PROGRAMACIÓN WEB

PARA PROGRAMADORES

HTTP HTML CSS JAVASCRIPT JQUERY PHP MYSQL

MAURO GULLINO

PROGRAMACIÓN WEB

Este libro introduce los *conceptos Web fundamentales*, haciendo uso de las siguientes tecnologías y presentando la relación entre ellas: HTTP, HTML, CSS, JavaScript, jQuery, PHP y MySQL.

Como su nombre lo indica, es un libro pensado para quienes ya tienen *alguna* experiencia en en el desarrollo de software. Ya sea para programadores que desean incursionar en Web o para alumnos en una carrera informática, este libro fue pensado para ellos.

Con ejercicios y bibliografía adicional en cada capítulo, el lector encontrará rápidamente las herramientas que necesita para comenzar a implementar aplicaciones de pequeña y mediana escala con un espíritu moderno. Se han incluido apéndices que presentan temas adicionales como JSON y AJAX.

Mauro Gullino es Maestrando en Ingeniería del Software (UNLP) y Licenciado en Comunicación Visual (UCES). Tiene una amplia trayectoria como desarrollador, consultor y docente en diversas universidades y empresas argentinas.

ISBN 978-987-33-4505-0

Sistema de consulta para lectores:
maurogullino.com.ar/pwpp

Gullino, Mauro
Programación web para programadores. - 1a ed. - Buenos Aires : el autor, 2014.
192 p. ; 25x18 cm.

ISBN 978-987-33-4505-0

1. Programación. 2. Internet. I. Título
CDD 004

Fecha de catalogación: 20/02/2014

© Mauro Gullino, 2014.

maurogullino.com.ar/pwpp

Las familias tipográficas utilizadas en el interior son Poly (*Nicolás Silva*), Exo 2 (*Natanael Gama*) y Envy Code R (*Damien Guard*).

Hecho el depósito que prevé la ley 11.723.

Reservados todos los derechos.

Impreso en Argentina.

PROGRAMACION WEB

PARA PROGRAMADORES

MAURO GULLINO

OBJETIVOS

Este material fue pensado para introducir los conceptos fundamentales de la programación web. Estos conceptos *básicos* constituyen justamente la *base* para entender la práctica de la vida profesional real en el ámbito de la programación web y su complejidad inherente. Con esto queremos anticipar que no es nuestro objetivo ser *exhaustivos*. Ya existe muy buena bibliografía sobre cada uno de los temas. Lo que no hemos encontrado es, justamente, una compilación introductoria que ofrezca un **panorama general**. Y esa tarea es la que nos hemos propuesto con esta obra.

El público principal que se tiene en mente es alumnos universitarios, pero este libro también puede ser utilizado por profesionales con experiencia en programación y que deseen incorporarse al mundo web. Con esto queremos decir que damos por descontada cierta experiencia en la informática en general y la programación en particular. No haremos aquí introducción de, por ejemplo, planteo de algoritmos para resolución de problemas. Nos focalizaremos en personas con experiencia previa *mínima* en lenguajes como C o Pascal.

Nos atendremos a las particularidades de la web y a sentar las bases para que cada uno pueda continuar aprendiendo por sí mismo, un desafío que sin dudas marca la actualidad y el futuro de los profesionales del software.

Este material es fundamentalmente apoyo bibliográfico para los cursos que el autor desarrolla en su actividad docente, pero fue pensado para funcionar también en su ausencia, por medio de la utilización de un lenguaje claro, directo, incluyendo ejercicios y referencias para tener un camino por donde continuar.

La presentación y el desarrollo de los temas toma en consideración que simultáneamente el lector irá experimentando frente a una computadora, a medida que avanza con la lectura del material. Es muy importante realizar la práctica, ya que *leer* y comprender código es una actividad cognitiva muy distinta a *escribirlo*.

HERRAMIENTAS

Las distintas tecnologías que se utilizan son sencillamente vehículos para concretar de manera práctica los ejercicios y programas pero, desde ya, no representan las únicas posibles y en parte están influenciadas por las preferencias personales. Es en este sentido que decimos que nuestro objetivo principal es presentar los fundamentos de la programación web, es decir, *"cómo funciona"* y no tecnologías ni versiones particulares de algún producto o programa. Para presentar esos fundamentos hemos elegido estas tecnologías:

- **Navegador web:** es una de las herramientas que más utilizaremos a lo largo de los diferentes temas, por lo que es muy importante contar con un navegador con el que nos sintamos a gusto y que nos provea de las herramientas y ayudas apropiadas. Elegimos Google Chrome o Mozilla Firefox.
- **Editor de texto:** es definitivamente necesario contar con un buen editor de código fuente. No es imprescindible para nuestros objetivos usar un IDE completo como

Eclipse o NetBeans. Programas como SublimeText, Notepad++ o gEdit son muy buenas opciones, sin complejidades inútiles por el momento.

- **Servidor web:** para los capítulos de PHP y MySQL necesariamente tendremos que hacer uso del software pertinente al lado servidor. Personalmente preferimos el paquete preconfigurado WampServer para instalaciones permanentes, o su equivalente Server2Go como opción portable. La elección misma de este lenguaje y esta base de datos se debe a su nivel de popularidad y a la mínima configuración necesaria para comenzar a trabajar. Debe decirse que existen otros lenguajes y bases de datos también muy utilizados en la práctica y que pueden ser perfectamente utilizados para enseñanza. Este es el punto donde más diversidad de tecnologías se puede encontrar y, por lo tanto, se debe realizar una elección y esforzarnos por concentrar la atención en los *conceptos* y no en las *particularidades* de estas tecnologías, en definitiva circunstanciales.

ORGANIZACIÓN

Cada capítulo se extiende sobre una tecnología en particular y corresponde a un concepto del mundo de la programación web. Si bien esta forma de tratar los temas es, entendemos, la más conveniente para *aprender*, es verdad que la web no funciona compartimentada de esta forma. Justamente, la dificultad inherente a la programación web tiene que ver con la cantidad de tecnologías que funcionan *a la vez*, tanto en los navegadores como en los servidores.

Por esta razón, es necesario aclarar que al principio se puede tener la sensación de estar haciendo cosas muy primitivas, feas o inútiles… y esto es totalmente verdad. No creemos que sea posible hacer ejemplos que se *parezcan* a las aplicaciones web reales sin antes emprender un pequeño viaje por cada una de las tecnologías involucradas. Es decir, no nos parece realmente productivo a largo plazo hacerlo de otra manera. Este esfuerzo por compartimentar los temas es fruto de varios años de experiencia en la enseñanza de la programación web.

Si no se comprenden las bases de cada tecnología, las razones de su diseño y los problemas que intenta abordar creemos que será muy difícil encarar luego de manera eficiente los problemas de las aplicaciones y clientes reales. Tal vez sea un poco frustrante al principio, pero en el mediano plazo es muchísimo más eficaz abordar con cierto detenimiento cada tema para *luego* integrarlos en una aplicación real y, lo más importante, aplicar el conocimiento a situaciones nuevas y distintas.

Trataremos de utilizar los términos en castellano siempre que sea posible. Cuando los términos o nombres en inglés sean imprescindibles, serán mencionados en ese idioma. Muchas palabras propias de los protocolos y estándares serán repetidas y utilizadas en distintos contextos y lenguajes de programación, por lo que se hace indispensable su mención en inglés ya que no podrán ser traducidas en el código.

Sin más introducción comencemos, obviamente, por el principio. ¡Bienvenidos!

