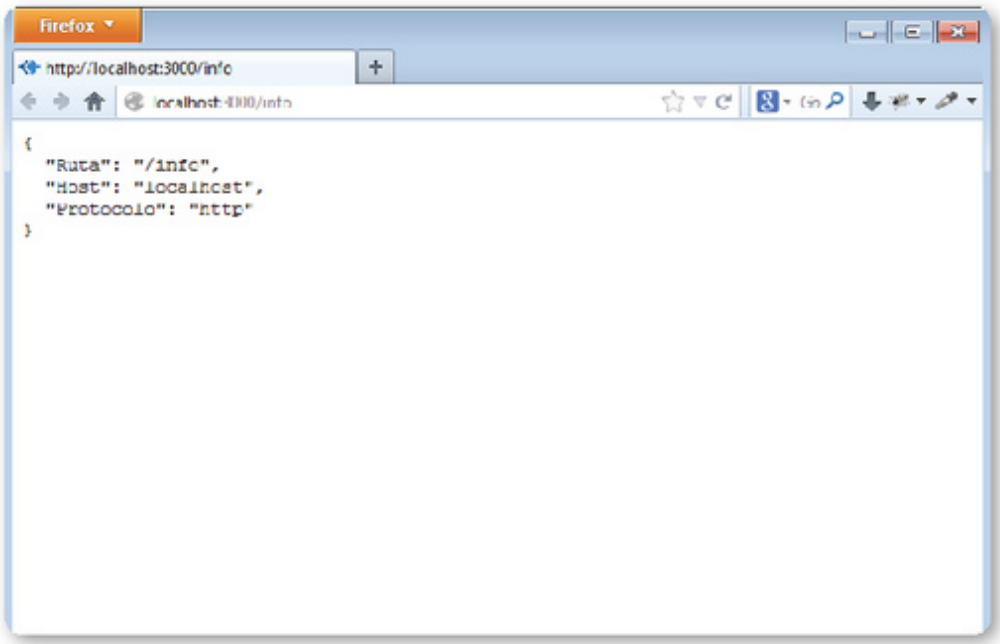


Figura 6. El servidor devuelve la información de ruta, el host y el protocolo en **formato JSON** cuando se accede a la URL **/info**.

Como vimos anteriormente, es posible definir comportamientos para cada ruta que necesitemos atender. Para incluir parámetros a una ruta, en el archivo **app.js**, debajo del método creado anteriormente, escribimos el código que mostramos a continuación:

```
app.get('/info/:parametro', function(req,res){
    var parametro = req.params.parametro;
    switch(parametro){
        case 'path':
            res.send('Path: ' + req.path);
            break;
        case 'host':
            res.send('Host: ' + req.host);
            break;
        case 'protocol':
            res.send('Protocolo: ' + req.protocol);
            break;
        default:
            res.send('Parámetro "' + parametro + '" no existe.');
    }
})
```

En este código hemos definido una nueva opción de ruteo para **/info**, donde agregamos la posibilidad de tener un parámetro opcional. Como pudimos ver en el código anterior, los parámetros se definen con el carácter dos puntos (:) seguido del nombre.

El valor del parámetro lo capturamos mediante **req.params.parametro** y luego verificamos si coincide con **path**, **host** o **protocol**. En caso de que coincida, devolvemos estos valores y, en el caso contrario, notificamos al usuario que el parámetro solicitado no existe.

Volviendo al archivo **app.js**, vemos que Express genera en forma predeterminada las rutas que mostramos a continuación:

```
app.get('/', routes.index);
app.get('/users', user.list);
```

Ahora vamos a analizar de qué manera se atiende cada una de ellas. Primero tenemos el ruteo para la página de inicio, en el cual el **callback** es **routes.index**. Esto quiere decir que se va a devolver el archivo **index.js** del directorio **routes**. En el archivo tenemos lo siguiente:

```
/*
 * GET home page.
 */

exports.index = function(req, res){
  res.render('index', { title: 'Express' });
};
```

Lo que encontramos aquí es una función llamada **index** que utiliza el método **render** del objeto **res**, propio de Express, que permite devolver una página construida a partir de una plantilla HTML mediante el motor de plantillas **jade**.

Los motores de plantillas son sistemas que nos permiten crear vistas dinámicas, donde el principal objetivo es separar la capa de negocio de la de presentación.

> PARA CREAR VISTAS DINÁMICAS DE LOS SISTEMAS USAMOS LOS MOTORES DE PLANTILLAS

Continuando con el método **render**, observamos que se definen dos parámetros: el primero es el nombre de la plantilla que se utilizará para mostrar el contenido (las plantillas deben estar ubicadas dentro del directorio **views/** y, para este caso, se utiliza el archivo **index.jade**) y el segundo parámetro es un objeto JSON que contiene una lista de claves que serán enviadas a la vista para ser mostradas en algún contenedor.

Si observamos el archivo **/views/index.jade**, tenemos lo siguiente:

```
extends layout

block content
  h1= title
  p Welcome to #{title}
```

Aquí nos encontramos con una estructura bastante diferente a la sintaxis HTML convencional, porque **jade** emplea una sintaxis simplificada basada en la indentación para el diseño y, al momento de renderizar el contenido, lo transforma en elementos HTML.

Sin tener en cuenta las dos primeras líneas del código, vemos que se define un elemento **h1** y un **p**, en el que se utiliza la variable **title** enviada desde el archivo **index.js**.

Para agregar variables a la vista solo es necesario definirlas en el **JSON** del archivo **index.js** y luego utilizarlas en algún equivalente al HTML en el archivo **index.jade**. Podemos encontrar los elementos que dispone **jade** accediendo a su página oficial: **http://jade-lang.com**.

Siguiendo con el archivo llamado **index.jade**, en la primera línea vemos que se define lo siguiente:

```
extends layout
```

Esto significa que se hace uso del contenido de la plantilla **layout** mediante una extensión. Si observamos **views/layout.jade** tenemos:

```
doctype 5
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

En este archivo está determinada la estructura de la página HTML; podemos ver que solo están definidos los nombres de las etiquetas y

SISTEMAS DESARROLLADOS CON NODE

Actualmente, existen varias opciones para alojar y probar en línea nuestros proyectos creados en Node. Entre los más recomendados se encuentran **Heroku**, **AppFog**, **OpenShift**, **Nodejitsu** y **Windows Azure**. Todos estos servicios ofrecen cuentas gratuitas o triales.

los cierres de cada una por su indentación. Para incluir archivos, como hojas de estilos, debemos definirlos en el directorio **public**. En este caso se incluye el archivo **style.css** que se encuentra en **/public/stylesheets/**.

Volviendo al archivo principal **app.js**, tenemos el ruteo para la dirección **/users**:

```
app.get('/users', user.list);
```

En este caso se utiliza la función **list** definida en el archivo **routes/user.js**, en el cual tenemos lo siguiente:

```
/*
 * GET users listing.
 */
exports.list = function(req, res){
  res.send("respond with a resource");
};
```

No se renderiza el contenido en ninguna plantilla, sino que se utiliza el método **send** para devolver la respuesta en formato **String**.

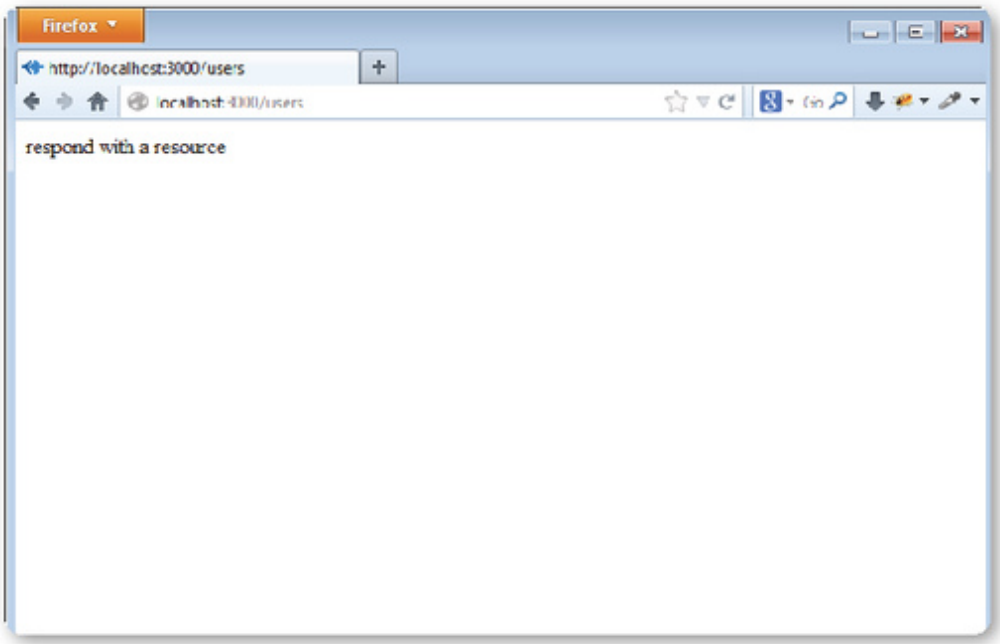


Figura 7. Al igual que **/users**, podemos definir cualquier ruta que necesitemos procesar y devolver una salida.

El método **res** de Express dispone de varias funciones y métodos para devolver la salida, entre los cuales se encuentran los siguientes:

- **res.send()**: es el método principal para devolver un respuesta al cliente.
- **res.sendfile()**: envía como respuesta un archivo al usuario.
- **res.JSON()**: envía una salida en formato JSON.
- **res.writeHeader()** y **res.contentType()**: definen el tipo de dato que devolverá el servidor.
- **res.redirect()**: redirecciona el navegador a una URL específica.
- **res.render()**: renderiza la salida en una plantilla.
- **res.end()**: finaliza la respuesta al cliente.

Nos quedan por analizar las últimas líneas del archivo **app.js:**

```
http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

Aquí se hace uso del método denominado **createServer()** correspondiente al objeto **http**, al cual se le pasa como parámetro el objeto **app** de Express, creado y configurado anteriormente. Luego se agrega el método **listen**, al cual se le indica que la aplicación va a estar disponible en el puerto definido mediante la variable **port**.

Express se presenta como uno de los frameworks más robustos y estables para desarrollar aplicaciones en Node, algo sumamente importante, ya que nos ahorra mucho tiempo gracias al conjunto de herramientas que dispone, a su integración con motores de plantillas y a su manera de gestionar el ruteo.

EMPRESAS QUE USAN EXPRESS

Entre las empresas más conocidas que actualmente tienen desarrollado algún servicio con Express se encuentran **MySpace**, **LearnBoost**, **Storify**, **Geekli.st**, **Prismatic**, **Persona**, **Countly**, **Apiary.io** y **Balloons.io**. Diariamente se reportan muchas aplicaciones distribuidas por el mundo que utilizan Express.

Socket.IO

Cada vez son más los sistemas web que implementan funciones en tiempo real, como las redes sociales **Facebook**, **Twitter** o **Google+**.

Socket.IO es una librería que nos permite crear sistemas de tiempo real. Posee dos componentes principales: uno para el cliente que opera en el navegador y otro para el servidor. Fue creada por Guillermo Rauch en el lenguaje JavaScript y es **Cross-platform** (es decir, funciona en todos los sistemas operativos).

