



Web-Based Projects that Rock the Class

Build Fully-Functional Web Apps and
Learn Through Doing

Christos Karayiannis

Apress®

Web-Based Projects that Rock the Class

**Build Fully-Functional Web Apps and
Learn Through Doing**

Christos Karayiannis

Apress®

Web-Based Projects that Rock the Class: Build Fully-Functional Web Apps and Learn Through Doing

Christos Karayiannis
Karditsa, Thessaloniki, Greece

ISBN-13 (pbk): 978-1-4842-4462-3 ISBN-13 (electronic): 978-1-4842-4463-0
<https://doi.org/10.1007/978-1-4842-4463-0>

Copyright © 2019 by Christos Karayiannis

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress LLC: Welmoed Spaehr
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484244623. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

For my children, Vasilis and Stamatia.

Table of Contents

About the Author	xiii
About the Technical Reviewer	xv
Acknowledgments	xvii
Introduction	xix
Chapter 1: The Apache Web Server	1
Getting Started with Apache	1
Installing and Testing Apache	2
Adding New Directories and Web Pages	4
Testing Your Web Site from Another Computer of Your LAN	7
Providing a Static Private IP Address to the Web Server.....	9
Using the Linux Firewall.....	14
Managing the Apache Process.....	16
Working with Virtual Hosts.....	19
Using IP-Based Virtual Hosts	21
Using Port-Based Virtual Hosts.....	25
Using Name-Based Virtual Hosts.....	30
Inspecting the Overall Virtual Host Configuration.....	35
Reading Apache Log Files	36
Summary.....	37
Chapter 2: Server-Side Programming with PHP	39
The PHP Engine.....	39
Installing and Testing PHP.....	41
Testing PHP Without a Web Server.....	43

TABLE OF CONTENTS

Running Your First PHP Examples from the Web Server	46
Working with Variables and Strings.....	46
Setting the PHP Variables with the GET Method	54
Setting the PHP Variables with the POST Method.....	58
Running Client-Side vs. Server-Side Programs	61
The JavaScript/PHP Addition Web Page	62
The Second Version of the JavaScript/PHP Addition Web Page.....	67
The Third Version of the JavaScript/PHP Addition Web Page.....	75
Form Validation with PHP.....	75
The validate.php Source Code Commentary.....	80
Summary.....	87
Chapter 3: Connecting Your Apache Server to the Internet	89
The NAT Protocol.....	89
Enabling Port Forwarding to Your Router	90
Implementing Port Forwarding with Apache Vhosts	94
Testing the New Virtual Host.....	96
Using the GeoIP Apache Module	100
Responding to the Visitor's Native Language.....	107
Using a Map to Display the Visitor's Location	113
A New Version of the Static Map Web Page	120
Summary.....	124
Chapter 4: Obtaining a Domain Name with DDNS.....	125
DNS and DDNS	125
Registering with a DDNS Service Provider	126
Configuring the Router's DDNS.....	131
Implementing Web Redirect	133
Implementing an Online Web Service	137
Editing the Apache Configuration File	139
Editing the Web Page for Submitting the User's Data.....	141
Working with Regular Expressions	145

TABLE OF CONTENTS

Testing the JavaScript Form Validation.....	146
Running whois from the Command Line.....	148
Editing the File That Processes the User Data.....	149
Testing the WHOIS Online Service.....	154
Adding a Favorite Icon to the Site.....	156
Summary.....	163
Chapter 5: The Lighttpd Web Server	165
Installing Lighttpd	165
Testing Lighttpd	166
Working in the Lighttpd Configuration File.....	168
Applying a Basic Configuration.....	170
Binding to a Specific IP Address or Hostname	171
Changing the Document Root.....	173
Enabling and Disabling the Directory Listing.....	174
Sending Custom-Made Error Replies to the Client.....	176
Accessing the Lighttpd Log Files	178
Using Virtual Hosts with Lighttpd.....	181
Using PHP with Lighttpd	189
Creating Online Services with Lighttpd.....	190
Creating the Directory Index of the Online Service	195
Creating the Action File for the Online Service	197
Enabling the Site to Serve Multiple Client Requests.....	203
Creating an Animated PNG Image	208
Summary.....	217
Chapter 6: The MySQL Database Server	219
Installing and Testing MySQL.....	219
Creating Your First MySQL Database	223
Creating and Deleting Tables of Your Database	225
Inserting, Displaying, and Deleting Records	228

TABLE OF CONTENTS

Altering the Table's Structure	230
Testing the Table Connection.....	233
Performing SQL Queries with the MySQL Server.....	235
Modifying Records with the update Command	238
Using the SQL like Operator.....	239
Web Scraping with MySQL and the Linux Shell	241
The URLs Describing the Resources.....	241
Designing the Web Scraping Project.....	245
Creating the MySQL Database Used for the Web Scraping Project.....	252
Implementing the Web Scraping Project.....	254
The Script's First Part.....	257
The Script's Second Part	260
Testing the Web Scraping Shell Program	264
Summary.....	268
Chapter 7: Creating a Dynamic Content Web Site.....	269
Search-Enabled Site Overview	269
Designing the Project.....	271
Creating the First Web Content Samples	272
Creating and Updating the Project's Database	273
Writing the Shell Script That Updates the Database	275
Automating the Database Updates with cron	279
Designing the Home Page of the Site	285
Creating the Directory Index of the Site	286
Creating the Action PHP Program	288
Testing the Dynamic Content Site	294
Making Modifications	298
Improving the Query Results Appearance with a Two-Colored Table.....	302
Implementing Pagination	304
The Pagination-Enabled Version of index.php.....	309
The Pagination-Enabled Version of search.php	311

TABLE OF CONTENTS

Using Images Instead of Submit Buttons	318
Implementing the Site with the GET Method	321
Summary.....	326
Chapter 8: Implementing Secure Sockets Layer on Your Site	327
Implementing SSL/TLS.....	327
SSL Certificates	328
Creating Self-Signed Certificates with OpenSSL	330
Configuring SSL for Lighttpd.....	334
Configuring SSL for Apache.....	336
Testing the Self-Signed Certificate	338
Enabling Your Site to Be Viewed Outside of Your LAN.....	345
HTTP Cookies and PHP Sessions	352
Setting a Cookie with PHP	353
Retrieving a Cookie Value from PHP	355
Removing Cookies with PHP.....	355
Creating a Site That Uses Cookies	355
Viewing the Cookie Details in Your Browser.....	361
Using Wireshark to View the HTTP Cookie Header	361
Using Browser Tools to View the HTTP Cookie Header	365
Using PHP Sessions	366
Running a PHP Session Example.....	368
The Source Code for index.php	371
The Source Code for page1.html and page2.html	376
Experimenting with the Sessions Project.....	380
Summary.....	383
Chapter 9: Running Your Site with a Certificate from a Certificate Authority	385
Obtaining Your Own Domain Name	386
Obtaining a CA SSL Certificate for Your Domain Name.....	387
Configuring SSL on the Web Servers	392
Installing the CA Certificate on the Apache Web Server	392
Installing the CA Certificate on the Lighttpd Web Server.....	399

TABLE OF CONTENTS

Testing the SSL CA Certificate	402
Project: Securely Logging In to a Site	406
Designing the Project's Site	407
The Source Code for the Home Page of the Site	409
The Web Page for Creating the User Account.....	411
Creating the Database Used for the Project	419
Testing the PHP to MySQL Connection	421
The Source Code of the Login Web Page.....	425
The Source Code for the User Profile Page	432
Allowing the User to Log Out.....	436
Testing the User Connection to the Site	437
Improving the profile.php Web Page.....	439
Summary.....	441
Chapter 10: Running Online Services Using PHP Sockets	443
Updating the Domain Name IP Address with ddclient.....	443
Utilizing PHP Sockets	449
The Code for the Command-Line PHP Socket Server.....	450
Testing the PHP Command-Line Socket Server.....	452
Implementing a Command-Line PHP Client.....	453
Configuring the Web Servers for the New Site	454
Create the Site That Interfaces with the Command-Line Server	455
A TCP Port Check Site	458
The Source Code for index.php and ports.php	460
Testing the Online Open Port Check Site Locally.....	464
Testing the Online Port Test Site Remotely	467
A Second Version of the Open Port Check Tool Source Code.....	471
Creating an Online Service Displaying QOTD Messages	473
The Source Code for the QOTD Project.....	475
Testing the QOTD Site	479
Using Different FQDNs for Your Sites	482
Summary.....	484

TABLE OF CONTENTS

Appendix: Exchanging Variables Between JavaScript and PHP	485
Example 1: Passing Variables from PHP to JavaScript Using the echo Command.....	485
Example 2: Passing Variables from JavaScript to PHP Using the location Object.....	486
Example 3: Passing Variables from JavaScript to PHP with HTML Form Submission.....	487
Example 4: Passing Variables from JavaScript to PHP and Back with Ajax	488
Example 5: Passing Variables from JavaScript to PHP with Cookies	492
Example 6: Passing Variables from PHP to JavaScript with Cookies	494
Index.....	497

About the Author



Christos Karayiannis has taught web development courses for more than 20 years in high schools and institutes of technology in Greece. He holds an MSc in computer science from the University of Wales and a physics degree from Aristotle University. His main interests are networking, operating systems, and programming. Christos has contributed to open source projects by documenting source code.

About the Technical Reviewer

François-Denis Gonthier is a graduate of the Université de Sherbrooke computer science program. He first worked for a startup company delivering cryptographic software using open source technologies. Since then, he has never strayed far from the Linux and open source world. He went from programming front ends in JavaScript and HTML 5.0 to coding back ends using Java, J2EE, JSF, or plain old Unix daemons. The cool Web 2.0 kids would call this being a *full-stack developer*. Nowadays, he mostly works on embedded Android projects and writes JavaScript running on Node.js in his spare time.

Acknowledgments

Writing this book would not have been possible without the contribution of the people of Apress. I would like to specially thank Louise Corrigan for entrusting me to turn an idea into this book, Nancy Chen and James Markham for their valuable advice and support, Francois-Denis Gonthier for providing his insightful comments, and finally Sathyamurthy Krishnan and Arockia Rajan for the proofreading of this book.

Introduction

This book includes several web-based projects. With each project you'll create a complete client-server application while learning about different aspects of web development. Each project includes detailed steps to build the software that executes on both the client and server sides. Introductory examples are also provided to highlight the techniques and programming concepts that are then used in the projects.

The projects are real-world web services that you would typically find on the Internet today. Here are a few examples:

- A site that pinpoints a visitor's location on a map and greets them in that country's native language
- An online WHOIS service
- A remote TCP port scan service
- A test service for displaying how a site is viewed remotely
- A dynamic content site where users can search with keywords and read online scientific papers, with a database automatically updated when new documents are inserted into a directory
- A project that implements web scraping techniques
- A site that uses cookies to remember a visitor
- A site that uses sockets to connect to other Internet services, like QOTD
- A site that implements PHP sessions, where visitors log in with an encrypted password and while connected browse the site with access only to their own personal details

The techniques you'll learn while creating these services can be applied to your own future projects.

About the Tools and Source Code

Each chapter contains all the steps for installing and setting up the required software tools for the project being discussed. For instance, two different web servers are used in the book, along with a PHP interpreter and the MySQL database server. The book also explains how to set up a router, and it contains a DDNS guide that explains how to configure the projects for the Internet.

The software tools used in this book are free and don't require you to purchase API keys, such as when using a static map. The only exceptions are the steps for purchasing a second-level domain name and an SSL certificate; however, obtaining these is optional for running the corresponding projects.

I've tried to keep the source code to a minimum so that the projects can be completed within a time-constrained web development course. All the projects are original in that they were designed just for this book. You will not find the source code used anywhere on the Web or in other textbooks.

However, just because these projects were designed for a classroom setting does not mean they are not professional-grade quality. Anyone interested in the underlying mechanisms of modern web development can also use this book.

Let's get started with the projects so you can design and build your first online service, used in the classroom or in the real world!

CHAPTER 1

The Apache Web Server

Most people are familiar with using a web browser. Even when they use a new browser for the first time, they are capable of visiting a web page and navigating to other sites immediately. This is because browsers usually encompass all the same handy functionalities. For example, all browsers include an address bar at the top of the window, a history option, a favorites folder, and some navigation buttons.

Web servers, on the other hand, rarely look similar to each other. However, servers do share some common ground. In this book, you'll be using two different web servers, and you'll learn about the common features of both as well as the differences. In this chapter, you'll use the Apache web server, and in Chapter 5 you'll use another popular web server, Lighttpd.

Getting Started with Apache

The Apache web server is one of the most popular web servers today. This is because of how easy the administrative features are to use and because of its flexibility based on its modular design. Apache got its name in 1995 when the Apache Project started. The Apache Project was a collaboration of many programmers, and hence the source code required many *patches* (software fixes). Through software modules, Apache offers additional functionality that administrators can install. Because of these added components, Apache isn't by default a huge, monolithic piece of software.

Apache was one of the first servers to support virtual hosts (*vhosts*). This notion—also referred as *virtual servers* on other web servers like Cherokee—denotes the capability of a single web server to run multiple sites simultaneously that are differentiated either by the IP address, by the port number, or by the domain name used in the client request.

Installing and Testing Apache

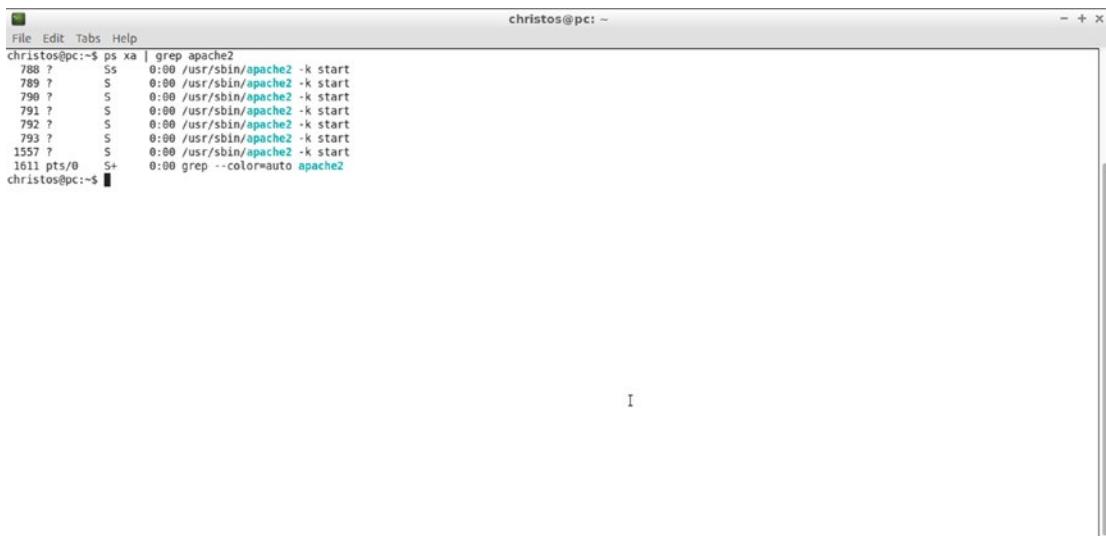
The Apache program is currently known as apache2, and you can download and install it from the Linux terminal using the following command:

```
$ sudo apt-get install apache2
```

The apache2 process starts running. Use the ps (process status) command from the Linux terminal to view the apache2 processes.

```
$ ps xa | grep apache2
```

More than one apache2 process is created instantly to be ready to serve further client requests. Figure 1-1 displays the process IDs (PIDs) for the Apache processes returned at this time by the command ps.



A screenshot of a terminal window titled "christos@pc: ~". The window contains the following text:

```
christos@pc:~$ ps xa | grep apache2
 788 ?      Ss    0:00 /usr/sbin/apache2 -k start
 789 ?      S     0:00 /usr/sbin/apache2 -k start
 790 ?      S     0:00 /usr/sbin/apache2 -k start
 791 ?      S     0:00 /usr/sbin/apache2 -k start
 792 ?      S     0:00 /usr/sbin/apache2 -k start
 793 ?      S     0:00 /usr/sbin/apache2 -k start
1557 ?      S     0:00 /usr/sbin/apache2 -k start
1611 pts/0  S+    0:00 grep -color=auto apache2
```

Figure 1-1. The command ps output lists the apache2 PIDs

The web server is up and running and ready to dispatch the first page. In the address bar of your browser, enter the loopback IP address, which is the IP address your PC owns even if no network card is installed, or its corresponding host name. Here are some examples:

```
127.0.0.1
http://127.0.0.1
localhost
```

Apache responds by providing the directory index since no actual web page was specified in the URL.

Hint! A default page set for a specific directory is called the *directory index* for this directory. Also, the directory index for the document root (i.e., the base directory for a website) is the directory index of the site.

The directory index contents include a short introduction to the Apache configuration options. Figure 1-2 displays the Apache default page for Ubuntu.

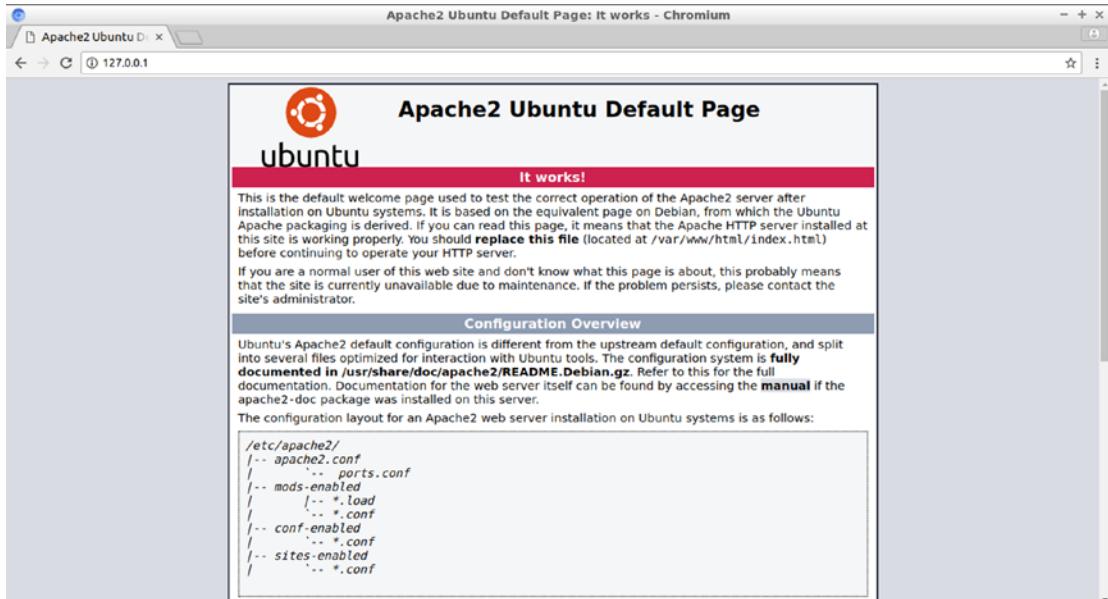


Figure 1-2. The Apache directory index for Ubuntu

As the first section's title states, "It works!" The bad news is that so far it works only for users who use their own PC and are interested in just the Apache configuration. There is good news, though. Following the steps in the book, you can customize your web server with your preferences, and you can make your site available to the whole Internet.

Adding New Directories and Web Pages

By reading the web page's content, you can find out the document root and the file name of the directory index. The following text:

```
"You should replace this file (located at /var/www/html/index.html)"
```

indicates that this path is the document root:

```
/var/www/html/
```

and indicates that this file is the directory index:

```
index.html
```

From the Linux terminal, change the current working directory to `/var/www/html`, as shown here:

```
$ cd /var/www/html
```

Create a copy of `index.html` to use your own directory index.

```
$ sudo cp index.html index_OLD.html
```

Use a Linux text editor such as gedit to edit the original `index.html`, as shown here:

```
$ sudo gedit index.html
```

In the gedit window, enter your own HTML source code, for instance:

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Apache</title>
</head>
<style>
p {
    font-size:24px;
    text-align:center;
    color:blue;
}
```

```
body{  
background-color:yellow;  
}  
</style>  
</head>  
<body>  
<p>Hello World!</p>  
</body>  
</html>
```

Click the Save button in the gedit window. Next enter the loopback IP address in the browser's address bar. As displayed in Figure 1-3, the Apache server displays your own web page.

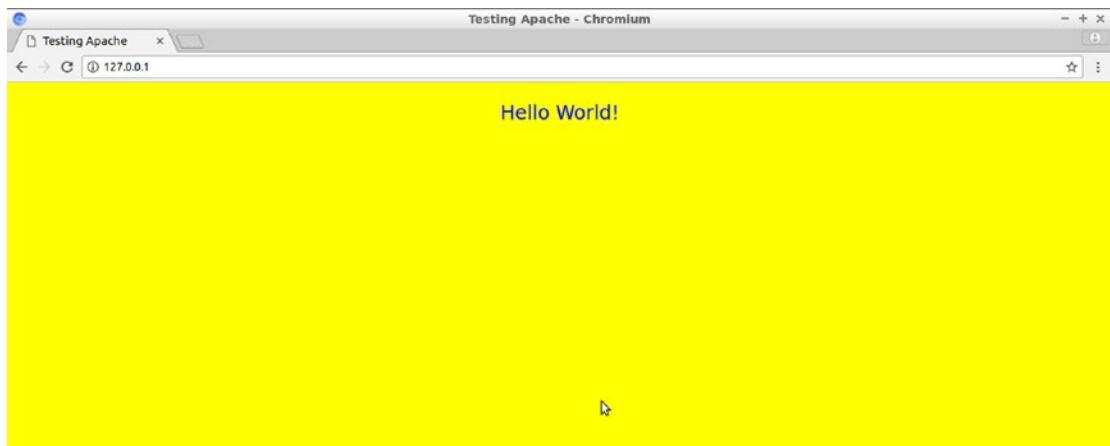


Figure 1-3. A customized directory index printing “Hello World!”

To request another web page, for instance `index_OLD.html`, you can provide the URL for the web page file by including the web page path starting from the document root. For instance, enter one of the following in the address bar of your browser:

`127.0.0.1/index_OLD.html`
`localhost/index_OLD.html`

Because `index_OLD.html` is in the document root, the path includes only the file name of the web page. By using this URL, the default web page of Apache for Ubuntu is displayed again, as shown in Figure 1-4.

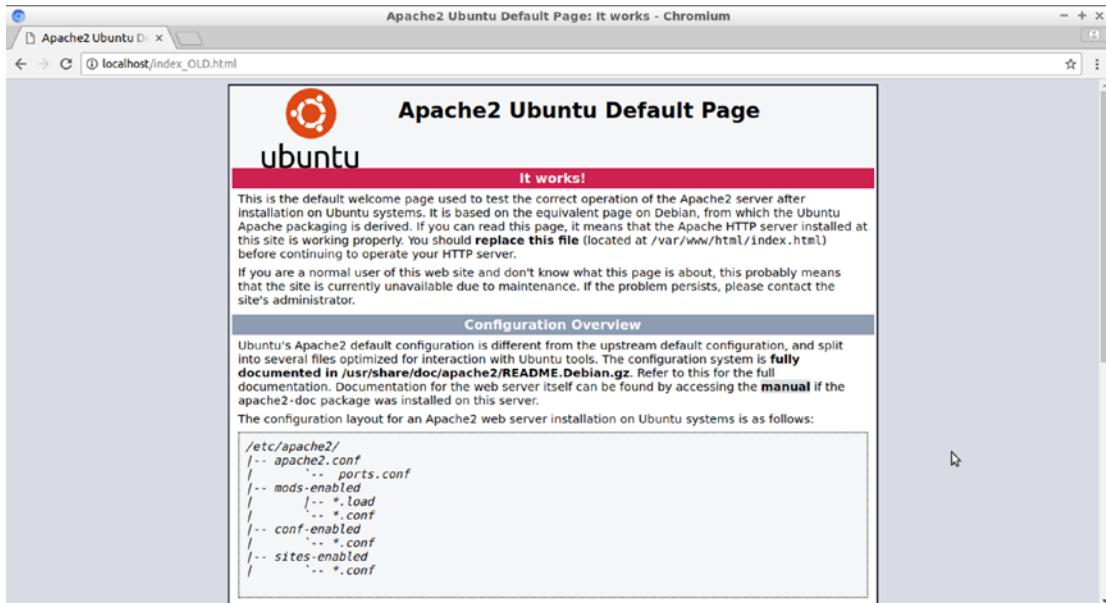


Figure 1-4. Downloading a specific web page by including its path in the URL

Start including subdirectories in the document root to better organize your web site. At the Linux terminal, change the working directory to the document root.

```
$ cd /var/www/html/
```

Create a new test directory. In the new directory, create `test.html`, another HTML file.

```
$ sudo mkdir test
$ sudo gedit test/test.html
```

In the gedit window, enter some HTML source code for `test.html` and click the Save button. Then enter the file name path of the web page, excluding the document root, in the address bar of your browser. The file name path for `test.html` is as follows:

```
/var/www/html/test/test.html
```

You can omit the document root part since on the web server any file reference starts from the document root; therefore, the URL of `test.html` is as follows:

```
localhost/test/test.html
or 127.0.0.1/test/test.html
```

The web page is loaded in your browser, as displayed in Figure 1-5.



Figure 1-5. Testing a web page located in the document root subdirectory

Testing Your Web Site from Another Computer of Your LAN

As another test, you can see whether Apache connects to other computers on the same local area network (LAN) as your PC. A connection between two workstations on the same LAN happens via the Internet layer of the TCP/IP protocol between their private IP addresses. The private IP addresses are valid only internally to the LAN. This is in contrast to public IP addresses that are globally accessible to the whole Internet. Put simply, all workstations in a LAN possess private IP addresses and are represented to the Internet with the public IP address of the LAN router (which also includes a private IP address for internal communication with the LAN workstations).

At the Linux terminal, use the `ifconfig` command to find out the private IP address of your computer.

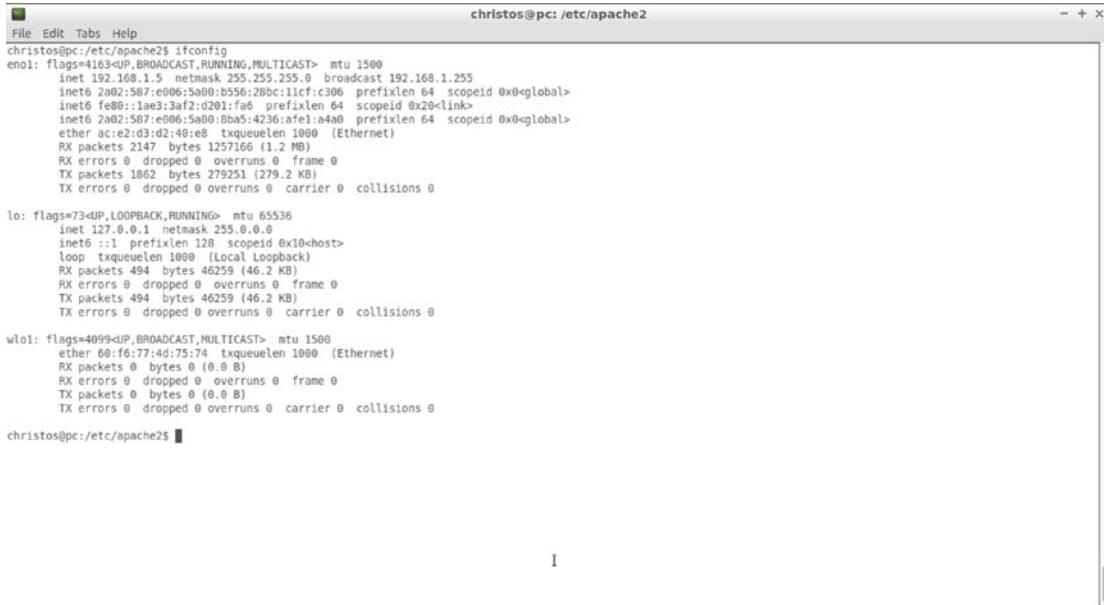
```
$ ifconfig
```

You can also use the `hostname` command.

```
$ hostname -I
```

The private IP address of my computer used at the given time was 192.168.1.5, as shown in Figure 1-6.

CHAPTER 1 THE APACHE WEB SERVER



```
christos@pc:/etc/apache2$ ifconfig
christos@pc:/etc/apache2$ ifconfig
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.5 netmask 255.255.255.0 broadcast 192.168.1.255
        inetb 2a02:587:0:0:5a00:b556:28bc:11fc:c306 prefixlen 64 scopeid 0x0<global>
        inet6 fe80::1ae3:3af2:d201:fa6 prefixlen 64 scopeid 0x20<link>
        inet6 fe80::1ae3:3af2:d201:fa6 prefixlen 64 scopeid 0x0<global>
        ether ac:2e:d3:d2:40:eb txqueuelen 1000 (Ethernet)
        RX packets 2147 bytes 1257166 (1.2 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 1862 bytes 279251 (279.2 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inetb ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 494 bytes 46259 (46.2 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 494 bytes 46259 (46.2 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlo1: flags=4009<UP,BROADCAST,MULTICAST> mtu 1500
    ether 00:f6:77:4d:75:74 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

christos@pc:/etc/apache2$
```

Figure 1-6. *ifconfig returning the private IP address*

Use the private IP address returned by `ifconfig` in the address bar of the local browser. The web page loads just like previously (with the loopback address), as shown in Figure 1-7.

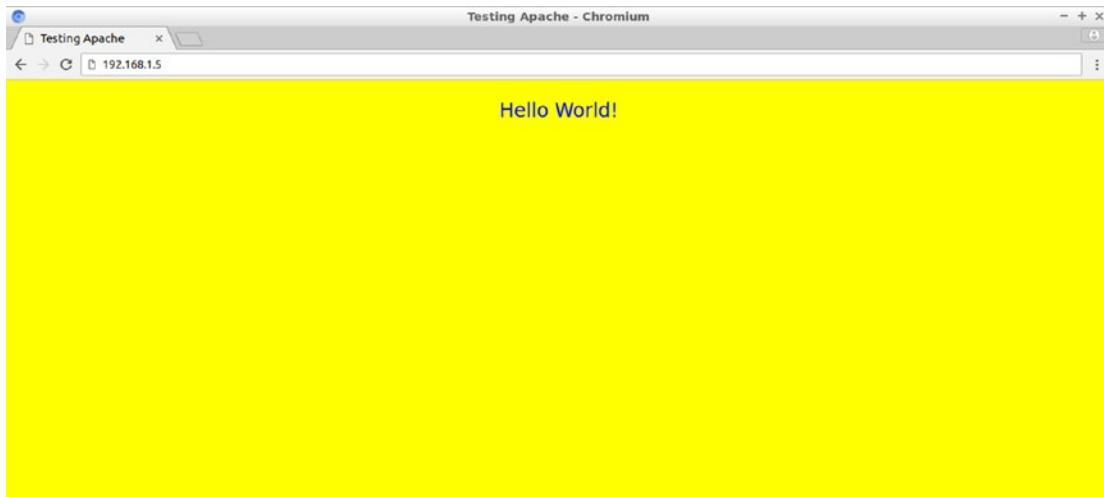


Figure 1-7. *Testing the web server using a private IP address*

Next try the same IP address from another computer of your LAN. Figure 1-8 displays the browser used on a Windows 7 system in the same LAN.



Figure 1-8. Testing the web server from another PC of your LAN

Providing a Static Private IP Address to the Web Server

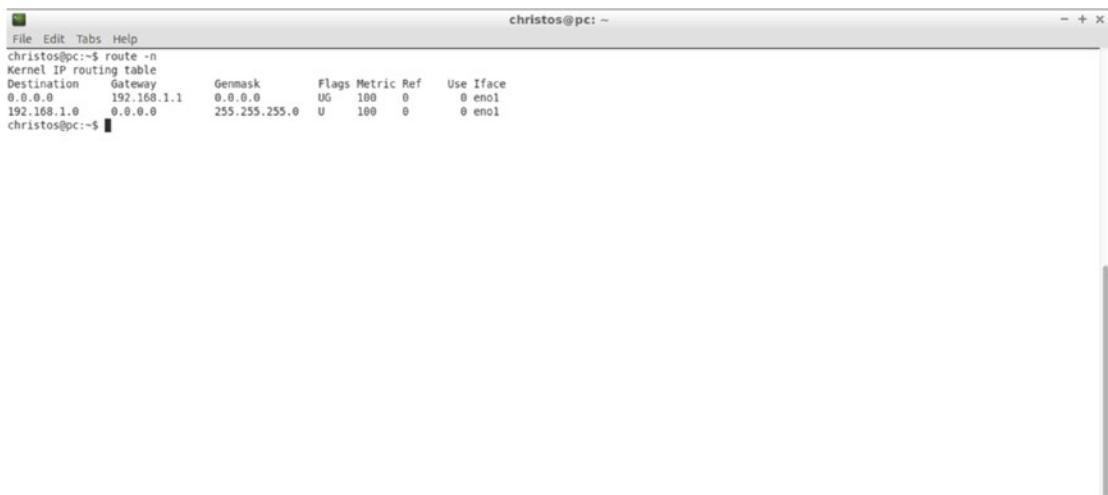
The next step for the web server configuration is to set up a static private IP address. This is a private IP address that remains the same each time the computer that hosts your web server restarts. This step is optional to test your server from your LAN, but it is essential for making the server available to the whole Internet. To provide a static private IP address, the network connection of the computer that hosts the web server has to properly configured.

The most straightforward configuration for a workstation in a LAN is to obtain its IP address with the Dynamic Host Configuration Protocol (DHCP). When a PC is configured with the DHCP option, on startup it is assigned by the router with an IP address, a network mask, the IP address of the default gateway (the router itself), and the DNS server used for performing the translations between domain names and IP

addresses. The drawback with DHCP is that your PC gets the next available IP address from an IP address pool, which may be a different address from the one obtained the previous time (a dynamic address).

Your other option is to use a static private IP address. Each time you start your PC, its IP address remains the same. A static IP address is mandatory for a server required to be available for connections across the Internet. To implement a static IP address, you have to configure manually the information previously retrieved by DHCP. Run first the `route` command from your terminal to find the default gateway (the router) in your LAN, as shown in Figure 1-9.