1. HTTP

El protocolo HTTP se creó a principios de la década de los '90 en los laboratorios del CERN, en Francia. Su objetivo es definir el **transporte** de documentos web a través de las entonces nacientes redes de computadoras. Su sigla proviene de *HyperText Transfer Protocol* (protocolo para transferencia de hipertexto), en donde *hipertexto* puede ser simplemente entendido por documento web, más puntualmente **documentos HTML**, o como los llamamos habitualmente: **páginas web**. Por lo tanto nos detendremos a observar cómo funciona este protocolo que se utiliza para controlar el viaje que cada página web emprende cada vez que navegamos.

El conocimiento de este protocolo es sumamente importante. Sin temor a exagerar, este sea quizá el tema más importante de este libro. El protocolo HTTP es el que determina el funcionamiento de la web y es el que funda las diferencias con el resto de la programación. HTTP es lo que nos define como *programadores web*. El conocimiento que implica este protocolo es de muy largo plazo porque es una tecnología que ha cambiado muy poco y que demuestra ser absolutamente funcional: la versión que utilizamos hoy en día es la 1.1, correspondiente a la última revisión aprobada en 1999.

1.1 LA WEB COMO TRÁFICO

Una de las cuestiones más radicales que implica trabajar programando en web es pensar en términos de **tráfico de red**. Es tan radical como fundamental porque, como dijimos, es lo que diferencia la programación web del resto de la programación. Tradicionalmente utilizamos los flujos de entrada y salida estándar para comunicarnos con el usuario, por ejemplo, en lenguajes como C, Pascal o Java en cursos introductorios de programación. Habiendo avanzado un poco más suelen introducirse las interfaces gráficas para escritorio, por ejemplo en VB.net, que un usuario utilizará en su computadora por medio del teclado y el mouse.

Pero al llegar a la programación web nos encontramos con sistemas que son necesariamente multiplataforma (todos tenemos distintos navegadores y distintas computadoras), con gran cantidad de asincronismo (programaremos tanto para el navegador del usuario como en el servidor) e ineludiblemente multiusuario y concurrentes (muchas personas están ejecutando nuestro programa simultáneamente). Estas son características que surgen del tráfico web, de la misma arquitectura, por lo que son conceptos medulares a la programación web.

Comprender este tráfico es fundamental para programar sistemas web. No habrá aplicación web sin tráfico de red, aunque se trate de una intranet muy pequeña o una aplicación pensada para nosotros mismos. Siempre tendremos tráfico cuando trabajemos en web, y este tráfico será HTTP.

Es importante destacar el uso que hacemos de la palabra **web**. Siempre que hablemos de web estaremos haciendo referencia al tráfico HTTP. Este tráfico HTTP es tráfico que

tiene como destino el **puerto 80** del protocolo TCP. Los servidores web están escuchando en este puerto 80. Este tráfico HTTP es entonces lo que llamamos *web*. Este tráfico web es encausado por la red de routers, empresas y subredes que llamamos ampliamente **Internet**.

Destacamos esto porque puede haber más ramas de la programación que trabajen sobre tráfico de Internet pero que no sean *web* específicamente. Por ejemplo, el protocolo FTP, los servicios de voz sobre IP, toda la infraestructura de correo electrónico o la resolución de DNS constituyen muestras de tráfico de red que no es web, ya que no es tráfico HTTP sino de otros protocolos con otros puertos. Decimos entonces que la web "corre" sobre Internet, utilizándola como infraestructura de comunicación para llegar de un punto al otro del planeta. El tráfico web es entonces una *parte* del tráfico total de Internet.

1.2 ARQUITECTURA CLIENTE-SERVIDOR

El protocolo HTTP responde a la arquitectura **cliente-servidor**. Esto quiere decir que el protocolo define dos roles diferentes que serán cumplidos por dos piezas de software distintas. Tendremos entonces los **clientes web** y los **servidores web**. Estos dos roles tienen distintas responsabilidades. Cuando un programa cliente dialoga con un programa servidor tenemos una comunicación HTTP, y toda esta comunicación y conjunto de responsabilidades es la que está definida en el protocolo.

Figura 1.1: HTTP es un protocolo cliente-servidor

En la figura 1.1 observamos que el cliente y el servidor están separados por una línea punteada. Esto intenta poner el foco en la distancia geográfica. Es decir que estos dos programas, el cliente y el servidor, están ejecutándose en computadoras distintas. Llamaremos a estas dos computadoras como **lado cliente** y **lado servidor**, o sencillamente *cliente* y *servidor*. Es importante visualizar que son dos computadoras diferentes. La única manera de intercambiar información que tienen estos dos programas es a través de mensajes HTTP definidos en el protocolo.

Es muy importante tener en cuenta que los roles no son intercambiables. Los clientes solo pueden (y deben) hacer las cosas que el protocolo define para los clientes, y lo mismo ocurre con los servidores. Cada rol tiene sus responsabilidades.

Hay distintos fabricantes de software que producen y comercializan clientes y servidores web. Los clientes web son muy conocidos por nosotros porque los utilizamos todos los días. Son llamados comúnmente **navegadores**. Por lo tanto, los navegadores que usamos en nuestras computadoras son, técnicamente, clientes web. Es decir que son programas que fueron creados para cumplir con lo que el protocolo HTTP exige para los

clientes web. Todas las empresas que crean navegadores deberán leer el protocolo y hacer un programa que cumpla las responsabilidades.

Lo mismo sucede con los servidores web que, salvo que trabajemos como programadores web, no son programas con los que estemos familiarizados normalmente. Las personas que navegan no trabajan de manera directa con estos programas, por lo que parecen no existir desde el lado cliente. Esto se debe a que el navegador hace el trabajo por nosotros, es decir, traduce nuestras acciones (teclas y clicks) en mensajes del protocolo HTTP que envía hacia un servidor web. Con el correr de nuestro estudio iremos "perdiendo la inocencia", y es uno de nuestros objetivos entender que la web es esencialmente HTTP, aunque "no lo veamos" de manera directa en nuestro uso cotidiano. Profundizaremos en el funcionamiento de HTTP para entender el gran trabajo que realizan los navegadores y servidores que luego programaremos.

Si todos los fabricantes de software se atienen a lo convenido en el protocolo, se tiene el gran beneficio de que los clientes y los servidores son **interoperables**. Esto es una realidad hoy en día, ya que prácticamente no encontramos diferencias al navegar con distintos clientes y no notamos cambios en los distintos sitios web que navegamos, aunque cada uno utilice un software servidor de distinto fabricante.

1.3 TIPOS DE PAQUETES

El tráfico HTTP se compone de **paquetes**. Estos paquetes o mensajes se dividen en dos grupos: **peticiones** (*requests*) y **respuestas** (*responses*). No hay otros elementos en el tráfico HTTP más que estos paquetes que viajan entre el cliente y el servidor.

Las peticiones son los paquetes que el cliente envía al servidor. Las respuestas son los paquetes que el servidor envía al cliente como consecuencia. Siempre la comunicación HTTP comienza en el lado cliente con una petición y concluye con una respuesta que proviene del lado servidor. Cada petición, si no hay problemas de comunicación, tendrá siempre una respuesta, y cada respuesta existirá porque antes hubo una petición. A este par petición-respuesta se lo llama **transacción HTTP**.