> SOCKET.IO UTILIZA EL PROTOCOLO WEBSOCKET, QUE CORRESPONDE A HTML5

En principio, Socket.IO utiliza el protocolo **WebSocket** de HTML5 para la comunicación con el navegador, pero en caso de que esto falle recurre a otras técnicas, como **Adobe Flash Sockets**, **Ajax long polling** o **JSONP Polling**. La ventaja que ofrece para los desarrolladores está en su total transparencia al programar, sin importar qué mecanismo se utiliza para la comunicación. A continuación, en la **Tabla 2**, detallamos los navegadores soportados:

NAVEGADORES SOPORTADOS POR SOCKET.IO

TIPO	NOMBRE
Escritorio	IE5.5 Safari 3 Google Chrome 4 Firefox 3 Opera 10.61
Móviles	iPhone Safari iPad Safari Android WebKit WebOs WebKit

Tabla 2. Navegadores en los que es posible utilizar Socket.IO.

SOCKET.IO NOS PERMITE CREAR SISTEMAS DE CHAT Y MESAS DE AYUDA EN LÍNEA

Los usos más comunes de Socket.IO se encuentran en la creación de sistemas de chat, mesas de ayuda en línea, notificación de estados y cualquier otro sistema de comunicación asincrónico. Sus métodos más importantes son **on()** y **emit()**, que se utilizan para esperar por un evento y emitir uno, respectivamente.

Socket.IO es totalmente compatible con Express y su implementación es muy sencilla pero, por razones de simplicidad, a continuación vamos a desarrollar un ejemplo sin utilizar el framework. El propósito será agregar nombres a una lista mediante un formulario web y que sean visibles para todos los clientes conectados.

Primero vamos a abrir una consola. Nos situamos en la unidad **C:** y creamos una carpeta con el nombre **EjemploSocketIO**, y luego instalamos Socket.IO mediante el siguiente comando:

```
npm install socket.io
```

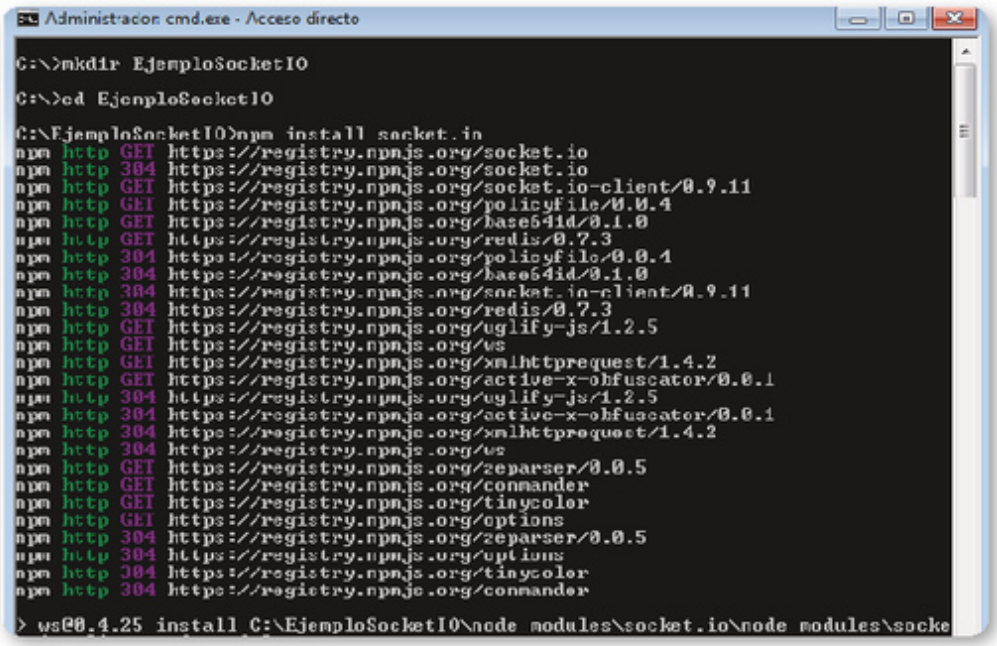


Figura 8. Al instalar Socket.IO estamos listos para empezar a desarrollar aplicaciones de tiempo real.

Es momento de crear el servidor, que se va a encargar de procesar las solicitudes hechas por el cliente y devolver la salida de manera

asincrónica. Para realizar esto, dentro del directorio denominado **EjemploSocketIO/**, creamos un archivo con el nombre **servidor.js** y el código que presentamos a continuación:

```
var io = require('socket.io').listen(4000);

io.sockets.on("connection", conexion);

function conexion(socket){
    socket.on("nombreNuevo", emitir);
    console.log('Se ha conectado un cliente.');
}

function emitir(data){
    io.sockets.emit("nombreRecibido",data);
    console.log('Se ha agregado el nombre: ' + data);
}
```

En este código, primero hemos creado una instancia de Socket.IO que va a funcionar en el puerto **4000**. Posteriormente, mediante el uso del método denominado **on()**, esperamos por el evento **connection**, que ocurrirá cuando un cliente se conecte desde el navegador; cuando esto suceda, se va a ejecutar la función **conexion**.

En **conexion** vemos que se pasa como parámetro el **socket** al cual se ha conectado y, dentro de la función que corresponde, tenemos nuevamente un método que espera por el evento **nombreNuevo**; luego, se muestra un mensaje en la consola de comandos.

Cuando el evento **nombreNuevo** ocurre, se llama a la función **emitir**, a la cual le pasamos como parámetro la información que proviene del cliente. Dentro de esta función hacemos uso del método denominado **emit** para publicar la información recibida de todos los clientes conectados bajo el evento **nombreRecibido**.

A continuación, vamos a crear el archivo de estilos para nuestro sistema. No lo explicaremos en detalle ya que no hace a su funcionamiento; simplemente, creamos un archivo llamado **style.css** en el directorio raíz del ejemplo y agregamos el siguiente código:

```
body{
    color: #333;
    background: #333;
    font-family: "Helvetica", Arial;
    font-size: 12px;
    text-align: center;
}
form{
    background: #CCC;
    border-radius: 10px;
    margin: 10px auto;
    padding: 10px;
    width: 40%;
}
form input{
    display: block;
    font-size: 12px;
    margin: 10px auto;
    padding: 0.5em;
    width: 70%;
}
h1{
    color:#FFF;
    text-shadow: 10px 10px 10px rgba(0,0,0,0.5);
}
div{
    text-align: left;
}
```

PROYECTOS QUE IMPLEMENTAN SOCKET.IO

En la wiki de Github de la cuenta de **LearnBoost** se detallan algunos de los proyectos que implementan Socket.IO. Actualmente, la lista se compone de más de veinticinco proyectos, sobre los cuales podemos basarnos para crear uno propio. Podemos verlos visitando la dirección **https://github.com/LearnBoost/Socket.IO/wiki/Projects-using-Socket.IO**.

El paso siguiente es crear la interfaz para el cliente. Para esto, creamos un archivo con el nombre **index.html**:

```
<!DOCTYPE html>
<html lang="es">
<head>
        <meta charset="utf-8" />
        <title>Ejemplo Socket.io</title>
        <link rel="stylesheet" type="text/css" href="style.css" />

        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.
js"></script>
        <script src="http://127.0.0.1:4000/socket.io/socket.io.js"></script>
        <script src="script.js"></script>
</head>
<body>
        <header>
                <h1>Socket.io</h1>
        </header>
        <section>
                <form id="formulario" name="formulario" action="/">
                        <label>¿Cuál es tu nombre?</label>
                        <input type="text" maxlength="10" id="nombre"
placeholder="Escribe tu nombre" required />
                        <div id="nombres"></div>
                </form>
        </section>
</body>
</html>
```

En este código hemos definido una estructura HTML5, en la cual hemos importado el archivo de estilos **css** y la librería **jQuery**. Luego, importamos la librería **socket.io** a través de la dirección IP y el puerto que hemos definido en el servidor, más la ruta **/socket.io/socket.io.js**.

A continuación, agregamos el archivo **script.js**, que vamos a ver luego de terminar la estructura HTML.

En la estructura hemos definido un formulario que tiene una caja de texto donde el usuario va a ingresar su nombre y, más abajo, definimos un contenedor con el id **nombres**, en el cual se irán agregando los otros usuarios que realicen un acceso al sistema.

Una vez definida la estructura que verá el cliente, vamos a crear la lógica de comunicación con el servidor. Para esto, creamos el archivo **script.js** con el código que vemos a continuación:

```
var sockets = io.connect('http://127.0.0.1:4000');

$(document).ready(function(){
   $('#formulario').on('submit', enviarNombre);

   sockets.on('nombreRecibido', mostrarNombre);
});

function enviarNombre(e){
   e.preventDefault();
   sockets.emit('nombreNuevo', $('#nombre').val());
   $('#nombre').val('');
}

function mostrarNombre(nombre){
   $('#nombres').append('Acaba de entrar: ' + nombre + '<br />');
}
```

En el código que presentamos, primero nos encargamos de crear una instancia del socket que se conectará al servidor correspondiente, que está esperando por una conexión en el puerto **4000**.

Luego, en el evento **ready()** del documento, se presenta una primera instrucción donde nos encargamos de indicar que, cuando se envía el formulario, ejecute la función **enviarNombre()**.

En **enviarNombre()** prevenimos el envío del formulario y emitimos el evento **nombreNuevo** con el valor ingresado en la caja de texto, y vaciamos el contenido del elemento. Es importante destacar que, cuando se ejecuta el método **emit()**, desde el cliente se envía la información al servidor bajo el nombre definido en el primer

parámetro. Continuando con la siguiente instrucción del evento **ready()**, ejecutamos el método **on()**, que espera a que suceda el evento **nombreRecibido** enviado por el servidor cuando recibe un nombre.

Cuando suceda el evento **nombreRecibido**, ejecutamos la función **mostrarNombre()**, que agrega el nombre recibido al contenedor **nombres**. Esto ocurre para todos los usuarios conectados, incluso para el actual.

Para probar el ejemplo abrimos una consola, vamos a la carpeta específica, ejecutamos **servidor.js** y abrimos **index.html** en un navegador.

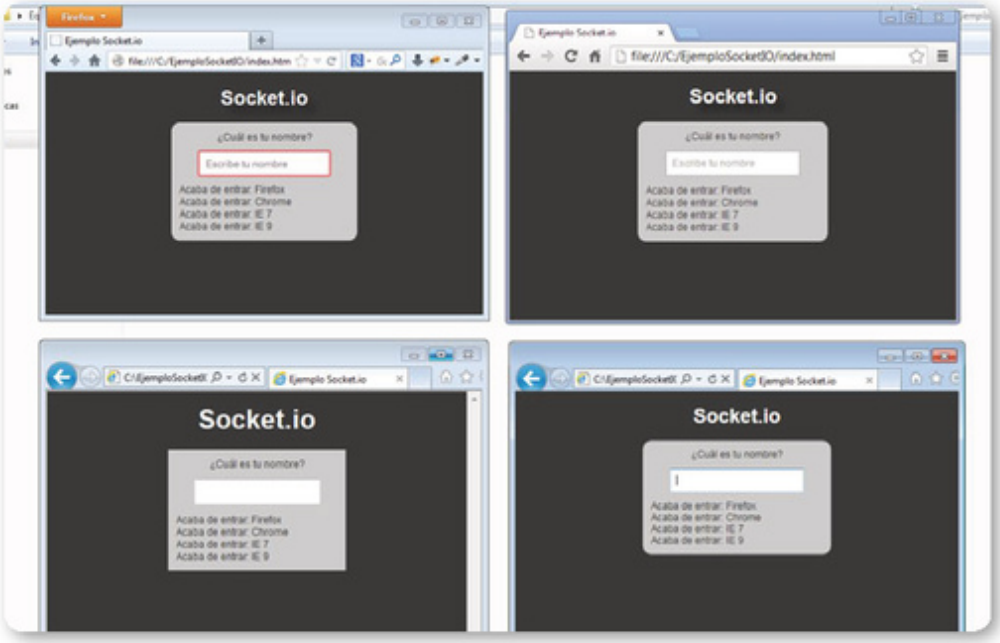


Figura 9. Vemos la comunicación en tiempo real desde los navegadores **Firefox**, **Chrome**, **IE7** e **IE9**, simultáneamente.