```
$ route -n
```



The screenshot shows a terminal window titled "christos@pc: ~". The window contains the command "route -n" followed by the kernel IP routing table. The table has columns: Destination, Gateway, Genmask, Flags, Metric, Ref, Use, and Iface. There are two entries: one for the local loopback (0.0.0.0) with gateway 192.168.1.1, and another for the subnet 192.168.1.0 with gateway 0.0.0.0. Both entries have a netmask of 255.255.255.0, flags UG, metric 100, reference 0, and interface en0.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	192.168.1.1	0.0.0.0	UG	100	0	0	en0
192.168.1.0	0.0.0.0	255.255.255.0	U	100	0	0	en0

Figure 1-9. The `route` command returns the default gateway IP address

The IP address for the gateway (the router) is therefore 192.168.1.1.

Run `ifconfig` again to gather information about the current IP address and the mask.

As you saw in Figure 1-6 earlier in the chapter, the IP address for the web server is currently 192.168.1.5 with a netmask of 255.255.255.0. The nonzero part of the netmask describes the part of the IP address that corresponds to the network, while the zero describes the part that corresponds to the computer in that network. By logical ANDing the IP address and its netmask, you get the network address.

The LAN in this example therefore has the IP address 192.168.1.0. This leaves the last byte to represent the computer in the network. This is therefore the byte that you will change to choose the static IP address for the web server. You can select any unused value from 1 to 254. For instance, for the value 100, you get the IP address 192.168.1.100.

With the information collected, click the Wired Network icon on your desktop and then click Next. Alternatively, select System Settings > Network > Wired or Settings > Preferences > Network Connections depending on your Ubuntu version. In the Network Connections window that appears, as shown in Figure 1-10, double-click “Wired connection 1” (or the option with a similar name in the Ethernet list).

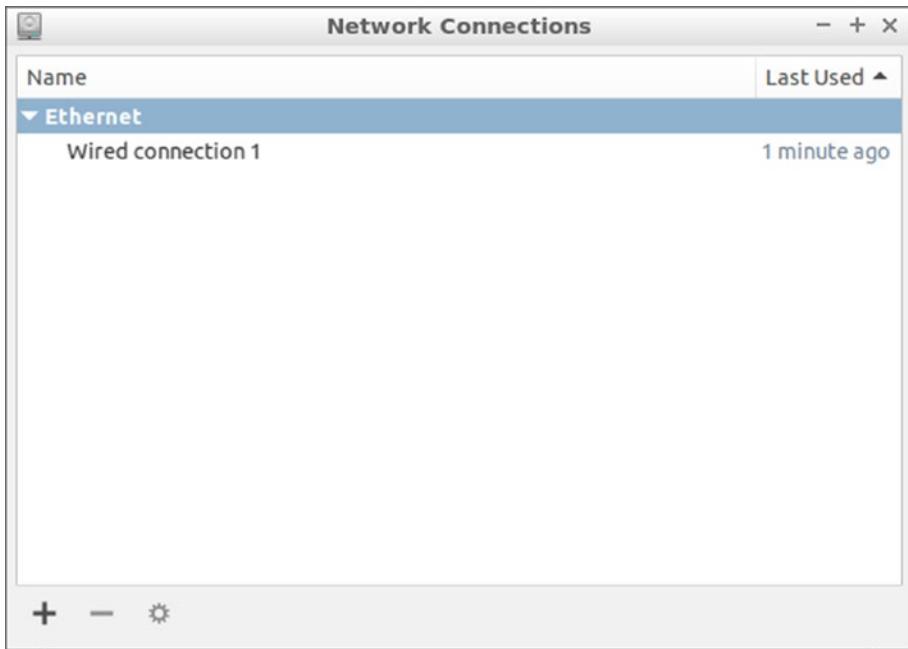


Figure 1-10. The Network Connections window

The “Editing Wired connection 1” window that appears next provides the IPv4 Settings and IPv6 Settings tabs for configuring IPv4 and IPv6 addresses, respectively (Figure 1-11).

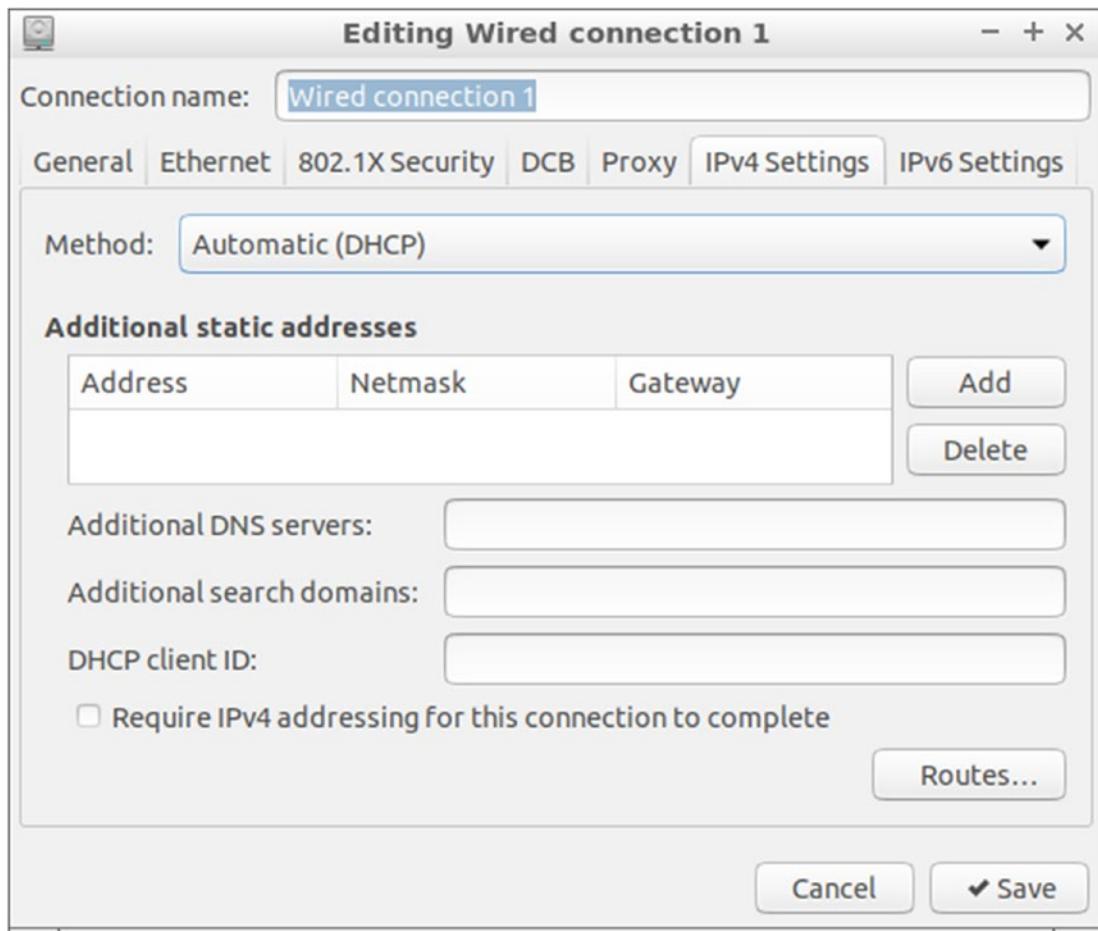


Figure 1-11. The “Editing Wired connection 1” window

On the IPv4 Settings tab, the method currently selected is Automatic (DHCP). Deselect this option and select instead Manual from the Method list (Figure 1-12).

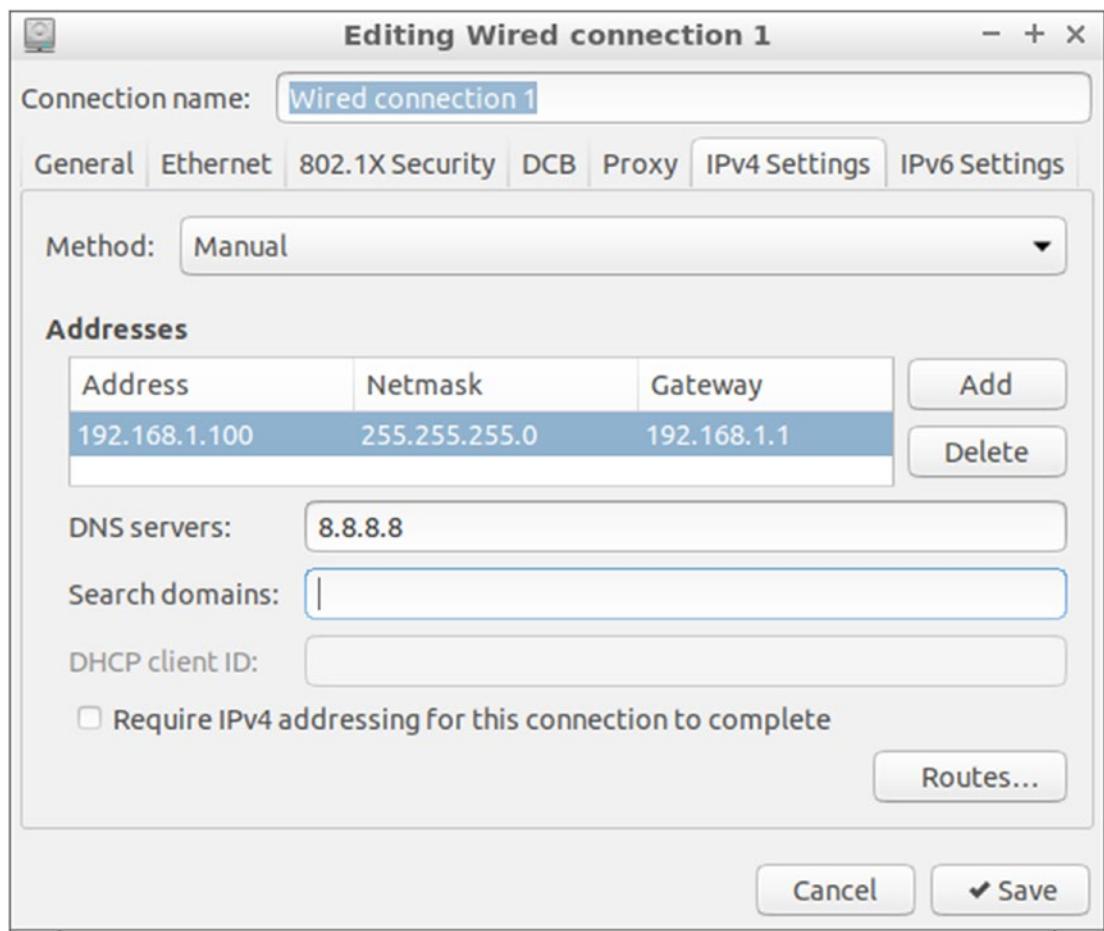


Figure 1-12. Applying the Manual method for configuring the IPv4 settings

Click the Add button to complete the required fields for manually setting the connection parameters. Under Address, enter your preferred IP address. For the given LAN with the IP address 192.168.1.0, you could enter, for instance, **192.168.1.100** as the new static IP address for the web server. Enter also the value for the netmask, **255.255.255.0**, for the Netmask option, and enter the IP address of the router, **192.168.1.1** in this example, for the Gateway option. You need also to provide the IP address (or addresses) for a DNS server in the “DNS servers” field. If you enter more than one IP address, separate them using commas. For the DNS server, you can use the IP address of the DNS server suggested by your Internet service provider (ISP) or use the IP address of a freely available DNS server like Google’s 8.8.8.8.

Click the Save button to confirm the new settings. To activate the new IP address, click the Network Manager icon and select Disconnect; then click again and select “Wired connection 1” or enter the following at the Linux terminal:

```
$ sudo service network-manager restart
```

You can try now the static IP address of your web server by entering it in the address bar of a browser and running it from another PC in your LAN, as indicated in Figure 1-13.



Figure 1-13. Testing the web server by using its static IP address

In the following section, you can configure your Linux firewall to block connections for all port numbers except the ones set by you.

Using the Linux Firewall

You should be able to view your web page on another PC in your LAN without a problem. However, if a firewall is enabled in your system, you will get the error message “This site can’t be reached” or a similar one in your browser. (A *firewall* is an application that checks the connections to your system and permits or bans them according to the rules that you set.)

In the system I used in the previous section, the pre-installed Ubuntu Linux firewall (ufw) was disabled by default. To disable it, you can use this command:

```
$ sudo ufw disable
```

Now try to connect and see if you get the error message. You can then restore it using the following command:

```
$ sudo ufw enable
```

To allow the Apache server to receive connections while a firewall is on, you can specify the appropriate rules. Apache, trying to make your work easy, has already set the basic rules for you when it was installed on your system. At the command line, enter the following:

```
$ sudo ufw app list
```

The command's output includes the ufw profiles for Apache shown in Figure 1-14.



```
christos@pc:~$ sudo ufw app list
[sudo] password for christos:
Available applications:
 Apache
 Apache Full
 Apache Secure
 CUPS
christos@pc:~$
```

Figure 1-14. Listing the ufw profiles

- Profile Apache is for the default HTTP port 80.
- Profile Apache Full is for both ports 80 and 443.
- Profile Apache Secure is for the default HTTPS port 443.

To allow incoming traffic for the Apache Full profile, enter the following at the Linux terminal:

```
$ sudo ufw allow 'Apache Full'
```

To verify this, enter the following:

```
$ sudo ufw status
```

The command's output is shown in Figure 1-15.

```
christos@pc:~$ sudo ufw status
Status: active

To                         Action      From
--                         --          --
Apache Full                ALLOW      Anywhere
Apache Full (v6)            ALLOW      Anywhere (v6)

christos@pc:~$
```

Figure 1-15. Displaying the status of the ufw firewall with the Apache Full profile set

An even better way to verify this is to connect to the Apache web server from another PC in your LAN. The web page should load as usual.

Managing the Apache Process

To apply new configuration rules while Apache runs and also to start and stop the Apache process (required, for instance, when you want to try other web servers), you need to run a few commands from the Linux terminal.

To stop your web server, enter the following at the command line:

```
$ sudo systemctl stop apache2
```

Or use:

```
$ sudo service apache2 stop
```

To start the Apache web server when it is already stopped, enter the following:

```
$sudo systemctl start apache2
```

Or use:

```
$ sudo service apache2 start
```

Do the following experiment to implement the previous commands. First run the following to view the apache2 processes:

```
$ ps xa | grep apache2
```

Then use the `systemctl stop` command and run the `ps` command again. You'll find that the apache2 processes have been "killed." You can also try a request from your browser. You'll find that the message "The site can't be reached" appears, as viewed in Figure 1-16.

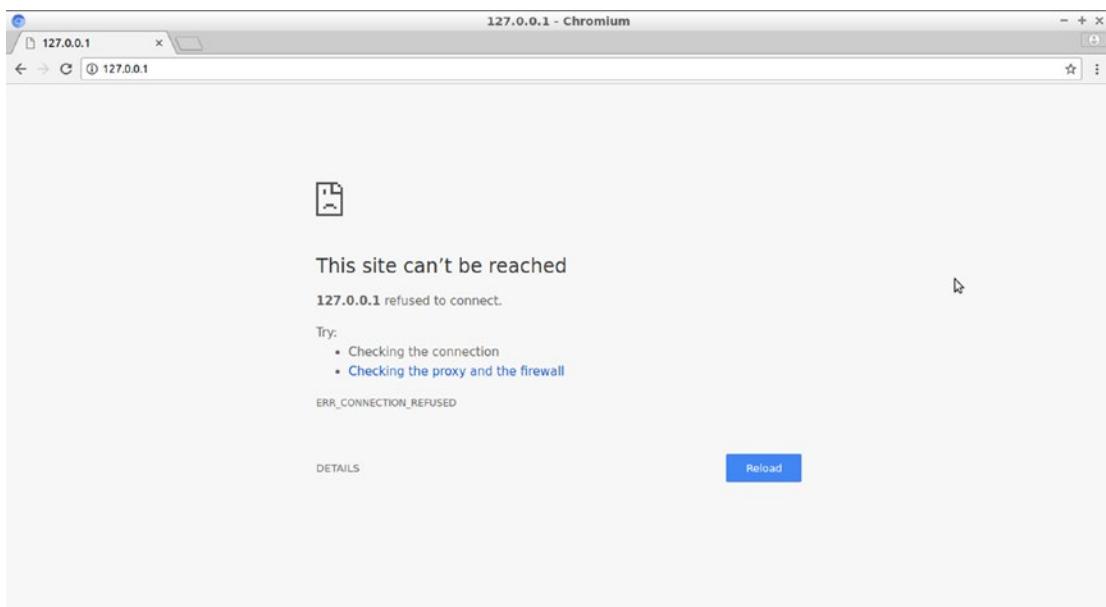


Figure 1-16. Trying to download a web page with the Apache server stopped

Use the `systemctl start` command to restore the Apache server.

Figure 1-17 shows the previous commands.



The screenshot shows a terminal window with the title bar "christos@pc: ~". The window contains a command-line session. The user runs "ps xa | grep apache2" to list all processes related to Apache. The output shows several processes with the command "/usr/sbin/apache2 -k start" and a status of "S". Then, the user runs "sudo systemctl stop apache2" to stop the server. After stopping, they run "ps xa | grep apache2" again, which shows no processes with "apache2" in the command line.

```
christos@pc:~$ ps xa | grep apache2
740 ? Ss 0:00 /usr/sbin/apache2 -k start
741 ? S 0:00 /usr/sbin/apache2 -k start
742 ? S 0:00 /usr/sbin/apache2 -k start
1546 pts/0 S+ 0:00 grep --color=auto apache2
christos@pc:~$ sudo systemctl stop apache2
christos@pc:~$ ps xa | grep apache2
1557 pts/0 S+ 0:00 grep --color=auto apache2
christos@pc:~$ sudo systemctl start apache2
christos@pc:~$ ps xa | grep apache2
1788 ? Ss 0:00 /usr/sbin/apache2 -k start
1789 ? S 0:00 /usr/sbin/apache2 -k start
1790 ? S 0:00 /usr/sbin/apache2 -k start
1847 pts/0 S+ 0:00 grep --color=auto apache2
christos@pc:~$
```

Figure 1-17. Displaying the Apache PIDs when the server runs and the absence of Apache PIDs when the server stops

To stop and then start the Apache server again, enter the following:

```
$ sudo systemctl restart apache2
```

To reload Apache (that is, to retain the process running and just update the configuration), enter the following:

```
$ sudo systemctl reload apache2
```

or you can enter the following:

```
$ sudo service apache2 force-reload
```

which is equal to the the following:

```
$ sudo service apache2 reload
```

The Apache server starts by default when the system boots. To disable that configuration, enter the following:

```
$ sudo systemctl disable apache2
```

To make Apache start on system boot, enter the following:

```
$ sudo systemctl enable apache2
```

Working with Virtual Hosts

Virtual hosts (vhhosts) or virtual servers enable a web server to host multiple sites simultaneously. Vhosts fall into one of the following categories:

- IP-based virtual hosts, where each vhost is dedicated to a site that makes use of a specific IP address from the web server IP addresses
- Port-based virtual hosts, where each vhost listens on a different port number for each site it hosts
- Name-based virtual hosts, where each vhost is dedicated to a site with a specific domain name

An Apache configuration can mix all these categories and also include a default host.

The initial Apache configuration file is called `000-default.conf` and is found in the `/etc/apache2/sites-available` directory. Apache requires a new configuration file to have a `.conf` file extension. At the Linux terminal, change the working directory to `sites-available`, which is the Apache directory that contains the configuration files.

```
$ cd /etc/apache2/sites-available
```

Open the default configuration file `000-default.conf` with the command `cat`, as shown here:

```
$ cat 000-default.conf
```

The file opens in a new window. The file contents are as follows:

```
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port
    # that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header
    # to match this virtual host. For the default virtual host (this
    # file) this value is not decisive as it is used as a last resort
    # host regardless.
```

```

# However, you must set it for any further virtual host explicitly.
#ServerName www.example.com

ServerAdmin webmaster@localhost
DocumentRoot /var/www/html

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host. For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf

</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

```

Each Apache configuration file includes three kinds of entries.

- Directives that define the behavior of the web server
- Containers, such as `VirtualHost`, that define blocks of directives for a vhost
- Comments, starting with a hash (#), that provide some aid in the usage of the directives

Container `<VirtualHost *:80>` pairs with `</VirtualHost>` just like an HTML start and end tag. The `VirtualHost` pair encloses all directives for the specific vhost and also defines the IP address and port number to which the given vhost should respond. In this

configuration file, the asterisk (*) corresponds to any IP address, and 80 corresponds to port 80, which is the default port for the HTTP protocol. You already tested this configuration in the previous sections when one of the following IP addresses was used in the address bar of your browser to download the directory index:

```
127.0.0.1  
192.168.1.100
```

Both addresses succeeded because the configuration accepted any valid IP address for this server. In the following section, you will create two different vhosts, each one serving a different IP address.

Using IP-Based Virtual Hosts

In this section, you'll create a new configuration file with gedit to implement two IP-based vhosts using the directives contained between the start/end `<VirtualHost>` containers. One vhost will be responsible for the loopback IP address, 127.0.0.1, and the other will be responsible for the private IP address of the web server, 192.168.1.100, both listening on port 80. The pair of IP address and port number for each vhost is indicated in the `<VirtualHost>` container. For instance, `<VirtualHost 127.0.0.1:80>` indicates the loopback IP address 127.0.0.1, and 80 is the default port number for the HTTP protocol. You can provide a different directory index to each vhost, such as `index1.html` for the first and `index2.html` for the second, using the `DirectoryIndex` directive. You can leave the other directives with their default values from the `000-default.conf` file.

At the Linux terminal, change the working directory to `sites-available` and use gedit to create the file `example1.conf`.

```
$ cd /etc/apache2/sites-available  
$ sudo gedit example1.conf
```

Enter the following configuration rules and save the file:

```
# 1st vhost  
<VirtualHost 127.0.0.1:80>  
    ServerAdmin webmaster@localhost  
    DocumentRoot /var/www/html  
    DirectoryIndex index1.html
```

```
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

# 2nd vhost
<VirtualHost 192.168.1.100:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html
    DirectoryIndex index2.html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Next you need to enable the new configuration using the `a2ensite` (apache2 enable site) command. At the Linux terminal, enter the following:

```
$ sudo a2ensite example1.conf
```

Or simply enter this:

```
$ sudo a2ensite example1
```

To enable the new Apache configuration, you need also to reload the web server.

```
$ sudo service apache2 force-reload
```

Create two web pages in the document root directory that will serve as the directory indexes for the two vhosts.

```
$ cd /var/www/html
$ sudo gedit index1.html
```

Enter the following HTML source code:

```
<!DOCTYPE html>
<html>
<head>
<title>Testing the Apache Web Server</title>
```

```
<style>
  body {
    background-color:lightblue;
  }
  p {
    background-color:red;
    font-size:20px;
  }
</style>
</head>
<body>
<p>Hello from 127.0.0.1</p>
</body>
</html>
```

In /var/www/html, create a web page called index2.html to be used as the directory index for the second vhost. Use the following HTML source code:

```
<!DOCTYPE html>
<html>
<head>
<title>Testing the Apache Web Server</title>
<style>
  body {
    background-color:red;
  }
  p {
    background-color:lightblue;
    font-size:20px;
  }
</style>
</head>
<body>
<p>Hello from 192.168.1.100</p>
</body>
</html>
```

CHAPTER 1 THE APACHE WEB SERVER

Open two tabs in your browser. On the first, enter the following address in the address bar:

127.0.0.1

On the second tab, enter the following address in the address bar:

192.168.1.100

With the current configuration, Apache displays different content for each request. The first request, with the IP address 127.0.0.1, resolves to index1.html (Figure 1-18).



Figure 1-18. Testing the first IP-based vhost

The second request with IP address 192.168.1.100 resolves to index2.html (Figure 1-19).

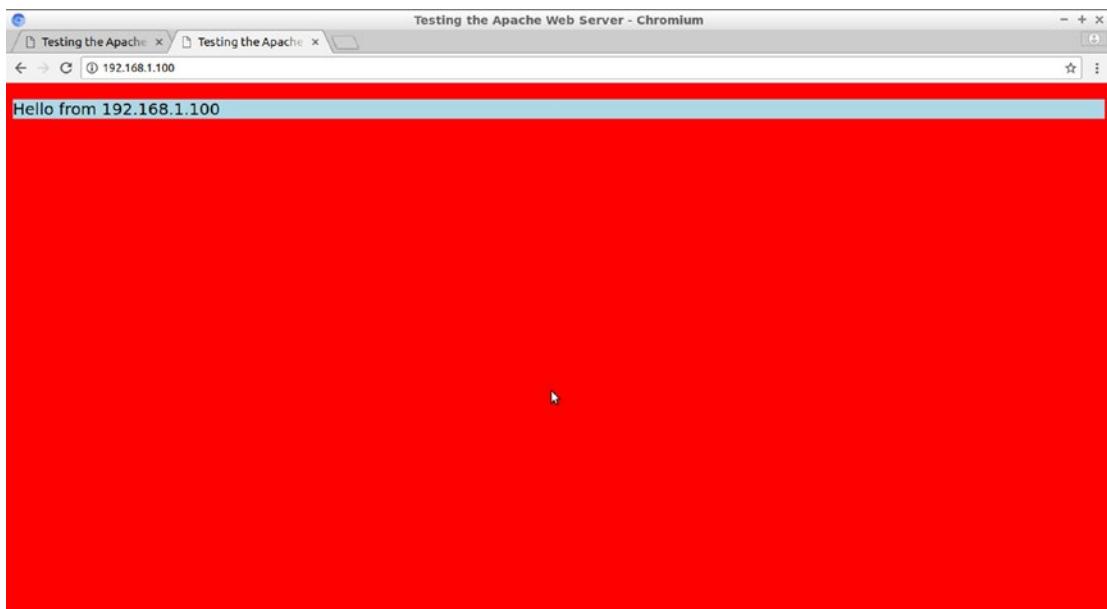


Figure 1-19. Testing the second IP-based vhost

Next you'll look at a similar example, this time using port-based virtual hosts.

Using Port-Based Virtual Hosts

The configuration used next will create two virtual hosts that listen on different ports, for instance port 8080 and port 8181. When anything other than the default port number for the HTTP protocol, port 80, is used, the port is required to be appended to the URL with a colon (:). Here's an instance:

192.168.1.100:8080

Using a different port than the default one makes the URL a bit more complicated, but that is not really a problem when you implement a DDNS service (see Chapter 4) that hides this complexity from the user.

In the `sites-available` directory, create a new configuration file.

```
$ cd /etc/apache2/sites-available  
$ sudo gedit example2.conf
```

In the file `example2.conf`, enter the following directives to direct the server to include port numbers 8080 and 8181 in the list of the listening ports:

```
Listen 8080
```

```
Listen 8181
```

To create a vhost that listens on port 8080 on any IP address of the server, you can use the asterisk notation in the `VirtualHost` container as `<VirtualHost *:8080>`. Similarly, for the vhost that listens on port 8181, also on any IP address of the server, use the `<VirtualHost *:8181>` container. The complete listing of `example2.conf` is as follows:

```
Listen 8080
Listen 8181
# 1st vhost
<VirtualHost *:8080>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html
    DirectoryIndex index3.html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

# 2nd vhost
<VirtualHost *:8181>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html
    DirectoryIndex index4.html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Save the new configuration file and enable it using the `a2ensite` command. Type the following:

```
$ sudo a2ensite example2.conf
```

or simply type the following:

```
$ sudo a2ensite example2
```

To enable the new Apache configuration, you also need to reload the web server.

```
$ sudo service apache2 force-reload
```

Sometimes the same configuration rules apply to different sites, causing unpredictable results. If you have to work on sites with conflicting rules, then you have to disable one of the sites. To disable, for instance, example1, use the a2dissite (apache2 disable site) command, as shown here:

```
$ sudo a2dissite example1
```

To have two new port numbers that the server will listen to, you need to set up ufw to permit connections for those port numbers. At the Linux command line, type the following:

```
$ sudo ufw allow 8080  
$ sudo ufw allow 8181
```

Create at the document root the two directory indexes, index3.html and index4.html, and refer to them in the example2.conf file. These indexes are for the vhosts that listen on port 8080 and on port 8181, respectively.

```
$ cd /var/www/html  
$ sudo gedit index3.html
```

In the gedit window that appears, enter the following HTML source code:

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Testing the Apache Web Server</title>  
<style>  
body {  
    background-color:red;  
}  
p {  
background-color:lightblue;  
font-size:20px;  
}  
</style>  
</head>
```

```
<body>
<p>Hello from 8080</p>
</body>
</html>
```

Next, in the document root directory, add the following HTML source code to index4.html:

```
<!DOCTYPE html>
<html>
<head>
<title>Testing the Apache Web Server</title>
<style>
    body {
        background-color:lightblue;
    }
    p {
        background-color:red;
        font-size:20px;
    }
</style>
</head>
<body>
<p>Hello from 8181</p>
</body>
</html>
```

To test the new vhosts, open two tabs in your browser. On the first tab, enter the following in the address bar:

127.0.0.1:8080

or enter the following:

192.168.1.100:8080

The directory index index3.html is displayed (Figure 1-20).

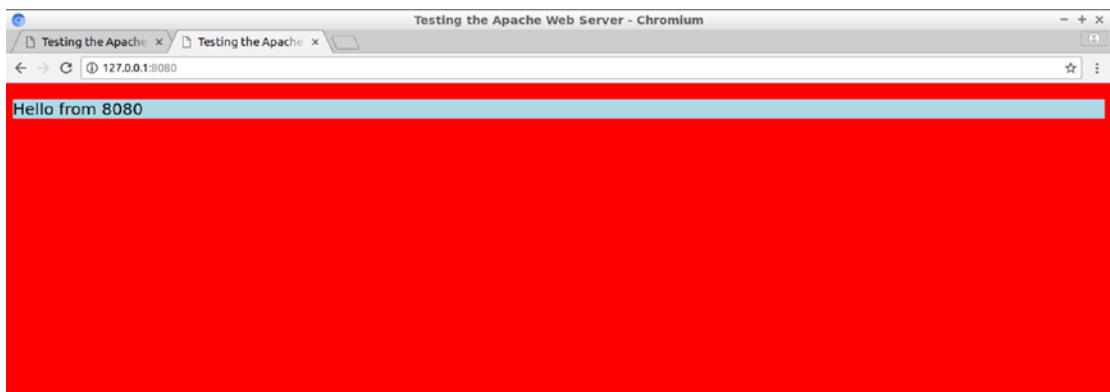


Figure 1-20. Testing the first port-based vhost

On the second tab, enter one of the following in the address bar:

127.0.0.1:8181

192.168.1.100:8181

The directory index `index4.html` is displayed (Figure 1-21).

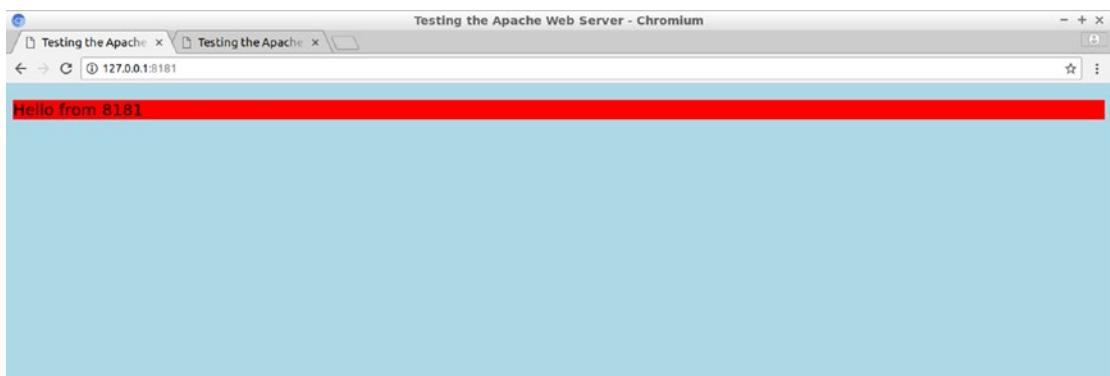


Figure 1-21. Testing the second port-based vhost

In the following section, you will create another pair of vhosts that serve requests for specific domain names in the Host field of the client's HTTP request.

Using Name-Based Virtual Hosts

The third approach for running multiple sites is to use name-based vhosts, where each vhost responds to the client's request if the client provides a host HTTP header value that matches its `ServerName` directive. You will not use the Domain Name System for your web server until Chapter 4, so for now you can only try the name-based virtual hosts locally, from the same computer as the server. To provide names to your computer, edit the `/etc/hosts` file as indicated in Figure 1-22. Use the following command at the terminal:

```
$ sudo gedit /etc/hosts
```

```
christos@pc: /etc/apache2/sites-available$ sudo cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      pc
192.168.1.100  webserver.com
192.168.1.100  myserver.com

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ffff::0 ip6-mcastprefix
ffff2::1 ip6-allrouters
ffff2::2 ip6-allrouters
christos@pc:/etc/apache2/sites-available$
```

Figure 1-22. Adding two more host names to `/etc/hosts`

As you can see, you have already used one entry from this file, `localhost`, which corresponds to IP address 127.0.0.1. Enter two more entries in `/etc/hosts`, both resolving to the web server's static private IP address.

```
192.168.1.100      webserver.com
192.168.1.100      myserver.com
```

Save the `/etc/hosts` file and try one of those names, for instance `webserver.com`, in your browser's address bar. The domain name resolves to the IP address 192.168.1.100, and with `sites` of the `example1` configuration disabled, the vhost that dispatches this request is the default one, `000-default.conf`. The web page downloaded is therefore the one indicated in Figure 1-23.



Figure 1-23. With no name-based vhost configured so far, the default vhost serves the webserver.com request

Create two name-based vhosts, each one responsible for one of the previous names. Use gedit to create the configuration file of the two vhosts.

```
$ sudo gedit example3.conf
```

In example3.conf, enter the following directives:

```
<VirtualHost *:80>
    ServerName webserver.com
    DocumentRoot "/var/www/html/test1"
</VirtualHost>

<VirtualHost *:80>
    ServerName myserver.com
    DocumentRoot "/var/www/html/test2"
</VirtualHost>
```

The first vhost listens to the default HTTP port, which is port 80. It is triggered when it receives an HTTP request with the value of the Host header set to `webserver.com`. The second vhost also listens to port 80. It waits for HTTP requests with the value of the Host header set to `myserver.com`.

The two sites use two different document roots, /var/www/html/test1 and /var/www/html/test2, and since no directory indexes are defined with the `DirectoryIndex` directive, the default one for Apache, `index.html`, will be used. Create next a file named `index.html` in each directory root. At the Linux terminal, enter the following:

```
$ cd /var/www/html/test1  
$ sudo gedit index.html
```

In the gedit window, insert the following HTML code and save the file:

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Testing the Apache Web Server</title>  
<style>  
body {  
    background-color:red;  
}  
  
p {  
    background-color:lightblue;  
    font-size:20px;  
}  
</style>  
</head>  
<body>  
<p>Hello from webserver.com</p>  
</body>  
</html>
```

Create a file called `index.html` in the second document root. At the Linux terminal, enter the following:

```
$ cd /var/www/html/test2  
$ sudo gedit index.html
```

In the gedit window, enter the following source code and save the file:

```
<!DOCTYPE html>
<html>
<head>
<title>Testing the Apache Web Server</title>
<style>
body {
    background-color:lightblue;
}
p {
    background-color:red;
    font-size:20px;
}
</style>
</head>
<body>
<p>Hello from myserver.com</p>
</body>
</html>
```

Enable the new configuration using the following:

```
$ sudo a2ensite example3
```

You also have to reload the web server to enable the new sites.