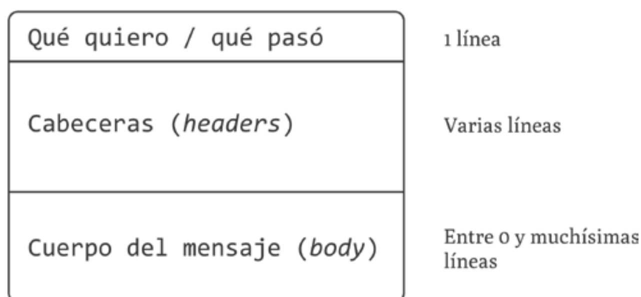

Figura 1.2: Estructura de los paquetes HTTP

En la figura 1.2 podemos observar un esquema que ilustra la estructura de los paquetes HTTP. Los paquetes, tanto peticiones como respuestas, tienen la misma estructura y se componen de tres partes.

La primera parte está constituida por una sola línea que indica en el caso de las peticiones qué es lo que estamos pidiendo al servidor. En el caso de las respuestas tenemos información sobre qué sucedió con nuestro pedido.

La segunda parte está compuesta de varias líneas que son llamadas **cabeceras** (*headers*). Estas cabeceras son información sobre el paquete (ya sea petición o respuesta) y contienen, por ejemplo, la fecha en la que se generó el paquete, el sitio web al que pertenece, el navegador que estamos usando, etc. Existen muchas cabeceras definidas en el protocolo HTTP. Las cabeceras que encontramos en las peticiones son distintas de las que encontramos en las respuestas. Por el momento no entraremos en más detalle; todas las cabeceras pueden consultarse en el protocolo.

La tercera y última parte del paquete, que llamamos **cuerpo** (*body*) es la que contiene efectivamente la información que está siendo transportada. Hemos visto que el protocolo HTTP fue creado para transportar páginas web. En el cuerpo de las respuestas están entonces las páginas web que solicitamos con las peticiones. Más adelante encontraremos que son muchos los recursos que deben transportarse entre clientes y servidores. Toda esta información es transportada en el cuerpo de las respuestas.

Veamos un ejemplo con paquetes reales. Estamos navegando en un sitio web que contiene recetas de cocina. Nos concentraremos en el lado cliente, es decir, en el navegador. En el menú lateral del sitio web encontramos categorías de recetas. Presionamos sobre un *link* para ir a la página con recetas que pertenecen al grupo "pastas".

```
GET /pastas.html HTTP/1.1
--------------------------------------------------
Host: www.todorecetas.net
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES; rv:1.9.....
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-es,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Cache-Control: max-age=0
--------------------------------------------------
```

Figura 1.3: Petición real enviada por un cliente

En la figura 1.3 observamos la petición que generó nuestro navegador. Aquí vemos en funcionamiento una **responsabilidad** que tienen los clientes: a partir de nuestros clicks, generar peticiones para enviar al servidor. Las líneas punteadas no son parte del paquete: las hemos incluido sólo para visualizar las partes más claramente.

La primera línea contiene la palabra "GET", el nombre de la página que queremos ver (hemos hecho click en pastas) y la versión del protocolo que el cliente prefiere utilizar. Decimos entonces que nuestro click sobre un link fue traducido por el navegador en una *petición tipo GET*.

En la segunda parte del paquete encontramos las cabeceras. Allí vemos que nuestro navegador es un Mozilla Firefox corriendo en un sistema Windows (*User-Agent*). Además vemos que el lenguaje preferido por el usuario es castellano (*Accept-Language*). Estas

cabeceras fueron armadas por nuestro navegador en función de nuestras preferencias y de su propia información. Existen muchas más cabeceras y puede profundizarse mucho en cada una de ellas, pero por el momento es suficiente.

La tercera parte (el cuerpo) de este paquete está vacía. Recordemos que esta parte de los paquetes se utiliza para enviar los datos que están siendo transportados. Por definición, el cuerpo de las peticiones GET siempre está vacío. Razonemos de esta forma: si estamos enviando un mensaje al servidor para que éste nos devuelva una página web, ¿qué incluiremos en el cuerpo del mensaje? La respuesta es: nada. *El cuerpo de las peticiones GET siempre está vacío.* A continuación veremos que hay más peticiones además de GET, en donde esto no siempre es así.

Una vez que la petición haya llegado al servidor, será procesada y, luego de otro tiempo de tránsito el navegador recibirá una respuesta. Pensemos en que estos paquetes tardan cierto tiempo en llegar al servidor y viceversa. Es importante pensar en este tiempo para estar atentos del **asincronismo** y de que son computadoras diferentes.

```
HTTP/1.1 200 OK
-------------------------------------
Date: Fri, 05 Aug 2013 11:23:24 GMT
Server: Apache
X-Powered-By: PHP/5.2.13
Keep-Alive: timeout=1, max=64
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
-------------------------------------
(en el cuerpo está la página que veo en el navegador, bytes...)
```

Figura 1.4: Respuesta real contestada por el servidor

En la figura 1.4 observamos la respuesta que corresponde a la petición anterior. Esta respuesta fue generada por el servidor, enviada y recibida en el lado cliente. Aquí tenemos una responsabilidad que el protocolo HTTP marca para los servidores web: contestar a las peticiones que envían los clientes.

En la primera parte encontramos la versión del protocolo, el número "200" y un breve mensaje "OK". Esto quiere decir que lo que hemos solicitado fue encontrado y nos está siendo enviado con esta respuesta.

En la segunda parte podemos observar las cabeceras, que son distintas a las cabeceras de la petición, y que tienen información sobre el paquete y sobre el servidor en sí. Podemos ver que el software del lado servidor se llama Apache (*Server*), la fecha en la que el servidor generó esta respuesta (*Date*) y el tipo de información que nos está enviando (*Content-Type*).

En el cuerpo del paquete, que en la figura fue recortado porque era muy extenso, encontramos lo que hemos pedido, es decir, la página web que se corresponde a las recetas de pastas. La página web está escrita siguiendo un estándar llamado HTML, que estudiaremos en el próximo capítulo.

El protocolo HTTP determina entonces de qué forma una persona que utiliza un cliente web puede solicitar a un servidor cierto documento. El protocolo también establece la forma en la que el servidor web enviará de vuelta ese documento. Nótese que el protocolo fue diseñado para ser leído directamente por seres humanos.

Enumeremos entonces las responsabilidades de los programas *cliente*:

- armar y enviar las peticiones al servidor
- entender las respuestas que vuelven del servidor
- atender la interfaz del usuario (notar la complejidad de este ítem)

Y las responsabilidades de los programas *servidores*:

- esperar peticiones escuchando el puerto 80
- concretar la acción que se solicita (por ejemplo, buscar una página web)
- armar la respuesta y enviarla al cliente (por ejemplo, con la página web)

1.4 MÉTODOS HTTP

El protocolo HTTP define ciertos **métodos**, también llamados comandos o verbos, que son los utilizados en la primera línea de las peticiones para indicar al servidor cuál es nuestro pedido. Estos métodos son los que cargan a la petición de un significado, porque según qué método utilicemos será el resultado que se obtenga del lado servidor.

Hemos hablado ya del método **GET**. Pero definamos ahora más precisamente: el método GET se utiliza cuando queremos que el servidor nos envíe cierto **recurso** (*resource*). Para el protocolo HTTP, un recurso puede ser una página web, una imagen, una canción, un video, o cualquier otra cosa. Sólo para visualizarlo, podemos pensar en un recurso como en un archivo. Por ejemplo, cada vez que vemos en nuestro navegador una página web que contiene imágenes, el cliente tuvo que pedir esas imágenes, una por una, al servidor. Para cada imagen construyó un paquete GET solicitándola. Hemos visto también que, cada vez que nos movemos de una página a la otra a través de los *links*, el cliente construye paquetes GET para pedir la página siguiente que queremos navegar.