Hemos visto que es muy fácil crear sistemas que se desempeñen en tiempo real. El funcionamiento es muy similar al de los sistemas de chat o notificaciones, por lo que no hemos puntualizado una acción u otra sino mostrado la base para cualquier implementación.

CREACIÓN DE PROTOTIPOS CON SOCKET.IO

Matisse es una herramienta que permite crear **wireframes** o prototipos, en tiempo real y colaborativo, mediante Socket.IO. El único requisito que debemos cumplir para empezar a utilizar la herramienta es utilizar cuentas de Facebook, Twitter o Google.

Nodemailer

Nodemailer es un módulo para Node que sirve para enviar correos electrónicos de una manera sencilla. Soporta diferentes métodos de transporte y permite utilizar cualquier tipo de caracteres en el mensaje.

Posee las siguientes características:

- Admite cualquier tipo de carácter Unicode.
- Es posible usar texto plano y contenido HTML en los mensajes.
- Soporta el envío de archivos adjuntos.
- Es posible adjuntar imágenes embebidas en el cuerpo del mensaje.
- Permite utilizar conexión segura.
- Nos ofrece soporte para tres tipos de métodos de transporte: SMTP, Sendmail y Amazon SES.
- El transporte por SMTP se encarga de mantener una conexión abierta para rehusar el mecanismo.
- Ofrece la posibilidad de utilizar servicios SMTP preconfigurados para Gmail, Hotmail, Yahoo, etcétera.
- Ofrece soporte para el protocolo de autenticación XOAUTH2.

Ejemplo de prueba

Realizaremos un ejemplo de prueba donde el objetivo es enviar un correo electrónico a un destinatario utilizando el servicio SMTP de Gmail.

Primero creamos un directorio en la unidad **C:** con el nombre **EjemploNodemailer**, abrimos una consola y vamos a la carpeta que creamos. Ahí instalaremos Nodemailer con la siguiente instrucción:

```
npm install nodemailer
```

NODE EN FUNCIONAMIENTO

Si queremos saber cómo utilizan Node las empresas más conocidas del mundo tecnológico (como Yahoo, Google, Mozilla y LinkedIn), recomendamos leer un interesante artículo, al que podemos acceder desde: **http://venturebeat.com/2012/01/24/node-at-google-mozilla-yahoo**.

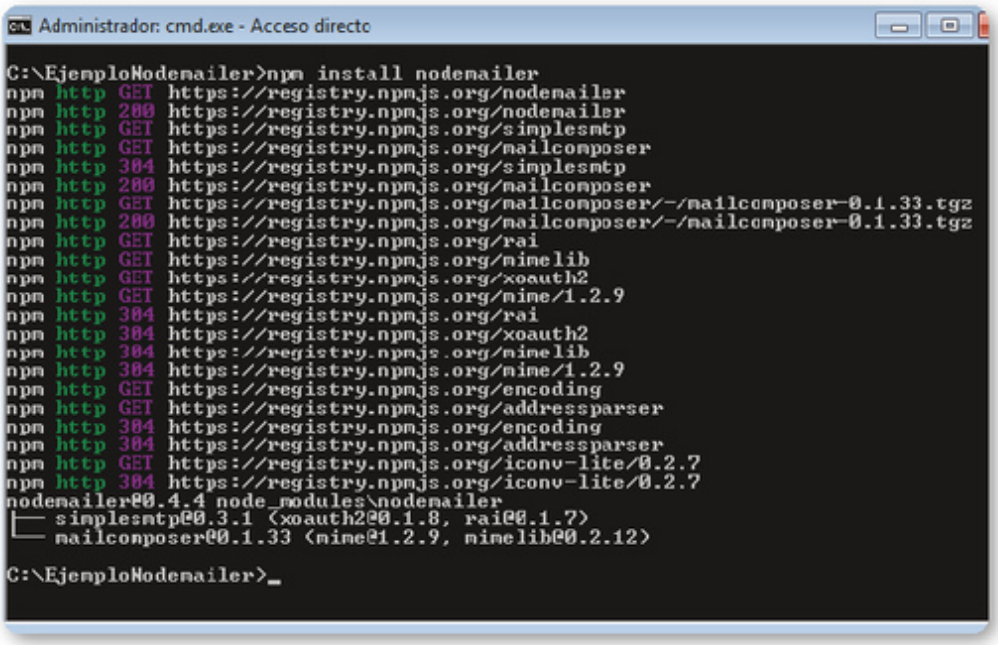


Figura 10. Una vez instalado Nodemailer, nos muestra las dos dependencias incorporadas, **`simplesmtp`** y **`mailcomposer`**.

A continuación vamos a crear un archivo con el nombre **app.js**:

```
var nodemailer = require("nodemailer");

var transporte = nodemailer.createTransport("SMTP",{
      service: "Gmail",
      auth: {
         user: "usuario@gmail.com",
         pass: "clave"
      }
   });

var opcionesMail = {
      from: "Carlos Alberto Benitez - <cabenitez83@gmail.com>",
      to: "cabenitez83@gmail.com",
      subject: "Prueba de Nodemailer",
      text: "Prueba de Nodemailer",
      html: "<b>Esto es una Prueba </b> ",
      attachments:[
         // Archivo de texto adjunto
```

```
            {
                fileName: 'adjunto.txt',
                contents: 'Contenido del archivo de texto Adjunto.',
                contentType: 'text/plain'
            },
            // Archivo binario adjunto
            {
                fileName: 'Check.png',
                contents: new Buffer('iVBORw0KGgoAAAANSUhEUgAAABAAAAAQ
AQMAAAAIPW0iAAAABIBMVEUAAAD/' +
                        '//+l2Z/dAAAAM0lEQVR4nGP4/5/h/1+G/58ZDrAz3D/
McH8yw83NDDeNGe4U' +
                        'g9C9zwz3gVLMDA/A6P9/AFGGFyj0XZtQA-
AAAAElFTkSuQmCC',  'base64'),
                cid: 'check@node'
            },
            // Archivo físico adjunto
            {
                fileName: 'logo-node.png',
                filePath: __dirname+"/logo-node.png",
                cid: 'logo-node@node'
            }
        ]
    };

    transporte.sendMail(opcionesMail, function(error, response){
        if(error)
            console.log(error);
        else
            console.log("Mensaje enviado: " + response.message);
        // Descomentar la siguiente línea si no se necesitan enviar mas mensajes.
        //transporte.close();
    });
```

En el código hemos incluido primero el módulo **nodemailer**, luego indicamos que el método de transporte va a ser **SMTP** y, mediante

una estructura JSON, definimos el servicio de envío y los datos de autenticación. En este caso, debemos usar una cuenta válida de Gmail.

A continuación, en la variable **opcionesMail**, configuramos los datos propios del correo electrónico, como quién lo envía, el o los destinatarios, el título y el cuerpo del correo.

Luego, en la clave **attachments**, indicamos las tres maneras de adjuntar un elemento. En la primera definimos un archivo de texto plano; luego adjuntamos una imagen en formato **PNG**, creada a partir de un código en **base64** y, por último, adjuntamos una imagen que se encuentra en el mismo directorio que nuestra aplicación.

> EN LA VARIABLE OPCIONESMAIL PODEMOS CONFIGURAR LOS DATOS DEL E-MAIL

Después de haber definido las opciones del correo, utilizamos el método **sendMail** del objeto **transporte** para enviar el correo. Aquí le pasamos como parámetro las opciones definidas anteriormente y, como segundo parámetro, ejecutamos una función para capturar el resultado de la ejecución en la consola.

Para probar el ejemplo que acabamos de realizar, debemos encargarnos de abrir una consola de comandos, situarnos en la carpeta **EjemploNodemailer/** y ejecutar **node app.js**.

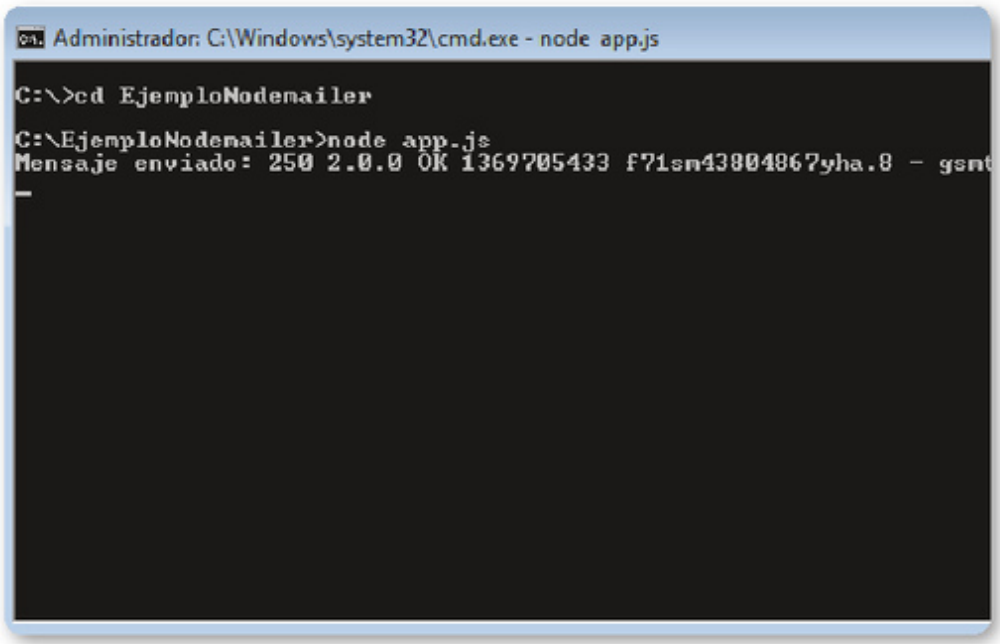


Figura 11. Al enviarse el correo electrónico, Nodemailer nos informa que todo ha salido bien.

Como podemos comprobar, Nodemailer es una excelente opción para enviar correos electrónicos de una manera muy fácil y segura.