```
$ sudo service apache2 force-reload
```

In a browser on the computer that hosts the web server, open two tabs. In the first tab, enter the following in the address bar:

`webserver.com`

The web page `index.html`, located in `/var/www/html/test1`, is displayed in the browser's window, as displayed in Figure 1-24.



Figure 1-24. Testing the first name-based vhost

Use the following URL on the second tab:

`myserver.com`

The web page `index.html`, located in document root `/var/www/html/test1`, is displayed in the browser's window, as viewed in Figure 1-25.

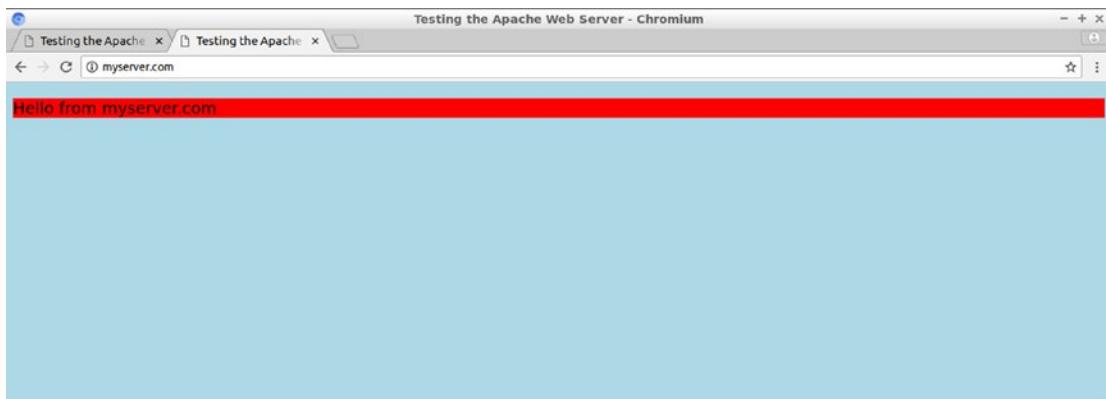


Figure 1-25. Testing the second name-based vhost

In Chapter 9 you'll test the `ServerName` directive with a globally valid domain name like `httpsserver.eu` and see how to obtain and apply such a domain name.

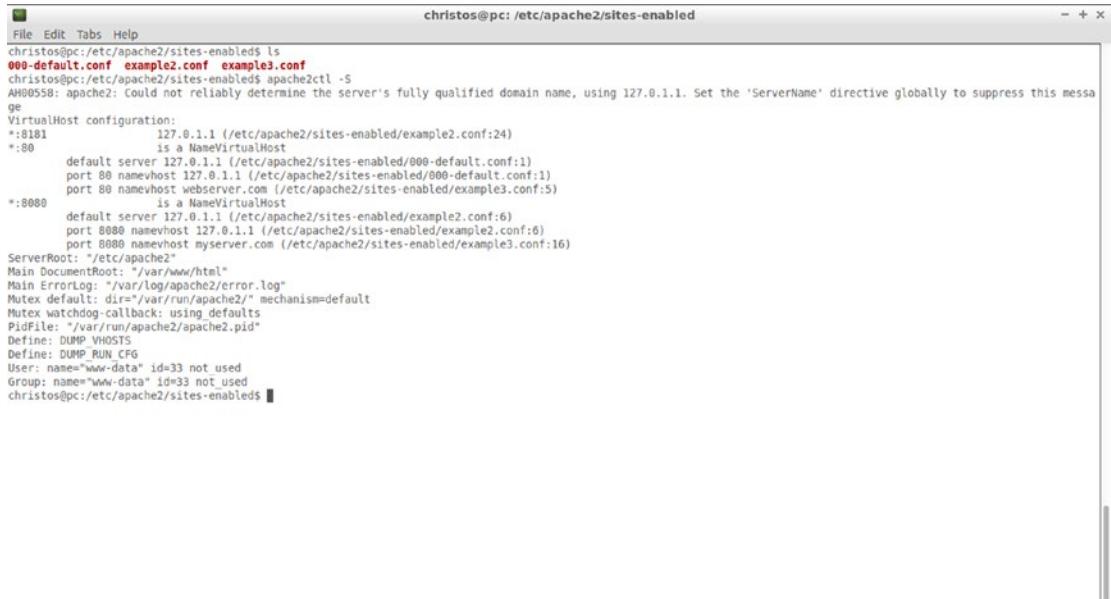
Inspecting the Overall Virtual Host Configuration

Each configuration enabled with the `a2ensite` command adds a symbolic link of itself to the directory `/etc/apache2/sites-enabled`. Use the following:

```
$ ls /etc/apache2/sites-enabled
```

As displayed in Figure 1-26, the `ls` command lists three configuration files: `000-default.conf`, `example2.conf`, and `example3.conf`. The configuration `example1.conf` is not included in this directory because it was disabled with the `a2dissite` command. Use the following to view the details of the overall configuration:

```
$ sudo apachectl -S
```



```
christos@pc:/etc/apache2/sites-enabled$ ls
000-default.conf example2.conf example3.conf
christos@pc:/etc/apache2/sites-enabled$ apache2ctl -S
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
VirtualHost configuration:
*:8181           127.0.1.1 (/etc/apache2/sites-enabled/example2.conf:24)
*:80            is a NameVirtualHost
      default server 127.0.1.1 (/etc/apache2/sites-enabled/000-default.conf:1)
      port 80 namevhost 127.0.1.1 (/etc/apache2/sites-enabled/000-default.conf:1)
      port 80 namevhost webserver.com (/etc/apache2/sites-enabled/example3.conf:5)
*:8080           is a NameVirtualHost
      default server 127.0.1.1 (/etc/apache2/sites-enabled/example2.conf:6)
      port 8080 namevhost 127.0.1.1 (/etc/apache2/sites-enabled/example2.conf:6)
      port 8080 namevhost myserver.com (/etc/apache2/sites-enabled/example3.conf:16)
ServerRoot: "/etc/apache2"
Main DocumentRoot: "/var/www/html"
Main ErrorLog: "/var/log/apache2/error.log"
Mutex default: dir="/var/run/apache2/" mechanism=default
Mutex watchdog/callback: using_defaults
PidFile: "/var/run/apache2/apache2.pid"
Define: DUMP_VHOSTS
Define: DUMP_RUN_CFG
User: name="www-data" id=33 not_used
Group: name="www-data" id=33 not_used
christos@pc:/etc/apache2/sites-enabled$
```

Figure 1-26. Listing the enabled sites and details of the Apache configuration

Figure 1-26 displays also the output of the `apachectl -S` command, which provides an overview of the vhost configuration.

Reading Apache Log Files

To get an idea of the number of visitors and see some of the connection details, such as the client IP address and the time, you can read the log files of Apache. For the vhosts examples of this chapter, the directives related to log files are used in the configuration files as follows:

```
ErrorLog ${APACHE_LOG_DIR}/error.log  
CustomLog ${APACHE_LOG_DIR}/access.log combined
```

To locate APACHE_LOG_DIR, use the following at the Linux terminal:

```
$ grep APACHE_LOG_DIR /etc/apache2/envvars
```

The Linux terminal responds with the following output:

```
export APACHE_LOG_DIR=/var/log/apache2$SUFFIX
```

With an empty value assigned for the variable \$SUFFIX in the file envvars, the Apache log directory resolves to /var/log/apache2.

Change the working directory to /var/log/apache2, where the file access.log is used to list the visitor details and the file error.log is used to report any errors.

```
$ cd /var/log/apache2
```

Move to another computer of your LAN and make an HTTP request to the Apache server. For instance, at the browser's address bar, type the following:

192.168.1.100

Read the access.log file using more, less, tail, or a similar command:

```
$ tail -n 1 access.log
```

The previous Linux shell command prints one line from the end, which corresponds to the last record of the access log. You can also use the -f argument to continuously print the last visitor's details of the access log:

```
$ tail -fn 1 access.log
```

The details about the new connection are displayed next, including the IP address of the computer used by the client, the date and time the request was issued, the URL used, the operating system, and the browser:

```
192.168.1.2 - - [14/Jun/2018:16:09:23 +0300] "GET /favicon.ico  
HTTP/1.1" 404 504 "http://192.168.1.100/" "Mozilla/5.0 (Windows NT 6.1)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36"
```

Summary

In this chapter, you set up and ran Apache, the first of two web servers used in this book. You created a number of virtual hosts (vhosts), each one corresponding to a different site. Two test sites were then run to display how Apache can discriminate between multiple sites belonging to each of the three main vhost categories: IP-based vhost, port-based vhost, and name-based vhost.

In the next chapter, you'll set up the PHP engine, namely, the interpreter that cooperates with Apache to create web pages with PHP source code that executes on the web server before the web page is dispatched to the client. With PHP, you enable sites to dispatch different content according to the client request and also to interface with other programs to work with the server, thus providing dynamic content to the visitors of the site.

CHAPTER 2

Server-Side Programming with PHP

In the beginning of the Internet, web servers dispatched only static content to the client. Soon, though, web servers became capable of dispatching dynamic content as well. To do this, a mechanism was needed that offered programming capabilities as well as the possibility of interfacing the terminal with other programs. Popular mechanisms for this purpose were the Common Gateway Interface (CGI) and later the PHP language. PHP is a recursive acronym for PHP: Hypertext Preprocessor. PHP is a general-purpose open source language that can be embedded in an HTML file or be included in its own stand-alone PHP file.

In this chapter, you will connect the Apache server to a PHP engine to create sites that accept client-specific data from HTML forms, which are processed, and then the results are dispatched back to the client's browser, thus serving dynamic content. You will be introduced to PHP, and then you will compare client-side programming in JavaScript to server-side programming in PHP. Last, you'll build a PHP-based site that validates the feature codes that can be entered online at a website in exchange for prizes.

The PHP Engine

A PHP-enabled web server can receive a request for a PHP file mainly in two ways.

- By using the URL of the PHP file in the web browser's address bar
- By submitting an HTML form, where the `action` attribute is set to the URL of the PHP file

On receiving the request for a PHP file, the web server locates the file and passes it to the PHP engine for processing. The PHP engine is a PHP interpreter that the web server is currently assigned. What the user actually views in the browser is not the

CHAPTER 2 SERVER-SIDE PROGRAMMING WITH PHP

PHP file itself but rather the output of all the PHP commands, sent back by the web server. For instance, the echo PHP command is the one used to output messages. Here's an example:

```
echo "Hello world!";
```

The start/end PHP tags (`<?php` and `?>`) are used as the delimiters for a PHP program.

```
<?php  
echo "Hello world!";  
?>
```

The PHP engine evaluates the previous program to a string, which is what the user actually sees when requesting `hello.php`.

Hello world!

Other than PHP commands, the PHP engine accepts HTML tags, for instance:

```
<?php  
echo "<b>Hello <br>world!</b>";  
?>
```

The user views the following message in bold:

**Hello
world!**

This capability makes PHP an ideal language for use on the Web.

PHP is a full-fledged programming language that you can use to make programs that run on the server side, limited only by your creativity. Here are a few epigrammatic uses for PHP:

- Interfacing with the operating system's shell, for instance the Linux terminal, and running terminal commands
- Interfacing with user programs, most commonly databases, and thus allowing the user to conduct a search and have the web server offer dynamic content

- Interfacing with the filesystem and allowing for reading and writing files on the server side
- Interfacing with remote servers by establishing network connections

You'll see examples of all these uses for PHP in the following chapters.

Installing and Testing PHP

In this book, PHP version 7.0 is used. To install PHP 7.0, follow the steps in this section.

Add the ondrej/php personal package archive (PPA).

```
$ sudo add-apt-repository ppa:ondrej/php
```

Run the update.

```
$ sudo apt-get update
```

Install PHP 7.

```
$ sudo apt-get install php7.0
```

Enable the Apache module `php7.0`, also installed with the previous command.

```
$ sudo a2enmod php7.0
```

Ensure that the `mpm_prefork` (prefork multiprocessing) Apache module is enabled and the `mpm_event` module is disabled.

```
$ sudo a2dismod mpm_event && sudo a2enmod mpm_prefork
```

Install also the CLI (Command Line Interface) PHP:

```
$ sudo apt-get install php7.0-cli
```

Restart Apache.

```
$ sudo systemctl restart apache2
```

To test your newly installed PHP engine, create a PHP file, named `info.php`, at the document root directory. Enter the following PHP source code:

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

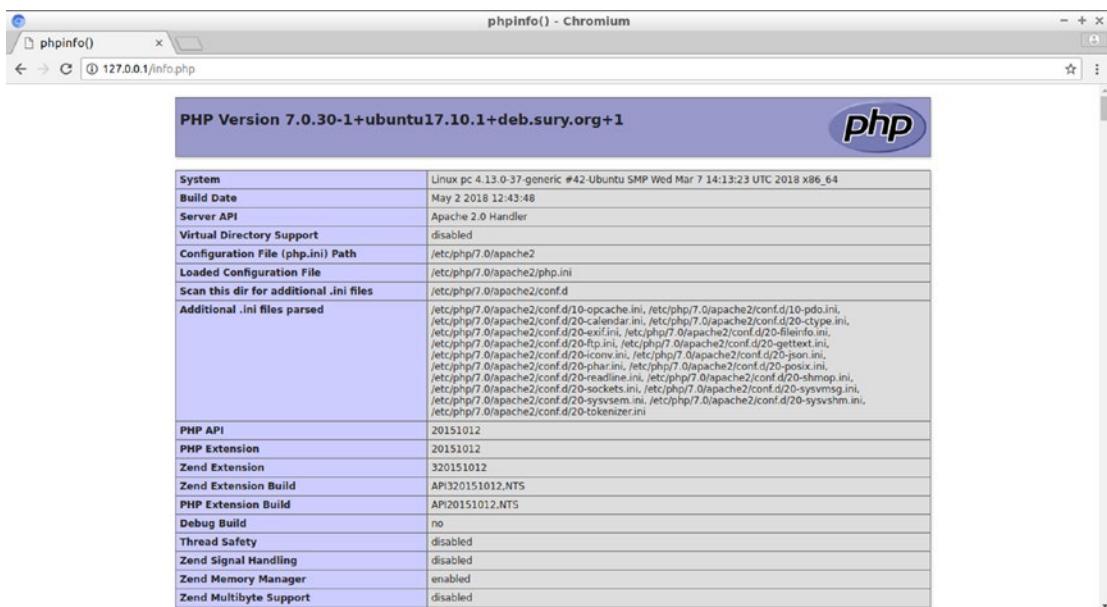
The previous code snippet includes just one command, the `phpinfo()` function call. As with each PHP command, it ends with a semicolon. The function `phpinfo()` returns a plethora of information about the PHP version, the operating system, the web server, etc., that spans multiple screens.

In your browser's address bar, type the URL that leads to `info.php`, for instance:

`192.168.1.100/info.php`

`192.168.1.100` in this example is the private IP address of the web server.

Figure 2-1 shows the info page.



PHP Version 7.0.30-1+ubuntu17.10.1+deb.sury.org+1	
System	Linux pc 4.13.0-37-generic #42-Ubuntu SMP Wed Mar 7 14:13:23 UTC 2018 x86_64
Build Date	May 2 2018 12:43:48
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/apache2
Loaded Configuration File	/etc/php/7.0/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/apache2/conf.d
Additional .ini files parsed	/etc/php/7.0/apache2/conf.d/10-opcache.ini, /etc/php/7.0/apache2/conf.d/10-pdo.ini, /etc/php/7.0/apache2/conf.d/20-calendar.ini, /etc/php/7.0/apache2/conf.d/20-ctype.ini, /etc/php/7.0/apache2/conf.d/20-datetime.ini, /etc/php/7.0/apache2/conf.d/20-dom.ini, /etc/php/7.0/apache2/conf.d/20-fp-ini.ini, /etc/php/7.0/apache2/conf.d/20-gettext.ini, /etc/php/7.0/apache2/conf.d/20-iconv.ini, /etc/php/7.0/apache2/conf.d/20-json.ini, /etc/php/7.0/apache2/conf.d/20-phar.ini, /etc/php/7.0/apache2/conf.d/20-posix.ini, /etc/php/7.0/apache2/conf.d/20-readline.ini, /etc/php/7.0/apache2/conf.d/20-shmop.ini, /etc/php/7.0/apache2/conf.d/20-sockets.ini, /etc/php/7.0/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.0/apache2/conf.d/20-sysvsem.ini, /etc/php/7.0/apache2/conf.d/20-sysvshm.ini, /etc/php/7.0/apache2/conf.d/20-tokenizer.ini
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	APR320151012.NTS
PHP Extension Build	APR20151012.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled

Figure 2-1. The `info.php` web page displayed in a browser

The information on the `info.php` web page will be used in the following examples. In the following section, you will try PHP independently from a web server; you'll use the command line to simulate the process followed by a web server equipped with a PHP engine.

Testing PHP Without a Web Server

Sometimes it is useful to run and test PHP source code and view the result as it would appear in the user's browser without requiring a web server. There are PHP interpreters, like CLI PHP, that run from the command line of your operating system; however, they do not deliver the exact effect because they do not process HTML tags. There are also online PHP editors, some of which like phptester.net process HTML tags, but you can't combine them with the software of your computer. For this purpose, the following example can be used, which relies on CLI PHP.

To test whether you have already installed PHP CLI installed, enter the following at the command line:

```
$ php -v
```

The output on my system is as follows:

```
PHP 7.1.17-0ubuntu0.17.10.1 (cli) (built: May 9 2018 17:28:01) ( NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2018 Zend Technologies
with Zend OPcache v7.1.17-0ubuntu0.17.10.1, Copyright (c) 1999-2018, by
Zend Technologies
```

To view the available options, use the `-h` (help) switch.

To simulate the process of using PHP code inside a web page and viewing the result with a browser, without requiring a web server, use the following steps. First create a PHP file that uses HTML tags and also PHP source code. Specifically, create a file called `a.php` in your home directory using these commands:

```
$ cd ~
$ gedit a.php
```

Enter the following source code and save the file:

```
<html>
<head>
<title>Testing PHP without a web server</title>
</head>
<body>
<b>
```

```
<?php
$command = "users";
$output = shell_exec($command);
echo $output;
?>
</b>
</body>
</html>
```

Three PHP commands are included between the start (`<?php`) and the end (`?>`) PHP tags. The first one sets the `$command` variable to the value of the command name that will be executed by the operating system. In this example, the shell command `users` was used, which lists the logged-in users. The second command uses `shell_exec()`, which is the PHP function that interfaces with the shell and executes the command given in the argument, in this case `users`, which is the value of `$command`. As a result, the `users` command is executed, and its output is stored to the `$output` variable. The third command prints the contents of the variable `$output` in bold since the PHP block is included between the `` start and end tags.

The file `a.php` is the original PHP file that will be evaluated and viewed from a browser; however, for this file, the auxiliary file `c.php` is required to run. Create the file `c.php` next.

```
$ gedit c.php
```

Enter the following source code and save the file:

```
<?php
shell_exec("php a.php > b.html");
shell_exec("firefox b.html");
?>
```

What the program `c.php` does is run the code of file `a.php` with PHP CLI and then redirect (`>`) the output to the HTML file `b.html`.

```
$ php a.php > b.html
```

Next the Firefox web browser is invoked from the command line to open the web page `b.html`. This is the equivalent of running the next command from the command line:

```
$ firefox b.html
```

After creating `a.php` and `c.php`, the command you issue from the command line that starts everything is as follows:

```
$ php c.php
```

Figure 2-2 displays the result of running the previous command: the Firefox window opens and lists the currently logged-in users.



Figure 2-2. Running `c.php` from CLI PHP results in displaying file `a.php` in a web page

A simple but efficient way to debug PHP code received from a web server by the user browser is to right-click the web page and select View Page Source (or similar) from the pop-up menu. As shown in Figure 2-3, a new tab opens with the evaluated PHP code.

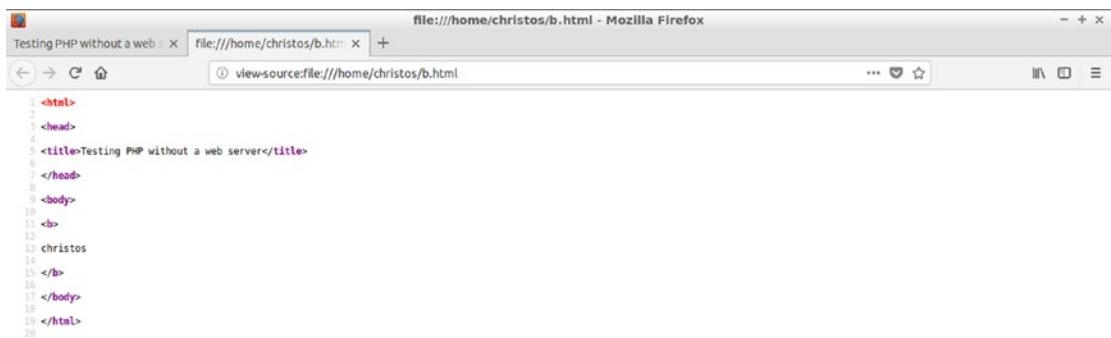


Figure 2-3. The source code of the evaluated PHP code

As shown in the web page source resulting from the evaluated PHP code, just one user is currently logged in, and the username is included between the start/end bold tags in the body section of the HTML source code. Starting in the next section, all PHP examples will run through the web server. The Apache web server, introduced in the previous chapter, will be used here (starting in Chapter 5 you will use another open source web server, the Lighttpd).

Running Your First PHP Examples from the Web Server

You will now test PHP files evaluated by the PHP engine and dispatched by the Apache web server. The following examples show how variables are combined with strings and the difference between double and single quotes when used with strings. Also, I explain how you can mix JavaScript and PHP in the same command. Understanding those details in the source code is essential for understanding the web-based projects included in the rest of the book.

Working with Variables and Strings

First you'll see how PHP strings and variables can be combined. Create a new file with a .php file extension in your document root directory.

```
$ cd /var/www/html  
$ sudo gedit test.php
```

Enter the following source code that does not include any strings and save the test.php file:

```
<!DOCTYPE html>  
<html>  
<head>  
<title>  
Testing PHP  
</title>  
<style>  
p {
```

```
font-size:24px;  
color:blue;  
</style>  
</head>  
<body>  
<?php  
// working with variables  
$var1 = 5;  
$var2 = 8;  
$var3 = $var1 + $var2;  
echo $var3;  
?  
</body>  
</html>
```

Test the PHP file by entering its URL in the address bar of your browser. If you test from the web server, use the following:

`localhost/test.php`

If you also test from the web server or from another computer of your LAN, you can use the following:

`192.168.1.100/test.php`

Here, 192.168.1.100 is the private IP address of the computer that hosts the web server. Figure 2-4 shows the web page downloaded to the browser.

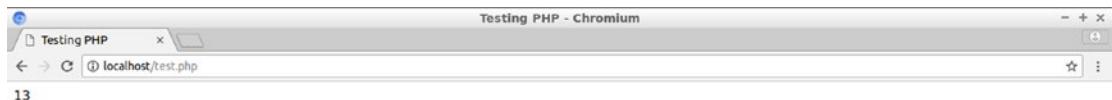


Figure 2-4. The evaluated `test.php` web page

The PHP program uses three variables: \$var1 and \$var2 that are initialized to specific values, and \$var3, whose value is calculated by the addition of the values of the previous two variables. The echo PHP command is used to output the \$var3 variable's value to the web page displayed in your browser. Also, a double forward slash (//) is used to mark a line as a comment. To include multiple continuous lines in a comment, use the slash and star notation, that is, /* for starting and */ for ending.

Notice that a variable name in PHP starts with the dollar sign (\$). Prepending a variable with the dollar sign may look like overhead, but it is handy when you include the variable in a string. To display this feature, replace the previous PHP code in test.php with the following:

```
<?php
// working with variables
$var1 = 5;
$var2 = 8;
$var3 = $var1 + $var2;
echo "The sum of $var1 and $var2 is $var3";
?>
```

Strings in PHP can be included between double ("") or single ('') quotes. When a variable is inserted in a string with double quote delimiters, it is evaluated to its actual value. The web page in this example displays the following:

The sum of 5 and 8 is 13

This is not the case, however, when single quotes are used as string delimiters. Replace the previous PHP code in test.php with the following:

```
<?php
// working with variables
$var1 = 5;
$var2 = 8;
$var3 = $var1 + $var2;
echo 'The sum of $var1 and $var2 is $var3';
?>
```

The web page displays the following:

The sum of \$var1 and \$var2 is \$var3

You can also use the concatenation operator (.) to concatenate strings and variables. Change the previous PHP code in `test.php` to the following:

```
<?php
// working with variables
$var1 = 5;
$var2 = 8;
$var3 = $var1 + $var2;
echo "The sum of " . $var1 . " and ". $var2 . " is " . $var3;
?>
```

The web page `test.php` evaluates now to the following:

The sum of 5 and 8 is 13

You can instead use single quotes to have the same effect.

```
<?php
// working with variables
$var1 = 5;
$var2 = 8;
$var3 = $var1 + $var2;
echo 'The sum of ' . $var1 . ' and ' . $var2 . ' is ' . $var3;
?>
```

Escaping Double or Single Quotes in PHP

To include double quotes in a string printed in the echo command without getting them mixed up with the message's double quotes used as delimiters, you can prepend them with the backslash (\) character. This is referred as *escaping* the double quotes. Another method is to include the message with the double quote characters in single quote delimiters. For instance, the following PHP file source code allows the double quote to be included in the echo messages:

```
<?php
echo "This message includes a double quote(\"<br>\"");
echo 'This message also includes a double quote(")<br>';
?>
```

Similarly, you can include single quotes in a string by escaping them with the backslash character in a single-quoted string or use double quotes as string delimiters. The following PHP example displays this technique:

```
<?php  
echo 'This message includes a single quote(\')<br>';  
echo "This message also includes a single quote(')";  
?>
```

Mixing JavaScript and PHP

While JavaScript is used for the client-side programming language and PHP is used for the server-side language, there are cases where HTML and JavaScript have to be mixed with PHP. This section outlines the general rules for mixing JavaScript, PHP, and HTML tags.

To start with, a PHP file may include HTML tags such as a common HTML file, for instance:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
p{  
font-size:24px;  
}  
</style>  
</head>  
<body>  
<?php  
echo "<p>Hello World!</p>";  
?>  
</body>  
</html>
```

You can instead use a PHP file with HTML-free content like the following:

```
<?php  
echo "Hello World!";  
?>
```

A PHP file may also include JavaScript source code, as in the following example:

```
<script>
document.write("Hello ");
</script>

<?php
echo "World!";
?>
```

In the previous PHP file, the two source code blocks are not mixed, and both functions individually print their own part on the web page, with JavaScript running when the browser loads the page and PHP running before the web page is submitted to the user's browser. Next you will see some cases where the PHP and JavaScript source code are mixed. As a general rule, HTML tags, like `<script>`, `<style>`, etc., may be included in a PHP file, however, not inside a PHP source code block, defined between the `<php` and `?>` delimiters. Otherwise, the PHP code won't run, and the evaluated web page will display an error message (HTTP error 500). The following example will run because the start and end script tags are outside the PHP block and the JavaScript commands are printed in their place with a PHP echo command:

```
<script>
<?php
echo '
alert("Hello World!");
';
?>
</script>
```

Also, the following source code snippet will run because all script tags are included in the echo PHP message, and therefore the whole script is printed to the web page when the PHP source code evaluates:

```
<?php
echo '
<script>
alert("Hello World!");
</script>
';
?>
```

In the next example, the source code will not run because the script's start and end tags are included inside the PHP block, however, without being included in the echo command:

```
<?php
<script>
echo '
alert("Hello World!");
';
</script>
?>
```

In this book, you will see a lot of cases where you have to use PHP and JavaScript in the same command. For instance, sometimes a PHP variable needs to appear to the user in a pop-up window instead of being typed into the browser's window with the echo command. With PHP belonging to the server side, you don't give control to the user's browser to create a pop-up window. You instead include JavaScript in the PHP evaluated page. The following PHP file contains PHP code in two blocks. In the first block, for simplicity, the variable \$name is hardwired to a specific value, and in the second block, a PHP block is combined with JavaScript code as part of the alert() text so that the PHP variable \$name appears in a pop-up window message:

```
<?php
$name = "Madison";
?>

<script>
alert("Hello <?php echo $name; ?> ");
</script>
```

Consider also the example where the pop-up window appears only if the value of \$name is 'Jennifer'. In this case, the whole script needs to be conditionally included in the PHP source code, but it may never run.

```
<?php
$name = 'Susan';
if ($name === Susan) {
echo '
```

```
<script>
alert("Hello ' . $name . ' ");
</script>
';
} else {
echo 'Hello';
}
?>
```

In the previous example, for simplicity, the `$name` variable was hardwired to the source code instead of allowing for multiple values, e.g., values derived from an HTML form. Like with the `script` tag, PHP code can be mixed with other HTML tags.

Here's an example of source code that won't run:

```
<?php
<b>
echo "Hello World!";
</b>
?>
```

On the other hand, the following example runs as expected:

```
<b>
<?php
echo "Hello World!";
?>
</b>
```

Or you can also use the following:

```
<?php
echo "<b>Hello World!</b>";
?>
```

In the examples used in this section, the PHP program initialized the PHP variables to their values. Next you will use HTML forms that the site visitor uses to submit user-specific data to the web server and then set the PHP variables according to the user preferences.

Setting the PHP Variables with the GET Method

The most common way to allow the user to send data to the web server is to use an HTML form that implements the GET or POST method and also refers to the URL of the PHP program that receives and processes the user data from the `action` attribute of the form element. In the following example, you will create a web page that includes an HTML form with the `method` attribute set to GET. First create the HTML file `get.html` that includes a form to enable the user to insert numbers in two fields and also to select an arithmetic operator with a drop-down list. When the submit button is clicked, the user data is transmitted to the `get.php` program, which is located, as the relative URL of the `action` attribute indicates, on the web server that dispatched the current web page and in the same directory as the current web page.

Use the following commands to switch to the server's document root and create the file `get.html`:

```
$ cd /var/www/html  
$ sudo gedit get.html
```

Insert the following HTML source code:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
input, select{  
font-size:24px;  
}  
</style>  
</head>  
<body>  
<form method="get" action="get.php">  
<input type="number" name="n1">  
<select name="operator">  
  <option value="add">+</option>  
  <option value="subtract">-</option>  
  <option value="multiply">*</option>  
  <option value="divide">/</option>  
</select>
```

```
<input type="number" name="n2">
<input type="submit" value="Calculate">
</form>
</body>
</html>
```

Create the get.php file in the same directory.

```
$ sudo gedit get.php
```

Insert the following HTML source code that embeds the PHP source code:

```
<!DOCTYPE html>
<html>
<head>
<style>
p{
font-size:24px;
}
</style>
</head>
<body>
<p>
<?php
$num1 = $_GET["n1"];
$num2 = $_GET["n2"];
$operator = $_GET["operator"];
switch($operator) {
    case "add":
        echo $num1 + $num2;
        break;
    case "subtract":
        echo $num1 - $num2;
        break;
    case "multiply":
        echo $num1 * $num2;
        break;
}
```

```

case "divide":
    echo $num1 / $num2;
    break;
}
?>
</p>
</body>
</html>

```

With the GET form method array `$_GET`, a global PHP variable is passed by the PHP engine to `.php`, the PHP action program. The `$_GET` array elements consist of the data sent by the form. By using the name of the form's element with `$_GET` as the index to submit a value, the specific value is returned. For instance, with the following command, the PHP variable `$num1` gets the value submitted by the form's field named `n1`:

```
$num1 = $_GET["n1"];
```

Similarly, the value entered by the user in the second field of the type number is assigned to the PHP variable `$num2`.