Las peticiones GET están definidas en el protocolo HTTP como *seguras*. Esto es porque por definición deben ser **idempotentes**, es decir, siempre generar las mismas respuestas. Si solicitamos a un servidor una página web diez veces, utilizando diez peticiones GET, el servidor nos enviará diez respuestas con la misma página web. Otra forma de observarlo es que las peticiones GET no deben cambiar el estado interno del servidor, por lo que a idéntica petición debe corresponder idéntica respuesta. El método GET debe ser idempotente según marca el protocolo. Recordemos que más adelante seremos nosotros quienes programemos el software que está en el servidor, por lo que tendremos responsabilidad en cumplir con este requisito.

Hablemos ahora de un método HTTP que no es seguro, porque no es idempotente: el método **POST**. Las peticiones POST se utilizan cuando es el cliente el que está enviando información al servidor. Por lo general el funcionamiento de la web es con peticiones GET, ya que la mayor parte del tiempo estamos navegando entre páginas y, por ende, es

el servidor el que nos envía la información a los clientes. Pero cada tanto somos nosotros, desde el lado cliente, quienes enviamos información hacia el servidor. El servidor tomará la información y realizará alguna acción, por lo que probablemente cambiará su estado interno. De aquí que no sea *seguro* como GET.

Veamos un ejemplo clásico de paquete POST: un formulario de contacto. Es conocido por todos el formato en donde se nos pide nuestro nombre, email y mensaje de contacto. Este mensaje se envía al dueño del sitio web, que más tarde nos contesta.

¿Cómo se implementa el formulario de contacto? Cuando introducimos nuestros datos y el mensaje que queremos enviar en el formulario, y presionamos "Enviar", el navegador construye una petición POST.

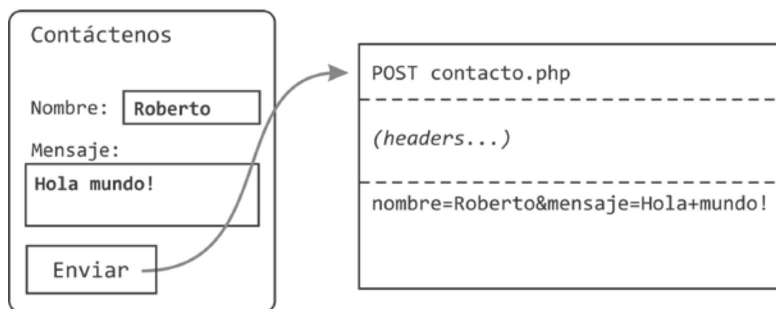

Figura 1.5: Esquema de paquete POST generado con un formulario

Vemos en la figura que los paquetes, como dijimos, tienen siempre la misma estructura de tres partes. En la petición que se construyó observamos que el método es POST y no GET. En segundo lugar tenemos las cabeceras y, a diferencia del paquete GET que estudiamos antes, encontramos que en el cuerpo sí hay información.

La información que encontramos en el cuerpo de los paquetes POST es la información que, en efecto, estamos enviando al servidor. Una petición POST le indica al servidor que es el cliente quien le está enviando algo para procesar. En el cuerpo está dicha información; en nuestro caso se trata de los datos para realizar el contacto.

El formato que tienen los datos presentes en el cuerpo se denomina **URL encoding**, y lo estudiaremos más adelante junto a HTML.

Los navegadores tienen una función muy conocida por todos: al presionar F5 hacen una **recarga de página**. Por definición, lo que los navegadores realizan al presionar F5 es, en realidad, volver a enviar el último paquete que hayan enviado al servidor. Si el último paquete enviado fue de tipo GET, se producirá la conocida "recarga" de página. Pero si el último paquete corresponde a un POST el navegador nos consultará si realmente queremos volver a enviarlo. Esto es así, justamente, porque el método POST no es idempotente. Traducido a nuestro ejemplo del formulario de contacto, si enviamos nuevamente el POST (porque presionamos F5) ¡estaremos generando otro contacto en el sitio! Para ponerlo en mayor perspectiva, pensemos que los formularios de compra online (cuando ingresamos los datos de nuestra tarjeta de crédito) también se envían con POST, por lo que si presionamos F5 ¡estaremos comprando varias veces lo mismo! De esta manera observamos que, con acierto, se denomina al método POST como no-seguro (*unsafe*).

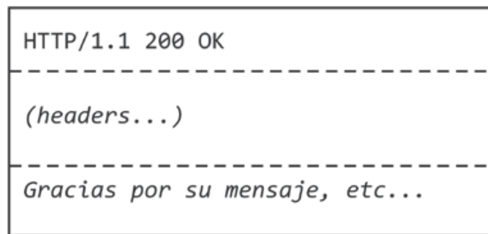

Figura 1.6: Ejemplo de posible respuesta a la petición POST

En la figura 1.6 observamos que la respuesta que nos envía el servidor a nuestra petición POST es como todos los paquetes. Tiene tres partes e indica qué sucedió con nuestra petición precedente, en nuestro ejemplo, el POST del formulario de contacto.

Además de los métodos GET y POST, el protocolo define algunos más y permite la extensión a través de métodos personalizados. Algunos métodos usados, aunque muy escasamente en comparación a los dos que mencionamos son HEAD, PUT y DELETE. Estos métodos exóticos son utilizados en algunos diseños modernos de software denominados *RESTful*.

1.5 CÓDIGOS DE ESTADO

Las respuestas que envía el servidor contienen en la primera línea un número de tres dígitos que llamamos **código de estado**. Este código es una de las partes más importantes de la respuesta ya que indica claramente al cliente qué ocurrió con su petición al llegar al servidor.

Los códigos son números que se clasifican en grupos según su primer dígito. Los códigos que comienzan con **2** son los que indican que la operación se concretó con éxito. Este es el código que nunca vemos al navegar, ya que nuestro cliente no nos indica si una página web solicitada se pudo conseguir con éxito: sencillamente se limita a mostrarla en pantalla.

Los códigos que comienzan con **4** son los errores del cliente. El célebre error **404** es el más conocido. Lo que estos errores agrupan son las situaciones de error que el servidor reconoce como problema del cliente. Por ejemplo, solicitar una página web que no existe generará que el servidor nos devuelva una respuesta 404, que indica justamente esto. El servidor está funcionando bien: somos nosotros quienes nos estamos equivocando al pedir algo que no existe. Otro error de este tipo puede ser solicitar una página para la cual no tenemos permisos de acceso.

Los códigos que comienzan con **5** son los errores del servidor. El más conocido es el error **500**, que quiere decir, de manera difusa, que hay un problema en el servidor y no se ha podido completar nuestro pedido. Si estamos solicitando una página web y obtenemos una respuesta 500 tendremos que intentar más tarde. Se trata de una indicación de que no es nuestra culpa (¡salvo que seamos nosotros los programadores *del servidor*!).