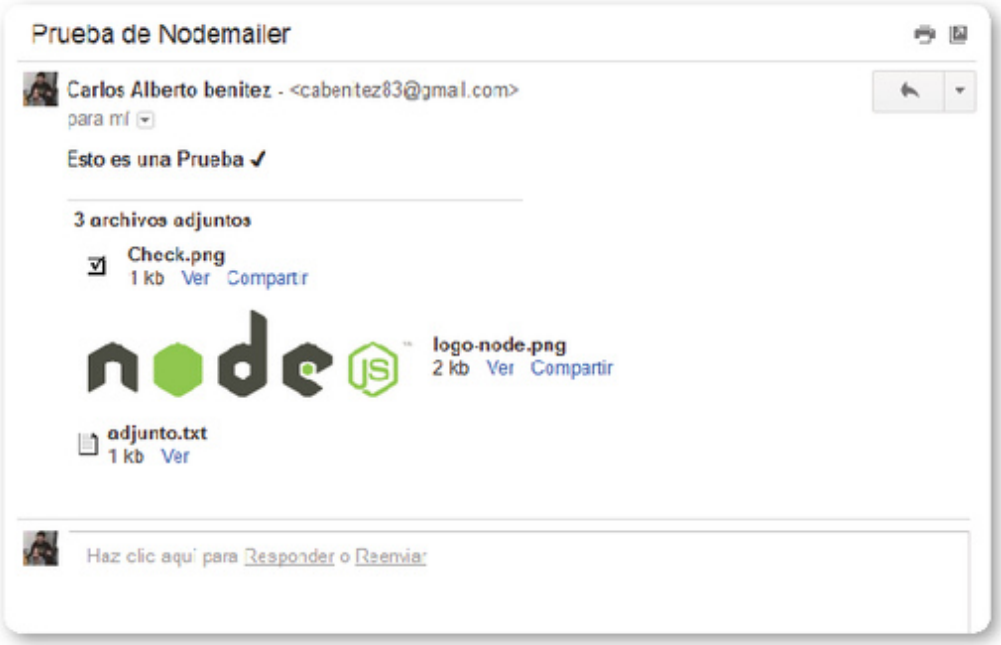


Figura 12. En el cuerpo del mensaje recibido, vemos el texto en formato **Unicode** y los archivos adjuntos.

node_redis

Como su nombre lo resume, **node_redis** es el cliente Redis para Node recomendado por el sitio oficial de Redis. Soporta todos los comandos, incluso los agregados recientemente como **EVAL**. Este cliente fue desarrollado por Matt Ranney y puede ser instalado como módulo con npm. El único requisito que necesitamos cumplir es tener instalado Redis en nuestro servidor. Para conocer su funcionamiento, vamos a crear un ejemplo sencillo donde trabajaremos con los tipos de datos de Redis.

Primero crearemos una carpeta en la unidad **C:** con el nombre **EjemploNodeRedis**. Luego deberemos abrir una consola e instalar el módulo con el siguiente comando:

```
npm install redis
```

El paso siguiente es crear el archivo **app.js**:

```
var redis = require('redis'),
cliente   = redis.createClient();
// cliente.select(3, function() { /* ... */ });

cliente.on('error', function (err) {
    console.log('Error ' + err);
});

// Strings ================================
cliente.set('node_redis:String', 'hola Mundo!', redis.print);
cliente.get('node_redis:String', redis.print);

// Hashes =================================
cliente.hmset('node_redis:Hash', 'Nombre', 'Juan', 'Apellido', 'Perez', redis.print);
cliente.hgetall("node_redis:Hash", function (obj) {
    console.log(obj);
});

// Lists ==================================
for (var i = 0; i <= 5; i++) {
    cliente.rpush('node_redis:List', 'Valor ' + i, redis.print);
};

// Sets ===================================
cliente.sadd('node_redis:Set:Set1', 'Hola', redis.print);
cliente.sadd('node_redis:Set:Set1', 'Mundo', redis.print);
cliente.sadd('node_redis:Set:Set2', 'Mundo', redis.print);
cliente.sadd('node_redis:Set:Set2', 'Maravilloso', redis.print);
cliente.sinter('node_redis:Set:Set1', 'node_redis:Set:Set2', redis.print);
cliente.sunion('node_redis:Set:Set1', 'node_redis:Set:Set2', redis.print);

// Sorted sets ============================
cliente.zadd('node_redis:SortedSet', 1, 'Uno', redis.print);
cliente.zadd('node_redis:SortedSet', 2, 'Dos', redis.print);
cliente.zadd('node_redis:SortedSet', 3, 'Tres', redis.print);
cliente.zrange('node_redis:SortedSet', 0, -1, redis.print);
cliente.quit();
```

En el código, primero incluimos el módulo y luego creamos una instancia con el método **createClient()**. En este método podemos definir opcionalmente el puerto y el host de conexión a Redis (por defecto, se utilizan **6379** y **localhost**, respectivamente).

A continuación, nos encontramos con una línea comentada donde podemos seleccionar la base de datos a la que nos vamos a conectar. Y en la siguiente instrucción, en caso de existir algún error, lo capturamos y lo mostramos en la consola.

En las siguientes líneas del bloque de código nos encargamos de trabajar con los tipos de datos **string**, **hash**, **list**, **set** y **sorted set**, donde los comandos utilizados en Node son los mismos que se definen en Redis. Como último parámetro de cada método, definimos el comando **redis.print** para imprimir el resultado en la consola.

Para ejecutar la aplicación, abrimos una consola, nos situamos en el directorio del ejemplo y ejecutamos **node app.js**.

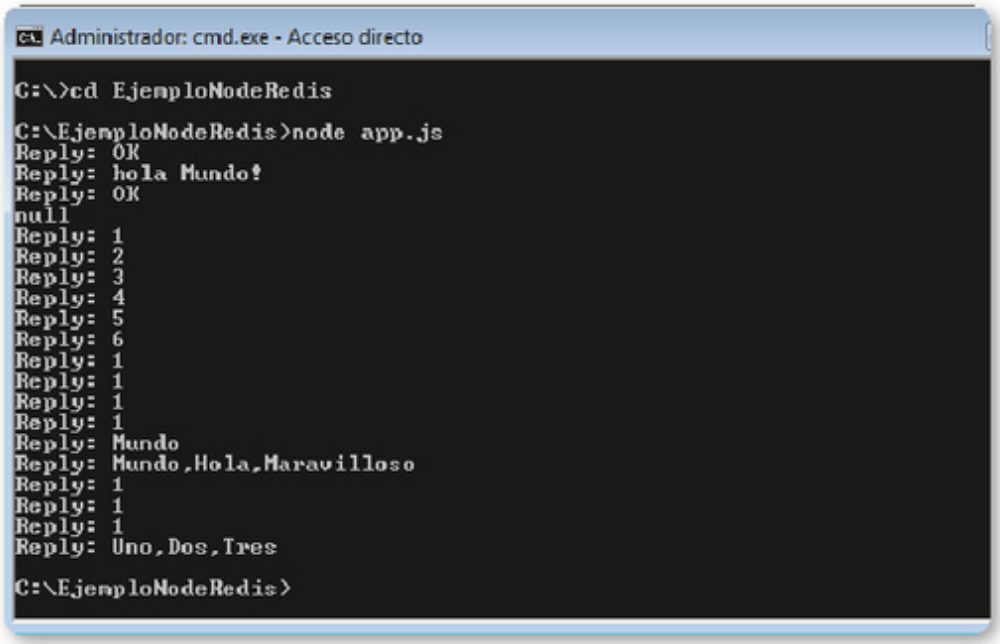


Figura 13. Mediante el comando **redis.print** vemos el resultado de cada interacción de Node con Redis.

Podemos verificar que, efectivamente, hemos creado las claves en Redis mediante la herramienta **phpRedisAdmin**. En nuestro caso, abrimos un navegador y nos dirigimos a **http://localhost/phpRedisAdmin**.

Como hemos visto, el cliente **node_redis** es muy simple de instalar y utilizar. Es la alternativa perfecta para interactuar con Redis desde Node.

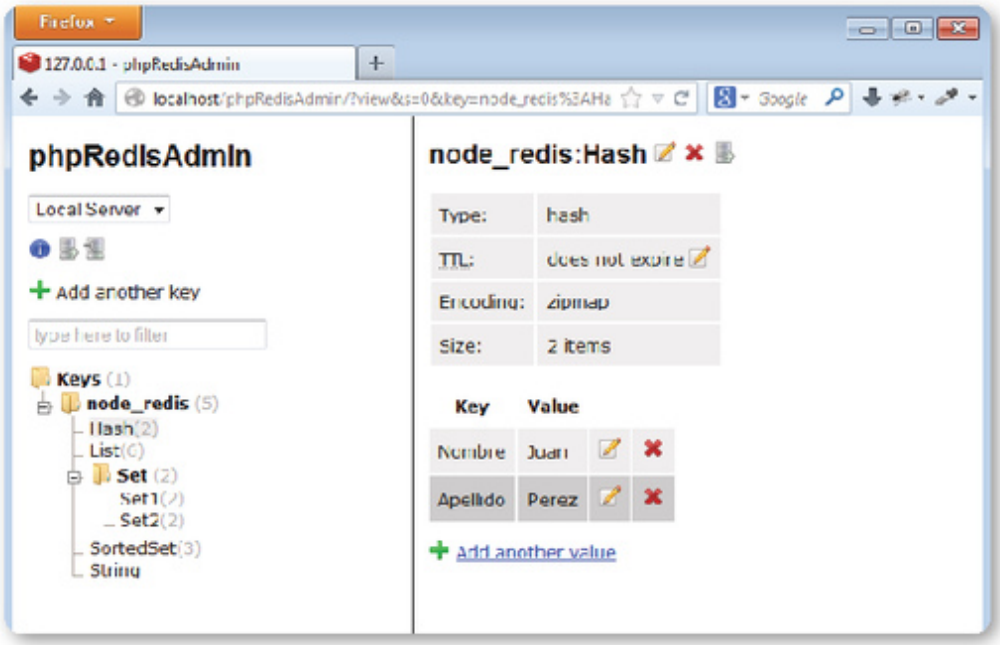


Figura 14. Mediante **phpRedisAdmin** podemos ver y administrar las claves creadas desde Node.

Redis Commander

Redis Commander es un módulo para Node que permite ver, editar y administrar bases de datos creadas en Redis. Su instalación es sencilla y similar a cualquier módulo de Node, solo tenemos que abrir una consola y escribir el siguiente comando:

```
npm install -g redis-commander
```

> REDIS COMMANDER PERMITE TRABAJAR CON BASES DE DATOS CREADAS EN REDIS

Luego de haber instalado el módulo de manera global, lo ejecutamos a través del comando:

```
redis-commander
```

Por defecto, el módulo se conecta a la base de datos **0** en el servidor local y es posible acceder a la interfaz web desde el navegador a través del puerto **8081** de **localhost**.

La interfaz web de Redis Commander ofrece características muy útiles, como la posibilidad de agregar servidores locales y remotos, y permite administrar todas las claves y bases de datos. Además, dispone de una consola para ejecutar directamente los comandos de Redis y una ventana interactiva para consultar los comandos disponibles.

Podemos conocer más sobre Redis Commander accediendo al sitio web que se encuentra en la siguiente dirección: **http://nearinfinity.github.io/redis-commander**.