```
$num2 = $_GET["n2"];
```

The drop-down list of the form submits also a value that corresponds to one of the following arithmetic operators: addition (+), subtraction (-), multiplication (*), division (/). The user choice for the operator is then accessed with the PHP `$operator` variable as follows:

```
$operator = $_GET["operator"];
```

Next, `$operator` enters a PHP switch command that according to the `$operator` value performs the corresponding arithmetic operation between `$num1` and `$num2`.

To test the client-server interaction of this example, use another computer in your LAN and enter the following in the browser's address bar:

`192.168.1.100/get.html`

`192.168.1.100` is in this example the private IP address of the computer that hosts your web server. Alternatively, you can test the example from the same computer where your web server resides by entering the following:

`127.0.0.1/get.html`

Fill in the fields of the form with numeric values and select an arithmetic operator. Figure 2-5 displays the web page `get.html` with the values 200 and 100 entered in the number fields and also the multiplication sign selected from the drop-down list. Click the Calculate button to submit the form data to `get.php`.

A screenshot of a Chromium browser window. The title bar says "localhost/get.html - Chromium". The address bar shows the URL "localhost/get.html". Below the address bar are two input fields: the first contains "200" and the second contains "100". To the right of these fields is a "Calculate" button. The rest of the screen is mostly blank.

Figure 2-5. The web page `get.html` with the form fields completed

Figure 2-6 displays the web page resulting from the evaluation of `get.php`, the action program in the web server, that received the data from the `get.html` form. The PHP code multiplies the two numeric variables `$num1` and `$num2`, and the result is printed with the `echo` command to the output returned by the web server to the client browser.



Figure 2-6. The evaluated `get.php` web page as it appears in the user browser

The data submitted by the user is appended as the query string in the URL. The query string, for this example, is as follows:

`n1=200&operator=multiply&n2=100`

This includes the name-value pairs of all form fields submitted to the server and comprises the info received by the PHP program, which is used to fill the `$_GET` array and provide the PHP program with the user-defined variables.

Setting the PHP Variables with the POST Method

Let's now try the POST method of the form element, which is another option for submitting data to the web server. Create the file `post.html` as follows:

```
$ cd /var/www/html  
$ sudo gedit post.html
```

Enter the following HTML source code, which differs from the `get.html` code only by the values of the form attributes `method` and `action`:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
input, select{  
font-size:24px;  
}  
</style>  
</head>  
<body>  
<form method="post" action="post.php">  
<input type="number" name="n1">  
<select name="operator">  
 <option value="add">+</option>  
 <option value="subtract">-</option>  
 <option value="multiply">*</option>  
 <option value="divide">/</option>  
</select>  
<input type="number" name="n2">  
<input type="submit" value="Calculate">  
</form>  
</body>  
</html>
```

The method used is therefore POST, and the PHP file post.php is used to handle the user request. Create the file post.php also at the document root as follows:

```
$ cd /var/www/html  
$ sudo gedit post.php
```

Enter the following source code and save the file:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
p{  
font-size:24px;  
}  
</style>  
</head>  
<body>  
<p>  
<?php  
$num1 = $_POST["n1"];  
$num2 = $_POST["n2"];  
$operator = $_POST["operator"];  
switch($operator) {  
    case "add":  
        echo $num1 + $num2;  
        break;  
    case "subtract":  
        echo $num1 - $num2;  
        break;  
    case "multiply":  
        echo $num1 * $num2;  
        break;
```

```
case "divide":  
    echo $num1 / $num2;  
    break;  
}  
?>  
</p>  
</body>  
</html>
```

To test the client-server interaction of this example, use another computer in your LAN and in the browser's address bar enter the following:

192.168.1.100/post.html

Here, 192.168.1.100 is the private IP address of the computer that hosts your web server. Alternatively, you can test the example from the same computer where your web server resides by entering the following:

127.0.0.1/post.html

Provide a numeric value for each of the two fields and choose also an arithmetic operator from the drop-down list. In Figure 2-7, the two form fields of the web page post.html are completed with the numeric values 3000 and 5, respectively, and the division operator is selected from the drop-down list.

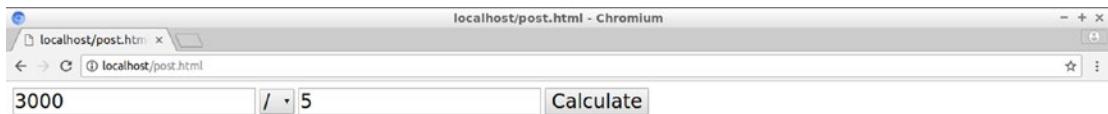


Figure 2-7. The web page post.html with the form completed

Click the Calculate button. The form values are submitted to post.php. On the web server, the PHP engine is invoked, and the form values are retrieved from the global PHP variable `$_POST`, which is an array similar to `$_GET` with the values submitted from the

form elements. The PHP source code performs the division between 3000 and 5. The result is output with the echo command to the web page, as displayed in Figure 2-8, and is sent back from the web server to the user.



Figure 2-8. The evaluated post.php web page as it appears in the user's browser

Notice that in the address bar of your browser the URL remains the same. With the POST method, the data is not included in the URL of the HTTP request line, as with the method GET, but is rather appended to the body of the HTTP request sent to the web server. In the following section, another form is used to submit data to a PHP program on the server, and this process is compared with the JavaScript code that performs the corresponding calculation locally.

Running Client-Side vs. Server-Side Programs

You will next compare how server-side programs and client-side programs act, using JavaScript and PHP source code embedded in the same web page. Both languages are used to perform the same operation, specifically an arithmetic addition. JavaScript runs on the client side in the client's browser and displays the web page. PHP runs on the server side since the addition is performed on the web server that provides the specific web page.

A simple pattern for defining the programming process is to input the data to the source code and output the results. With JavaScript, the web server submits the source code via the network, and then the input/output data is used locally on the client's system. With PHP, the contrary applies: the client submits the data to the server, which makes the calculation remotely to the client, and the output is sent back to the client, as shown in Figure 2-9.

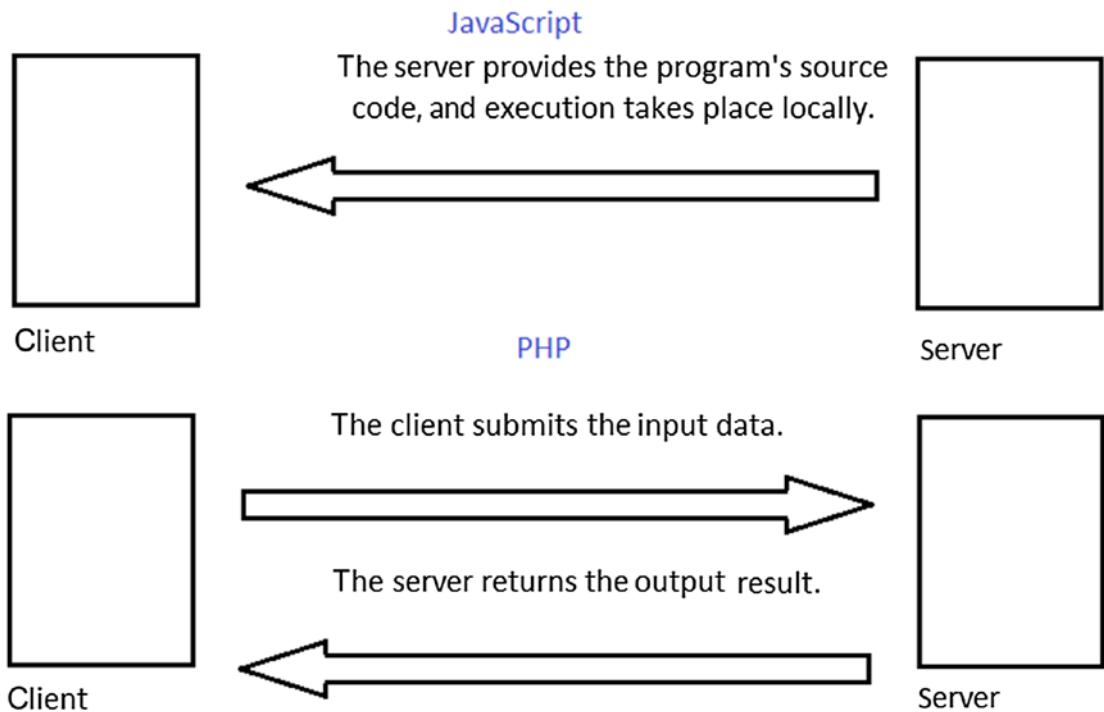


Figure 2-9. Defining the programming process

The JavaScript/PHP Addition Web Page

In this section, you'll create two calculators, which for simplicity here only do addition, in the same web page, `program.html`. The first is a client-side calculator that runs JavaScript, and the second is a server-side calculator that runs PHP. At the Linux terminal, create `program.html` in the document root of the web server with the following terminal commands:

```
$ cd /var/www/html
$ sudo gedit program.html
```

In the gedit window, enter the following HTML page that embeds the PHP and JavaScript source code and save the file:

```
<!DOCTYPE html>
<html>
<head>
<title>Client side and Server side programs</title>
```



```
<input type="text" name="t4" size=5>
<span>+</span>
<input type="text" name="t5" size=5>
<input type="submit" value="=">
</form>
</span>
</div>
<script>
function f1() {
var x = Number(document.getElementById("t1").value);
if (isNaN(x)) {
alert("Please enter a number");
return false;
}
var y = Number(document.getElementById("t2").value);
if (isNaN(y)) {
alert("Please enter a number");
return false;
}
var z = x + y;
document.getElementById("t3").value = z;
}
</script>
</body>
</html>
```

The HTML file is used to perform addition in two ways: by using JavaScript and by using PHP. For JavaScript, the fields t1 and t2 are used to fill the operands and t3 is used to output the sum. When the button with the equal sign is clicked, the calculation is performed. This button is assigned function f1() for handling the onclick event. The JavaScript section implements function f1(). This routine sets the JavaScript variable x to the current value of the element t1 (the first textbox) and sets the JavaScript variable y to the current value of the element t2 (the second textbox). The JavaScript function Number() is then used to convert the textbox value to a numeric value. Without using Number(), the value 5 would represent the character 5, and the addition 5+5 would result

in 55. Because with JavaScript the calculation is performed locally in the browser, no data is transferred to and from the web server, and therefore when using JavaScript, the URL never changes.

The second way to do the addition is to use PHP. With PHP the values of textboxes t4 and t5 are transferred to the web server via a form submission when the submit button is clicked. The submit button looks like the equal button from the JavaScript technique, but it functions by sending the data to the web server contrary to the JavaScript button that invokes a function to make the calculation locally. The program that receives the data on the web server is determined by the value of the action attribute of the form. This is set to addition.php, which is a PHP file that should be found in the document root of the web server, as indicated by the relative URL that includes only the file name.

Create this program with the following command:

```
$ sudo gedit addition.php
```

In the gedit window, enter the following source code and click the Save button:

```
<!DOCTYPE html>
<html>
<head>
<style>
body{
background-color:lime;
font-size:24px;
}
</style>
</head>
<body>
<?php
$x = $_GET["t4"];
$y = $_GET["t5"];
$z = $x + $y;
echo $z;
?>
</body>
</html>
```

The PHP file `addition.php` includes a PHP block where the value sent by textbox `t4` is retrieved from the `$_GET[]` global PHP array, and it is assigned then to the PHP variable `$x`. Similarly, the value sent by textbox `t5` is stored in the PHP variable `$y`. Next, `$x` and `$y` are added together, and the result is assigned to the variable `$z`. The value of `$z` is printed with the `echo` command to the web page sent back to the browser.

To try the calculators, enter the following URL in your browser's address bar:

`localhost/program.html`

On the web page you can test the JavaScript code first. In the example in Figure 2-10, the result of adding 6 and 9 appears when the first equal sign button is clicked.

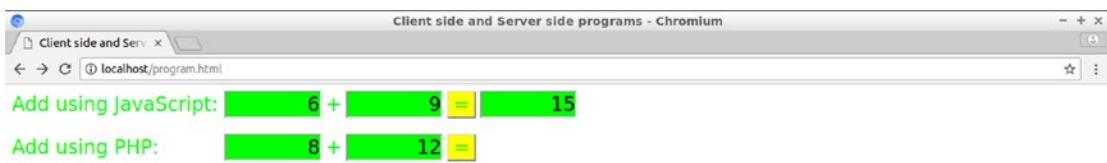


Figure 2-10. Performing the addition locally with the JavaScript calculator

Test the PHP addition next. Enter two numbers in the PHP fields. In the example in Figure 2-9, the numbers 8 and 12 were entered. Click the button with the equal sign, which is an HTML form submit button. The evaluated source code of the action file, `addition.php`, appears, as shown in Figure 2-11.



Figure 2-11. The result of the PHP addition as returned from the web server

In the case of JavaScript, the source code is executed locally on the same computer, and the web page remains the same. For the PHP code, the code is executed in the web server, and the reply is submitted to the client with a new page. In the previous example, it may look like an unfair comparison because PHP provides the result in a web page that is not consistent with the initial one. With the next versions of the files `program.html` and `addition.php`, this can be fixed.

The Second Version of the JavaScript/PHP Addition Web Page

Create a file called `program2.html` with gedit or any other text editor. Enter the following source code in `program2.html`:

```
<!DOCTYPE html>
<html>
<head>
<title>Client side and Server side programs</title>
<style>
input{
font-size:24px;
text-align:right;
background-color:lime;
}
input[type="submit"], input[type="button"] {
background-color:yellow;
color:lime;
}
button{
font-size:24px;
color:lime;
background-color:yellow;
}
span{
font-size:24px;
color:lime;
}
```

CHAPTER 2 SERVER-SIDE PROGRAMMING WITH PHP

```

var z = x + y;
document.getElementById("t3").value = z;
}
</script>
</body>
</html>

```

The HTML file `program2.html` differs from `program.html` because of the value of the `action` attribute of the `form` element. Instead of `addition.php`, the program `addition2.php` is used. But there is another difference as well: the first three textboxes, `t1`, `t2`, and `t3`, used for the JavaScript section now use a `name` attribute, which will be used to transfer their values to the web server when the submit button is clicked. Those values will not be used for any calculation by the PHP web server program, `addition2.php`, but the values will be submitted to the `addition2.php` source code to be returned as values in the corresponding textboxes in the evaluated page. Thus, a continuity will be retained with the state of the original page, `program2.html`.

Create a file named `addition2.html` in the document root of your server with the following commands:

```

$ cd /var/www/html
$ sudo gedit addition2.php

```

Enter the following code and save the file:

```

<!DOCTYPE html>
<html>
<head>
<title>Client side and Server side programs</title>
<style>
input{
font-size:24px;
text-align:right;
background-color:lime;
}
input[type="submit"], input[type="button"] {
background-color:yellow;
color:lime;
}

```

CHAPTER 2 SERVER-SIDE PROGRAMMING WITH PHP

```
button{
font-size:24px;
color:lime;
background-color:yellow;
}
span{
font-size:24px;
color:lime;
}
</style>
</head>
<body>
<?php
if (isset($_GET["t1"])){
$t1 = $_GET["t1"];
}
if (isset($_GET["t2"])){
$t2 = $_GET["t2"];
}
if (isset($_GET["t3"])){
$t3 = $_GET["t3"];
}
if (isset($_GET["t4"])){
$t4 = $_GET["t4"];
}
if (isset($_GET["t5"])){
$t5 = $_GET["t5"];
}
if (isset($t4) && isset($t5)){
$t6 = $t4 + $t5;
}
?>
<form name="f1" action="addition2.php" method="GET">
<div>
<span>Add using JavaScript: </span>
```


CHAPTER 2 SERVER-SIDE PROGRAMMING WITH PHP

In this new version of the PHP file, the result of the PHP addition appears as the value of a new textbox, named t6, visually simulating the JavaScript calculator. You can test both calculators in this new version. As displayed in Figure 2-12, the fields of the JavaScript calculator reflect the addition previously issued, and in the PHP calculator the operand fields are set for the addition.

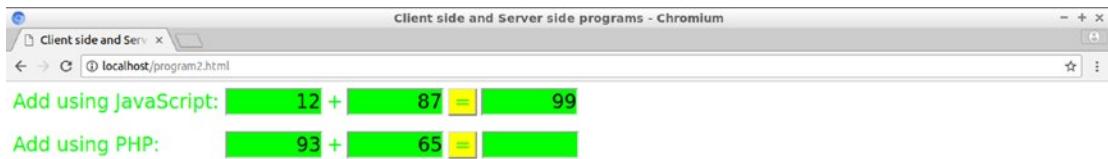


Figure 2-12. The web page program2.html before submitting the form

After clicking the submit button, the result appears in the evaluated addition2.php web page. As displayed in Figure 2-13, addition2.php has the same look and feel as the previous web page, program2.html. Moreover, all textbox values from both JavaScript and PHP sections are retained.

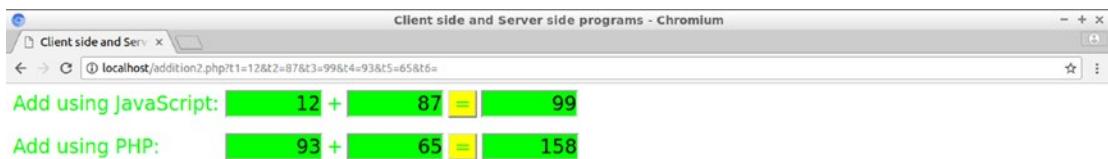


Figure 2-13. The evaluated web page addition2.php when the form is submitted

This is achieved with the following PHP block:

```
<?php
if (isset($_GET["t1"])){
$t1 = $_GET["t1"];
}
if (isset($_GET["t2"])){
$t2 = $_GET["t2"];
}
if (isset($_GET["t3"])){
$t3 = $_GET["t3"];
}
if (isset($_GET["t4"])){
$t4 = $_GET["t4"];
}
if (isset($_GET["t5"])){
$t5 = $_GET["t5"];
}
if (isset($t4) && isset($t5)){
$t6 = $t4 + $t5;
}
?>
```

In the previous PHP block, the values of the JavaScript textboxes, t1, t2, and t3, are submitted along with two of the values of the PHP section, t4 and t5. The value for the result of the PHP section, appearing in t6, is calculated next from the values of t4 and t5. Function `isset()` is used to check whether a variable is set.

To return the values of t1 up to t6 in the corresponding textboxes, an echo command is included for each text input element in a separate PHP block. For instance, for t1, its last value, maintained in the PHP code in variable \$t1, is printed in the first textbox with the following PHP block contained in the input element:

```
<input type = "text" id="t1" name = "t1" size=5 value = "<?php echo $t1 ?>" >
```

The previous input tag processed by the PHP engine, for instance when t1 currently has the value 12, is as follows:

```
<input type = "text" id="t1" name = "t1" size=5 value = "12" >
```

This value, carried to the server side and returned to the client side, is maintained to retain the state of the two calculators.

To enable the evaluated page to use the JavaScript calculator, the script of `program2.html` is also included in `addition2.php`.

```
<script>
function function1() {
var x = Number(document.getElementById("t1").value);
if (isNaN(x)) {
alert("Please enter a number");
return false;
}
var y = Number(document.getElementById("t2").value);
if (isNaN(y)) {
alert("Please enter a number");
return false;
}
var z = x + y;
document.getElementById("t3").value = z;
}
</script>
```

Figure 2-14 shows the results. You can access the JavaScript fields in `addition2.php` to make further local additions.



Figure 2-14. The new version `addition2.php` enables both remote PHP additions and local JavaScript additions

Next you'll create the final version of this site.

The Third Version of the JavaScript/PHP Addition Web Page

Since addition2.php evaluates to the program2.php web page that submits the data, program2.php can be completely omitted from the site. Therefore, the only PHP file required, addition.php, can replace program2.php as the home directory of the site. The following URL will be used:

localhost/addition2.php

In the following section, you will create a site that validates promotional codes dispatched from an HTML form and uses PHP to validate the data submitted by this form.

Form Validation with PHP

So far you have used HTML and JavaScript to validate a web page. Another option you have is to use PHP. With PHP you validate the web page remotely. To test this scheme, create a new PHP site that simulates a scenario where promotional codes are redeemed so that the user can win free products. At the Linux terminal, create the validate.php file in the document root of the web server.

```
$ cd /var/www/html  
$ sudo gedit validate.php
```

Enter the following source code and save the file:

```
<html>  
<head>  
<title>PHP Form Validation</title>  
<style>  
h1{  
color:orange;  
}  
.error{  
color:red;  
font-size:20px;  
}
```

```
label{
color:blue;
font-size:24px;
}
input{
color:blue;
font-size:24px;
background-color:orange;
}
</style>
</head>

<body>
<?php
$errmsg1="";
$errmsg2="";
$errmsg3="";
$valid1=false;
$valid2=false;
$valid3=false;

if (isset($_POST['submit']))
{
    $name=$_POST["name"];
    $email=$_POST["email"];
    $code=$_POST["code"];

    if(empty($name) || is_numeric($name))
    {
        $errmsg1.='<p class="error">* Please enter a valid name.</p>';
        $valid1=false;
    }
    else
    {
        if(is_string($name))
        $valid1=true;
    }
}
```

```
else
{
$errmsg1.='

* Please use valid characters.</p>';
$valid1=false;
}
}

if(empty($email) || is_numeric($email))
{
$errmsg2.='

* Please enter your e-mail.</p>';
$valid2=false;
}
else
{
if(is_string($email))
$valid2=true;
else
{
$errmsg2.='

* Please use valid characters.</p>';
$valid2=false;
}
}

if(empty($code))
{
$errmsg3.='

* Please enter your code number.</p>';
$valid3=false;
}
else
{
$len=strlen($code);

if($len==10)
$valid3=true;
else
{
$errmsg3.='

* Code should be in alphabetic letters and
numerical digits format with 10 characters in it.</p>';
}


```

CHAPTER 2 SERVER-SIDE PROGRAMMING WITH PHP

```
$valid3=false;
}
}

if($valid1==true && $valid2==true && $valid3==true)
header("Location:process.php? code=$code&name=$name&email=$email");
}

?>

<h1>Submit your name, your e-mail, and your code</h1>
<form name="form1" method="post" action=<?php echo htmlspecialchars(
$_SERVER["PHP_SELF"]);?>">

<label for="name">Full Name:</label>
<input type="text" name="name"><br>
<?php
if((errmsg1!="") && isset($_POST['submit']))
echo $errmsg1;
?>
<label for="email">E-mail:</label>
<input type="text" name="email"><br>
<?php
if((errmsg2!="") && isset($_POST['submit']))
echo $errmsg2;
?>
<label for="code">Code:</label>
<input type="text" name="code"><br>
<?php
if((errmsg3!="") && isset($_POST['submit']))
echo $errmsg3;
?>
<input type="submit" name="submit" value="Go">
</form>

</body>
</html>
```

As displayed in Figure 2-15, validate.php creates a web page that includes a form with three fields and a submit button, with the caption Go.

The screenshot shows a web browser window titled "PHP Form Validation - Chromium". The address bar displays "localhost(validate.php)". The main content area contains the following text and form fields:

Submit your name, your e-mail, and your code

Full Name: [redacted]
 E-mail: [redacted]
 Code: [redacted]

Go

Figure 2-15. The web page created by validate.php

For this form, all fields are required to be filled in, and the code must be an alphanumeric string of 10 characters. Figure 2-16 displays the error messages generated by the PHP code when the user clicks the Go button without completing all the fields.

The screenshot shows a web browser window titled "PHP Form Validation - Chromium". The address bar displays "localhost(validate.php)". The main content area contains the following text and form fields, with error messages displayed below them:

Submit your name, your e-mail, and your code

Full Name: [redacted]
 * Please enter a valid name.
 E-mail: [redacted]
 * Please enter your e-mail.
 Code: [redacted]
 * Please enter your code number.

Go

Figure 2-16. The error messages displayed in the web page when fields are incomplete

Test the form by completing just two of the fields, e.g., Full Name and Code, with the latter including only three characters. Figure 2-17 displays an example. Click the Go button.

PHP Form Validation - Chromium
localhost/validate.php

Submit your name, your e-mail, and your code

Full Name: Christos Karayiannis

E-mail:

Code: XV1

Go

Figure 2-17. Testing the form with only two fields completed

The PHP source code validates the form according to the rule set and displays two warnings, one for the empty field and one for the length of the string entered in the Code textbox. Figure 2-18 displays the warnings.

PHP Form Validation - Chromium
localhost/validate.php

Submit your name, your e-mail, and your code

Full Name: [highlighted]

E-mail: [highlighted]
* Please enter your e-mail.

Code: [highlighted]
* Code should be in alphabetic letters and numerical digits format with 10 characters in it.

Go

Figure 2-18. The warnings displayed for the entries of Figure 2-16

In the following section, I'll discuss the PHP source code for the form validation.

The validate.php Source Code Commentary

The Go button of the form, which is of type submit, relays the data to the PHP file indicated by the value of the form's action attribute. This is set to \$_SERVER["PHP_SELF"], which is another variable of the global PHP \$_SERVER[] array. This value is filled

by the PHP engine on the fly relative to the document root path of the current file, for this example `/validate.php`. Notice that the `action` attribute of the form is set as follows:

```
action=<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>
```

The value of the action is enclosed in the PHP tags so that the value of `htmlspecialchars($_SERVER["PHP_SELF"])`

is actually printed as the action's value. Function `htmlspecialchars()` is used to sanitize the file's pathname so that symbols like the double quote ("") is converted to ", which is the equivalent HTML entity. The purpose of this is to avoid a possible `$_SERVER["PHP_SELF"]` exploit, which is a cross-site scripting (XSS) exploit, where a hacker injects JavaScript code into web pages viewed by other users.

At the beginning of the PHP source code, three fields are evaluated. For instance, consider the following validation code for the value of the first field sent as `$_POST["name"]`:

```
if (isset($_POST['submit']))
{
    $name=$_POST["name"];
    $email=$_POST["email"];
    $code=$_POST["code"];

    if(empty($name) || is_numeric($name))
    {
        $errmsg1.= '<p class="error">* Please enter a valid name.</p>';
        $valid1=false;
    }
    else
    {
        if(is_string($name))
            $valid1=true;
        else
        {
            $errmsg1.= '<p class="error">* Please use valid characters.</p>';
            $valid1=false;
        }
    }
}
```

With function `isset()`, it is ensured that the evaluation takes place only when the submit button is clicked and the submit method is POST. Otherwise, the fields would get evaluated even before the user had the chance to fill them.

```
if (isset($_POST['submit']))
```

The variables `$valid1`, `$valid2`, and `$valid3` are used as flags to signal a valid (true) or an invalid (false) value for the first, second, and third fields, respectively. Also, the variables `$errmsg1`, `$errmsg2`, and `$errmsg3` are used to specify the appropriate error message for the first, second, and third fields, respectively. For instance, with the following commands, `errmsg3` appears when the value of the code does not have a length of ten characters:

```
if($len==10)
$valid3=true;
else
{
$errmsg3='<p class="error">* Code should be in alphabetic letters and
numerical digits format with 10 characters in it.</p>';
```

The concatenating assignment operator (`.=`) is a PHP string operator that appends the argument on the right side to the argument on the left side.

The form submits to the same PHP file, where it is included, while the user does not provide the expected values, thus allowing for validating the form. When all three fields are filled with valid values, the following if condition holds true and the `header()` function runs:

```
if($valid1==true && $valid2==true && $valid3==true)
header("Location:process.php? code=$code&name=$name&email=$email");
```

The `header()` function, which sends raw HTTP headers, is used here with the `Location` HTTP header. This function redirects the browser to the URL indicated by the `Location` value. In this example, the value includes the attached query string with the variable-value pairs required to be forwarded to the destination file, `process.php`. Thus, the function `header()` provides the escape mechanism to validate `.php` to break out of the loop of continuously submitting data to itself.

Create the file process.php in the document root with the following commands:

```
$ cd /var/www/html  
$ sudo gedit process.php
```

Enter the following source code and save the file:

```
<html>  
<head>  
<title>Code evaluation</title>  
<style>  
p{  
color:green;  
font-size:32px;  
}  
</style>  
</head>  
<body>  
<p>  
<?php  
  
$var=$_GET['code'];  
$var2=$_GET['name'];  
$var3=$_GET['email'];  
if(isset($var) && $var == 'SX1DF908RW')  
{  
echo nl2br("$var2 congratulations you have entered the lucky code: $var\nYou have won one T-shirt. We will contact you soon.");  
  
// Save the user's e-mail for contacting him/her  
$filename = "code.txt";  
$handle = fopen($filename, "w") or die(" Unable to open file!");  
if ($handle)  
{  
fwrite($handle, $var2);  
fwrite($handle, PHP_EOL);  
fwrite($handle, $var3);
```

```

fwrite($handle, PHP_EOL);
fclose($handle);
}
}
else
{
    echo "You didn't win, please try another time!";
}

?>
</p>
</body>
</html>

```

The three variable values submitted (code, name, and email) from header() in file validate.php are now retrieved as var, var2, and var3, respectively. The following if condition looks for the lucky code:

```
if(isset($var) && $var == 'SX1DF908RW')
```

If the condition is true, the following message is returned to the browser:

```
echo nl2br("$var2 congratulations you have entered the lucky code: $var\n"
You have won one T-shirt. We will contact you soon.");
```

The function nl2br() is used to translate escape characters to their meaning; for instance, \n is treated now as a newline.

Test the validate.php form by providing the correct code, SX1DF908RW, as shown in Figure 2-19.

The screenshot shows a web browser window titled "PHP Form Validation - Chromium". The address bar displays "localhost(validate.php)". The page content is a form with the following text:
Submit your name, your e-mail, and your code
Full Name: Christos Karayiannis
E-mail: christos@somesite.com
Code: SX1DF908RW

Figure 2-19. The validate.php form filled with the correct code

Click the Go button. The form is submitted, and the browser redirects to process.php, which displays the message shown in Figure 2-20 to the client browser.

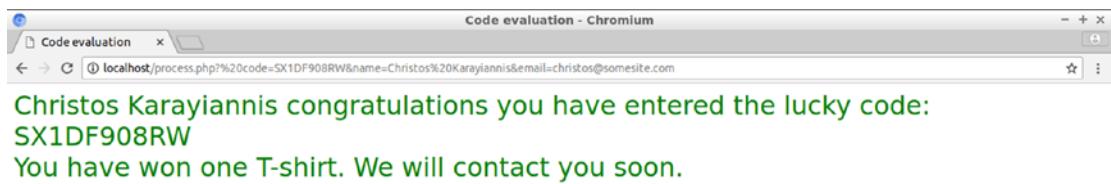


Figure 2-20. The message displayed from process.php for the correct code

Try validating validate.php with a wrong code, as displayed in Figure 2-21.

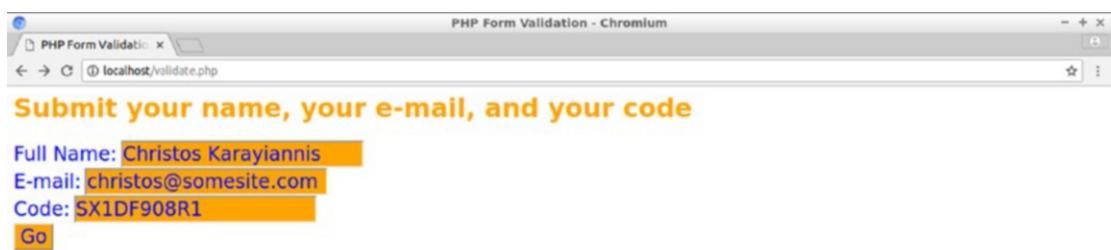


Figure 2-21. Testing validate.php with a wrong code

Click the Go button to redirect to process.php. The message displayed to the client's browser is shown in Figure 2-22.

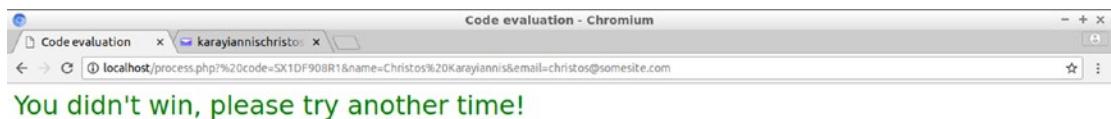


Figure 2-22. The reply viewed in the user's browser for a wrong code

Let's not forget the promise to get back to the user. To save the visitor's details, the PHP source code is used to provide the interface between the web server and the local filesystem. A new file called `code.txt` is required to store the winner's personal details: the full name and the e-mail. With the following PHP code file, `code.txt` is used to store this information. The file handle `$handle` is created with `fopen()`, which in this example is used for writing (`w` mode) to the file, with the name indicated in the value of `$filename`. The function `fwrite()` is used twice to fill the winner's name and e-mail and also the `PHP_EOL` value between. This inserts a line break.

```
// Save the user's e-mail for contacting him/her
$filename = "code.txt";
$handle = fopen($filename, "w") or die(" Unable to open file!");
if ($handle)
{
    fwrite($handle, $var2);
    fwrite($handle, PHP_EOL);
    fwrite($handle, $var3);
    fwrite($handle, PHP_EOL);
    fclose($handle);
}
```

To enable the web server (and the PHP engine) to access file `code.txt`, create it first with the `touch` command and then change its ownership to belong to the user `www-data`, a member of the group `www-data`, which is the user assumed by the web server. At the command line, enter the following:

```
$ sudo touch /var/www/html/code.txt
$ sudo chown www-data:www-data /var/www/html/code.txt
```

Figure 2-23 displays the content of `code.txt`, including the personal details for the user who provided the correct code.



Figure 2-23. The personal details of the winner stored in file code.txt

The user's personal details are thus saved for contacting the user to claim their prize.

Summary

In this chapter, you saw some examples of the Apache web server utilizing the PHP engine to create programs that run on the server side, and you implemented similar-functioning programs with JavaScript that run on the client side. You also created a site where the form data was validated with PHP, and also PHP code was used to redirect to another web page and to interface the web server with the filesystem. In the following chapter, you will continue to use PHP in combination with the GeoIP geolocation module that you set up for Apache so that information about a visitor's location can be retrieved and used from the PHP source code on the site's web pages. But first you will set up your router to officially assign the Apache host as the web server of your LAN and thus enable your site to be accessible from the whole Internet.

CHAPTER 3

Connecting Your Apache Server to the Internet

In this chapter, you'll configure your router to forward any incoming packet with a TCP port number from the web service to the computer hosting your web server. TCP port 80 is the default port number for the HTTP protocol; however, other port numbers can also be used. Another port number used often for the HTTP protocol is port 8080. This port number, sometimes called the *HTTP alternate port*, is used when the ISP does not allow incoming traffic for port 80.

With the router configured to forward requests for the port you prefer to the web server, your site becomes available to the whole Internet. In this chapter, you'll use a DNS domain name for the server instead of an IP address and thus make the URL for your site more user-friendly.

By allowing the Apache web server to connect to the Internet, your next project is to utilize the GeoIP Apache module to allow the server to locate the geographic origin of the site visitors, pinpoint them on a map, and even respond in their native language.

The NAT Protocol

The router device that connects two networks requires two IP addresses. Like Janus, the two-faced ancient Roman god of gates, it looks in two directions: externally and internally to the network. These addresses are referred to as *public* and *private* addresses, respectively. All workstations in the LAN allocate private IP addresses. The router represents all the private IP addresses of the LAN with its unique public IP address by implementing the Network Address Translation (NAT) protocol. NAT was

CHAPTER 3 CONNECTING YOUR APACHE SERVER TO THE INTERNET

developed to address the problem of the IPv4 address shortage. With a single public IP address for the router, multiple computers and Internet-enabled devices can connect to the Internet in a seamless manner.

NAT comes in two variations: static and dynamic. With static NAT, each host requires one public IP address for its private IP address to be reachable over the Internet.

With dynamic NAT, multiple hosts can translate their private IP address to the public IP address of the router. An extension of dynamic NAT is port forwarding, aka port mapping, which consists of a router service. With port forwarding, the router is configured to dispatch packets with a specific destination port number to a specific machine on the LAN. For instance, the router in Figure 3-1 is set to forward all packets with destination port 80 to the host with the private IP address 192.168.1.100.

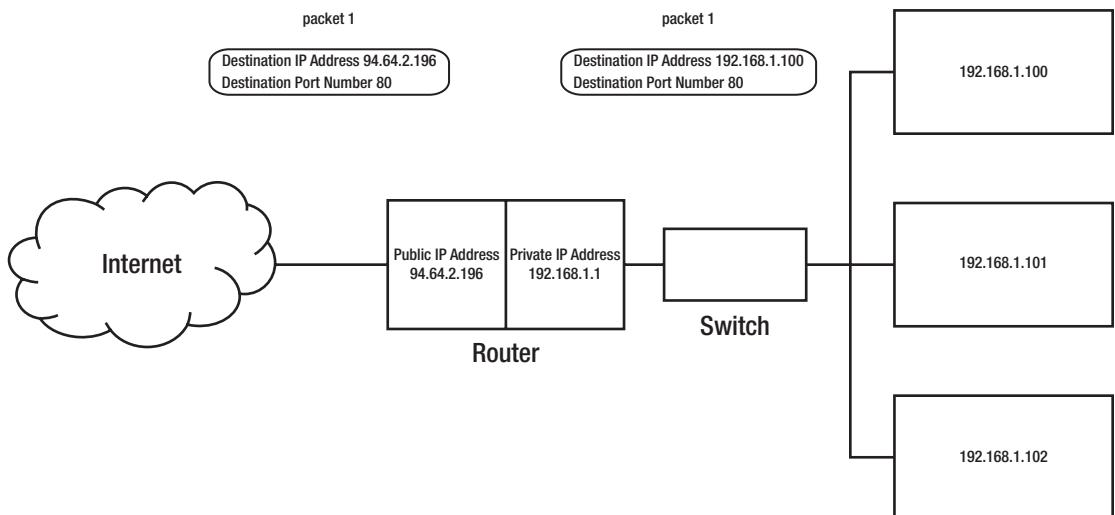


Figure 3-1. A packet with destination addresses the public IP address of the router and port number 80 with the dynamic NAT service is forwarded to the private IP address of the web server

Enabling Port Forwarding to Your Router

To relay HTTP requests aimed at the public IP address of your router with the HTTP port number, you have to apply port forwarding on your web server. When an HTTP request with port 80 (or any other port number that you choose) arrives at the router, it will be forwarded to the private IP address of your web server. Most routers today include port forwarding not only because it's a feature of dedicated web servers but also of many home appliances that act as servers, for instance IP cameras.

Most router configuration options are accessed through the HTTP or Telnet protocol. To use the first approach, enter the private IP address of your router in the address bar of your browser and provide the username and the password to go to the web-based router setup page. Locate the Port Forwarding option in the main menu of your router. For the examples in this book, the private IP address of the router is 192.168.1.1. When using this address in the browser's address bar, the web page for the router appears; Figure 3-2 shows the page for the Speedport Entry 2i VDSL router.

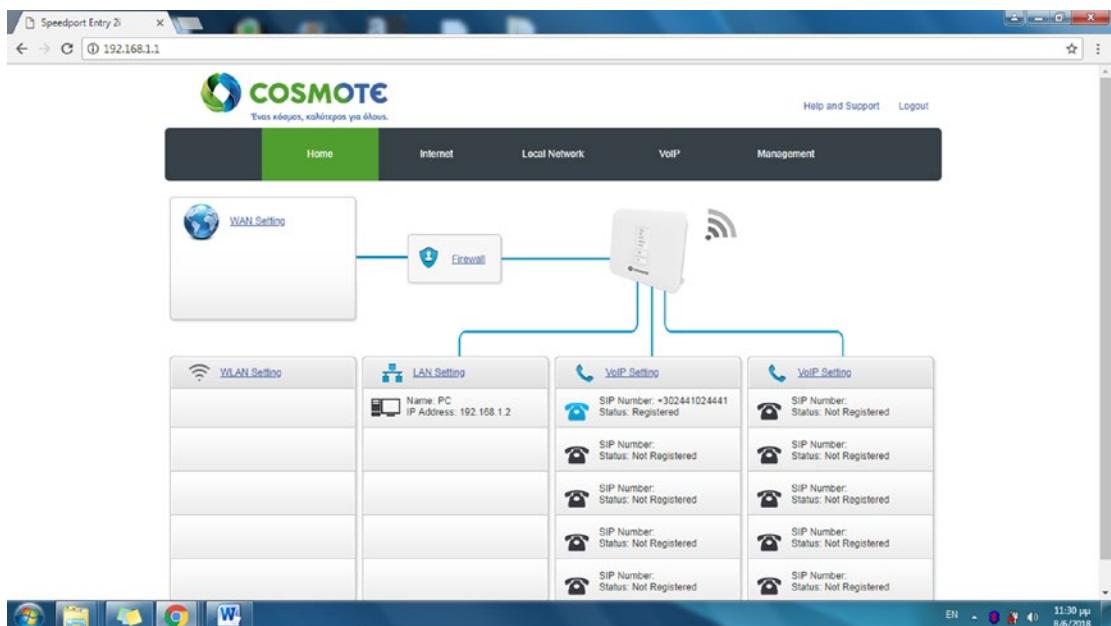


Figure 3-2. The home web page for the web-based configuration interface of the test router

Click the Firewall link to go to the web page where the Port Forwarding option is available below the main navigation bar.

Click the Port Forwarding link to go to the web page displayed in Figure 3-3.

CHAPTER 3 CONNECTING YOUR APACHE SERVER TO THE INTERNET

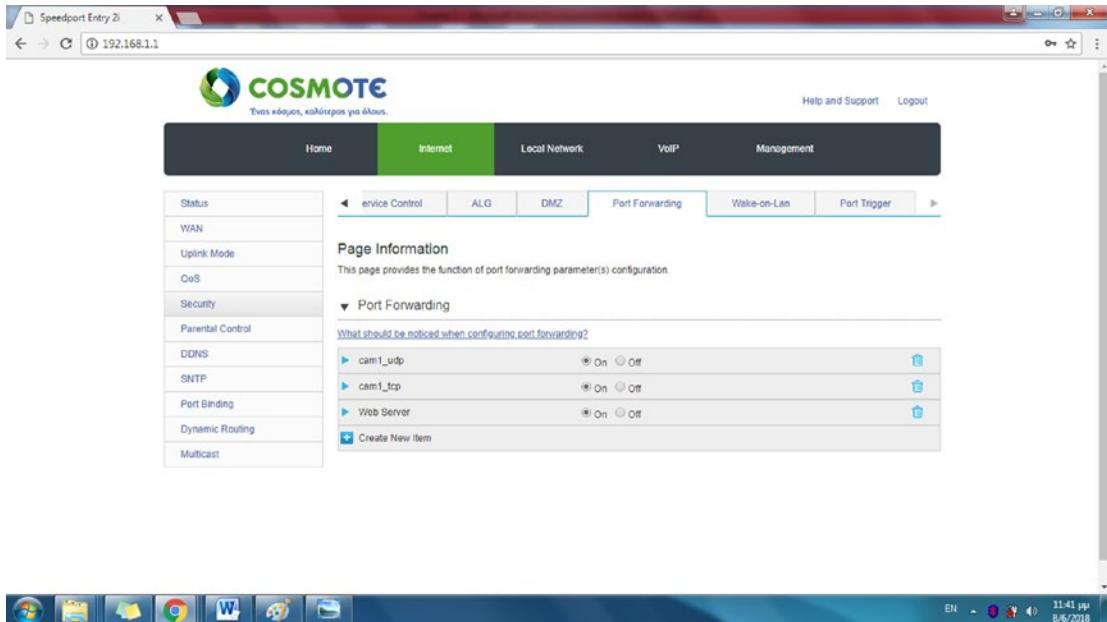


Figure 3-3. The Port Forwarding configuration web page

Click the Create New Item button to create a new entry for the Port Forwarding list. The form elements for the Port Forwarding settings appear, as shown in Figure 3-4.

This screenshot shows the same web interface as Figure 3-3, but with a new entry being created in the Port Forwarding list. The "Create New Item" button was clicked, which expanded the "Web Server" section. The expanded section includes fields for Name (set to "Web Server"), Protocol (set to "TCP"), WAN Connection (set to "PTM_DSL"), WAN Host IP Range (IPs 0.0.0.0 to 0.0.0.0), MAC Mapping (radio button set to "Off"), LAN Host IP Address (IP 192.168.1.100), WAN Port Range (Ports 8080 to 8080), and LAN Host Port Range (Ports 8080 to 8080). The status bar at the bottom shows system information: EN, 11:43 PM, 8/6/2018.

Figure 3-4. The form with the port forwarding details

Enter a name for the service you implemented with port forwarding in the Name field, e.g., **Web Server**. Enter the private IP address for this computer, e.g., **192.168.1.100** (in the field LAN Host IP Address), and enter the port numbers you will use for the web service, e.g., **8080** (in the fields WAN Port Range and LAN Host Port Range). For the Protocol option, select TCP from the drop-down list to implement the HTTP protocol. Click the Apply button to confirm the settings. The new entry appears in the Port Forwarding list.

This is a good chance to add all the port numbers used for the HTTP and HTTPS protocols for the examples in this book. You can assign the computer with the private IP address of 192.168.1.100 the responsibility of receiving the TCP/IP packets on your LAN for the port numbers 80, 8080, 8181, and 443.

Each router model uses a different GUI environment to provide the Port Forwarding settings. Figure 3-5 shows the configuration page of the FBR-1161 LevelOne ADSL router, which displays the settings under the NAT option of the Advanced Setup menu.

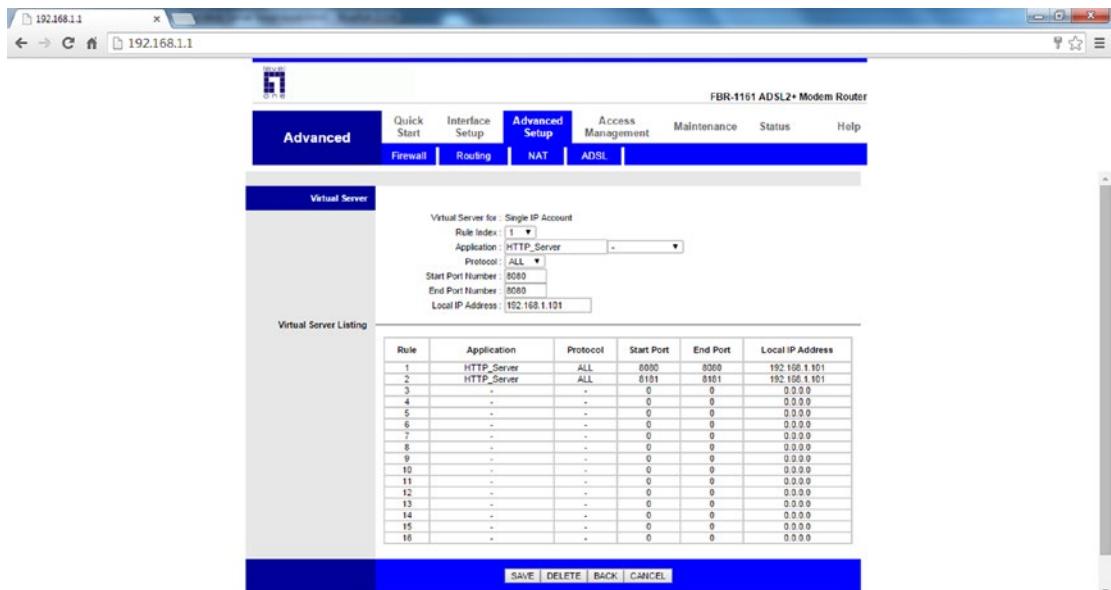


Figure 3-5. The port forwarding web page for the LevelOne FBR-1161 ADSL router

The previous configuration assigns ports 8080 and 8181 to be forwarded to the LAN computer with IP 192.168.1.101 for *all* protocols (both TCP and UDP).

As a general rule, you have to provide three main settings: the port number on which the port forwarding applies for the router's public IP address, the private IP address of the LAN computer assigned with the task to dispatch client requests for this specific port, and TCP as the transport protocol used for the HTTP and HTTPS services. As each router implements a different user interface for the web-based settings, it is useful to locate your specific router model options for port forwarding at <https://portforward.com/router.htm>.

Implementing Port Forwarding with Apache Vhosts

To implement the port forwarding feature, you can use the `example1.conf` configuration file from Chapter 1, which is a simple copy of the file `000-default.conf` and which utilizes the default HTTP port 80. If your ISP blocks requests to this port, you can use another Apache virtual host that serves another port, for instance the HTTP alternate port, 8080. In that case, use the following commands at the Linux terminal to create a new configuration file by copying and modifying the default Apache configuration file `000-default.conf`:

```
$ cd /etc/apache2/sites-available  
$ sudo cp 000-default.conf example4.conf
```

Use a text editor to edit the file `example4.conf`.

```
$ sudo gedit example4.conf
```

Change just the first two lines as indicated next:

```
Listen 8080  
<VirtualHost *:8080>  
    # The ServerName directive sets the request scheme, hostname and port that  
    # the server uses to identify itself. This is used when creating  
    # redirection URLs. In the context of virtual hosts, the ServerName  
    # specifies what hostname must appear in the request's Host: header to  
    # match this virtual host. For the default virtual host (this file) this
```

```

# value is not decisive as it is used as a last resort host
# regardless.
# However, you must set it for any further virtual host explicitly.
#ServerName www.example.com

ServerAdmin webmaster@localhost
DocumentRoot /var/www/html

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host. For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf

</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

```

The first directive, Listen 8080, instructs Apache to listen on TCP port 8080. The starting <VirtualHost *:8080> container is used with the end </VirtualHost> container to enclose the group of directives that apply to the new vhost. The wildcard asterisk (*) with the port number 8080 appended with a colon indicates that the enclosed directives apply to incoming requests destined to any one of the available IP addresses (*) of the web server over port 8080.

Click the Save button of the gedit window to apply the configuration to the vhost. Next you need to enable the new configuration with the a2ensite command.

```
$ sudo a2ensite example4
```

To disable any previous configuration files that utilize the same port, for instance `example2.conf`, to avoid conflicts, use the `a2dissite` command.

```
$ sudo a2dissite example2
```

To enable the new configuration file, you need to reload the web server.

```
$ sudo service apache2 force-reload
```

If you intend to use port 8080 with the Apache server, you should configure ufw, the Linux firewall, to permit it. For port number 80, no actions are required because you already allowed this port in Chapter 1. Enter the following at the Linux terminal:

```
$ sudo ufw allow 8080
```

Testing the New Virtual Host

To implement the router's port forwarding service, you have to access your site from an external LAN using the public IP address of your router. The common case is your ISP assigns your router a dynamic public IP address that changes, e.g., each time the router restarts. Alternatively, an ISP can offer static IP addresses, usually after the client's request. In each case, to find the current public IP address of your router, use an online service like <https://bearsmyip.com/>. The current public IP address of your router is displayed in your browser, as shown in Figure 3-6.

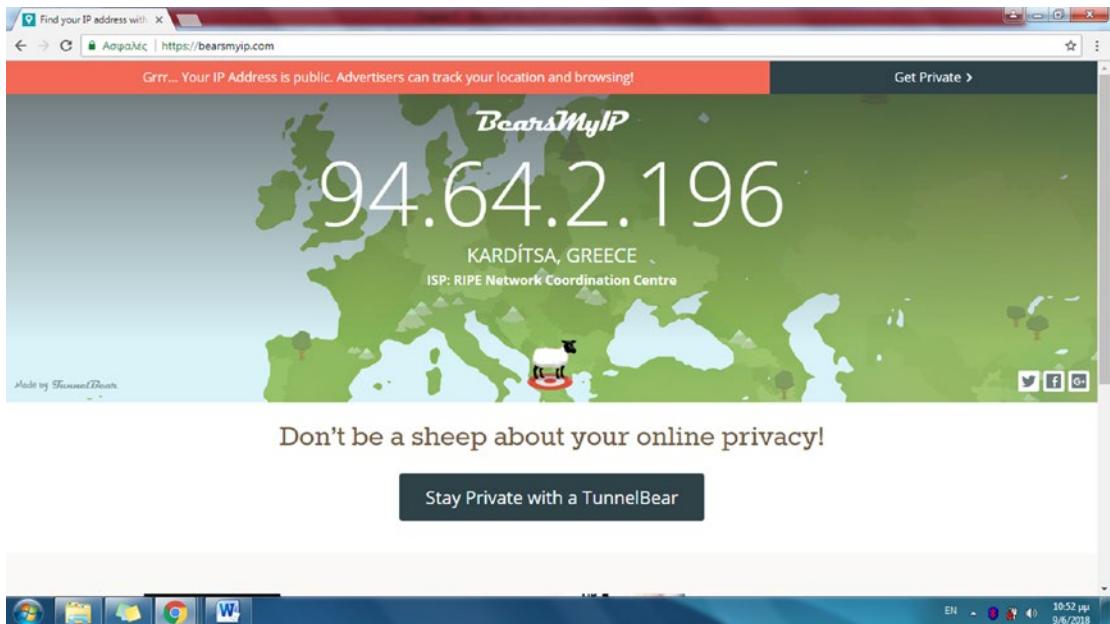


Figure 3-6. Discovering the public IP address of the router with an online service

Enter the URL consisting of this IP address in your browser's address bar from a LAN computer.

`http://94.64.2.196`

Alternatively, if you utilize port 8080 for your web server, enter the following:

`http://94.64.2.196:8080`

Your directory index is displayed in your browser, as shown in Figure 3-7.

CHAPTER 3 CONNECTING YOUR APACHE SERVER TO THE INTERNET



Figure 3-7. Testing the site using the public IP address of your router, from a computer of your LAN

To make the test definitive, you next have to request the web page from an external computer to your LAN. You can either use your mobile phone's Internet connection and view the web page with the previous URL or have a remote computer make the connection for you. There are many online services for testing web pages from various positions around the world. Actually, in Chapter 5, you will follow the steps to create a similar Internet service all by yourself. Such an online service is www.webpagetest.org. Using the images sent to your browser by this site, you can view the web page the way it is downloaded to the site's remote browser. You can even choose the country, city, and browser, and also view statistics such as the download time. Figure 3-8 shows the [webpagetest.org](http://www.webpagetest.org) home page.

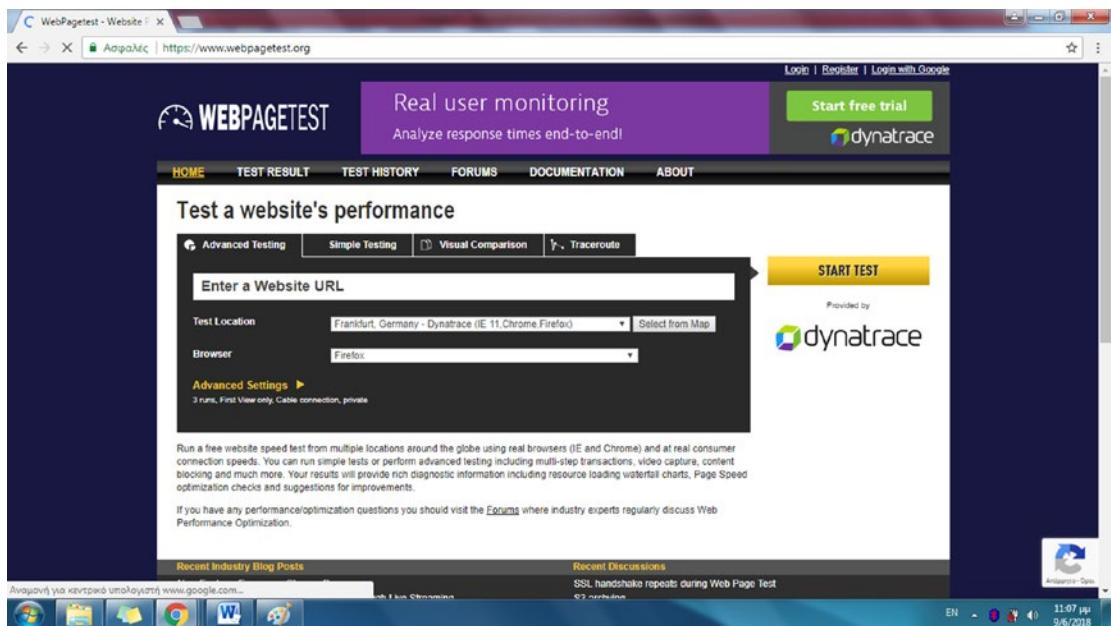


Figure 3-8. The home page of [webpagetest.org](https://www.webpagetest.org)

Enter the URL you want to test in the main textbox.

94.64.2.196

Or if you use a different port such as port 8080 instead of the default HTTP port, append the port number to the IP address with a colon. Here's an example:

94.64.2.196:8080

Use the Test Location drop-down list to select a location (country and city) where the request to your server will originate from, and select the browser used from the Browser drop-down list. Click the Start Test button. After a few seconds, images of your web page are displayed along with info about the download time and the web page's size. Figure 3-9 displays the result web page.

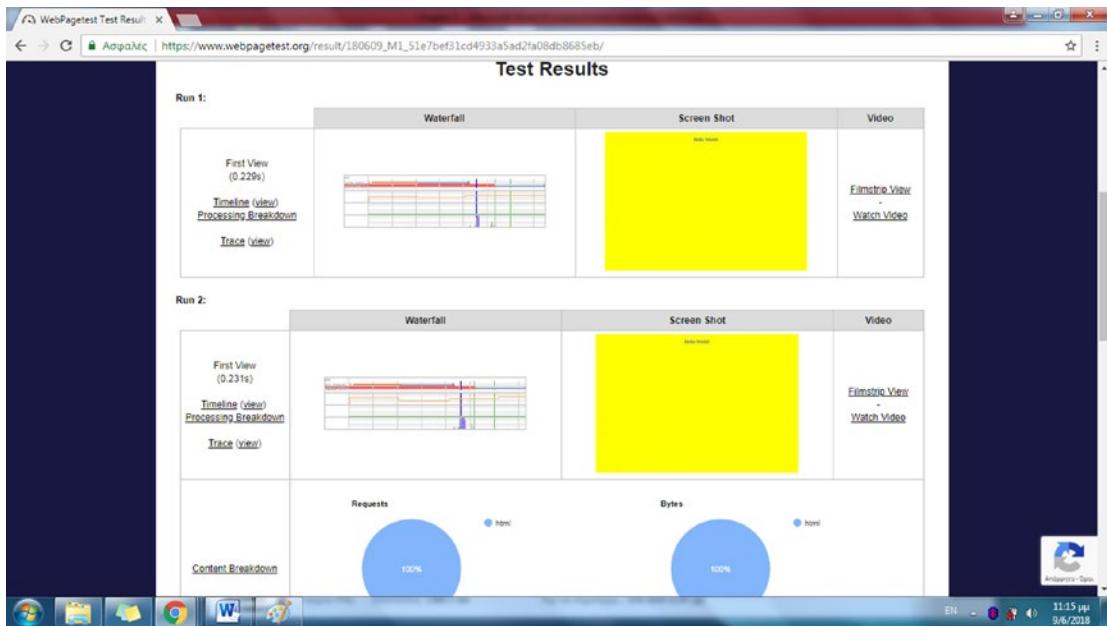


Figure 3-9. The result web page for testing your site remotely with [webpagetest.org](https://www.webpagetest.org)

On the web page in your browser, you can view screenshots of the directory index, each one taken from different tests for your site. You can click those images to enlarge them. This test verifies that your web site is able to be viewed externally.

With your site available to users throughout the world, you can make use of the GeoIP Apache module to give your web server information about the visitor's location based on their IP address. The PHP source code will use these geographical details and return geographical information in the web page sent back to the user.

Using the GeoIP Apache Module

With the online service proving that your web pages are available to the whole Internet, it is a good chance to use a geolocation service to retrieve information about the origin of the users visiting your site and at the same time try an Apache module. With a geolocation service, the IP addresses of your visitors are compared to a database and are matched to specific countries, cities, geographical coordinates (latitude or longitude), etc. Those values are assigned to the Apache server variables for each client's request and become available from the PHP engine.

With the following steps, you will implement a geolocation service with MaxMind's Apache module `geoip_module`.

Hint! To avoid prepending the `sudo` keyword in the following commands, enter the following at the Linux terminal:

```
$ sudo su
```

Install `geoip_module` with the following command:

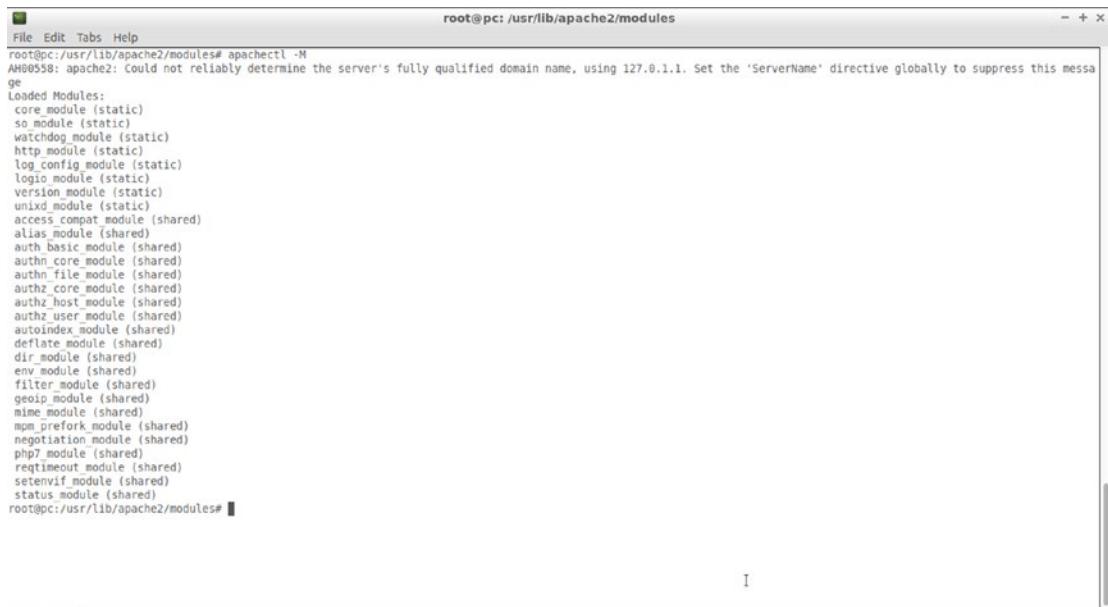
```
$ apt-get install libapache2-mod-geoip
```

The `geoip_module` shared object file (`mod_geoip.so`), which includes the compiled source code used by Apache, is installed in the directory `/usr/lib/apache2/modules`.

The `geoip_module` module is included among the other Apache modules loaded from Apache. Use the following command to view the list of loaded modules:

```
$ apache2ctl -M
```

Figure 3-10 displays the list of modules currently loaded.



```
root@pc:/usr/lib/apache2/modules# apache2ctl -M
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Loaded Modules:
core module (static)
so module (static)
watchdog module (static)
http module (static)
log config module (static)
logio module (static)
version module (static)
unixd module (static)
access_compat module (shared)
alias module (shared)
auth basic module (shared)
authn_core module (shared)
authn_file module (shared)
authz_core module (shared)
authz_host module (shared)
authz_user module (shared)
autoindex module (shared)
deflate module (shared)
dir module (shared)
env module (shared)
filter module (shared)
geoip module (shared)
mime module (shared)
mpm_prefork module (shared)
negotiation module (shared)
php7 module (shared)
reqtimeout module (shared)
setenvif module (shared)
status module (shared)
root@pc:/usr/lib/apache2/modules#
```

Figure 3-10. Command `apache2ctl -M` returns the list of loaded Apache modules

You can find the module's configuration files in the directory called `mods-available`. Using gedit from the Linux terminal, edit the configuration file `geoip.conf` of the `geoip`_module as follows:

```
$ gedit /etc/apache2/mods-available/geoip.conf  
Set the GeoIPEnable directive from Off to On. The configuration file  
includes then the following directives:  
<IfModule mod_geoip.c>  
    # For performance reasons, it's not recommended to turn GeoIP on  
    # serverwide,  
    # but rather only in <Location> or <Directory> blocks where it's actually  
    # needed.  
    GeoIPEnable On  
    GeoIPDBFile /usr/share/GeoIP/GeoIP.dat  
</IfModule>
```

Enable the new configuration file using the `a2enmod` (`apache2 enable module`) command:

```
$ a2enmod geoip
```

Hint! If you need to disable the GeoIP module, you can use the `a2dismod` (`apache2 disable module`) command as follows:

```
$ a2dismod geoip
```

`geoip_module` is included now as a symbolic link to the directory `/etc/apache2/mods-enabled`.

Reload Apache using this:

```
$ sudo service apache2 force-reload
```

`geoip_module` looks up the client's IP address from MaxMind's database. The following are the steps to download the GeoIP database.

Change the working directory to `GeoIP`.

```
$ cd /usr/share/GeoIP/
```

wget is a command-line utility for downloading web pages and other resources with web protocols. You will use wget next to download the GeoIP.dat.gz zipped database file:

```
$ wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz
```

Decompress with the gunzip utility the zipped file GeoIP.dat.gz to the GeoIP.dat database file.

```
$ gunzip GeoIP.dat.gz
```

Create a PHP file for testing the GeoIP database.

Use gedit to create a PHP file called geo1.php.

```
$ gedit geo1.php
```

Enter the following PHP source code in the file geo1.php, which includes the function apache_note() and can be used from the Apache server:

```
<!DOCTYPE html>
<html>
<body>
<?php
$country_name = apache_note("GEOIP_COUNTRY_NAME");
print "Your Country: " . $country_name;
?>
</body>
</html>
```

The previous source code makes use of apache_note(), an Apache-specific function that accesses the Apache notes table to retrieve the information exchanged between Apache modules. The file geo2.php, on the other hand, includes PHP source code that can be used independently from other web servers as well.

```
<!DOCTYPE html>
<html>
<body>
<?php
$country_name = $_SERVER["GEOIP_COUNTRY_NAME"];
```

```
print "Your Country: " . $country_name;  
?>  
</body>  
</html>
```

With the GeoIP module enabled, the web server looks up the IP address of the client in the database files and sets environment variables such as `GEOIP_COUNTRY_NAME` used next from the PHP global variable `$_SERVER`, for instance `$_SERVER["GEOIP_COUNTRY_NAME"]`.

Hint! You can test the environment variables with the following PHP code:

```
<?php  
echo 'Client IP Address: ' . getenv('GEOIP_ADDR') . "<br>";  
echo 'Client Country Code: ' . getenv('GEOIP_COUNTRY_CODE') . "<br>";  
echo 'Client Country Name: ' . getenv('GEOIP_COUNTRY_NAME') . "<br>";  
echo 'Client City: ' . getenv('GEOIP_CITY');  
?>
```

Insert the previous source code in a PHP file located in the document root of your web server and request this file next in your browser. Here is some sample output:

```
Client IP Address: 206.189.190.142  
Client Country Code: US  
Client Country Name: United States  
Client City: Chicago
```

Test the files `geo1.php` and `geo2.php` using a web page tester like webpagetest.org. On the home page of this site, enter a URL that includes the current IP public address of your router and the port set in the port forwarding service of your router, relative to the document root path of the PHP file you are testing. Here's an example:

94.64.2.196/geo1.php

If you are using another port other than the default, append this port number with a colon to the IP address, as shown here:

94.64.2.196:8080/geo1.php

Select a country in the Test Location drop-down list, as shown in Figure 3-11, where the test will run from. For instance, in the following test, Ireland was selected:

Ireland – EC2

Select also the browser's name in the Browser drop-down list.

Chrome

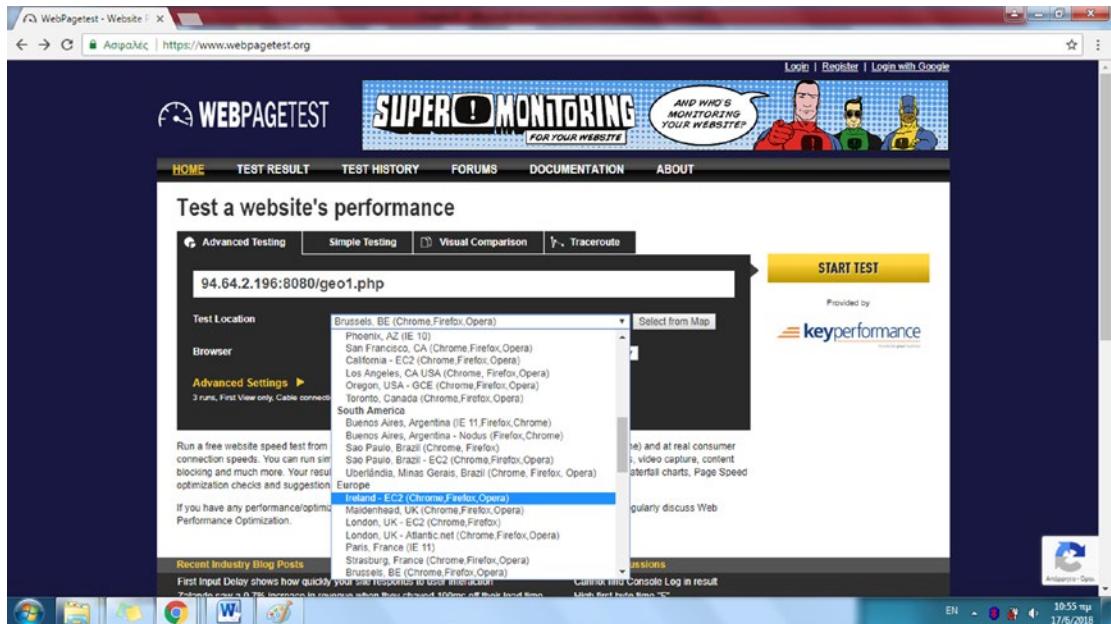


Figure 3-11. Selecting at the [webpagetest.org](https://www.webpagetest.org) site a specific computer in a country where the test will run

Alternatively, you can select the country from a map using the Select from Map button. The available geographical locations are pinned on the map, as displayed in Figure 3-12. Click a specific pin to use the corresponding remote computer for your test.

CHAPTER 3 CONNECTING YOUR APACHE SERVER TO THE INTERNET

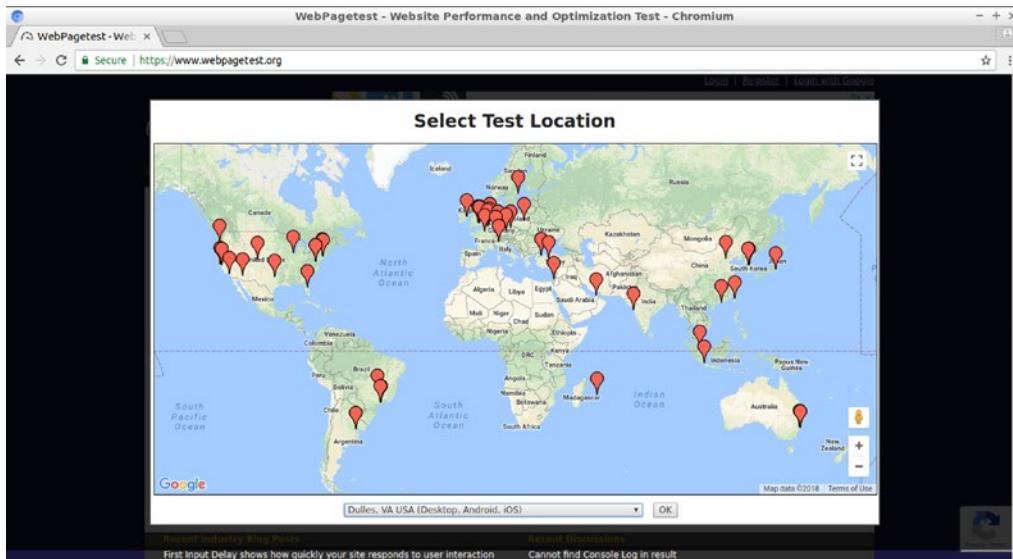


Figure 3-12. Selecting a test computer from the [webpagetest.org](https://www.webpagetest.org) map

Click the Start Test button.

Figure 3-13 displays the web page, returned when the test runs, enlarged by clicking it. This is the web page that will appear in the web browser of the user in Ireland.

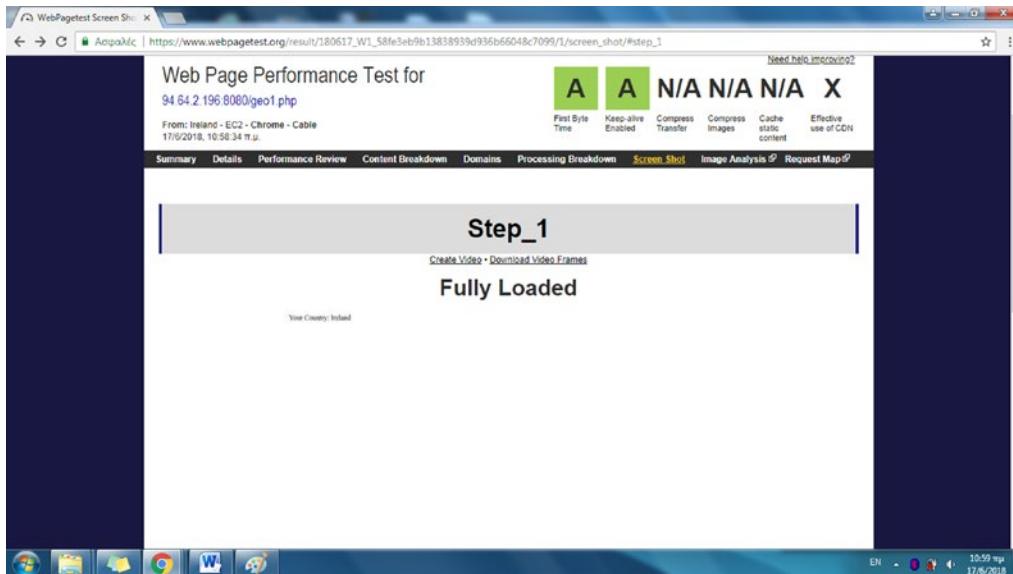


Figure 3-13. The web page displaying "Your Country: Ireland" as it appears on the computer located in Ireland

The web page displays the following:

Your Country: Ireland

Hint! Notice that the geolocation service is not always accurate. Also, the GeoIP database file needs to be constantly updated, something that can be automated with a software tool like cron.

In the site in the following section, the geographical information for the user is taken from the PHP source code and processed to display web pages with text corresponding to the time in the official language of the country associated with the user's IP address.

Responding to the Visitor's Native Language

With this next project, you'll create a web site that looks up the IP address of the visitor to the GeoIP database, retrieves the visitor's two-letter country code (ISO 3166-1 alpha-2), and responds by saying "hello" in the official language of that country. This is a simplified example; however, similar techniques are used on the sites of international corporations to provide their content, e.g., device manuals, in different areas of the world.

In the document root directory /var/www/html, create a file called geo3.php with the following source code:

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
font-size:80px;
color:red;
}
</style>
</head>
<body>
<?php
$cc = $_SERVER["GEOIP_COUNTRY_CODE"];
```

CHAPTER 3 CONNECTING YOUR APACHE SERVER TO THE INTERNET

```
switch ($cc) {  
    case "US":  
    case "UK":  
        print "<p>hello</p>";  
        break;  
    case "FR":  
        print "<p>bonjour</p>";  
        break;  
    default:  
        print "<p> &#x1F600; </p>";  
}  
?  
</body>  
</html>
```

The PHP code retrieves the `GEOIP_COUNTRY_CODE` environmental variable, whose value is a two-letter country code. A `switch` statement then checks this value against some predefined country codes, for instance, US, UK, and FR. You can add more `switch` statements for more country codes. For each country, a “hello” message in the corresponding language is printed on the screen. The `default` statement, which applies for all cases not covered in the previous `switch` statements, prints a Unicode smiley instead of a greeting. The specific smiley used in the example corresponds to Unicode hex code 1F600. To print the characters in a large red font, CSS style properties were used.