Los códigos que comienzan con **3** son llamados **redirecciones**. Las redirecciones son respuestas que llegan al cliente y que indican que debe generar una nueva petición. De

alguna manera, es una instrucción que el servidor envía al cliente, indicando que para completar la petición debe dirigirse hacia otro lado. Se utiliza cuando un recurso cambia de dirección. Si enviamos una petición, por ejemplo GET, para solicitar una página que ha cambiado de nombre, es posible que el servidor nos devuelva una respuesta **301**, indicando una nueva dirección para la página. Nuestro navegador identificará la respuesta 301 y, automáticamente, se dirigirá a la nueva dirección realizando un nuevo GET, comenzando el diálogo nuevamente. Este mecanismo de redirección es fácilmente reconocible al realizar *login* en los correos electrónicos. El efecto visual que tiene en el navegador es que la barra de direcciones cambia varias veces hasta que llegamos a ver nuestros correos. Esas constituyen varias redirecciones en el diálogo HTTP que nuestro navegador ha tenido con el servidor.

Código	**Texto que acompaña**	**Significado**
200	OK	Se pudo completar lo solicitado
301	Moved permanently	Redirección permanente
302	Found	Redirección temporal
403	Forbidden	Se debe proveer una contraseña
404	Not Found	Recurso no existente en el servidor
500	Internal Server Error	El servidor está fuera de servicio

Figura 1.7: Códigos de estado HTTP más utilizados

1.6 INSPECCIÓN DE TRÁFICO

A continuación utilizaremos el navegador Chrome para realizar inspecciones de tráfico HTTP. Esta inspección también se puede realizar en Firefox y otros, además de existir diversas extensiones para los navegadores que nos permiten distintos análisis. Nuestro objetivo es verificar las propiedades del tráfico que hemos estudiado y familiarizarnos con esta herramienta tan útil en la práctica profesional.

Primero desplegaremos la herramienta para desarrolladores presionando las teclas `CTRL+MAYUS+J` o eligiendo "Herramientas del desarrollador" en el menú. Luego seleccionaremos la pestaña "*Network*" o "Red", según el idioma.

En principio no visualizaremos nada porque recién hemos activado la herramienta. Pediremos al navegador que envíe un paquete GET. Haremos esto escribiendo en la barra de direcciones la siguiente URL: `es.wikipedia.org/wiki/HTTP`.

Al presionar `ENTER`, el navegador genera el paquete GET y espera la respuesta del servidor. Si todo funciona bien, luego de un breve tiempo veremos en pantalla la entrada de Wikipedia para el protocolo HTTP. Lo más interesante es que nuestra herramienta de inspección de tráfico de red se ha llenado de información.

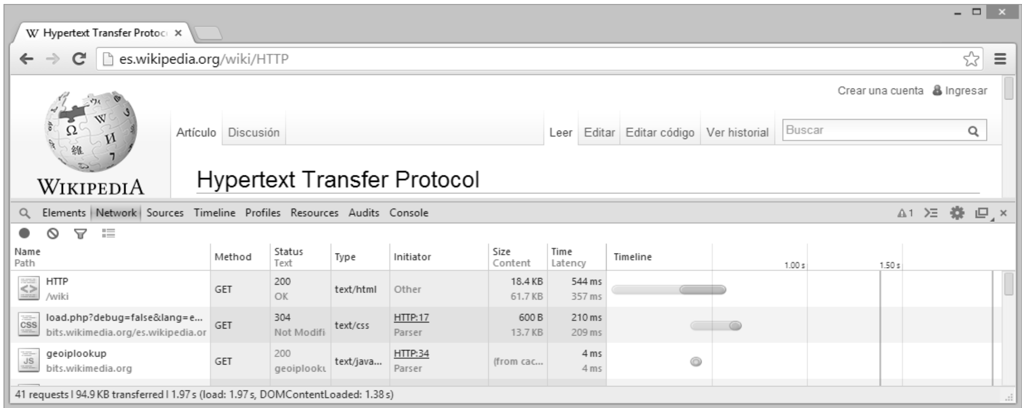

Figura 1.8: Inspección de tráfico HTTP con Chrome

Cada línea corresponde a un paquete enviado al servidor, con su indicación de método, código de estado de la respuesta, tiempos, pesos y demás información. Es evidente la potencia que tiene esta herramienta de análisis.

Si hacemos click en un paquete accedemos a información más detallada, además de poder ver las cabeceras tanto de la petición como de la respuesta correspondiente. El navegador nos está mostrando el tráfico generado a partir de nuestra solicitud original, que se inició con un simple `ENTER`. Estamos viendo el tráfico HTTP concreto.

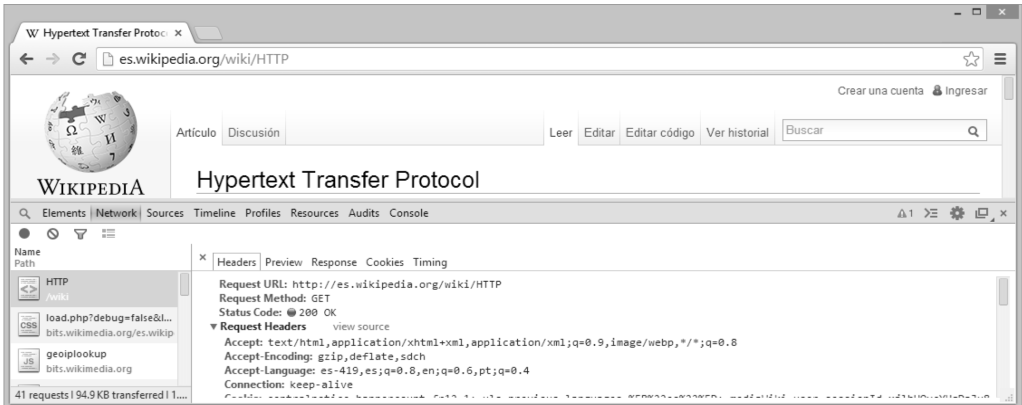

Figura 1.9: Detalle de petición HTTP

Es interesante notar la cantidad de solicitudes que el navegador debe realizar para cargar una sola página web. Muchas de las peticiones no tendrán sentido en este momento, pero es de destacar que, por ejemplo, cuantas más imágenes posea una página web, más solicitudes deberá realizar el cliente. Ello se traduce en ancho de banda y en tiempo de carga. El tráfico de red es aquí fundamental para comprender cómo funciona la navegación a través de la web.

La mayoría de los paquetes enviados serán GET y la mayoría de las respuestas serán "200 OK". Esto quiere decir, básicamente, que estamos navegando y que no hay problemas. Al final del capítulo se podrán encontrar ejercicios para probar distintas situaciones y continuar trabajando con esta importante herramienta.

1.7 PRODUCTOS MÁS UTILIZADOS

La existencia de un protocolo, aceptado mundialmente, permite que distintos fabricantes de software creen clientes y servidores web, y que todos ellos sean interoperables. Si bien en las primeras épocas de Internet esto todavía era un desafío, hoy es prácticamente una realidad.

Los clientes HTTP más utilizados son Google Chrome, Microsoft Internet Explorer, Mozilla Firefox, Opera y Apple Safari. Los servidores HTTP más utilizados son Apache, Nginx y Microsoft IIS (*Internet Information Server*).

Mes a mes se modifica la tasa de participación de cada uno en el mercado y los distintos fabricantes van desarrollando sus productos de acuerdo a las tendencias tecnológicas que surgen cada vez más rápidamente. Otro factor de decisión importante entre las distintas opciones es el tiempo que toma cada organización para corregir las vulnerabilidades de seguridad que se descubren en todo software.