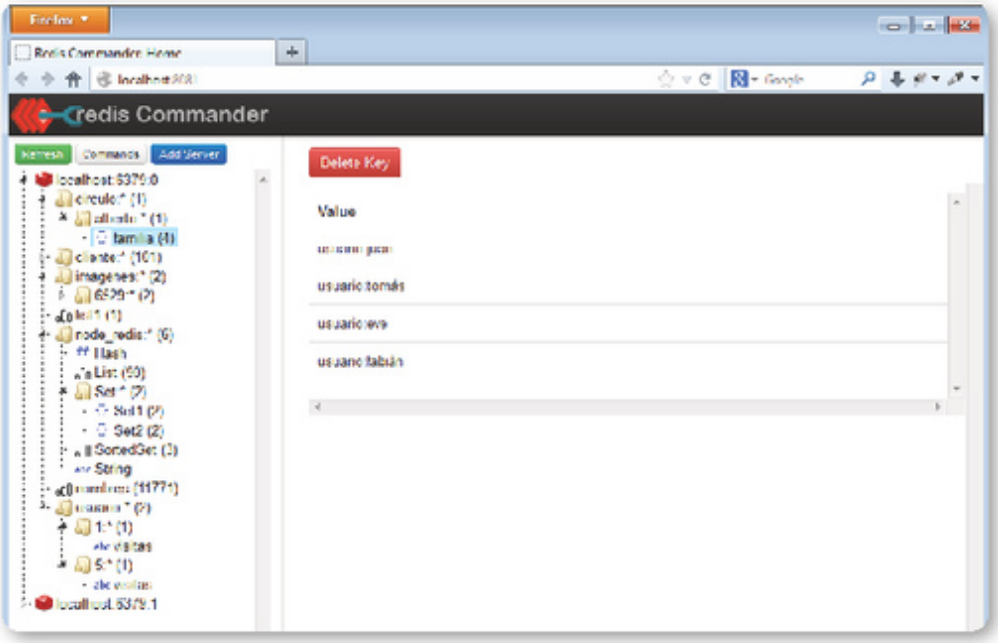


Figura 15. Con Redis Commander podemos administrar las bases de datos de Redis desde un navegador web.

Nodemon

Nodemon es una herramienta que permite monitorear aplicaciones desarrolladas con Node, con la particularidad de que, ante cualquier cambio producido en el directorio, reinicia automáticamente la aplicación. No es necesario realizar ningún cambio en las aplicaciones para implementarla, sino reemplazar el comando **node [nombre de la aplicación]** por **nodemon [nombre de la aplicación]** para iniciar la aplicación.

Para probar Nodemon, crearemos un simple ejemplo donde imprimiremos la fecha y hora actual en la consola, y lo iniciaremos con

el módulo. Luego vamos a realizar un cambio en el archivo principal para verificar que ha reiniciado de manera automática.

Primero debemos instalar el módulo. Para esto, utilizamos npm y lo hacemos de manera global mediante la siguiente instrucción:

```
npm install -g nodemon
```

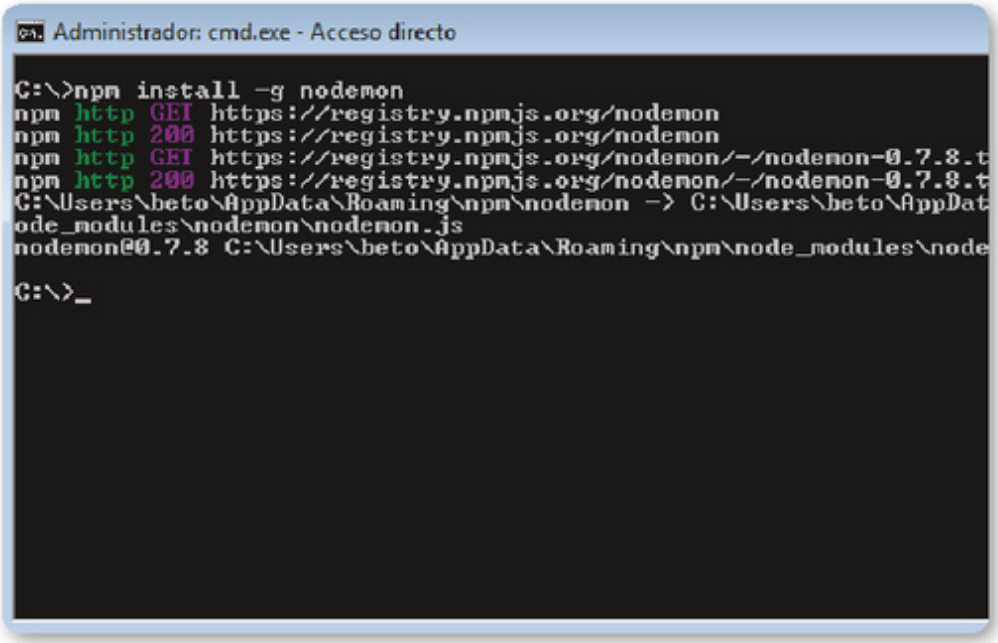


Figura 16. Al instalar **Nodemon** globalmente, podemos monitorear cualquier aplicación.

Después de instalar el módulo vamos a crear una carpeta en la unidad **C:**, con el nombre **EjemploNodemon**, y dentro de ella **app.js**:

```
var fecha   = new Date();

var anio    = fecha.getFullYear();
var mes     = fecha.getMonth();
var dia     = fecha.getDate();
var hora    = fecha.getHours();
var minutos = fecha.getMinutes();
var segundos = fecha.getSeconds();
```

```
    var fechaSalida = dia + '/' + mes + '/' + anio + ' ' + hora + ':' + minutos + ':' +
segundos;

    console.log('Esta app se ejecutó: ' + fechaSalida);
```

En este código primero hemos declarado una variable que contendrá la fecha actual, de la cual obtendremos el año, el mes, el día, la hora, los minutos y los segundos, para después concatenarlos en una variable y mostrarlos junto a un texto en la consola. Para ejecutar la aplicación abrimos una consola, ejecutamos el comando que sigue:

```
nodemon app.js
```

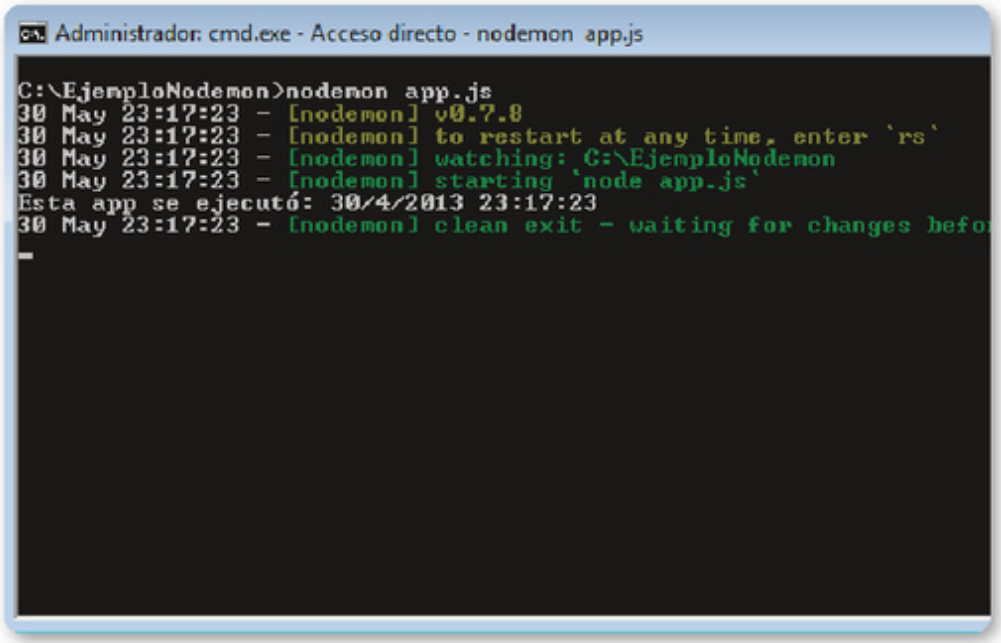


Figura 17. Al ejecutar nuestra aplicación, Nodemon nos indica que se está monitoreando el directorio en espera de cambios.

Para verificar que nuestra aplicación reinicie de manera automática, cambiamos el archivo **app.js** reemplazando la siguiente línea:

```
    console.log('Esta app se ejecutó: ' + fechaSalida);
```

Por esta otra:

```
console.log('Esta aplicación se ejecutó: ' + fechaSalida);
```

Guardamos los cambios y observamos en la consola el comportamiento.

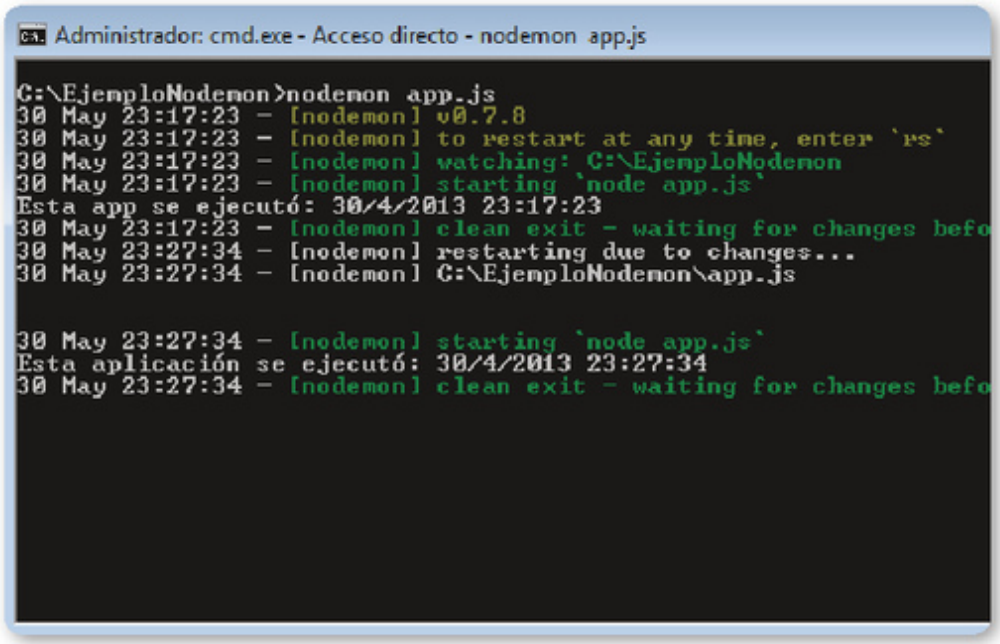


Figura 18. Podemos observar que, al detectarse un cambio en el archivo **app.js**, el módulo **nodemon** reinició automáticamente la aplicación.

Nodemon es útil en la fase de desarrollo, ya que no necesitamos estar pendientes de reiniciar la aplicación ante cada cambio efectuado.