Visit the [webpagetest.org](http://www.webpagetest.org) home page to test your server and select a U.S. city from the Test Location list. In the example shown in Figure 3-14, a computer located in California was used.

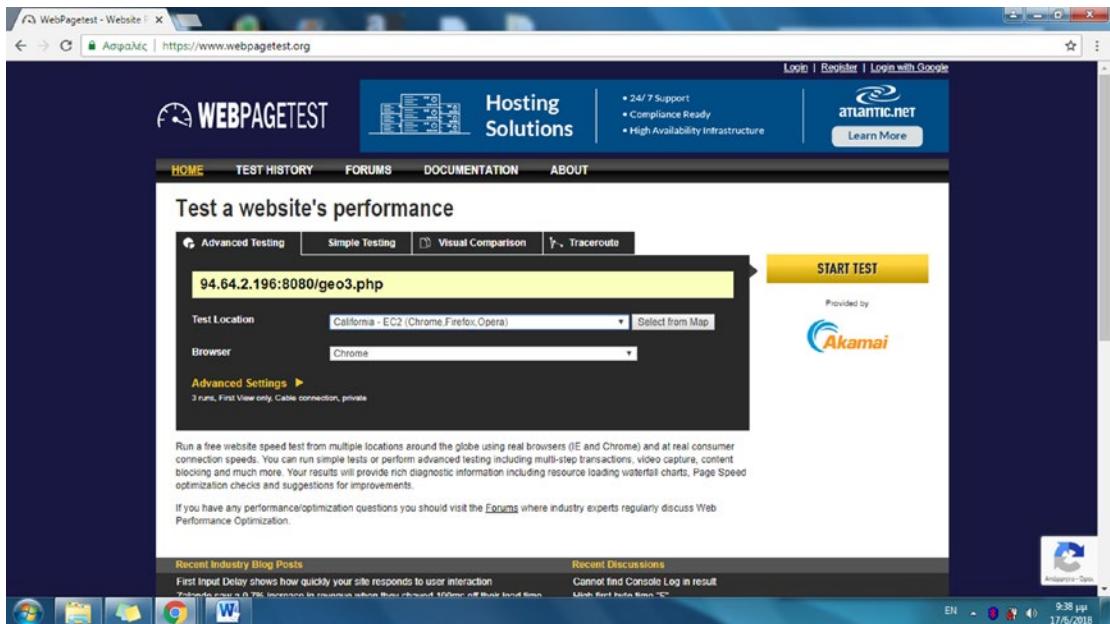


Figure 3-14. Running the test from a computer located in California

Enter the following URL at the main textbox:

94.64.2.196/geo3.php

Or if your server configuration implements port number 8080 use:

94.64.2.196:8080/geo3.php

The IP address 94.64.2.196 used in this example should be substituted with the public IP address of your router. Click the Start Test button. The web page in Figure 3-15 displays the content of the web page viewed from the remote computer in California.

CHAPTER 3 CONNECTING YOUR APACHE SERVER TO THE INTERNET

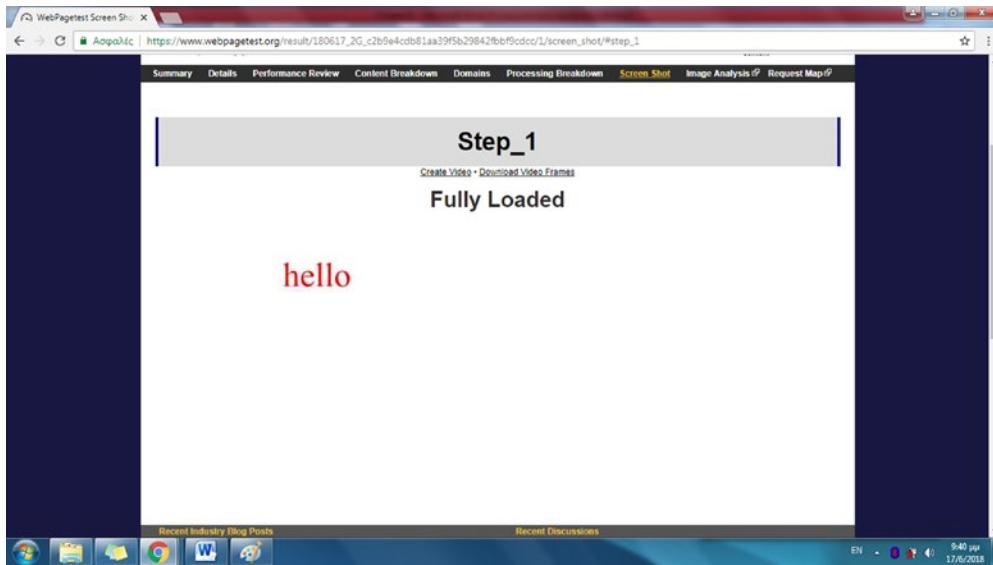


Figure 3-15. The message “hello” displayed in the browser of a test computer located in California

Perform the test again, this time from a French location. On the home page of [webpagetest.org](https://www.webpagetest.org), displayed in Figure 3-16, a computer from Strasburg is selected to download your specified web page.

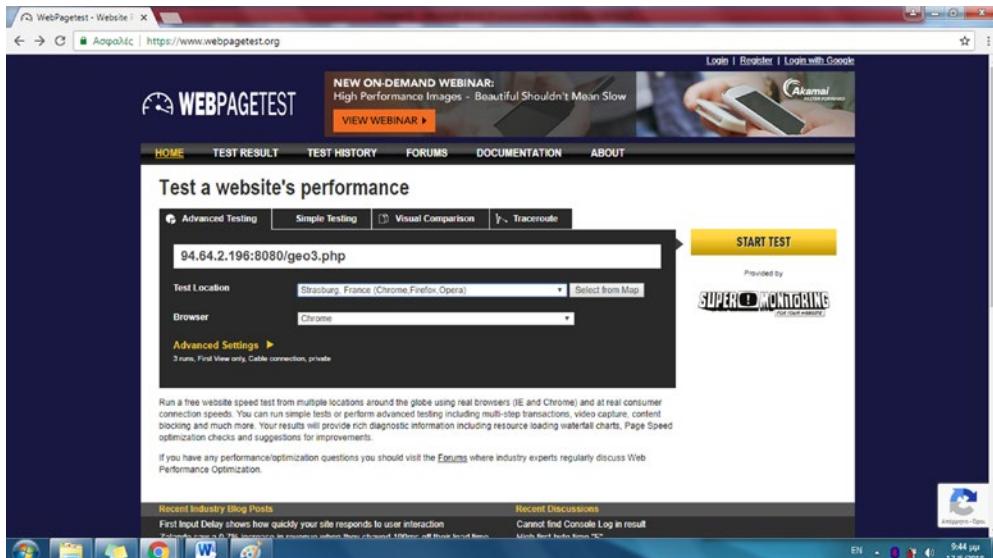


Figure 3-16. Running the test from a computer located in Strasburg

Click the Start Test button to run the test. As displayed in Figure 3-17, the web page of your site greets the visitor with “bonjour.”

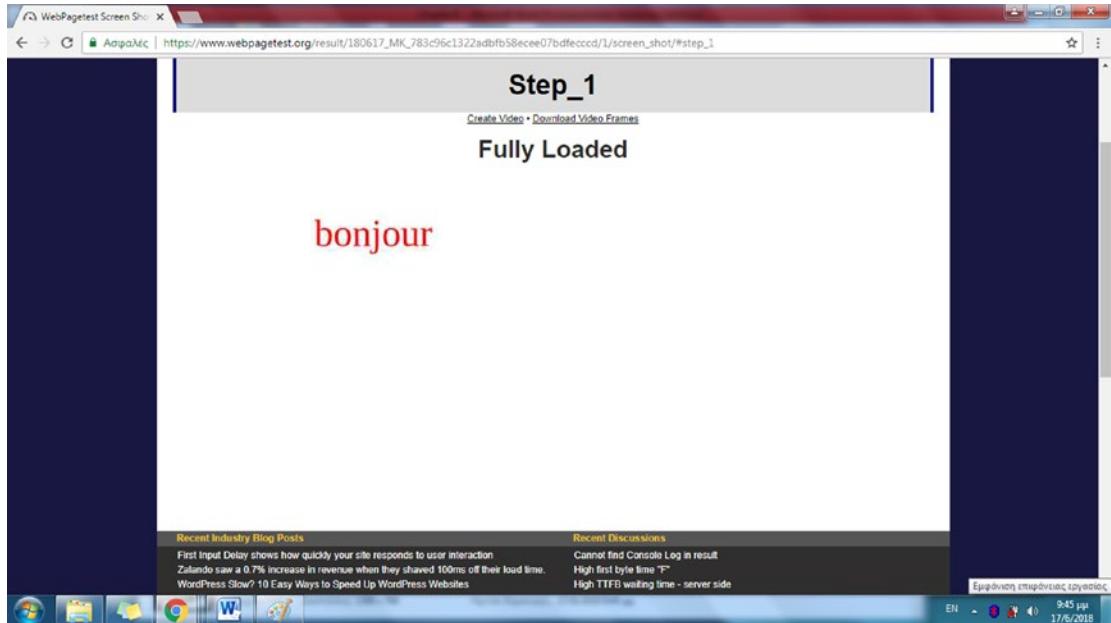


Figure 3-17. The message “bonjour” displayed in the browser of a computer located in France

Do a final test using a South Korean site. In the example in Figure 3-18, a computer located in Seoul was selected. For simplicity, the PHP code of geo3.php supports only three countries, leaving all other country visitors to be welcomed with a smiley.

CHAPTER 3 CONNECTING YOUR APACHE SERVER TO THE INTERNET

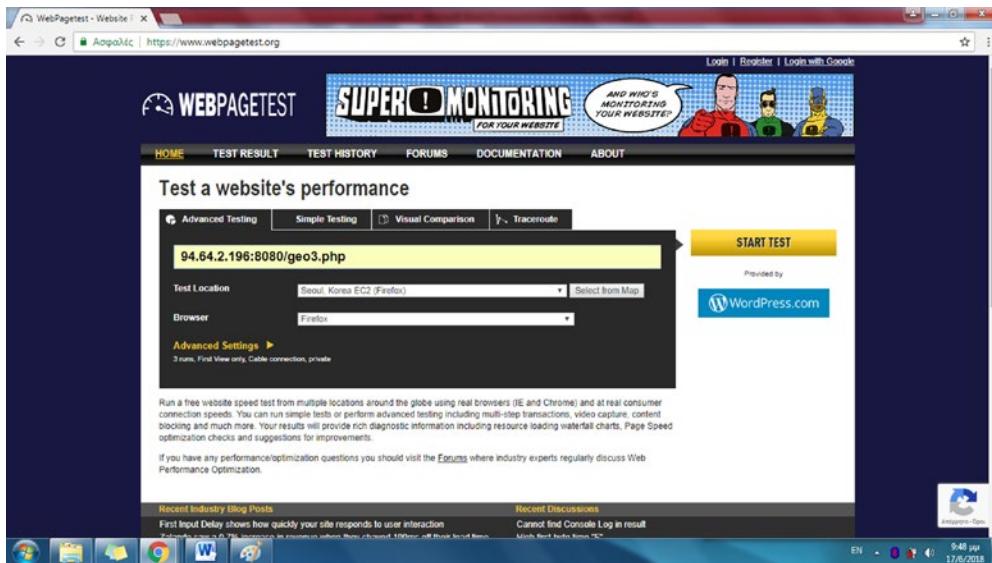


Figure 3-18. Running the test from a computer located in Seoul

As displayed in Figure 3-19, for a non-US, UK, or France visitor, instead of a written message, the downloaded web page from your site displays a broad smile.

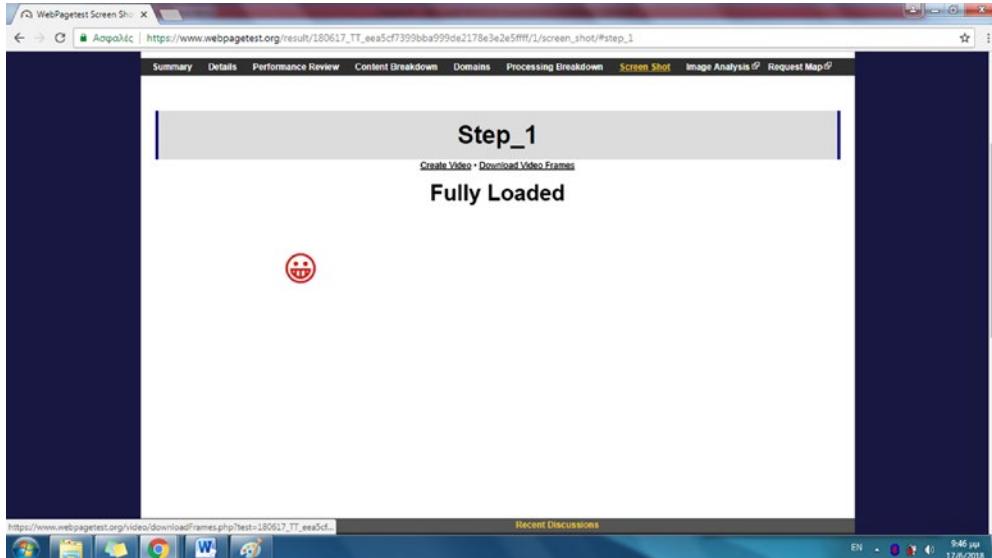


Figure 3-19. The smiley displayed in the browser of a computer located in Korea

Another site for testing your page is www.geoscreenshot.com. It performs multiple tests from different locations and displays the results in a single web page. Figure 3-20 shows a test for the geo3.php web page.

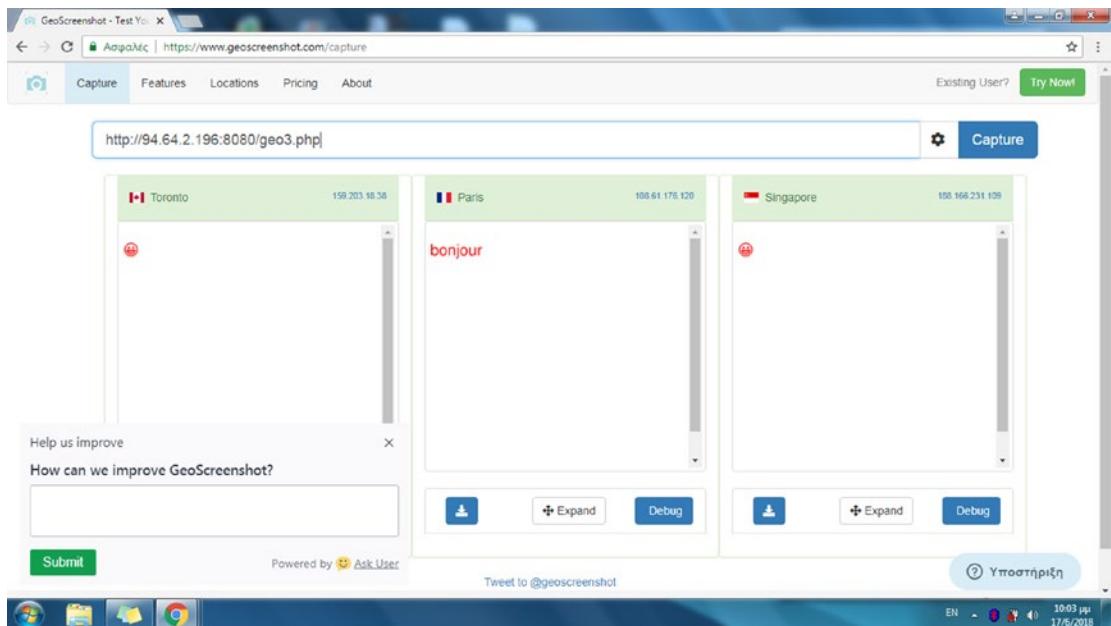


Figure 3-20. Using geoscreenshot.com to test your site remotely

In the following section, you'll install another database, GeoIPCity, to look up the IP address of the visitor and retrieve the latitude and longitude associated with the IP address so that the visitor's location on the map is returned to the user.

Using a Map to Display the Visitor's Location

In this project, you'll gather more information about the visitor, such as the city registered with the IP address used and also the geographical coordinates. The latitude and the longitude associated with the IP address of the client can be used to pinpoint the location of the client on a map. To use this information, another database file should also be downloaded and included in the files that the Apache server looks up.

Like previously, type `sudo su` to avoid prepending most of the following commands with `sudo`.

```
$ sudo su
```

Move to the GeoIP directory.

```
$ cd /usr/share/GeoIP/
```

Use the wget utility from the command line to download `GeoLiteCity.dat.gz` in the current directory.

```
$ wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
```

To uncompress the database file, use the following command:

```
$ gunzip GeoLiteCity.dat.gz
```

Rename the uncompressed database file as required by the Apache module.

```
$ mv GeoLiteCity.dat GeoIPCity.dat
```

Enter the `GeoIPCity.dat` entry in the `geoip.conf` file.

```
$ cd /etc/apache2/mods-available
```

```
$ gedit geoip.conf
```

Insert the new entry in `geoip.conf` and save the file. The contents of the configuration file are now as follows:

```
<IfModule mod_geoip.c>
  # For performance reasons, it's not recommended to turn GeoIP on
  # serverwide,
  # but rather only in <Location> or <Directory> blocks where it's actually
  # needed.
  GeoIPEnable On
  GeoIPDBFile /usr/share/GeoIP/GeoIP.dat
  GeoIPDBFile /usr/share/GeoIP/GeoIPCity.dat
</IfModule>
```

Reload Apache to enable the new configuration.

```
$ service apache2 force-reload
```

Create another PHP file, called `geo4.php`, in the document root of the web server.

```
$ cd /var/www/html
```

```
$ gedit geo4.php
```

Enter the following source code and save the geo4.php file:

```
<!DOCTYPE html>
<html>
<head><title>Location on the map</title>
</head>
<body>
<h1>Your position on the map</h1>
<?php
echo '
';
?>
</body>
</html>';
```

The previous PHP source code injects the basic HTML tags with the echo command. By using a single quote (') instead of a double quote (") in echo, you can maintain the double quotes of the HTML source code without having to escape them with a backslash (\ ") in order not to intervene with the PHP string delimiter double quotes.

The `$_SERVER['GEOIP_LONGITUDE']` and `$_SERVER['GEOIP_LATITUDE']` parts that correspond to the visitor's longitude and latitude, respectively, are the PHP global variables. The Apache module sets them by looking up the client's IP address in the GeoIPCity.dat database. The coordinates will be used in the URL value of the `src` attribute of the image element (`img`) to create the static map that will be displayed in the web page of the user. Static maps are image files, commonly in PNG or JPEG format, that do not require a mapping library. However, they lack interactivity with controls like the Zoom control or the Map Type control. According to the coordinates in the image URL, a specific area will be displayed that will reflect the origin of the client's host computer.

CHAPTER 3 CONNECTING YOUR APACHE SERVER TO THE INTERNET

There are many static map APIs you can use; some require a registration and a fee. In this project, Yandex will be used. At <https://staticmapmaker.com/yandex/>, you can find the following example for using Yandex to create a static map:

```

```

The previous tag is inserted into the HTML source code and creates an image with a width of 600 pixels and a height of 300 pixels, with English language labels. It focuses on longitude 73.7638 and latitude 42.6564 with a zooming level set to 13. There is also alternative text (alt) in case a problem occurs and the map does not load. Figure 3-21 displays how the user visiting staticmapmaker.com can interact with the Static Map Maker online tool. By using the provided form, the user can use specific geographical coordinates, zoom level, width, and height for the image created, and they can define the category of the map formed, e.g., satellite or traffic. So, you can include this map on a site, a URL with the required query string according to the user settings is also created automatically. The image tag that includes the previous URL as the value of the src attribute is also formed to be used readily in the HTML source code.

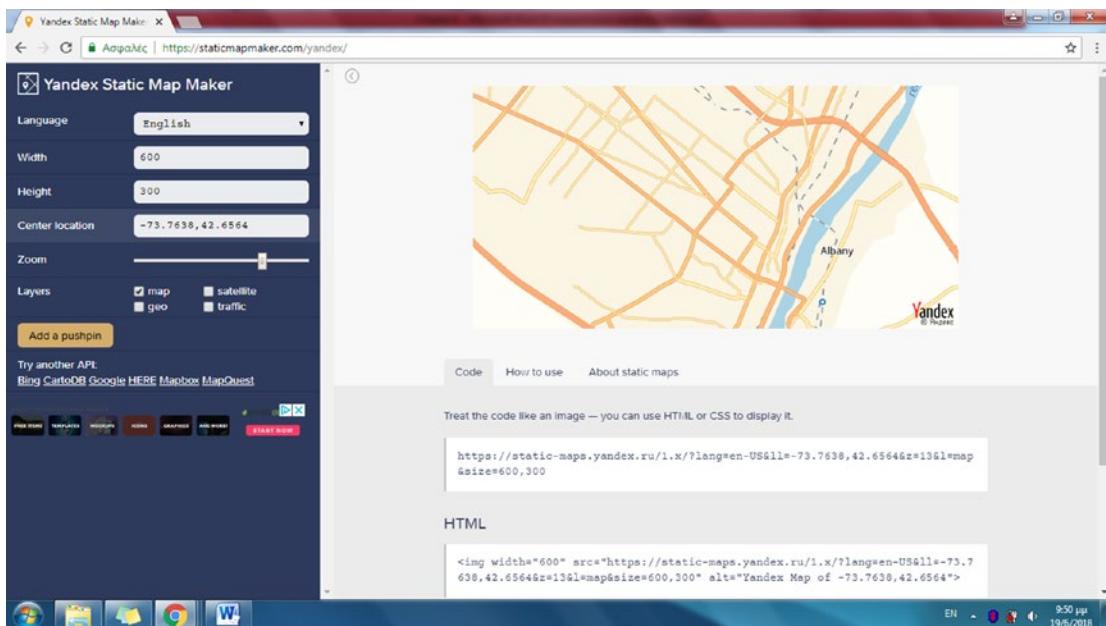


Figure 3-21. Using the Static Map Maker tool to create a static map image

You can use the format of the image element from Static Map Maker for the static map included in your web page. The basic attributes of the static map are set in the query string of the URL value of the `src` attribute. For instance, in the source code of `geo4.php`, the zoom level was set to 8 (`z=8`). The longitude and the latitude were replaced by the PHP global variables `$_SERVER['GEOIP_LONGITUDE']` and `$_SERVER['GEOIP_LATITUDE']` to hold the values associated with the client's IP address. To form the `src` attribute, which includes the PHP global variables, a number of echo commands are used.

```
<?php
echo '

';
?>
```

To test this web page, use an online service like whatismyipaddress.com to find the public IP address of your router and replace it with the IP address used in this example, 87.202.116.192:

87.202.116.192/geo4.php
Or if you utilize port number 8080 use:
87.202.116.192:8080/geo4.php

Figure 3-22 displays the requested web page `geo4.php` from a computer in the LAN where the web server is hosted for this example.

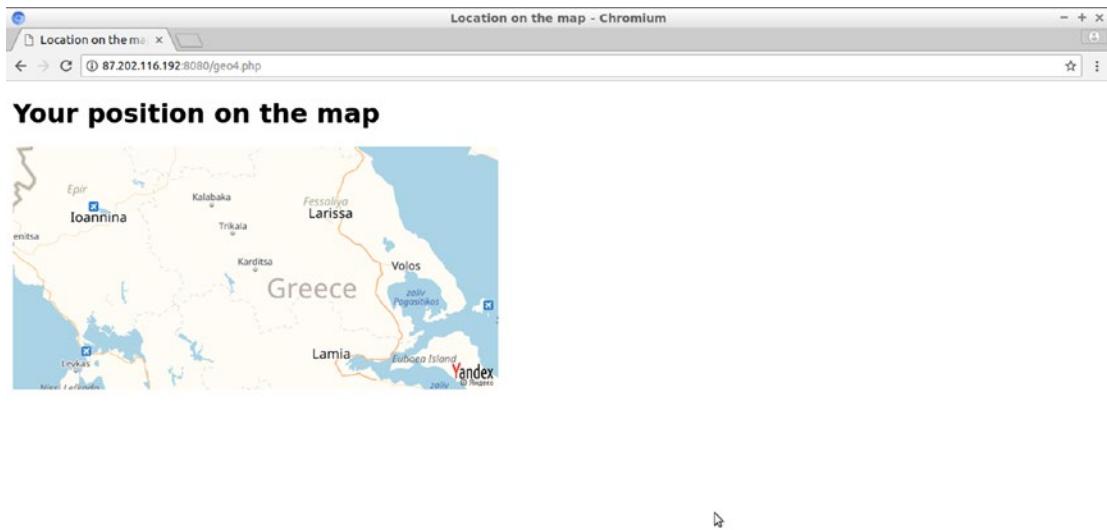


Figure 3-22. Testing geo4.php from a computer of your LAN

Right-click the web page and select View Page Source to display the web page source code and find the values for `$_SERVER['GEOIP_LONGITUDE']` and `$_SERVER['GEOIP_LATITUDE']` as included in the query string of the `src` attribute in the `img` element. For the previous example, the `img` element is formed as follows:

```

```

Rename the `geo4.php` file as `geo5.php` and use it to make a new version of the previous source code.

```
$ cp geo4.php geo5.php
$ gedit geo5.php
```

The contents of `geo5.php` are displayed here:

```
<!DOCTYPE html>
<html>
<head>
<title>Location on the map</title>
</head>
<body>
<h1>Your position on the map is in 
```

```

<?php
echo $_SERVER['GEOIP_CITY']
. ', ';
. $_SERVER['GEOIP_COUNTRY_NAME']
. '</h1></body></html>';
?>
</body>
</html>

```

Two more PHP global variables are used in the previous source code: `$_SERVER['GEOIP_CITY']`, which holds the name of the city and `$_SERVER['GEOIP_COUNTRY_NAME']`, which corresponds to the name of the country as registered with the client's IP address.

Using the URL of file `geo5.php`, the web page is displayed, as shown in Figure 3-23.

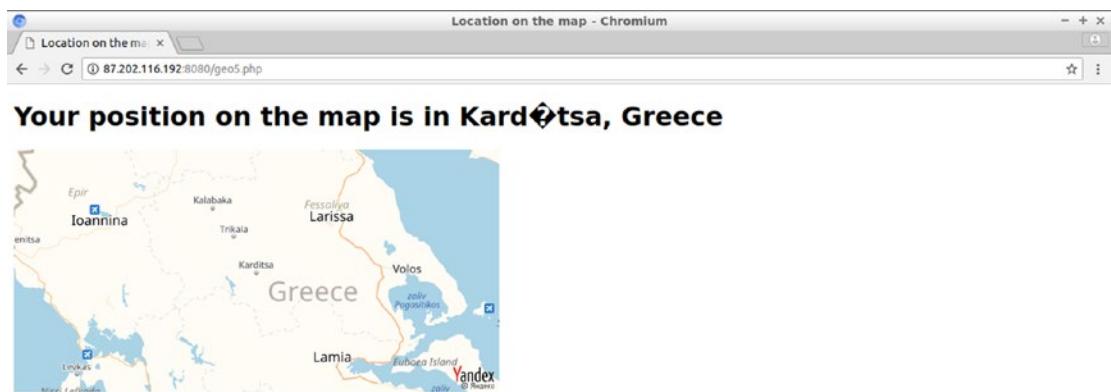


Figure 3-23. Testing `geo5.php`

In the next section of this chapter, you'll make changes to improve the appearance of the previous site.

A New Version of the Static Map Web Page

Here you'll use gedit to create in the document root of your web server a new version of the geo5.php file, called geo6.php.

```
$ cd /var/www/html  
$ sudo gedit geo6.php
```

Enter the following source code and save the file:

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Your Location</title>  
<style>  
body{  
background-color:#B9D6E5;  
}  
.center {  
margin: auto;  
}  
p{  
text-align: center;  
}  
</style>  
</head>  
<body>  
<div class="center">  
<p>  
<?php  
echo '  
<h1 style="text-align:center">Your position on the map is in '  
. $_SERVER['GEOIP_CITY']  
. ', '
```

```
. $_SERVER['GEOIP_COUNTRY_NAME']
. '</h1><p></p></body></html>';
?>
</p>
</div>
</body>
</html>
```

Test geo6.php in your browser, as shown in Figure 3-24.



Figure 3-24. The map as displayed in the new version, geo6.php

In the new file geo6.php, there are a few significant differences from the web page created with the file geo5.php. First, the image is centered on the web page. A `div` element of the `class center` is used to include the image and the heading. The class is defined with CSS as follows:

```
.center {
margin: auto;
}
```

The second difference in the geo5.php source code is the zoom level, set now to 4, to include a larger area on the map. By zooming out on the map, the name of the city is not printed anymore, and instead a map pin is used to indicate the city's location. There are a lot of different placemarks offered for the Yandex maps. You can view them at <https://tech.yandex.com/maps/doc/staticapi/1.x/dg/concepts/markers-docpage/>. The format for the value of the `pt` variable on the image's URL, used for placing a specific marker, is as follows:

```
{longitude},{latitude},{style}{color}{size}{content}
```

In the URL of the map, the value of `pt` includes the coordinates of a city to pinpoint the marker at an exact location.