1.8 UBICACIÓN DE RECURSOS

Hemos dicho que HTTP trabaja con recursos. Cuando visitamos una página web, nuestro navegador solicitará al servidor correspondiente esa página, a través de una petición GET. Decimos que ese servidor web **aloja** (*hosts*) a ese recurso. Para indicar qué recurso solicitamos se necesita una URL, sigla de *Uniform Resource Locator*, que indica qué recurso (*resource*) buscar y en dónde está alojado.

Figura 1.10: Partes de una URL

En la figura 1.10 observamos las partes que componen una URL. Analicemos algunos ejemplos:

- `http://es.wikipedia.org` El dominio donde se aloja el recurso es `es.wikipedia.org`, es decir, ese es el "nombre" de la máquina servidora. Como no se indica ninguna página en especial, se nos responderá con el **índice** (*index*) de la carpeta raíz. El índice es una página web creada con un nombre especial, que el servidor envía cuando un cliente accede a una carpeta sin especificar un archivo.
- `ftp://ftp.servidor.com/imagenes/hola.jpg` Se indica otro protocolo, el FTP. El nombre de la máquina es `ftp.servidor.com` y el recurso, que es una imagen, se encuentra dentro de una carpeta llamada "imagenes" dentro de la raíz.

- `https://seguro.afip.gob.ar/monotributo` Se indica una variante de HTTP que es el **HTTP seguro** (de allí la "S"). Esta variante se utiliza, por ejemplo, en las comunicaciones bancarias y basa su funcionamiento en la criptografía asimétrica. No se indica el nombre de un archivo, por lo que se accederá, como en el primer ejemplo, al **índice**, pero de la carpeta "monotributo".
- `http://secreto.com:81/ingreso.php` Aquí encontramos una URL que señala que el puerto de escucha no es el 80 sino el 81. Es decir que tenemos un servidor HTTP que fue configurado para escuchar un puerto que no es el estándar.

Para indicar cuál es el servidor que aloja el recurso se suele utilizar un **dominio** (*domain*). El dominio es un nombre, generalmente palabras en cierto idioma, fácilmente recordable para las personas, que está asociado a un **número IP**. Dentro de toda red de computadoras, incluyendo a la gran red Internet, las computadoras se identifican con un número específico para cada una, que es su IP. Sin embargo, este número es muy difícil de recordar, por lo que el dominio nos provee de un atajo mucho más fácil en el uso cotidiano.

1.9 DOMINIOS Y SU TRADUCCIÓN A IP

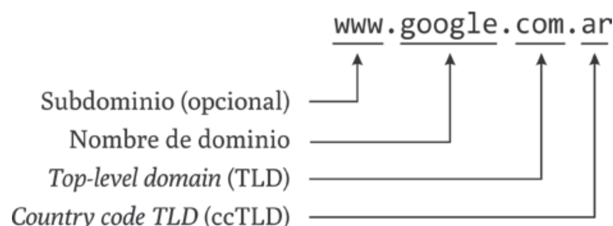

Figura 1.11: Partes de un dominio

En la figura 1.11 se detallan las distintas partes que componen un dominio de los regularmente utilizados por nosotros. El **nombre de dominio** propiamente dicho es el que aparece justo antes del **top level domain**, que es regulado internacionalmente. Cada país cuenta además con un código adicional que también está regulado. El organismo que coordina y mantiene el sistema es IANA (*Internet Assigned Numbers Authority*).

Pero hemos dicho que los dominios son en realidad nombres más memorizables por nosotros, cuando en realidad las computadoras de una red se identifican y comunican utilizando su número IP. Cuando indicamos al navegador qué página queremos ver lo hacemos a través de una URL que expresa el dominio. Evidentemente debe existir una forma de traducir los dominios en números IP para localizar los recursos en las computadoras correspondientes. Esta traducción se llama **resolución de nombre**, y es llevada a cabo por los **servidores DNS** (*domain name server,* servidor de nombre de dominio).

La infraestructura de DNS es muy importante para el funcionamiento de Internet y de todas las redes, no sólo de los servicios web. Por cierto, su complejidad escapa a nuestro foco de atención, pero haremos una breve reseña de su funcionamiento.

Los servidores DNS básicamente reciben preguntas del tipo ¿qué IP corresponde a este dominio? Estas preguntas se envían con cierto formato, para lo cual existe también un

protocolo. El servidor *resuelve* el nombre buscándolo en una tabla y devuelve la IP correspondiente. La infraestructura es muy compleja porque todo el sistema está diseñado para ser tolerante a fallas, y ese objetivo se consigue con una fuerte redundancia de la información. Cada servidor de DNS tiene un "padre" a quien puede consultar, y cada vez que resuelve un nombre lo almacena en una **caché**. Cuando se le pide resolver un dominio que no encuentra en su tabla, el servidor le consulta a su padre y guarda el resultado para próximas preguntas. El padre sigue la misma lógica. De esta forma, todos los servidores tienen a quién preguntarle en caso de "no saber" qué IP corresponde a un dominio. El último eslabón de esta cadena de servidores DNS se denomina **root server**. Estos servidores están divididos por continente, y son el último padre regular al que se le puede pedir que convierta un dominio a una IP. ¿Qué sucede si el root server tampoco puede convertir un dominio en IP?

La fuente última de conocimiento sobre qué IP corresponde a un dominio la tiene un servidor especial denominado **servidor autoritativo**. Este servidor es el "dueño" del dominio y es el que tiene la información real y más actualizada. Cuando un root server encuentra que no conoce la IP de cierto dominio, lo que hace es analizar a qué zona geográfica corresponde. Si es un dominio con código de país ".ar" lo que hará es delegar la resolución del dominio en el sistema argentino. Argentina, como todos los países, administra un registro que indica cuál es el servidor autoritativo de cada dominio. De esa manera se llega al servidor autoritativo, que conoce el número IP que corresponde a un dominio. Y todo el proceso continúa en sentido inverso, hasta que el navegador que originó toda esta cascada recibe cuál es la IP del dominio que posee en su URL. Todos los servidores DNS intermedios mantendrán en caché este número, para que este largo proceso no se tenga que repetir.

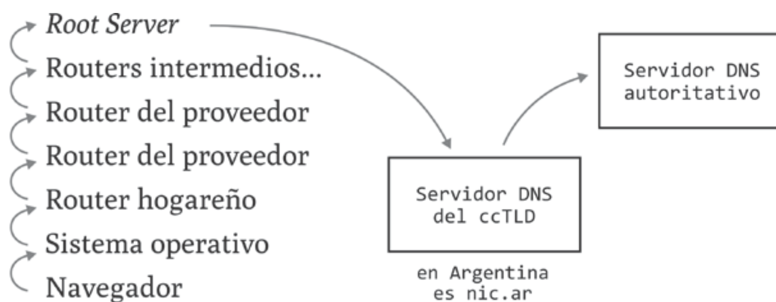

Figura 1.12: Procedimiento de consulta entre servidores DNS

Analicemos un posible caso de resolución de dominio y sigamos los pasos en cada servidor de DNS. Supongamos que hemos creado un sitio y su dominio es `prueba.com.ar`. Abrimos nuestro navegador e intentamos navegar por primera vez.

1. El navegador intenta enviar una petición GET hacia `prueba.com.ar`, pero no conoce la IP del servidor con el cual debe dialogar.
2. El navegador le consulta al sistema operativo cuál es la IP de `prueba.com.ar`, pero éste tampoco lo sabe porque nunca se lo han preguntado y, por ende, no lo tiene en su caché.