RESUMEN

Conocimos algunos de los módulos más importantes de Node. Entre ellos, aprendimos a estructurar un sistema con Express, a manejar eventos de tiempo real mediante Socket.IO y a enviar correos electrónicos directamente desde Node. Como vimos, es posible conectarnos a bases de datos como Redis simplemente con instalar un módulo, y además disponer de un administrador para gestionar los datos. Por último, conocimos Nodemon, un módulo muy útil en el proceso de desarrollo de las aplicaciones, que reinicia automáticamente el sistema ante algún cambio.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Un framework MVC define cómo estructurar un sistema?
2. ¿Existen otros frameworks MVC diferentes a Express?
3. ¿Express permite utilizar sesiones, cookies y variables POST y GET?
4. ¿Es posible pasar parámetros a las rutas en Express?
5. ¿Socket.IO permite crear un sistema de chat o juego en línea?
6. ¿Cuáles son los dos métodos fundamentales que se utilizan en Socket.IO?
7. ¿Socket.IO es soportado por navegadores móviles?
8. ¿Es posible enviar correos electrónicos utilizando una cuenta de Gmail mediante Nodemailer?
9. ¿Cuáles son los tres tipos de métodos de transporte que soporta Nodemailer?
10. ¿Nodemon debe instalarse de manera global?¿Por qué?

EJERCICIOS PRÁCTICOS

1. Implemente un sistema de ruteo en el ejemplo de Express para gestionar usuarios, únicamente devolviendo una plantilla.
2. Implemente un sistema de notificaciones con Socket.IO.
3. Modifique el ejemplo de Nodemailer para enviar un correo a varios destinatarios con copia a un tercero.
4. Implemente un sistema de persistencia de usuarios en Redis utilizando Express.
5. Implemente Nodemon en cada ejercicio creado anteriormente.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

Servicios al lector

En esta sección incluimos un completo índice temático para encontrar, de manera sencilla, los conceptos fundamentales que están presentes en el libro.

▾ Índice temático.....312

Índice temático

A

Accesos ... 229
Afinidad de ubicación ... 20
Ajax ... 186
Ajax versus Comet ... 186
Almacenamiento de contenido ... 17
AOF ... 134
App.js ... 283
Aspectos para el diseño ... 23
Autenticación ... 131, 140

B

Bases de datos ... 29
Benchmarks ... 141
Binary JSON ... 35
BSON ... 35

C

Cacheo de módulos ... 248
Cambio de la codificación ... 235
Canales y patrones ... 73
Capacidad del servicio ... 19
Captcha ... 176
Características de Node.js ... 183
Clientes PHP ... 76
Cloud 9 ... 172
Clúster ... 25
Codificación ... 235
Código ... 210
Comandos ... 61
Comandos de npm ... 258
Comet ... 186
Configurar Redis ... 126
Convertir el servidor en un módulo ... 213
Costo ... 15
Creación de módulo ... 249
Creación de una aplicación ... 279
Cuándo no usar Node ... 189

C

Cuándo usar Node ... 188
Cuándo usar NoSQL ... 32

D

Datacenter ... 25
Declarativa ... 174
Demanda de los usuarios ... 17
Desarrollo de sistemas ... 172
Descentralización ... 25
Deshabilitar comandos ... 140
Diferencia entre Node y Apache ... 185
Diseño de un sistema ... 22
Disponibilidad ... 14
Documentos HTML ... 231
Dummy data ... 177

E

El archivo app.js ... 283
Empresas que usan Node ... 190
Escalabilidad ... 18
Escalabilidad horizontal ... 20
Escalabilidad vertical ... 18
Esclavo a maestro ... 132
Esclavo de solo lectura ... 130, 131
Escribir código ... 20
Estructuración de código ... 210
Estructuración y creación de módulos ... 249
Estructurada ... 175
Express ... 277

F

File ... 247
Flow3 ... 177
Formato de los mensajes ... 69
Frameworks MVC ... 276
Funcionamiento de la replicación ... 130
Funcionamientos de Node.js ... 184
Funciones.php ... 112

G
Gestor de paquetes npm 258
GOTO 175
Google Analytics 14
Google File System 31

H
Habilitar AOF 136
Hadoop 33
Hashes 60

I
Imperativa 173
Index.html 90
Instalación de dependencias 280
Instalación de Express 278
Instalar Node 190
Instalar Node en Linux 195
Instalar Node en Mac 196
Instalar Node en Windows 197
Instalar Redis 47

L
Ley de Moore 18
Línea de comandos 126
Lists 56

M
Manejabilidad 15
Manejadores 219
Manejo de peticiones POST 238
Mensajes 69
Mini-twt 142
Módulo 213
Módulo File 247
Módulos 246
Módulos propios de Node 247
Mouseover 184
MVC 276

N
Node en Linux 195
Node en Mac 195
Node en Windows 195
Node y Apache 185
Node.js 182
Node_modules 248
Node_redis 302
Nodemailer 298
Nodemon 306
NoSQL 29

O
Onclick 184
Orientación a aspectos 176
Orientación a eventos 178
Orientación a objetos 172

P
Package.json 267
Paquetes npm 258
Paradigmas 172
Parámetros 126
Pasar parámetros 126
Patrón de coincidencia 71
Persistencia 132
Persistencia mediante AOF 134
Persistencia mediante RDB 133
PHPRedis 77
PHPRedisAdmin 82
POST 238
Predis 76
Primeros pasos con Node 202
Programación declarativa 174
Programación estructurada 175
Programación imperativa 173
Programación orientada a eventos 178
Programación orientada a aspectos 176
Programación orientada a objetos 172
Prueba de la aplicación 281
Publicación 68

R
RDB 133
README.md 265

R

Recursos particionados 24
Redis 44
Redis Commander 305
Redis como caché 128
Registro de accesos 229
Rendimiento 15
Rendimiento de Ajax 187
Rendimiento de Comet 187
Rendimiento de Redis 141
REPL 202
Replicación 129
Replicación con Redis 130
Response 211
Richard Stallman 222
Ruta 232
Ruteo 215, 234
Ruteo por defecto 234

S

SaaS 204
Salida.html 231
Script.js 103
SD 26
Seguridad 139
Seguridad de acceso 139
Servidor 210
Servidor de documentos HTML 231
Servidor.js 205
Servidor Nginx 225
Servidor web 210
Servidor web iniciado 217
Sets 57
Shared Disk 26
Shared Memory 26
Shared Nothing 27
Sidewiki 221
Sistemas conmutables 24
Sistemas escalables 14
Sistemas intercambiables 24

S

SM 26
Smalltalk 172
SN 27
Socket.IO 291
Solaris 195
Soluciones de Node.js 183
Sorted sets 58
SQL 29
Storyfy 194
Strings 56
Style.css 97
Suscripciones coincidentes 73
Suscripción 68

T

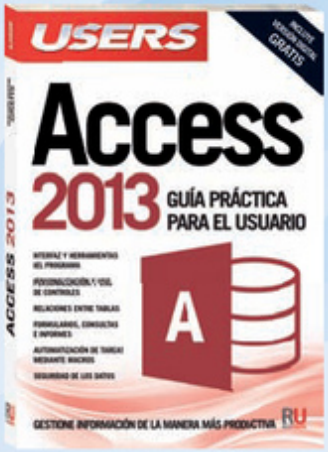
Términos 254
Tiempo de ejecución 127
Tiempos de espera 23
Tipos de bases de datos NoSQL 29
Tipos de datos 55
Tipos de persistencia 132
Traductor.js 251
Transloadit 194

U

Unicode 235
Uso de Node 189
Uso de NoSQL 32
UTF-8 235
Utilizar Redis 46

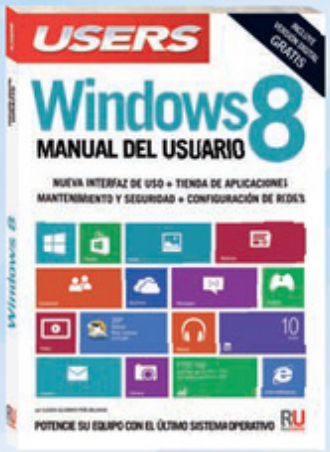
W

Warm() 234
Web semántica 227
Websockets 188
Windows Azure 191
Write 211
WriteHead() 211



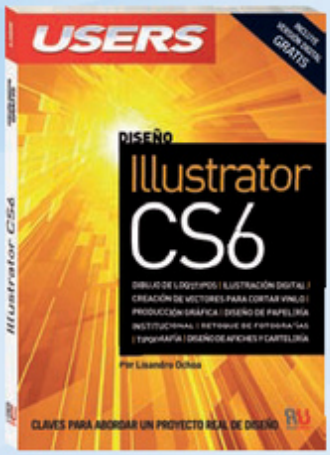
Aproveche todo el potencial del programa y obtenga información clave para la toma de decisiones.

→ 320 páginas / ISBN 978-987-1949-17-5



Conozca las claves y los consejos necesarios para aprovechar al máximo todo el potencial del sistema operativo más utilizado.

→ 320 páginas / ISBN 978-987-1949-09-0



La mejor guía a la hora de generar piezas de comunicación gráfica, ya sean para web, dispositivos electrónicos o impresión.

→ 320 páginas / ISBN 978-987-1949-04-5

Aprenda a simplificar su trabajo, convirtiendo sus datos en información necesaria para solucionar diversos problemas cotidianos.

→ 320 páginas / ISBN 978-987-1949-08-3

Acceda a consejos útiles y precauciones a tener en cuenta al afrontar cualquier problema que pueda presentar un equipo.

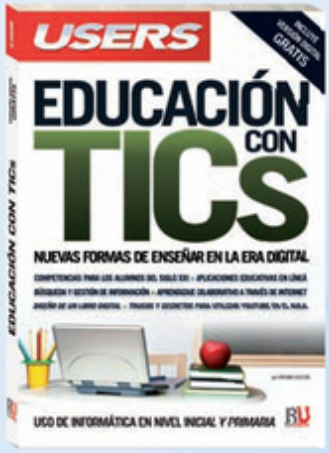
→ 320 páginas / ISBN 978-987-1949-02-1

El libro indicado para enfrentar los desafíos del mundo laboral actual de la mano de un gran sistema administrativo-contable.

→ 352 páginas / ISBN 978-987-1949-01-4

Un libro ideal para ampliar la funcionalidad de las planillas de Microsoft Excel, desarrollando macros y aplicaciones VBA.

→ 320 páginas / ISBN 978-987-1857-99-9



Un libro para maestros que busquen dinamizar su tarea educativa integrando los diferentes recursos que ofrecen las TICs.

→ 320 páginas / ISBN 978-987-1857-95-1

Libro ideal para introducirse en el mundo de la maquetación, aprendiendo técnicas para crear verdaderos diseños profesionales.

→ 352 páginas / ISBN 978-987-1857-74-6

Esta obra reúne todas las herramientas de programación que ofrece Unity para crear nuestros propios videojuegos en 3D.

→ 320 páginas / ISBN 978-987-1857-81-4

Esta obra nos enseña sobre el diseño y prueba de circuitos electrónicos, sin necesidad de construirlos físicamente.

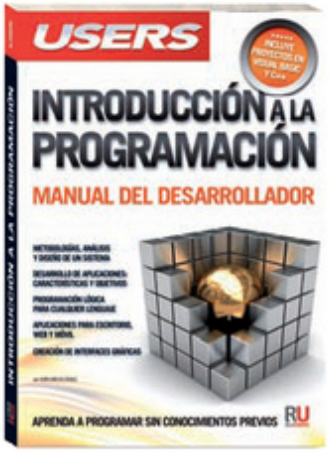
→ 320 páginas / ISBN 978-987-1857-72-2

Obra imperdible para crear infraestructura virtual con las herramientas de Vmware según los requerimientos de cada empresa.