3. El sistema operativo le pregunta al router que la empresa de internet ha dejado en nuestra casa cuando realizó la instalación (es decir, su padre). Este aparato tampoco sabe la IP, por lo que deberá preguntar a su propio padre.
4. En la empresa que nos provee Internet hay más routers, que tampoco saben y deben seguir preguntando a sus padres.
5. La empresa que nos provee de Internet tiene también sus propios proveedores, que son mayoristas de telecomunicaciones. Estas empresas, a las que los usuarios finales no tenemos acceso directo, proveen de ancho de banda a las empresas minoristas que nosotros contratamos. Ellos también tienen routers que, igualmente, tampoco conocen en que IP está nuestro sitio web.
6. Siguiendo hacia niveles superiores llegaremos al root server del continente que, naturalmente, tampoco sabe la IP de nuestro sitio.
7. El root server evaluará que nuestro dominio es de Argentina y consultará al servidor `nic.ar`, que es la entidad que registra los dominios en nuestro país.
8. `Nic.ar` dirá que el *servidor autoritativo* para nuestro dominio es 200.10.10.10 (ejemplo). Esto lo hemos configurado nosotros cuando creamos el dominio.
9. El root server le consultará al servidor DNS localizable en la IP 200.10.10.10 cuál es la IP del dominio `prueba.com.ar`.
10. Como este servidor es el *autoritativo* debe conocer la IP de nuestro dominio, por lo que contesta, por ejemplo, 200.40.40.40.
11. El diálogo inicia el camino inverso, almacenando el resultado en las cachés de todos los participantes intermedios, hasta llegar a nuestro navegador.
12. El navegador envía el paquete GET a la IP 200.40.40.40.
13. Nuestro servidor web atiende la petición y devuelve una respuesta.
14. Vemos el índice en nuestro navegador.

1.10 PASOS PARA PUBLICAR UN SITIO WEB

Si bien todavía no hemos incursionado en las tecnologías necesarias para crear las páginas web, ya hemos introducido los conceptos mínimos para comprender cómo funcionará un sitio web nuevo, desde el punto de vista de la infraestructura de red.

El primer paso será contratar un servicio de **hosting**. Este servicio nos proveerá de **alojamiento** para nuestras páginas. Alojamiento quiere decir, básicamente, espacio en el disco rígido de un servidor HTTP. La empresa de hosting nos alquilará espacio en su servidor y, la mayoría de las veces, nos brindará además el servicio de DNS autoritativo. Esta empresa es la que conoce la IP que tendrá el servidor (porque el servidor les pertenece) y mantendrán el servidor DNS autoritativo que indica cuál es la IP del dominio. Al momento de contratar el servicio, nos preguntarán cuál es el dominio de nuestro sitio web.

Por otro lado, para registrar un dominio debemos hacer una solicitud en una **entidad registrante**. Las entidades registrantes dependen del país al que pertenece el dominio. Como mencionamos, en el caso de Argentina los dominios son los `.ar` y se registran en `nic.ar`. El registro puede conllevar un costo, que varía de país en país y según el tipo de dominio que registremos.

Cuando realicemos el registro del dominio en la entidad correspondiente, uno de los datos solicitados será el servidor DNS autoritativo. Nosotros indicaremos el que nuestro hosting nos informe. Este paso se denomina **delegación de dominio**.

Revisemos los pasos con un ejemplo:

1. Nos acercamos a una empresa de hosting y contratamos espacio de alojamiento para nuestras páginas.
2. La empresa nos da acceso al disco rígido de uno de sus servidores HTTP. Allí subimos nuestras páginas. La IP de este servidor es 200.40.40.40.
3. La empresa nos pregunta cuál será nuestro dominio. Supongamos nuevamente `prueba.com.ar`
4. La empresa posee un servidor de DNS escuchando en la IP 200.10.10.10. En ese servidor crea un registro en la tabla de nombres, que dice que el dominio `prueba.com.ar` corresponde a la IP 200.40.40.40 y nos informa que el servidor autoritativo de nuestro dominio está en 200.10.10.10
5. Nosotros registramos nuestro dominio ingresando a `nic.ar`. Allí debemos ingresar datos personales e indicar la delegación, o sea, que el servidor autoritativo de DNS es 200.10.10.10
6. Cuando cualquier persona quiera navegar `prueba.com.ar` verá nuestras páginas web, alojadas en el servidor HTTP del hosting que hemos contratado.

Es importante notar que toda esta complejidad está diseñada para ser tolerante a fallas y para permitir una modificación sencilla de las IP sin que los usuarios finales vean cambios en los dominios.

Si en algún momento precisamos cambiar de empresa de hosting, el dominio no se verá afectado, porque las modificaciones serán en el servidor autoritativo, que no afectan al usuario de manera directa. El servidor HTTP, el servidor DNS autoritativo y la entidad registrante son los tres componentes de este diseño flexible.

1.11 CONSERVACIÓN DE ESTADO (COOKIES)

Como último punto a considerar sobre el protocolo HTTP, observaremos una de sus características de diseño y una técnica muy difundida denominada *cookie*.

El protocolo HTTP es un **protocolo sin estado**. Esto quiere decir que el protocolo no obliga al servidor a realizar ningún seguimiento sobre las peticiones que contesta. En otras palabras, no es una de sus responsabilidades. Esto resulta muy conveniente desde el punto de vista de la performance, pero es problemático para ciertas implementaciones. Por ejemplo, si solicitamos a un usuario su contraseña de acceso ¿cómo recordaremos que este cliente es quien está autorizado si perdemos su rastro en la próxima petición? ¿cómo saber que el cliente que nos envía la contraseña también es el que, dentro de un minuto, nos pide una página protegida?

Este problema surgió muy temprano en la historia de la web y los primeros desarrolladores de software lo solucionaron con una técnica llamada **cookie**.

Las cookies son porciones de información que se transmiten cada vez que un cliente realiza una petición al servidor. Esta información viaja en los paquetes en forma de cabecera. Es correcto decir entonces: las cookies son cabeceras.

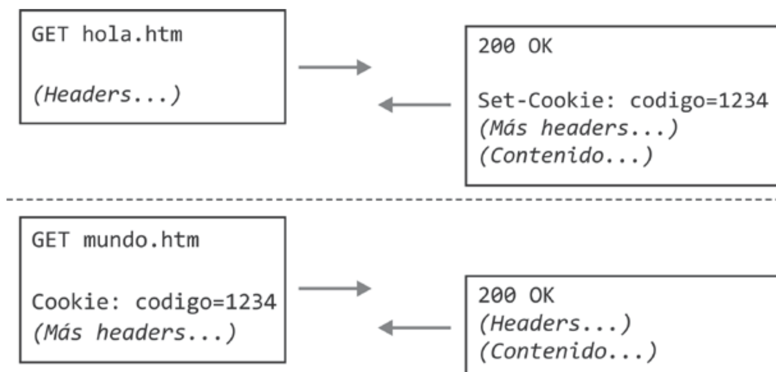

Figura 1.13: Proceso de creación de cookies con la primera respuesta y posterior envío con peticiones

Supongamos que ingresamos a un sitio web por primera vez. El servidor verificará que "no nos conoce", por lo que en la respuesta nos enviará un cabecera llamada *Set-Cookie*, que le solicita al navegador que cree una cookie. La cookie es, en definitiva, una **variable**, que tiene un nombre y un valor.

La primera vez que ingresemos a un sitio, lo más probable es que el servidor nos asigne un número. Este identificador se utilizará para, en la primera respuesta, crear esta cabecera *Set-Cookie*. Al recibir esta cabecera, el navegador creará un registro y almacenará el nombre y el valor de la cookie.

Cada vez que el navegador envíe una nueva petición a este mismo servidor enviará una cabecera *Cookie* con el nombre y valor almacenados previamente. De esta forma el servidor, en la segunda petición y sucesivas, podrá deducir que se trata del mismo usuario. De esta forma se logra conservar el estado ya que cada usuario tendrá un número distinto y el servidor podrá ofrecerles páginas personalizadas para cada uno.

Es importante destacar que la cabecera *Set-Cookie* es sólo un pedido para el navegador y que éste no está obligado a guardar la información ni retransmitirla. Los navegadores pueden configurarse para no almacenar cookies, aunque esto generará hoy en día que prácticamente no podamos utilizar de forma razonable ningún sitio.

Existen muchos mitos sobre las cookies, relacionados con la publicidad, la privacidad y el *malware*. Durante mucho tiempo han estado en boca de los usuarios de Internet y en las noticias. Lo concreto es que son mecanismos utilizados para conservar estados en las aplicaciones que utilizan el protocolo HTTP.

1.12 EL FUTURO

Actualmente se encuentra en desarrollo la **versión 2.0** del protocolo HTTP, 15 años después de la última revisión que ha tenido lugar, la actual 1.1.

Esta nueva versión está basada en desarrollos de la industria, que ya habían puesto manos a la obra en resolver ciertos cuellos de botella. Para reducir el tráfico y los tiempos de latencia, el nuevo protocolo indica comprimir las cabeceras en lugar de trabajar con texto plano. Además establece mecanismos para eliminar la repetición de información idéntica entre sucesivas peticiones y busca un mejor uso de las conexiones TCP, que son costosas de establecer.

En definitiva, la versión 2 supone una mejora que no altera la esencia del protocolo, por lo que las aplicaciones de versión 1.1 seguirán funcionando de la misma manera, tunelizadas a través del nuevo protocolo. Los navegadores más modernos ya soportan funcionalidad que acabará siendo el protocolo versión 2, y en algunos servicios como Google Gmail está en funcionamiento efectivo.

1.13 BIBLIOGRAFÍA

Un excelente libro, a la vez directo, completo y ameno es **Gourley y Totty**, *"HTTP, The Definitive Guide"*, O'Reilly, Sebastopol: 2002.

El protocolo 1.1 propiamente dicho está disponible en `tools.ietf.org/html/rfc2616`

1.14 EJERCICIOS

1. Nombre dos clientes y dos servidores HTTP que no estén listados en este capítulo entre los más usados.

2. Utilice la herramienta de inspección de tráfico HTTP de su navegador e ingrese a dos de sus sitios favoritos. ¿Cuántas peticiones se realizan para la carga completa del índice o página principal?

3. ¿Cuántas de las peticiones encontradas en el ejercicio 2 corresponden a imágenes y cuántas a los demás recursos?

4. Al realizar la experiencia del punto 2, ¿existe alguna respuesta con código distinto de 200? En ese caso ¿qué código se indica y qué significa según el protocolo?

5. Ingrese al índice de 4 sitios distintos e indique qué contiene la cabecera *Server* de las respuestas. ¿Son iguales?

6. Encuentre algún paquete POST utilizando sus sitios favoritos.

7. ¿Qué es la criptografía asimétrica?

8. ¿Para qué se utiliza la herramienta `nslookup` de la línea de comandos de Windows?

Índice

Objetivos, herramientas y organización 4

1. HTTP 6
1.1 La Web como tráfico 6
1.2 Arquitectura cliente-servidor 7
1.3 Tipos de paquetes 8
1.4 Métodos HTTP 11
1.5 Códigos de estado 13
1.6 Inspección de tráfico 14
1.7 Productos más utilizados 16
1.8 Ubicación de recursos 16
1.9 Dominios y su traducción a IP 17
1.10 Pasos para publicar un sitio web 19
1.11 Conservación de estado (cookies) 20
1.12 El futuro 21
1.13 Bibliografía 22
1.14 Ejercicios 22

2. HTML 23
2.1 Lenguaje de marcas 23
2.2 Sintaxis de las marcas 24
2.3 Semántica vs. grafica 25
2.4 Estándares 26
2.5 Estructura base 27
2.6 Árbol del documento 29
2.7 Etiquetas para marcado de texto 30
2.8 Etiquetas de agrupamiento 34
2.9 Etiquetas para datos tabulares 35
2.10 Etiquetas de sección 38
2.11 Contenedores genéricos 39
2.12 Código fuente, renderización y espacio blanco 40
2.13 Entidades HTML 43
2.14 Formularios HTML 45
2.15 Campos de formulario 48
2.16 Resumen de etiquetas y atributos más utilizados 52
2.17 Bibliografía 52
2.18 Ejercicios 53

3. CSS 55
3.1 Estándares, versiones y niveles 55
3.2 Formas de inclusión en HTML 57
3.3 Sintaxis 58
3.4 Los selectores 60
3.5 Especificidad de los selectores 63
3.6 Pseudoclases 67

3.7 Selectores avanzados ... 68
3.8 Herencia de propiedades ... 69
3.9 Introducción a la maquetación ... 70
3.10 Propiedades más utilizadas ... 77
3.11 Bibliografía ... 78
3.12 Ejercicios ... 78

4. JAVASCRIPT ... 80
4.1 Formas de inclusión en HTML ... 81
4.2 Cliente/servidor ... 82
4.3 Sintaxis básica ... 82
4.4 Utilización de la consola ... 84
4.5 Variables y tipos ... 85
4.6 Funciones ... 88
4.7 Acceso al DOM ... 92
4.8 Eventos ... 98
4.9 El objeto String ... 100
4.10 El objeto Date ... 102
4.11 Validación de formularios y HTML5 ... 104
4.12 El futuro ... 105
4.13 Bibliografía ... 105
4.14 Ejercicios ... 106

5. JQUERY ... 107
5.1 Uso de selectores ... 107
5.2 Manejo de eventos ... 109
5.3 *Traversing* ... 111
5.4 Otros usos de jQuery ... 114
5.5 Bibliografía ... 115

6. PHP ... 116
6.1 Programación del lado servidor ... 116
6.2 Relación con HTML ... 118
6.3 Variables y tipos ... 124
6.4 Arrays ... 126
6.5 Datos de formulario ... 131
6.6 Conservación de variables ... 135
6.7 Algunas funciones útiles ... 137
6.8 División del código en partes ... 140
6.9 Bibliografía ... 141
6.10 Ejercicios ... 142

7. MYSQL ... 143
7.1 Persistencia de información ... 143
7.2 Cliente/servidor ... 144
7.3 Usuarios y privilegios ... 145
7.4 Diseño relacional ... 146
7.5 Tipos de datos ... 149
7.6 Lenguaje SQL ... 150
7.7 Uso de phpMyAdmin ... 151

7.8 Consultas de selección 155
7.9 Consultas de modificación 162
7.10 Manejo de texto 163
7.11 Formato de fechas 164
7.12 Exportación e importación 164
7.13 Consultas desde PHP 166
7.14 Advertencia de seguridad 168
7.15 Bibliografía 169
7.16 Ejercicios 170

APÉNDICES

A. JSON 171
B. AJAX 175
C. Encodings 180
D. Rutas absolutas y relativas 185

Sistema de consulta para lectores: maurogullino.com.ar/pwpp