→ 320 páginas / ISBN 978-987-1857-71-5

Esta obra reúne todos los conocimientos teóricos y prácticos para convertirse en un técnico especializado en Windows.

→ 320 páginas / ISBN 978-987-1857-70-8

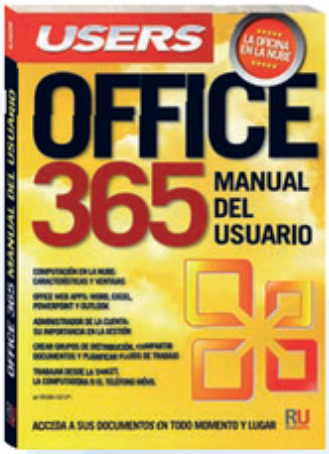


Libro ideal para iniciarse en el mundo de la programación y conocer las bases necesarias para generar su primer software.

→ 384 páginas / ISBN 978-987-1857-69-2

Presentamos una obra fundamental para aprender sobre la arquitectura física y el funcionamiento de los equipos portátiles.

→ 352 páginas / ISBN 978-987-1857-68-5

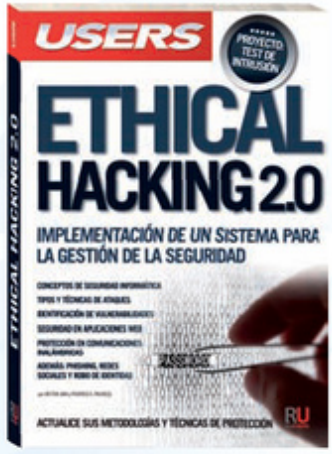


Una obra ideal para aprender todas las ventajas y servicios integrados que ofrece Office 365 para optimizar nuestro trabajo.

→ 320 páginas / ISBN 978-987-1857-65-4

Esta obra presenta las mejores aplicaciones y servicios en línea para aprovechar al máximo su PC y dispositivos multimedia.

→ 320 páginas / ISBN 978-987-1857-61-6



Esta obra va dirigida a todos aquellos que quieran conocer o profundizar sobre las técnicas y herramientas de los hackers.

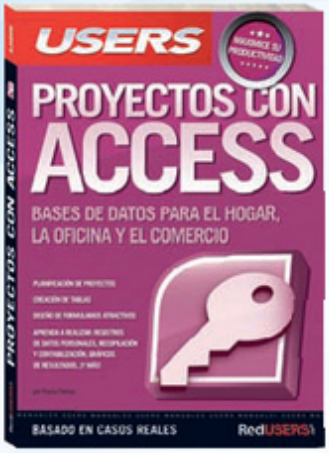
→ 320 páginas / ISBN 978-987-1857-63-0

Este libro se dirige a fotógrafos amateurs, aficionados y a todos aquellos que quieran perfeccionarse en la fotografía digital.

→ 320 páginas / ISBN 978-987-1857-48-7

En este libro encontraremos una completa guía aplicada a la instalación y configuración de redes pequeñas y medianas.

→ 320 páginas / ISBN 978-987-1857-46-3



Esta obra está dirigida a todos aquellos que buscan ampliar sus conocimientos sobre Access mediante la práctica cotidiana.

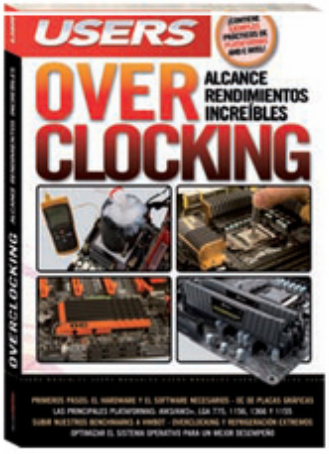
→ 320 páginas / ISBN 978-987-1857-45-6

Este libro nos introduce en el apasionante mundo del diseño y desarrollo web con Flash y AS3.

→ 320 páginas / ISBN 978-987-1857-40-1

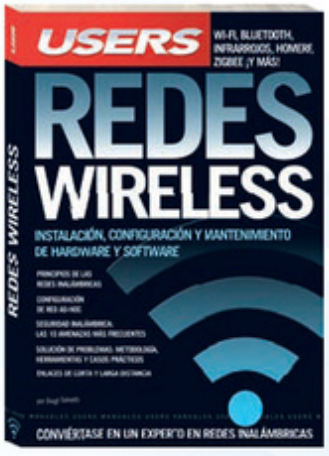
Esta obra presenta un completo recorrido a través de los principales conceptos sobre las TICs y su aplicación en la actividad diaria.

→ 320 páginas / ISBN 978-987-1857-41-8



Este libro está dirigido tanto a los que se inician con el overclocking, como a aquellos que buscan ampliar sus experiencias.

→ 320 páginas / ISBN 978-987-1857-30-2



Este manual único nos introduce en el fascinante y complejo mundo de las redes inalámbricas.

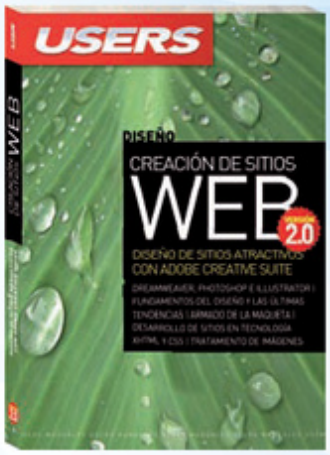
→ 320 páginas / ISBN 978-987-1773-98-5

Esta increíble obra está dirigida a los entusiastas de la tecnología que quieran aprender los mejores trucos de los expertos.

→ 320 páginas / ISBN 978-987-1857-01-2

Esta obra se encuentra destinada a todos los desarrolladores que necesitan avanzar en el uso de la plataforma Adobe Flash.

→ 320 páginas / ISBN 978-987-1857-00-5

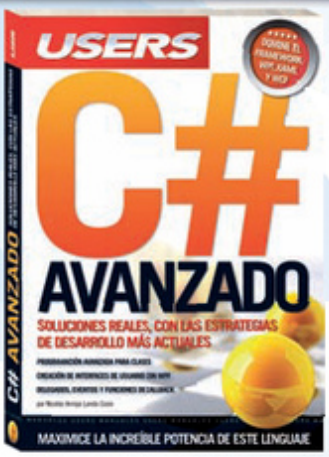


Un libro clave para adquirir las herramientas y técnicas necesarias para crear un sitio sin conocimientos previos.

→ 320 páginas / ISBN 978-987-1773-99-2

Una obra para aprender a programar en Java y así insertarse en el creciente mercado laboral del desarrollo de software.

→ 352 páginas / ISBN 978-987-1773-97-8

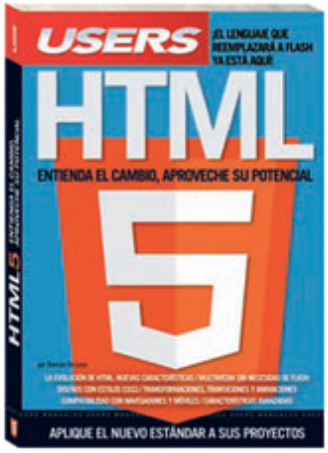


Este libro presenta un nuevo recorrido por el máximo nivel de C# con el objetivo de lograr un desarrollo más eficiente.

→ 320 páginas / ISBN 978-987-1773-96-1

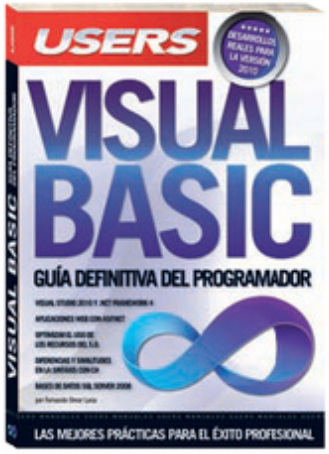
Esta obra presenta todos los fundamentos y las prácticas necesarios para montar redes en pequeñas y medianas empresas.

→ 320 páginas / ISBN 978-987-1773-80-0



Una obra única para aprender sobre el nuevo estándar y cómo aplicarlo a nuestros proyectos.

→ 320 páginas / ISBN 978-987-1773-79-4



Un libro imprescindible para aprender cómo programar en VB.NET y así lograr el éxito profesional.

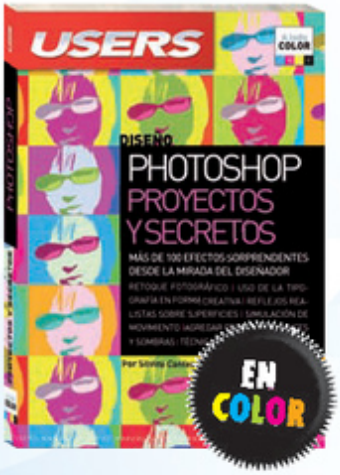
→ 352 páginas / ISBN 978-987-1773-57-2

Una obra para aprender los fundamentos de los microcontroladores y llevar adelante proyectos propios.

→ 320 páginas / ISBN 978-987-1773-56-5

Un manual único para aprender a desarrollar aplicaciones de escritorio y para la Web con la última versión de C#.

→ 352 páginas / ISBN 978-987-1773-26-8



Un manual imperdible para aprender a utilizar Photoshop desde la teoría hasta las técnicas avanzadas.

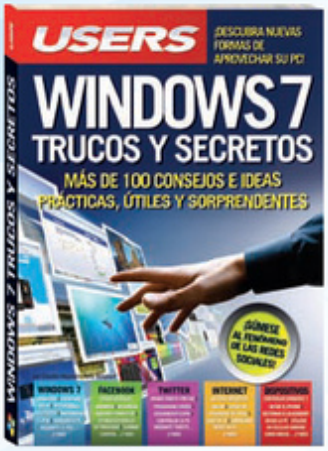
→ 320 páginas / ISBN 978-987-1773-25-1

Una obra imprescindible para quienes quieran conseguir un nuevo nivel de profesionalismo en sus blogs.

→ 352 páginas / ISBN 978-987-1773-18-3

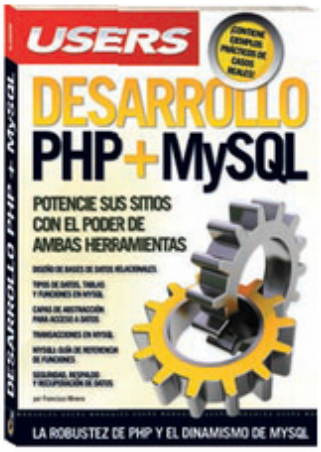
Un libro único para ingresar en el apasionante mundo de la administración y virtualización de servidores.

→ 352 páginas / ISBN 978-987-1773-19-0



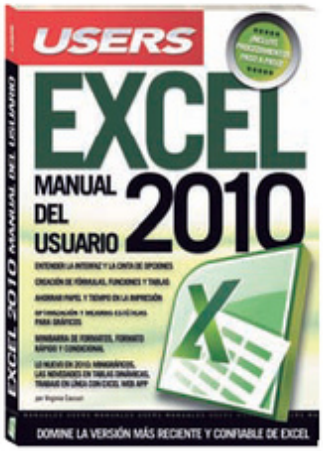
Esta obra permite sacar el máximo provecho de Windows 7, las redes sociales y los dispositivos ultraportátiles del momento.

→ 352 páginas / ISBN 978-987-1773-17-6



Este libro presenta la fusión de las dos herramientas más populares en el desarrollo de aplicaciones web: PHP y MySQL.

→ 432 páginas / ISBN 978-987-1773-16-9



Este manual va dirigido tanto a principiantes como a usuarios que quieran conocer las nuevas herramientas de Excel 2010.

→ 352 páginas / ISBN 978-987-1773-15-2

Esta guía enseña cómo realizar un correcto diagnóstico y determinar la solución para los problemas de hardware de la PC.

→ 320 páginas / ISBN 978-987-1773-14-5

Este libro brinda las herramientas para acercar al trabajo diario del desarrollador los avances más importantes en PHP 6.

→ 400 páginas / ISBN 978-987-1773-07-7

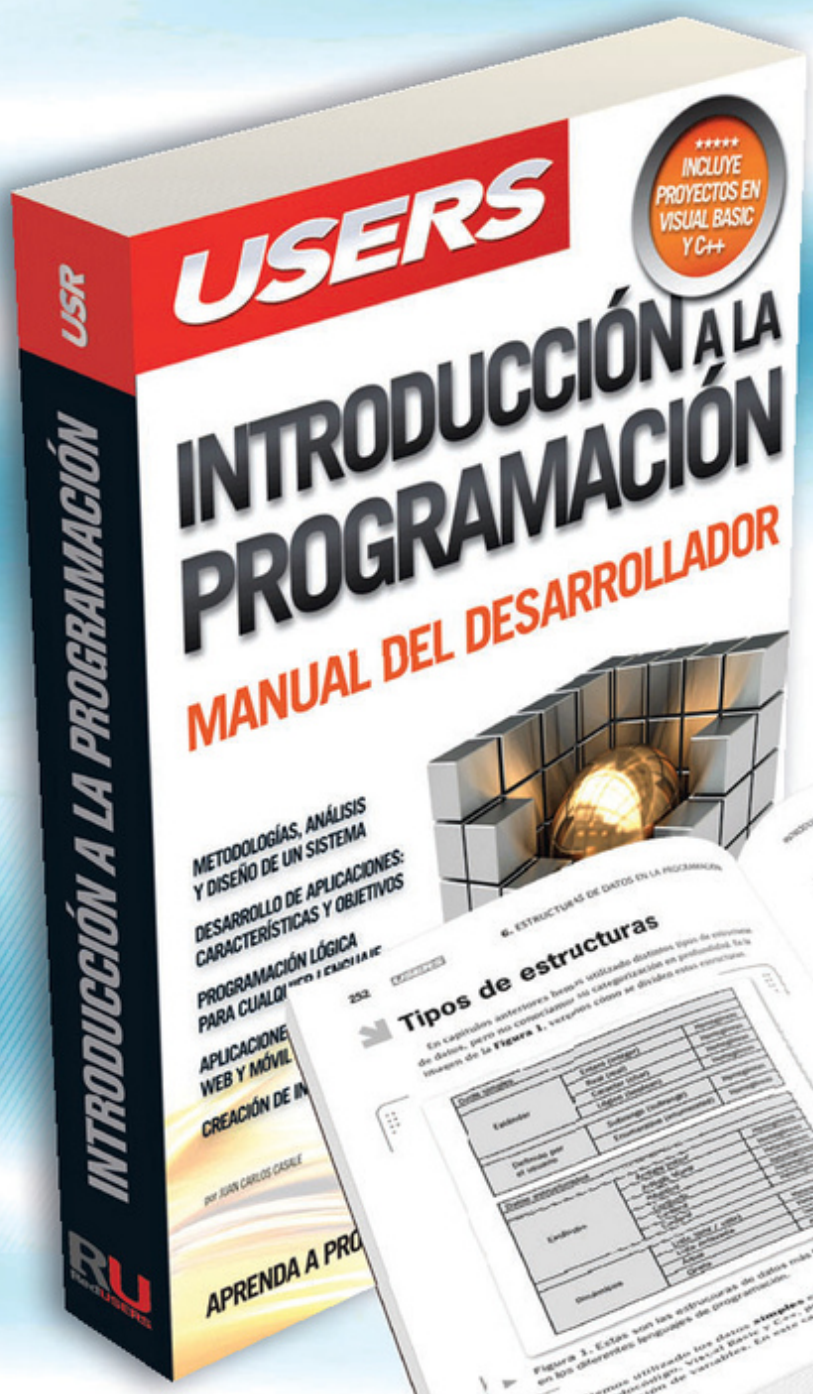
Un libro imprescindible para quienes quieran aprender y perfeccionarse en el dibujo asistido por computadora.

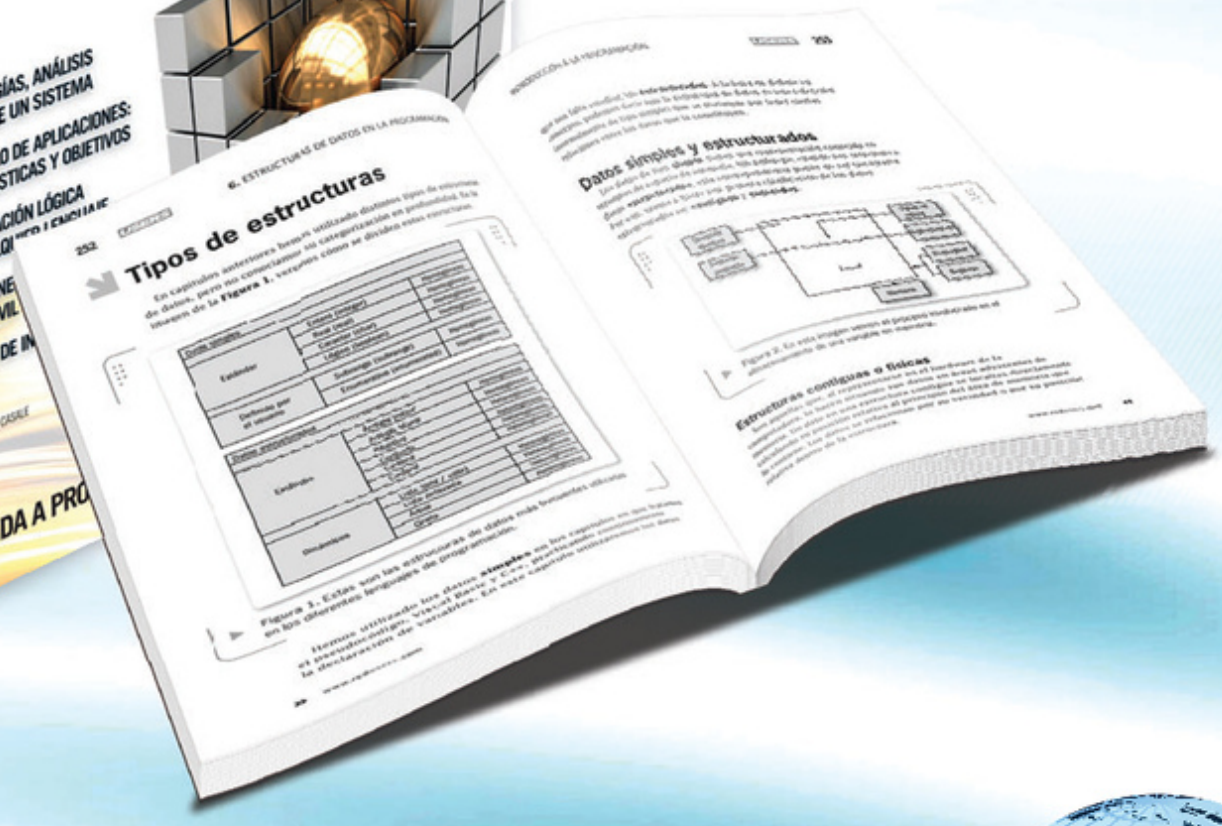
→ 384 páginas / ISBN 978-987-1773-06-0

Este libro único nos permitirá alcanzar el grado máximo en el manejo de Windows: Administrador Profesional.

→ 352 páginas / ISBN 978-987-1773-08-4

CONÉCTESE CON LOS MEJORES LIBROS DE COMPUTACIÓN





Presentamos un libro ideal para quienes quieran iniciarse en el mundo de la programación y generar su primer software.
A través de explicaciones sencillas, guías visuales y procedimientos paso a paso, el lector adquirirá conocimientos sólidos y aprenderá a programar de manera eficiente.

» DESARROLLO
» 384 PÁGINAS
» ISBN 978-987-1857-69-2

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

MÁS INFORMACIÓN / CONTÁCTENOS

usershop.redusers.com +54 (011) 4110-8700 usershop@redusers.com

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

USERS

INCLUYE VERSIÓN DIGITAL GRATIS

SISTEMAS WEB ESCALABLES

Este material está dirigido a desarrolladores interesados en implementar tecnologías de última generación. En sus páginas, el autor explica en qué consiste la escalabilidad y cómo crear sistemas de alto rendimiento.

A lo largo de los capítulos se verá cómo desarrollar sistemas escalables mediante ejemplos con PHP, uno de los lenguajes más populares de la web, y utilizando una de las bases de datos NoSQL más veloces del momento: Redis. Al mismo tiempo, se trabajará con Node.js, la tecnología que está revolucionando la experiencia de los usuarios en sistemas de tiempo real.

Al finalizar la lectura el lector podrá aplicar todo lo aprendido en el desarrollo de su propia red social, mediante una guía paso a paso con indicaciones detalladas.

> Implementando sistemas capaces de evolucionar en el tiempo obtenemos la flexibilidad necesaria para responder al crecimiento del tráfico.

EN ESTE LIBRO APRENDERÁ:

- **Introducción a los sistemas escalables:** tipos de escalabilidad y aspectos a considerar en el diseño. Bases de datos NoSQL: MongoDB, Cassandra y Redis.
- **Redis:** uso conveniente e instalación. Tipos de datos y comandos. Clientes disponibles para PHP. Administración: replicación, persistencia y opciones de seguridad.
- **Paradigmas de programación:** orientada a objetos, imperativa, declarativa, estructurada, orientada a aspectos y orientada a eventos.
- **Node.js:** características y soluciones que ofrece. Cuándo conviene utilizar esta tecnología y cómo instalarla. Estructuración del código, manejo de peticiones y gestión de módulos.
- **Desarrollo de una red social:** aplicación práctica de la teoría aprendida en un sistema que refleja todas las características que deben tener los escalables.

Carlos Benitez es Analista de Sistemas especializado en tecnologías web y entusiasta en la investigación e implementación de nuevas tecnologías. Ha dictado seminarios de desarrollo de sistemas web, trabaja como UNIX Developer y actualmente se encuentra desarrollando un sistema SaaS de gestión de tareas.

NIVEL DE USUARIO
Intermedio / Avanzado

CATEGORÍA
Desarrollo - Internet - Empresas

ISBN 978-987-1949-20-5

RU RedUSERS

REDUSERS.com
En nuestro sitio podrá encontrar noticias relacionadas y también participar de la comunidad de tecnología más importante de América Latina.

PROFESOR EN LÍNEA
Ante cualquier consulta técnica relacionada con el libro, puede contactarse con nuestros expertos: profesor@redusers.com.