```
...pt=' . $_SERVER['GEOIP_LONGITUDE'] . ',' . $_SERVER['GEOIP_LATITUDE'] .
',pm2rdl...
```

The value `pm2rdl` corresponds to a placemark that belongs to group 2 of the Yandex placemarks (`pm2`), is red (`rd`), and also has a large (1) size.

The third difference is the background color selected for the web page, which is actually the same color used for the oceans on the map. To find the specific color (RGB color code #B9D6E5 in this example), use the following instructions.

First make a screenshot of the geo6.php web page, using a Linux screenshot tool like Shutter. To download and install Shutter, use the following command at the Linux terminal:

```
$ sudo apt-get install shutter
```

Start the application from the Linux GUI or from the command line as follows:

```
$ shutter
```

The Shutter window opens, as displayed in Figure 3-25.

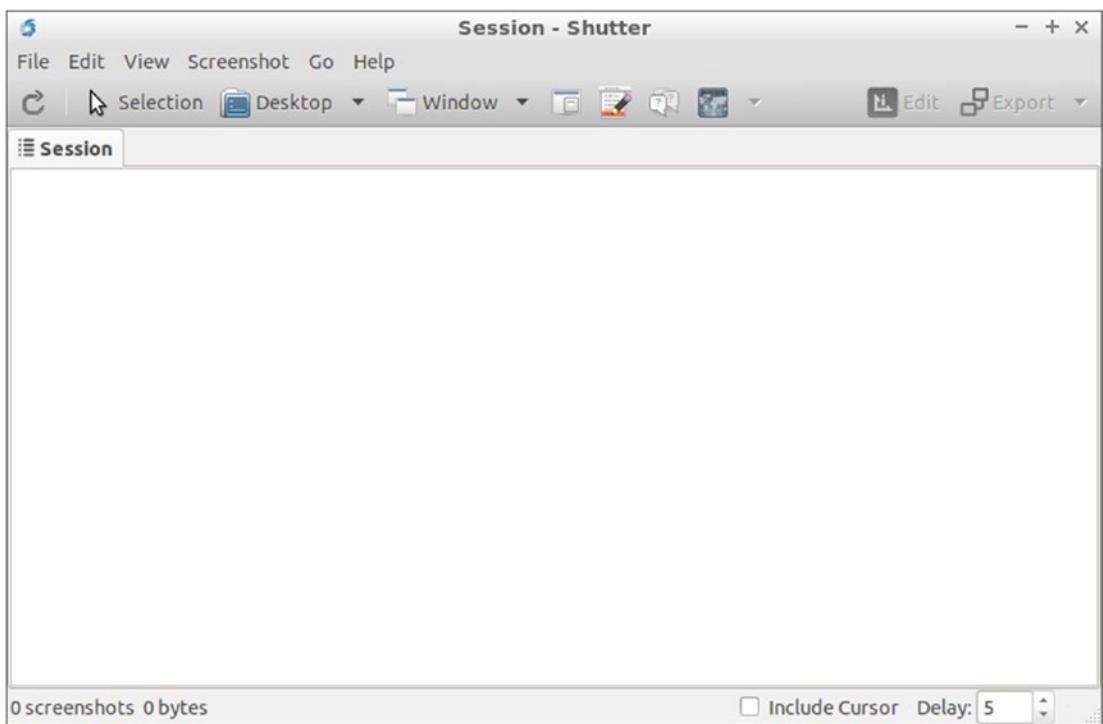


Figure 3-25. The Shutter window

Open with your browser the previous version, geo5.php, to detect the ocean color. Click the arrows for the up-down control at the bottom of the Shutter window to set the seconds (e.g., 5) of the Delay field. This corresponds to the delay timer for the shot. Click next to the Window menu button and select Active Window. The delay timer starts counting, and within this interval you have to select with the mouse the browser window, displaying geo5.php, for the screenshot. Click the browser window to select it. The image is saved in the Pictures folder of your home directory.

Use an image editor like mtPaint to open the image file. With the Eyedropper tool, click the color of the ocean. As displayed in Figure 3-26, the Eyedropper tool for mtPaint is included as a button in the Palette Editor window, which you open by selecting the Palette menu and then Palette Editor.

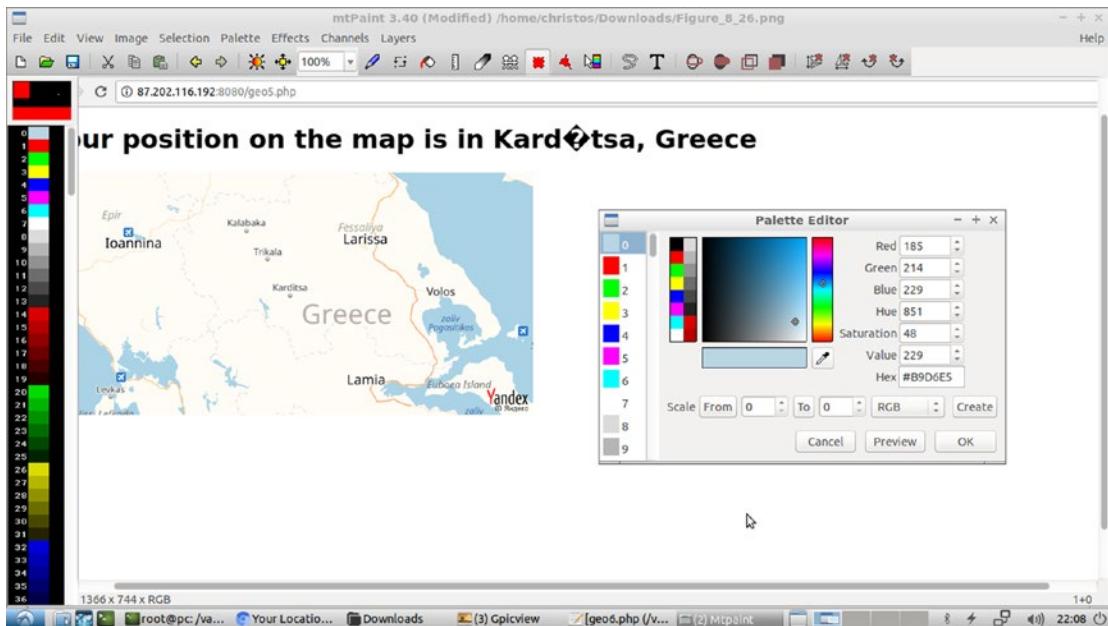


Figure 3-26. *The Eyedropper tool of mtPaint*

The color used for the oceans in geo5.php is used in the source code of geo6.php as the background color, by setting the value of the background color in the CSS section to the RGB value #B9D6E5.

Summary

This chapter covered the following:

- You enabled the web server to become accessible to the whole Internet by configuring the router to implement port forwarding.
- You installed the GeoIP Apache module that associates the visitor's IP address with geographic information.
- You ran examples that use the information provided from the GeoIP module with the PHP source code.
- You used online tools to test your site remotely.

CHAPTER 4

Obtaining a Domain Name with DDNS

In the previous chapter, your site passed the borders of your LAN and became available to the whole Internet. URLs that were not exactly user-friendly like the following were used:

`http://94.64.2.196:8080/`

In this chapter, you will learn how to use the Dynamic DNS (DDNS) service to obtain and use a domain name for your site; for example, you can use [webtoolsonline.servehttp.com](#) instead of an IP address in the URL.

With a proper URL for your site, you can set up PHP to interface with the Linux shell and use an online service like WHOIS. Also, you will further improve your site's appearance by creating a favicon icon, which is an icon that represents your site in clients' browser tabs and in the list of bookmarks.

DNS and DDNS

The Domain Name System (DNS) service translates the domain name of a computer to the corresponding static IP address. Purchasing a static IP address from your ISP is usually more expensive than having a dynamic IP address that often changes, for instance every time your router reboots. Correlating a domain name to a dynamic IP address is still possible with DDNS. There are two ways to update the DDNS service provider data with your changed IP address. The method discussed in this chapter is to use a DDNS service in your router. Today most routers support DDNS services. The only drawback is that some router models support DDNS for only certain DDNS service providers. If your router does not support your DDNS provider, you can use a DDNS

client program that runs in the background of your web server as a daemon process. This is the method implemented in Chapter 10, where you'll use the program ddclient. The program ddclient checks in constant time intervals to see whether the public IP address of your router has changed. If it has, an update message is sent to your DDNS service provider.

DDNS services can be free or require a monthly fee. A nice feature offered by most DDNS service providers is web redirect. Consider, for instance, the following URL with an IP address:

`http://94.64.2.196:8080/`

Using a DDNS service, the client can replace the public IP address of the router with a domain name and use the following in the address bar of a browser:

`http://christos.ddns.net:8080`

The domain name is used instead of the IP address; however, the port number is present. Web redirect helps you hide the port number and enables your visitors to use a more readable URL, as shown here:

`http://christos.ddns.net`

Registering with a DDNS Service Provider

This section describes the steps to register with a no-IP DDNS service provider. This is one of the providers supported by the router that is used in the examples of this book. You can register with any other provider supported by your router following a similar process. As with most online services, you sign up by providing your e-mail address and a unique password. For a DDNS service provider, you can choose a third-level domain name. For instance, I chose `christos` under the configuration authority of the second-level domain name `ddns` (the DDNS service provider itself), which is under the configuration authority of the top-level domain (TLD) or first-level domain `net`. The fully qualified domain name (FQDN), used in the site, therefore becomes the following:

`christos.ddns.net`

With this specific DDNS server provider, you can choose three hostnames and also implement web redirect for free.

Now visit the no-IP site.

<http://www.noip.com>

Once there, click the Sign Up button to begin the registration process (Figure 4-1).

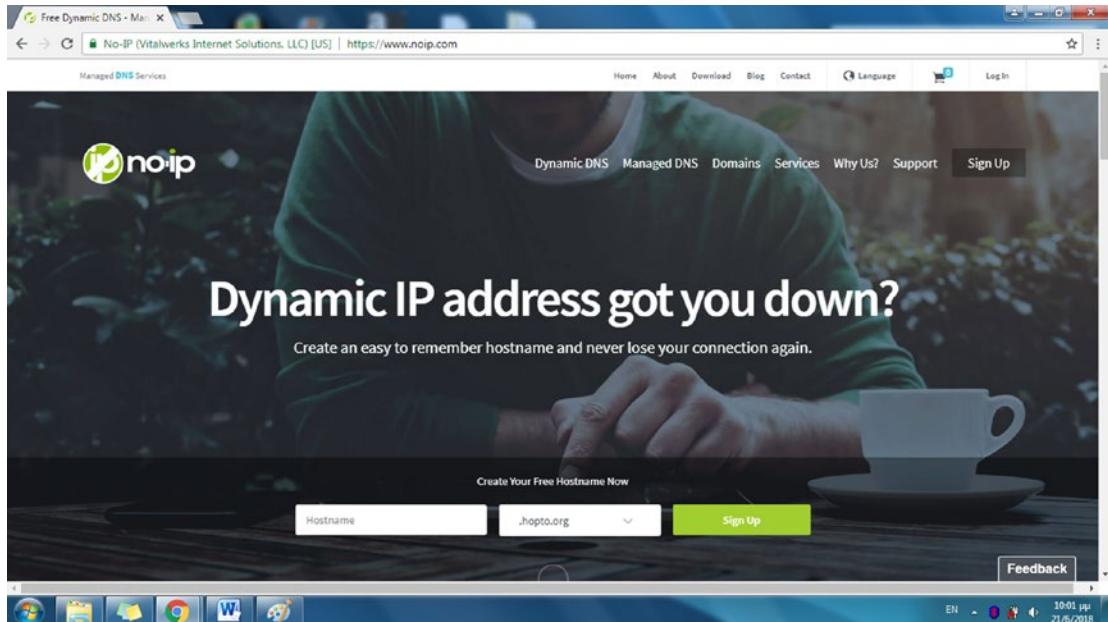


Figure 4-1. The Sign Up button of the no-IP home page

The next page, shown in Figure 4-2, allows you to select a hostname (e.g., christos) and the second-level domain name (for instance, zapto.org or ddns.net) from a drop-down list. You need also to provide your password and your e-mail. Click the Terms of Service and Privacy Policy box and then click the Free Sign Up button.

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

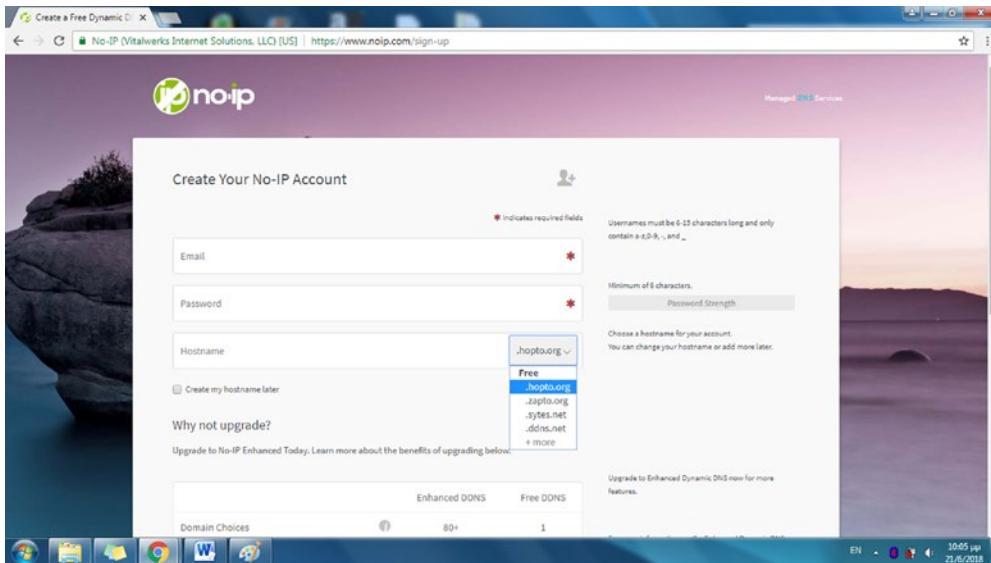


Figure 4-2. Setting your preferences for your hostname with no IP

The next page, shown in Figure 4-3, asks you to confirm your account by clicking the link of the e-mail sent to the account you provided. This allows the DDNS provider to verify that your e-mail account is a real one.

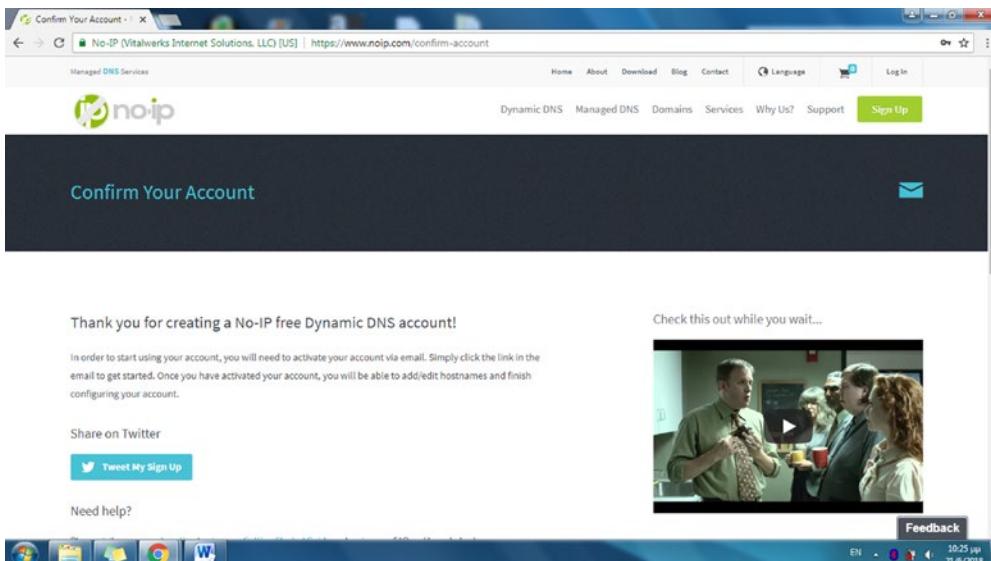


Figure 4-3. The web page requiring e-mail account confirmation

Figure 4-4 displays a snapshot of the e-mail received.

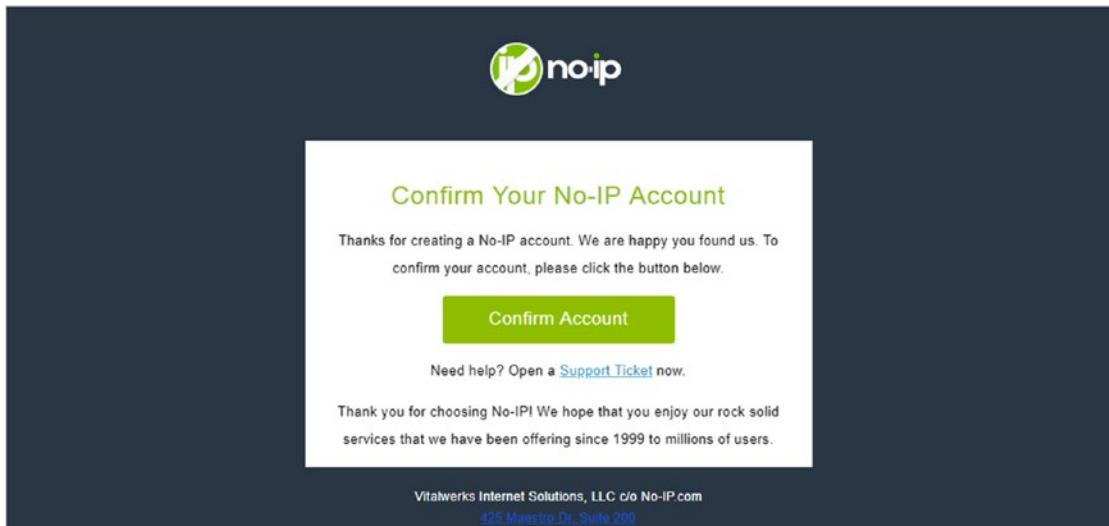


Figure 4-4. The e-mail, including a Confirm Account button for validating the no-IP account

After clicking the Confirm Account button, the window shown in Figure 4-5 appears in a new tab of your browser with the message “Your account is now active!”

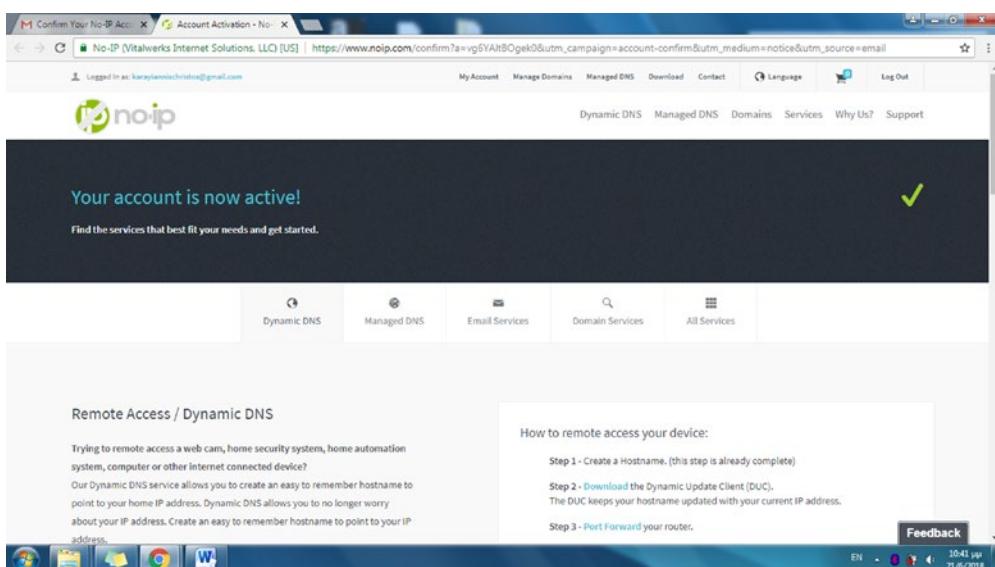


Figure 4-5. The web page informing you that the account is activated

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

Click the “Get started with Dynamic DNS” link to continue. The window displayed in Figure 4-6 informs you to create a username and a security question to complete your account configuration. Click the Add Now button to proceed.

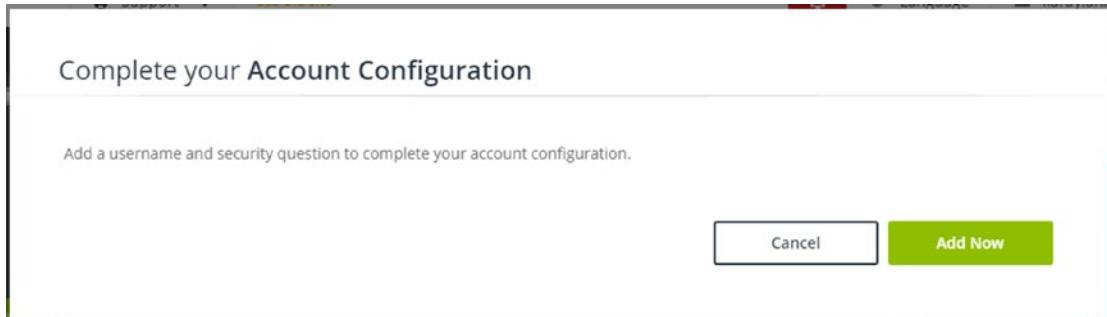


Figure 4-6. The dialog for proceeding to the final step of creating a DDNS account

In the web page shown in Figure 4-7, you set your username, security question, and also some personal information.

A screenshot of a web browser showing the "Account Info" page of the No-IP website. The URL is https://my.noip.com/#/account. The left sidebar shows navigation options like Dashboard, Dynamic DNS (Free), My Services, Account, Account Info (selected), Account Settings, Email Settings, Invoices, and Support Center. A "Upgrade to Enhanced" button is visible. The main content area has a red header bar saying "Complete your account. Create a username and security question now". It includes fields for "Username" (empty), "Email" (karayannischristos@gmail.com), and "Password" (hidden). Below this is a "Security Question" section with a dropdown menu for "Select A Security Question" and an empty "Answer" field. At the bottom is a "Personal Info" section with a dropdown menu for "Select an Option". On the right side, there's a "Service Level" section stating "Free Hostnames expire every 30 days. Enhanced Hostnames never expire." and a "Upgrade to Enhanced" button. There's also a "Quick Stats" section with "Last Login IP: N/A", "Last Login: 00:00:00", and "Support Type: Basic". A "Feedback" button is at the bottom right. The system tray at the bottom shows icons for network, battery, and date/time (10:54 μ, 21/6/2018).

Figure 4-7. The final web page of the DDNS account setup process

Click the Save button to save the information entered and thus complete the account setup process. You can use the hostname obtained here to configure the router of your LAN to implement the DDNS service.

Configuring the Router's DDNS

With a new domain name obtained in the previous section, you can now configure your router to relay its public IP address to the DDNS provider each time this IP address changes. Locate the DDNS section in the web-based configuration interface of your router. As shown in Figure 4-8, for the router used in the examples, you can view the DDNS configuration options by selecting Internet and then DDNS from the menu.

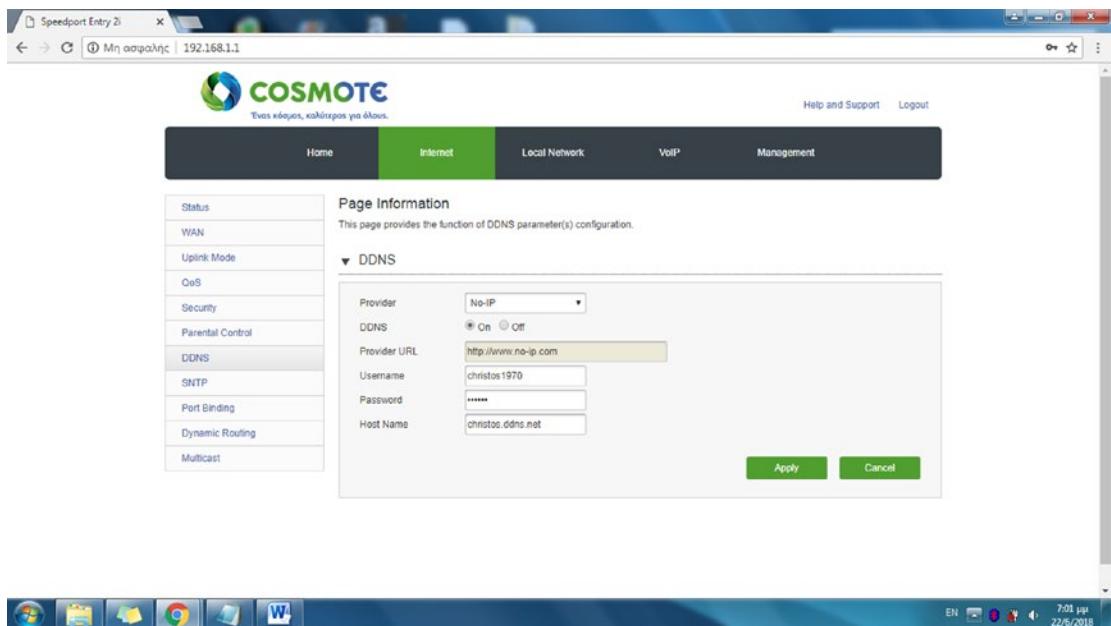


Figure 4-8. The DDNS configuration section for the router used in the examples

Select the DDNS provider from the Provider drop-down list, which has all the DDNS providers that the router supports. Enter the details required: your username, your password, and the hostname you chose at the DDNS registration process. Complete also the Provider URL field, and select the On DDNS radio button. Click the Apply button to confirm the new settings.

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

Test your new domain name (christos.ddns.net is used in the following examples). In this chapter, the Apache configuration created in Chapter 1 is used. You can test the URLs of the web pages created in the previous chapter, for instance geo5.php. Enter the following URL for geo5.php in the address bar of your browser, substituting the domain name used here with yours:

`http://christos.ddns.net/geo5.php`

If your ISP blocks inbound port 80, which is the default HTTP port, use an alternate port like 8080.

`http://christos.ddns.net:8080/geo5.php`

The web page geo5.php is displayed, as shown in Figure 4-9.

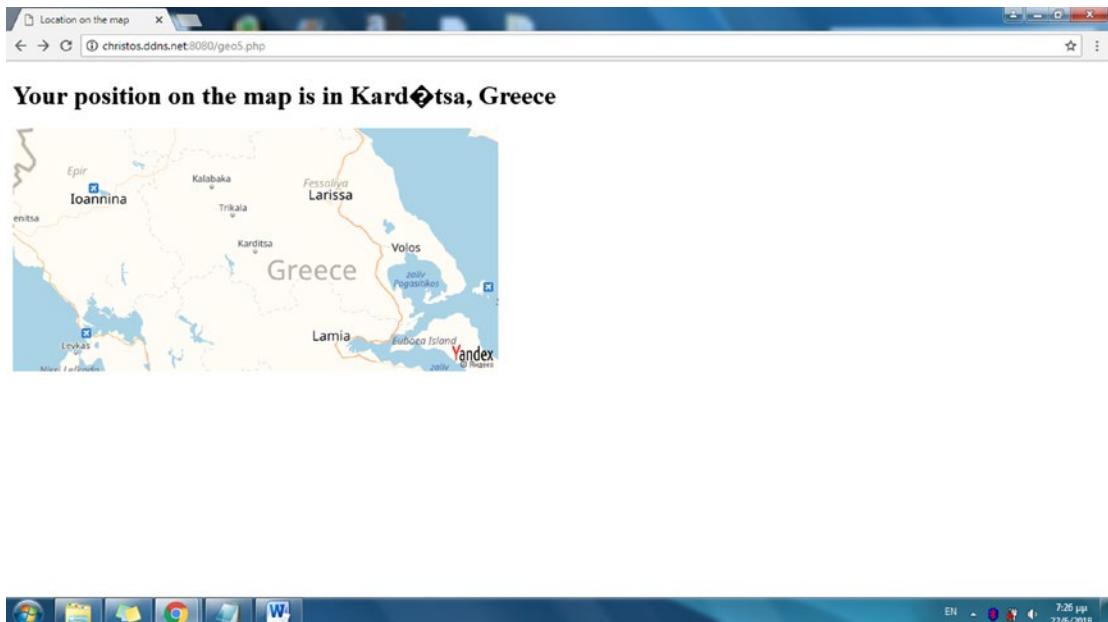


Figure 4-9. Testing the web page geo5.php with the new domain name

The web page geo5.php loads with the URL that includes the previously acquired domain name. In the following section, you will implement web redirect so that if you do not utilize the default HTTP port number, but another one such as 8080, this number is hidden from the URL.

Implementing Web Redirect

If your ISP blocks the default HTTP port for incoming connections to your LAN, you can still use URLs that do not append a colon and the port number after the domain name, although behind the scenes the specific port number *is* actually used. To include this feature, you need to configure the web redirect service offered by DDNS providers.

By setting a specific port number, such as port 8080 in the web redirect option, you are enabling clients to use a port-free URL. The DDNS provider redirects visitors to another domain name, where the port number is already appended with a colon. To test the web redirect feature, another domain name is required, so this is a good chance to utilize the second of the three domain names offered from the DDNS provider.

Log in on the DDNS provider home web page and click the Dynamic DNS link in the left panel. The first domain name, e.g., christos.ddns.net, appears as a link that leads to the configuration web page of this domain name, as displayed in Figure 4-10.

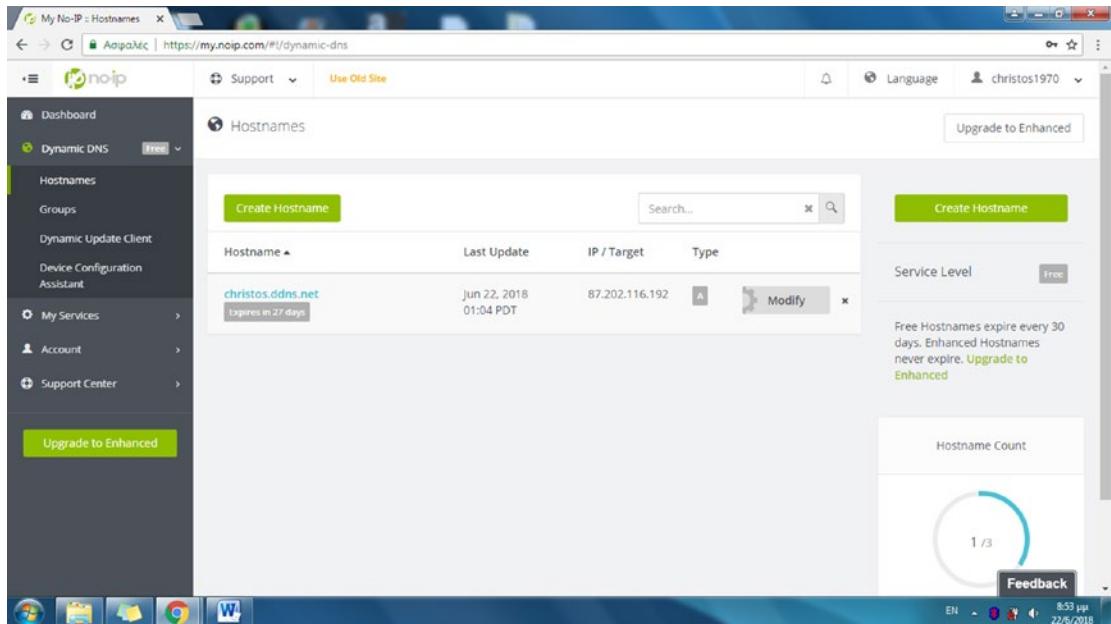


Figure 4-10. The Hostnames web page displays the hostnames created so far

Click the Create Hostname button to create your second domain name, the one used for web redirect. Enter the third-level domain name in the Hostname textbox, as shown in Figure 4-11. For instance, in this example, `webtoolsonline` was used.

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

To also change the second-level domain name, select a different one from [ddns.net](#) in the Domain drop-down list. In this example, [servehttp.com](#) was used. The FQDN therefore becomes as follows:

webtoolsonline.servehttp.com

For the Record Type radio buttons, select Web Redirect to implement this service for the new domain name. By selecting this option, a new textbox URL/IP address appears. Type the first domain name with port 8080 appended after a colon.

christos.ddns.net:8080

Each time the new hostname [webtoolsonline.servehttp.com](#) is used from a client, the DDNS service redirects the request to [christos.ddns.net:8080](#).

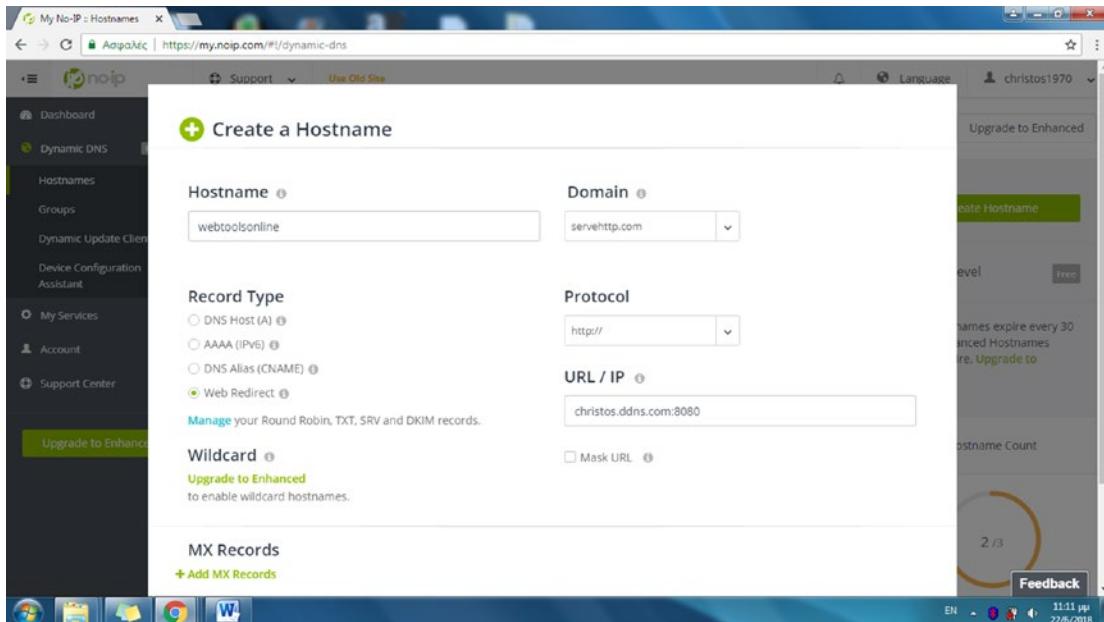


Figure 4-11. Creating the second hostname with the Web Redirect option set

Click the Create Hostname button. As viewed in Figure 4-12, the new domain name is added in the Hostname list.

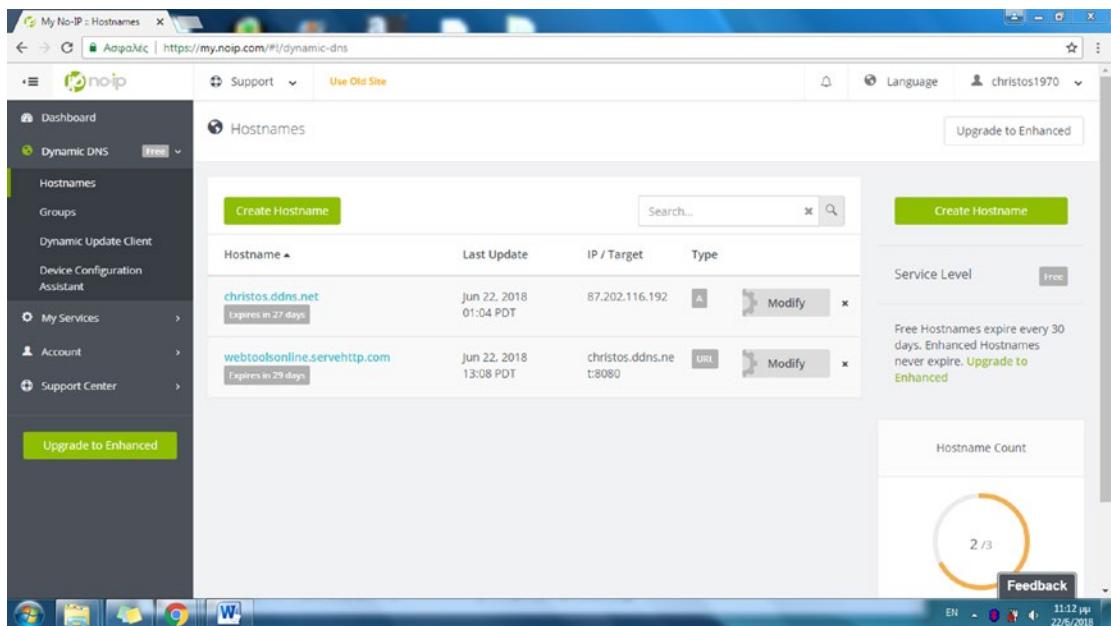


Figure 4-12. The Hostnames web page displays the hostnames created so far

To try the new domain name, enter the following in the address bar of your browser:

<http://webtoolsonline.servehttp.com>

With the web redirect service enabled, the URL <http://webtoolsonline.servehttp.com> redirects to christos.ddns.net:8080. The directory index of the Apache vhost listening to port number 8080 (used in Chapter 1) is displayed in the web browser, as shown in Figure 4-13.

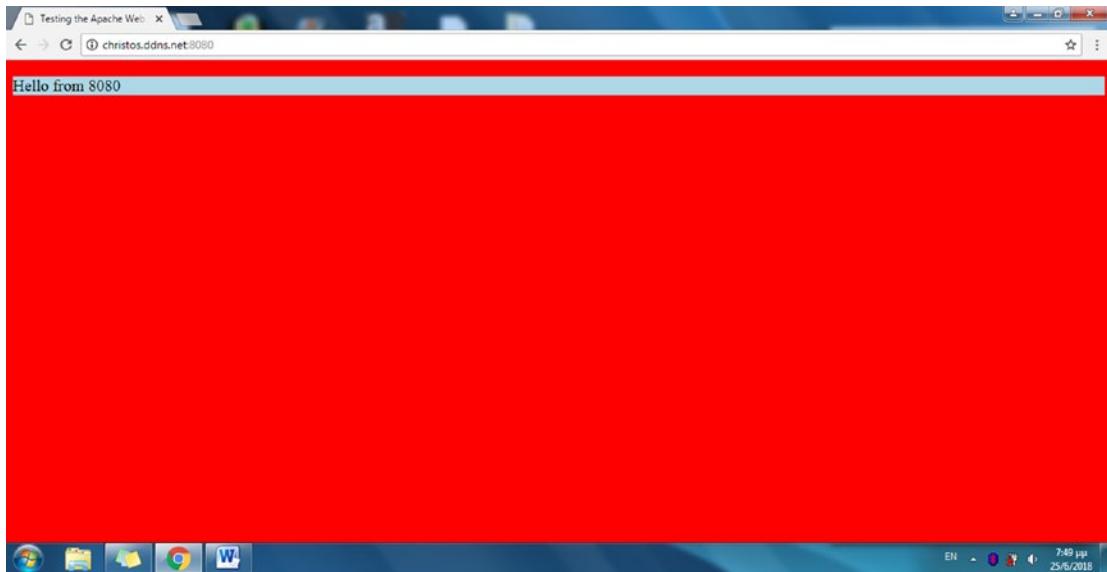


Figure 4-13. The URL webtoolsonline.servehttp.com redirects to christos.ddns.net:8080

To visibly retain the original URL, you have to implement the Mask URL feature by configuring the DDNS provider. Edit the hostname on the Hostnames web page of the DDNS provider's site and select the Mask URL box. Click the Update Hostname button to apply the configuration.

With the URL mask enabled, the DDNS service injects JavaScript into the source code of the web page to create an HTML frame that covers the entire browser window and includes the same content but a different URL in the address bar of the browser. The result, as displayed in Figure 4-14, is a web page with the same content; however, instead of the expected URL shown here where the client redirects:

`http://christos.ddns.net:8080`

the following URL appears in the address bar:

`http://webtoolsonline.servehttp.com`



Figure 4-14. Using the URL mask option to modify the URL displayed in the client browser

Hint! To view the frame created with the Mask URL option, right-click the web page and select View Page Source for Chrome or the corresponding option for other browsers. In this case, the URL of the address bar is altered with JavaScript code.

You have so far created and used two hostnames for your registered DDNS account. You will use the third hostname, e.g., secureserver.ddns.net, in Chapter 8. In Chapter 9, you will find the steps to obtain and use your own second-level domain name, e.g., httpsserver.eu, required for running a secure site. In the following section, you will put the hostnames created so far to use to create your next project.

Implementing an Online Web Service

With such a catchy domain name like webtoolsonline.servehttp.com, it is tempting to implement an actual online web service. You can implement, for instance, the WHOIS service, which applies whois, a query and response protocol that is widely used for querying databases that store the assignees of Internet resources, such as domain names

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

and IP addresses. You can run the `whois` command at the command line, but it is also offered as an online web service from various sites, e.g., whois.domaintools.com or ping.eu, which is shown in Figure 4-15.

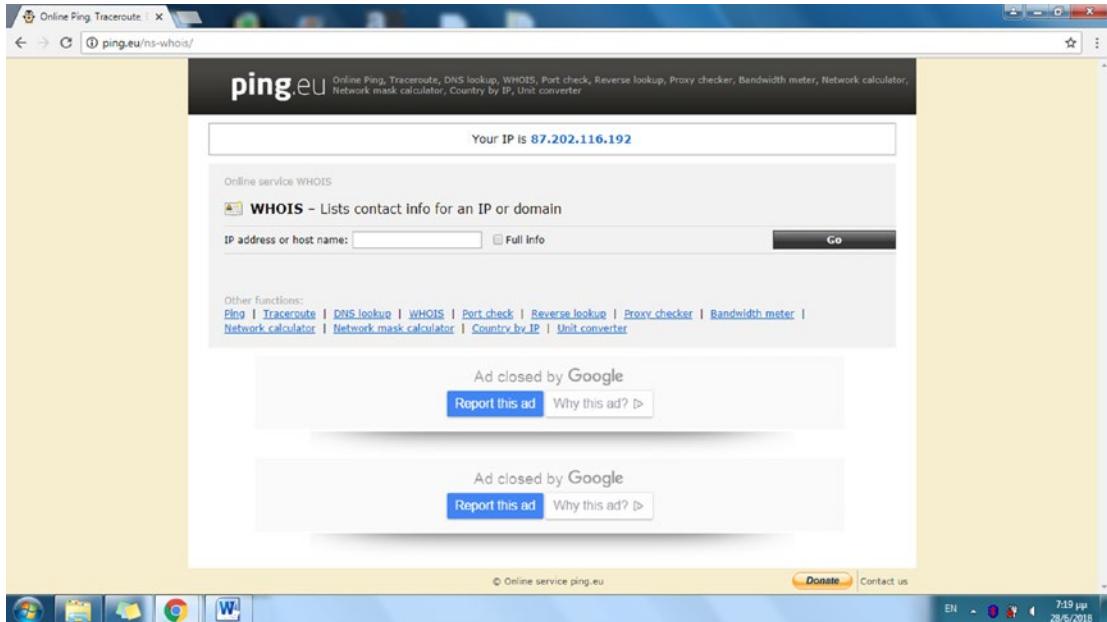


Figure 4-15. The web page of the ping.eu WHOIS service

Ping.eu, along with other similar sites, offers online web services, for instance DNS lookup, ping, traceroute, and port check. You will implement port checking in Chapter 10.

To create a site that offers WHOIS as an online service, you can use a web page with a form; the form will include a textbox to submit an IP address to the PHP program on the web server that interfaces with the Linux shell and returns information about the corresponding domain name.

The first step to implement the online service is to change the default web page that Apache sends in response to using port 8080. As explained in the previous sections of this chapter, the domain name webtoolsonline.servehttp.com resolves to christos.ddns.net:8080 and to the router's public IP address through port 8080. The Apache vhost configuration file for serving port 8080 for any domain name was set up in Chapter 1 to `example2.conf`. The directory index used in this file for port 8080 is `index3.html`, which is actually the web page shown in the previous section with the URL <http://webtoolsonline.servehttp.com>. As a second step, edit `example2.conf` and replace `index3.html` with another HTML file, for instance `onlineservice.html`.

Create a new web page called `onlineservice.html` that includes a form similar to the one on the `ping.eu` page. This form will be used to submit the user's data to the web server, and as viewed from `ping.eu`, two form elements are required at a minimum: the textbox that will receive the IP address or the hostname and a submit button. The form's `action` attribute will refer to a PHP file that receives the user's data, interfaces with the `whois` command-line program, and then passes the user's data as arguments. Finally, the `whois` output is included on the web page generated by PHP and sent back to the user. With similar steps, you can implement any other online service such as the ones offered by `ping.eu`.

Editing the Apache Configuration File

In this section, you'll edit the configuration file `example2.conf` used in Chapter 1 to implement two sites with two vhosts, one listening on port 8080 and one listening on port 8181. In this example, only the vhost listening to port 8080 is required; therefore, for this vhost the `DirectoryIndex` directive is edited to set the file `onlineservice.html` as the new directory index of the site.

At the Linux terminal, enter the following:

```
$ cd /etc/apache2/sites-available  
$ sudo gedit example2.conf
```

Replace `index3.html` with `onlineservice.html`.

```
Listen 8080  
Listen 8181  
# 1st vhost  
<VirtualHost *:8080>  
    ServerAdmin webmaster@localhost  
    DocumentRoot /var/www/html  
    DirectoryIndex onlineservice.html  
    ErrorLog ${APACHE_LOG_DIR}/error.log  
    CustomLog ${APACHE_LOG_DIR}/access.log combined  
</VirtualHost>  
  
# 2nd vhost  
<VirtualHost *:8181>
```

```
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html
DirectoryIndex index4.html
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Click the Save button in the gedit window to confirm the changes. To reflect the configuration changes for the vhost listening on port 8080, the Apache web server needs to be reloaded. At the Linux terminal, enter the following:

```
$ sudo service apache2 force-reload
```

Hint! In this example, an existing configuration file was used. Consider the case that you implement the site with a new configuration file for a vhost that listens to the default HTTP port, which is 80. You can create the configuration with the following commands:

```
$ cd /etc/apache2/sites-available
$ sudo gedit onlineservice.conf
```

Enter the following directives and save the file:

```
Listen 80
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html
    DirectoryIndex onlineservice.html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Then use the following:

```
$ sudo a2ensite onlineservice  
$ sudo service apache2 force-reload
```

To test the site, you create the first file called `onlineservice.html` as described in the following section, and you enter the first hostname, `christos.ddns.net`, in the address bar of the browser. The second one, `webtoolsonline.servehttp.com`, redirects to `christos.ddns.net:8080`, which listens to port 8080 instead of the default port.

The next step is to create the directory index used from the vhost. This is called `onlineservice.html` and is the web page that contains the form for submitting an IP address.

Editing the Web Page for Submitting the User's Data

The HTML file that includes the form used by the user to provide the data to the web server is `onlineservice.html`, which was set in the previous section in the vhost configuration file as the directory index for the site `webtoolsonline.servehttp.com`. Figure 4-16 displays the directory index.

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

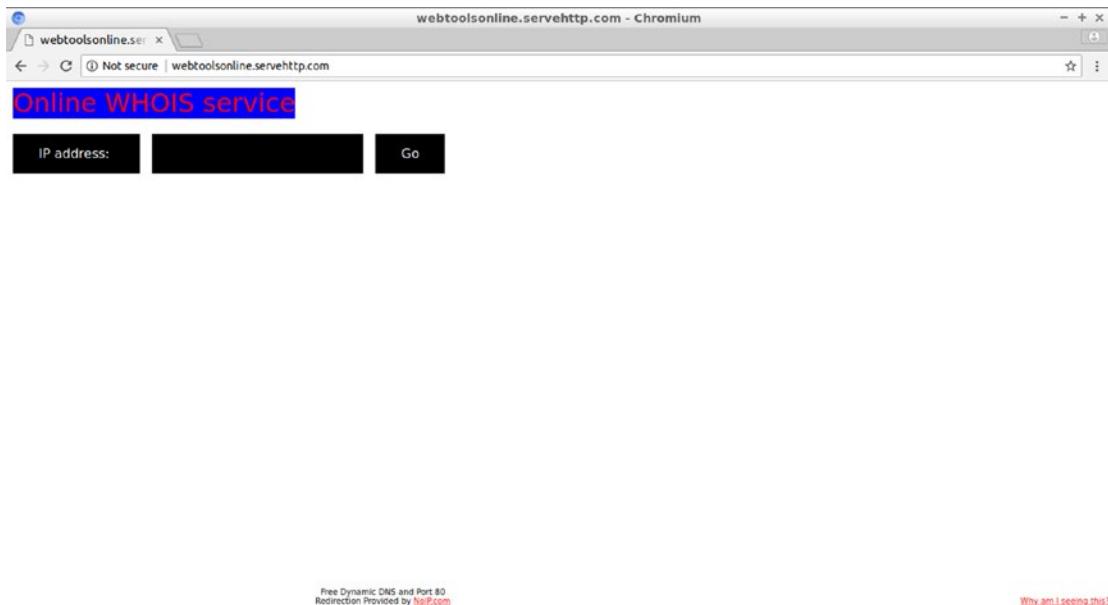


Figure 4-16. The browser displays *onlineservice.html*, the directory index of the site

Create the file *onlineservice.html* in the document root of the vhost. At the Linux terminal, enter the following:

```
$ cd /var/www/html  
$ sudo gedit /var/www/html/onlineservice.html
```

Enter the following lines and save the file:

```
<!DOCTYPE html>  
<html>  
<head>  
<title>  
Online WHOIS service  
</title>  
<style>  
p.large {  
font-size:32px;  
color:red;
```

```
background-color:blue;
display:inline;
}
p.small {
font-size:16px;
color:white;
background-color:black;
display:inline;
padding: 15px 32px;
}
input {
    background-color: black;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    font-size: 16px;
}
</style>
</head>
<body>
<p class="large">Online WHOIS service</p>
<br><br>
<form method="POST" action="result.php" name="f1" onsubmit="return validate()">
<p class="small">IP address: </p>&nbsp;&nbsp;
<input type="text" name="user_data">&nbsp;&nbsp;
<input type="submit" value="Go">
</form>

<script>
function validate() {
var f = document.forms["f1"]["user_data"].value;
f=f.trim();
var ipformat = /^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|([1-9][0-9])|[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|([1-9][0-9])|[0-9])$/
```

```

if(!(f.match(ipformat))){
alert("Enter a valid IP address");
return false;
}
}
</script>
</body>
</html>

```

The HTML source code includes a form that implements the method POST for submitting the data and sets the value of the action attribute to result.php. This is the program on the server side that will accept and process the data submitted.

There are two more attributes in the form element, both required for the form validation, which is implemented locally by JavaScript. The name attribute is used by the JavaScript function validate(), which validates the form, to refer to this form. The return value of validate() is set as the value of the fourth attribute of the form, the onsubmit event. Events are actions applied to the web page that JavaScript responds to. The onsubmit event defines the form submission as the exact instant in time that the source code bound to this event will run.

By setting the onsubmit event value to return validate(), the function validate() will run when the form is submitted, and also the form will be submitted only when validate() returns true. The function validate() is implemented in the following script:

```

<script>
function validate() {
var f = document.forms["f1"]["user_data"].value;
f=f.trim();
var ipformat = /^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|1[0-9][0-9])\{3\}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|1[0-9][0-9])$/;
if(!(f.match(ipformat))){
alert("Enter a valid IP address");
return false;
}
}
</script>

```

The `name` attribute of the form is used next to refer to the specific form in the HTML file and then return the value of the `user_data` textbox to variable `f`. For the variable `f`, which is an object of class `string`, the `trim()` method removes whitespace from both sides of the string. If the user accidentally enters spaces in the `user_data` textbox with the IP address, `trim()` is used so that `validate()` won't reject the IP address.

The `ipformat` regular expression (`regex`) is used next to check whether the value of variable `f` matches the pattern of a valid IP address according to `ipformat`. The details of this regex are explained in the following section.

The `match string()` method is used to search string `f` for a match against the regular expression. The following `if` command uses `match()` to verify whether the `ipformat` regex does not (!) match string `f`:

```
if(!(f.match(ipformat))){
```

If a match is not met, the IP address is considered invalid, and `validate()` returns false. In that case, the form is not submitted, and the user is asked to enter a valid IP address with a pop-up window message, created by `alert()`.

Working with Regular Expressions

Expressions are character sequences that define search patterns. They are extremely useful in validating or extracting information from text. The regular expression assigned to the JavaScript variable `ipformat` is used to describe the pattern for a valid IP address and validate the user data. The following command assigns the regex to the `ipformat` variable:

```
var ipformat = /^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|1[0-9][0-9])\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|1[0-9][0-9])\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|1[0-9][0-9])\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|1[0-9][0-9]))$/
```

A regex usually uses two slash characters, e.g., `/abc/`, to delimit the search pattern. The caret (^) and the dollar sign (\$) anchors match the start and the end, respectively, of the string the regex pattern is applied to.

The first block is repeated three times, as indicated by the quantifier {3}. A valid IP address consists of four numbers from 0 to 255, separated with three periods. Each repetition validates one number from 0 to 255 with a period appended. The bracket expressions ([]) are replaced by one of the characters specified in the range. The following expressions are used:

- 25[0-5] matches text numbers (in text form) from 250 up to 255.
- 2[0-4][0-9] matches numbers (in text form) from 200 up to 249.
- 1[0-9][0-9] matches numbers (in text form) from 100 up to 199.
- [1-9][0-9] matches numbers (in text form) from 10 up to 99.
- [0-9] matches numbers (in text form) from 0 to 9.

The or (|) operator is used to include just one number of the previous expressions. With the quantifier {3}, three numbers are required, each followed by a period. Here's an example:

89.4.100.

For the fourth number, a period is not allowed. An example of matching text is as follows:

223

The total regex validates the IP addresses in this form:

89.4.100.223

Testing the JavaScript Form Validation

You'll now test the form validation in the HTML file `onlineservice.html` using JavaScript. In the address bar of your browser, enter the URL of the site, as shown here:

`http://webtoolsonline.servehttp.com`

Enter in the textbox an invalid IP address, for instance:

256.5.46.7

or the following:

1.2

Click the Go button, which submits the IP address to the web server. A web server program to receive the data is not defined yet, but you can still view the warning message the JavaScript creates for an invalid IP address.

A pop-up window with the message “Enter a valid IP address” appears, as displayed in Figure 4-17. In the message, the domain name of the server is included as christos.ddns.net:8080, which is the domain name that webtoolsonline.servehttp.com actually redirects to.

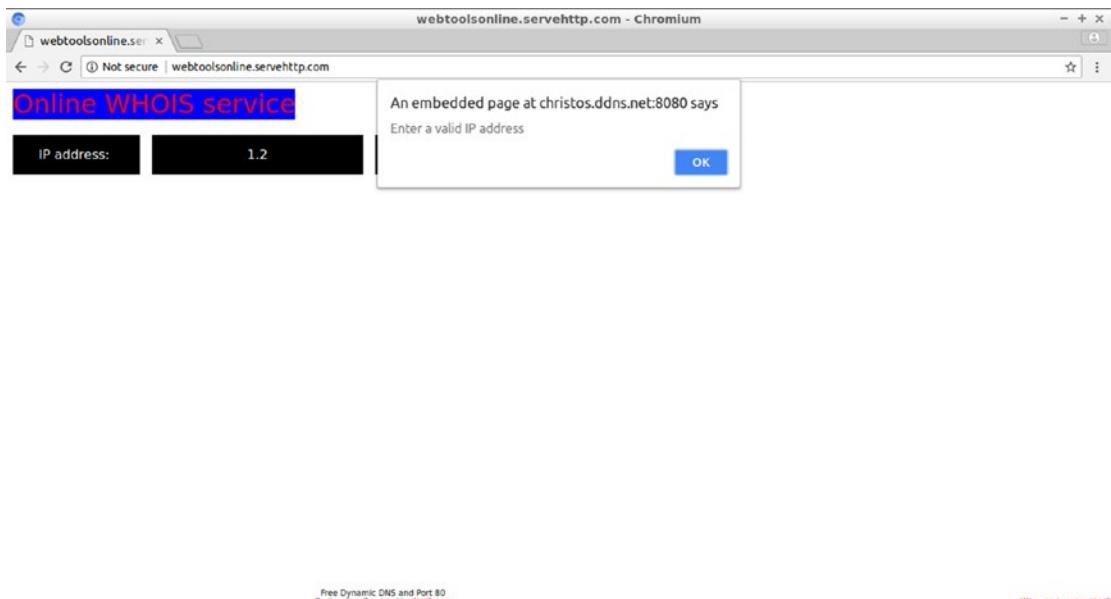


Figure 4-17. Testing the web page using an invalid IP address

Next you will create the action program for the form, `result.php`, which receives that IP address and uses it with the `whois` command-line program. But first you can try to run `whois` from the command line to find out how it outputs data and comments. The latter will be excluded from the output of the PHP program for the WHOIS service.

Running whois from the Command Line

The IP address entered by the user in the textbox of form f1 will be submitted to program result.php on the web server. The PHP engine that processes result.php will run the whois shell utility with this specific IP address as an argument. The output of whois will be entered in the web page the user receives from the echo PHP command.

Before using PHP to interface with the whois utility, try to run whois directly from the Linux terminal. To install whois at the command line, enter the following:

```
$ sudo apt install whois
```

Test whois with an IP address. You can try a random IP address, for instance:

```
$ whois 1.2.3.4
```

Figure 4-18 displays the whois output at the Linux terminal.

```
christos@pc:~$ whois 1.2.3.4
% [whois.apnic.net]
% Whois data copyright terms  http://www.apnic.net/db/dbcopyright.html

% Information related to '1.2.3.0 - 1.2.3.255'

% Abuse contact for '1.2.3.0 - 1.2.3.255' is 'abuse@apnic.net'

inetnum:      1.2.3.0 - 1.2.3.255
netname:      Debgogen-prefix
desc:         APNIC Debgogen Project
desc:         APNIC Pty Ltd
country:      AU
admin-c:      AR302-AP
tech-c:       AR302-AP
mnt-pw:       APNIC-HM
mnt-routes:   MAINT-AU-APNIC-GM85-AP
mnt-irt:      IRT-APNICRANDET-AU
status:       ASSIGNED PORTABLE
last-modified: 2011-09-22T03:42:30Z
source:       APNIC

irt:          IRT-APNICRANDET-AU
address:     PO Box 3646
address:     South Brisbane, QLD 4101
address:     Australia
e-mail:      abuse@apnic.net
abuse-mailbox: abuse@apnic.net
admin-c:      AR302-AP
tech-c:       AR302-AP
auth:        # Filtered
mnt-by:      MAINT-AU-APNIC-GM85-AP
last-modified: 2011-09-22T03:53:02Z
source:       APNIC

role:         APNIC RESEARCH
address:     PO Box 3646
address:     South Brisbane, QLD 4101
address:     Australia
country:     AU
```

Figure 4-18. The whois output with percent symbols indicating comments

Notice that some lines include comments starting with the percent (%) symbol. In the evaluated result.php, it is probably a good idea to remove those comments and make the output appear more professional.

Try another IP address, for instance 4.3.2.1:

```
$ whois 4.3.2.1
```

Figure 4-19 displays the whois output for the second IP address.

```
christos@pc:~
```

```
christos@pc:~$ whois 4.3.2.1

# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/whois_tou.html
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/whois_reporting/index.html
#

# start

NetRange: 4.0.0.0 - 4.255.255.255
CIDR: 4.0.0.0/8
NetName: LVL-T-ORG-4-8
NetHandle: NET-4-0-0-0-1
Parent: ()
NetType: Direct Allocation
OriginAS:
Organization: Level 3 Parent, LLC (LPL-141)
RegDate: 1992-12-01
Updated: 2018-02-29
Ref: https://whois.arin.net/rest/net/NET-4-0-0-0-1

OrgName: Level 3 Parent, LLC
OrgId: LPL-141
Address: 100 CenturyLink Drive
City: Monroe
StateProv: LA
PostalCode: 71203
Country: US
RegDate: 2018-02-06
Updated: 2018-02-22
Ref: https://whois.arin.net/rest/org/LPL-141
```

Figure 4-19. The whois output with hash symbols indicating comments

Notice in this case that the hash symbol is also used to indicate the start of the comments. Now it's time to create `result.php`, the program that receives the IP address entered by the user, processes it with the `whois` command, and returns to the user the output with the comments excluded.

Editing the File That Processes the User Data

You create next `result.php` in the document root (`/var/www/html`) of the vhost used.

At the Linux terminal, enter the following:

```
$ sudo gedit /var/www/html/result.php
```

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

Enter the following lines and save the file:

```
<!DOCTYPE html>
<html>
<body>
<span style="color:red;text-align:left;font-size:32px;background-
color:blue;float:left">
Online WHOIS Service
</span>
<span style="color:yellow;text-align:right;font-size:32px;background-
color:blue;float:right">
<?php echo "Your IP address: " . $_SERVER['REMOTE_ADDR']; ?>
</span>
<?php
echo '<h1 style="color:blue;text-align:center"> whois ' . $_POST['user_
data'] . '</h1>';
echo '<p style="font-family:monospace;font-size:16px;background-
color:black;color:white">';
$user_data = $_POST['user_data'];
if (isset($user_data)) {
    $exec_string = "whois $user_data";
    $output = shell_exec($exec_string);
    $stream = fopen('data://text/plain,' . $output, 'r');
    while(! feof($stream)) {
        $x = fgets($stream);
        if((strpos(trim($x), '%') === 0) || (strpos(trim($x), '#') === 0))
            continue;
        echo $x;
        echo "<br>";
    }
}
echo '</p>';
?>
</body>
</html>
```

In the `result.php` source code, the global PHP variable `$_SERVER`, set by the PHP engine, includes the element `$_SERVER['REMOTE_ADDR']`, which holds the visitor's IP address. This information is not required by the WHOIS service but could be used as additional info for users who view your site.

With the following source code, the IP address entered by the user in the `user_data` textbox in the file `onlineservice.html` is now received by the PHP engine as `$_POST['user_data']`. This IP address is part of the message displayed when `result.php` is evaluated. For instance, for the IP address 1.2.3.4, the message is `whois 1.2.3.4`, which indicates the query `whois` performed. This information is included as a header, because it consists of the content of an `h1` HTML element.

```
echo '<h1 style="color:blue;text-align:center"> whois ' . $_POST['user_data'] . '</h1>';
```

The `$_POST['user_data']` value is assigned then to the PHP variable `$user_data`.

The variable `$exec_string` is set to the string that represents the command that will be executed from the terminal at the web server. It is set to `whois $user_data`. This evaluates to `whois <IP address>`, where `<IP address>` is the specific value provided by the user, e.g., 87.202.116.192.

Hint! To implement a different service, you could change the command assigned as a value to `$exec_string`. For instance, for the ping service, you could use the following:

```
$exec_string = "ping -c 3 $user_data";
```

This sends three ICMP echo messages to the destination IP address and receives the reply. ICMP stands for Internet Control Message Protocol. It is part of the TCP/IP protocol suite and is used for diagnostic purposes.

The command defined in `exec_string` is passed to the Linux terminal with the function `shell_exec()`.

```
$output = shell_exec($exec_string);
```

The value stored in `shell_exec` is the output of the `whois` command that you ran previously at the terminal. The output of the command executed is returned to the `$output` variable. This is a long string that usually includes multiple lines. To process this string, one approach is to store it in a file. Another approach is to process it on the fly using the notion of a Unix stream. A stream treats a string like a file. To access the `$output` string as a stream, for a read (`r`) operation, the file descriptor `$stream` is used in the following command:

```
$stream = fopen('data://text/plain,' . $output, 'r');
```

For this stream, the data wrapper specifies the stream type, and the `text/plain` target specifies the stream format.

The following `while` loop is used to read one line at a time of the `$stream` and output all noncomment lines:

```
while(!feof($stream)) {
    $x = fgets($stream);
    if((strpos(trim($x), '%') === 0) || (strpos(trim($x), '#') === 0))
        continue;
    echo $x;
    echo "<br>";
}
```

The `while` condition, `! feof($stream)`, is true, while the end of the stream is not reached. Each line of the stream is returned in each `while` iteration by the `fgets()` function to variable `$x`.

```
$x = fgets($stream);
```

The following command ignores all the lines starting with the percent (%) or the hash (#) symbol:

```
if((strpos(trim($x), '%') === 0) || (strpos(trim($x), '#') === 0))
    continue;
```

The function `trim()` is used to remove the characters specified in its second argument. Since in this example no second argument exists, it returns a string stripped of any space, tab, or newline characters at the start and the end of the string. This is the corresponding PHP function to the JavaScript `trim()` method used previously in file `onlineservice.html`.

The two `strpos()` functions receive the returned string and check whether the character at position 0 (first position) is either the percent or the hash character. If this is the case, the command `continue` executes, and as a result, the `while` loop proceeds without executing the last two `while` commands in the current iteration. If the current line starts with the percent or hash character, this line is therefore ignored.

Hint! To check the result of the functions `trim()` and `strpos()` and of other string functions of PHP, use the PHP CLI utility. CLI stands for command-line interface, and if this program is not installed on your system, install it now by using this:

```
$ sudo apt-get install php7.0-cli
```

In the current directory, create a PHP file, for instance `test.php`.

```
$ gedit test.php
```

Enter the following code snippet and save the file:

```
<?php  
$str = "\n\n\t\tHello World!";  
echo $str;  
echo "\nPosition of character H: ";  
echo strpos($str, 'H');  
echo "\n";  
$str2 = trim($str);  
echo "$str2";  
echo "\nPosition of character H: ";  
echo strpos($str2, 'H');  
?>
```

The value of variable \$str is prepended with two newline characters (\n), which have the effect of pressing the Enter key of the keyboard, and two tab characters (\t), which have the effect of pressing the Tab key. The first strpos() call returns the position of character H in the string. The return value is 5 because character H is in the sixth position, which is position 5 starting counting from 0. Next, trim() is used to strip the tab and newline characters at the start of the string. The second strpos() therefore returns 0 because now H is the first character.

To run the program, use the following:

```
$ php test.php
```

The output in the Linux terminal is as follows:

```
christos@pc:~$ php test.php
```

```
    Hello World!
```

```
Position of character H: 5
```

```
Hello World!
```

```
Position of character H: 0
```

If the current line is not a comment, it is printed with the echo command, and also a break tag is used to change the line.

```
echo $x;  
echo "<br>";
```

All not-commented lines are thus included in the output generated by the PHP engine and returned to the user from the web server.

Testing the WHOIS Online Service

You can test next your online WHOIS service at webtoolsonline.servehttp.com. Enter one of the IP addresses previously tested with the whois command from the terminal, e.g., 1.2.3.4, as viewed in Figure 4-20.

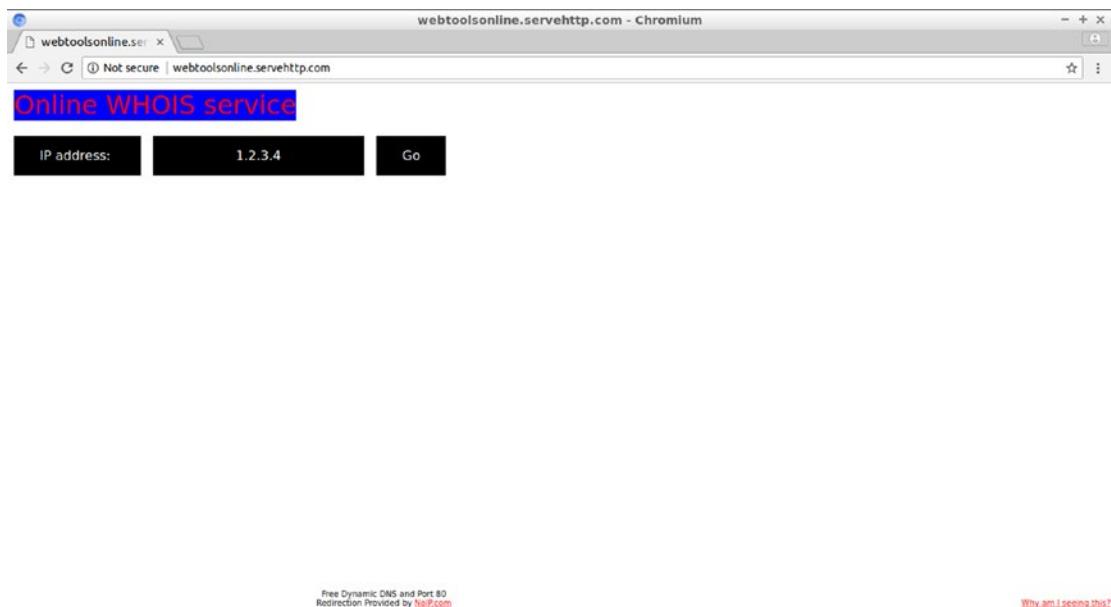


Figure 4-20. Testing a valid IP address with the online service

Click the Go button to submit the IP address to the web server.

The result.php evaluated web page for the IP address 1.2.3.4 is displayed in Figure 4-21.

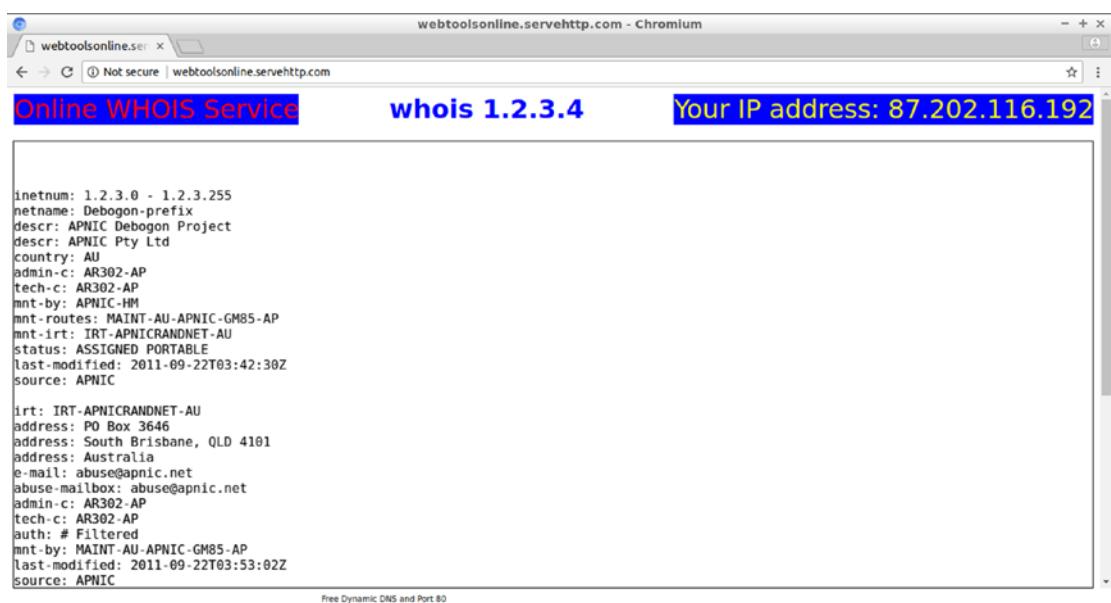


Figure 4-21. The online WHOIS service output

You will notice that the style of this web page resembles the Linux terminal. At the top of the web page, the message `whois 1.2.3.4` appears in the center, and the public IP address of the client is displayed on the right. The output differs from the `whois` terminal command because it lacks comment lines.

In the next section, you can add a favicon image to the web site to make the site instantly recognizable.

Adding a Favorite Icon to the Site

A favorite icon, referred as a *favicon* in its filename, is the tiny icon you see on the left of your browser tab when a web page loads. It makes a site recognizable because you can easily locate it when lots of tabs are open in the browser. Even before the web page loads, the favicon icon appears in the address bar of the browser, next to the address, which is completed with the autofill feature. This ensures that the address you are about to enter is the correct one. Favicons appear also in the bookmark list, where the URLs of your favorite sites are saved.

The first step of including a favicon image is to design it. In the Linux locate an image editor program or download and install one if necessary. The example shown here uses the mtPaint editor, which is the default image editor on the Linux system I'm using. Open mtPaint, and from the File menu select New. Favicons require a size of 16×16 pixels; therefore, in the dialog that appears (Figure 4-22), set the dimensions of the new image to 16 pixels width by 16 pixels height.

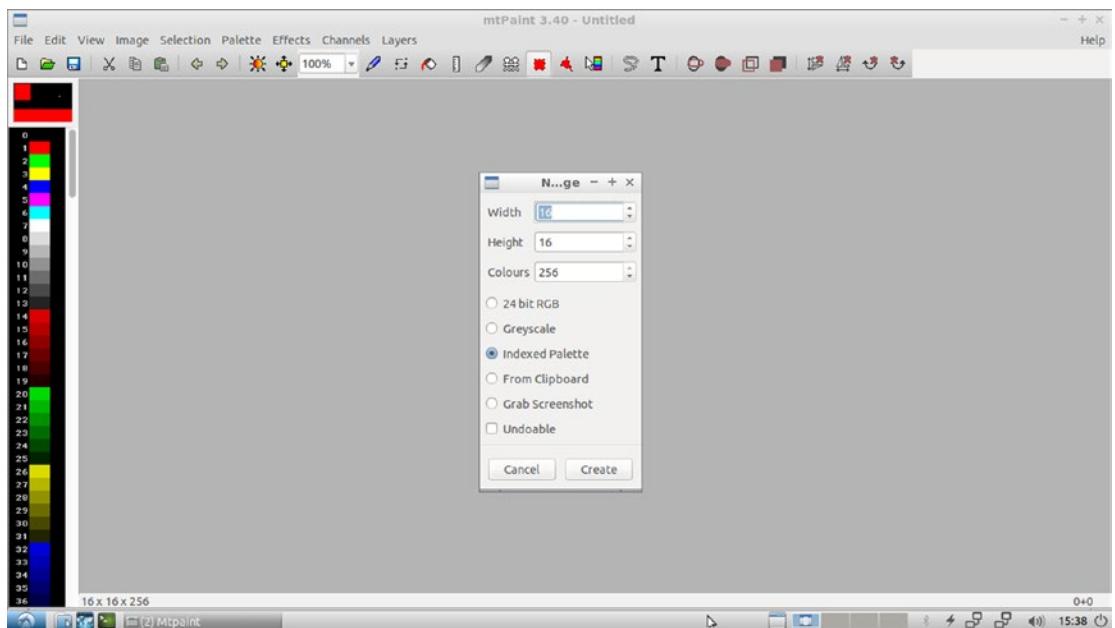


Figure 4-22. Setting the dimensions of the favicon icon to 16x16 pixels

mtPaint displays a black-colored canvas, sized 16×16 pixels, for the new image (Figure 4-23).

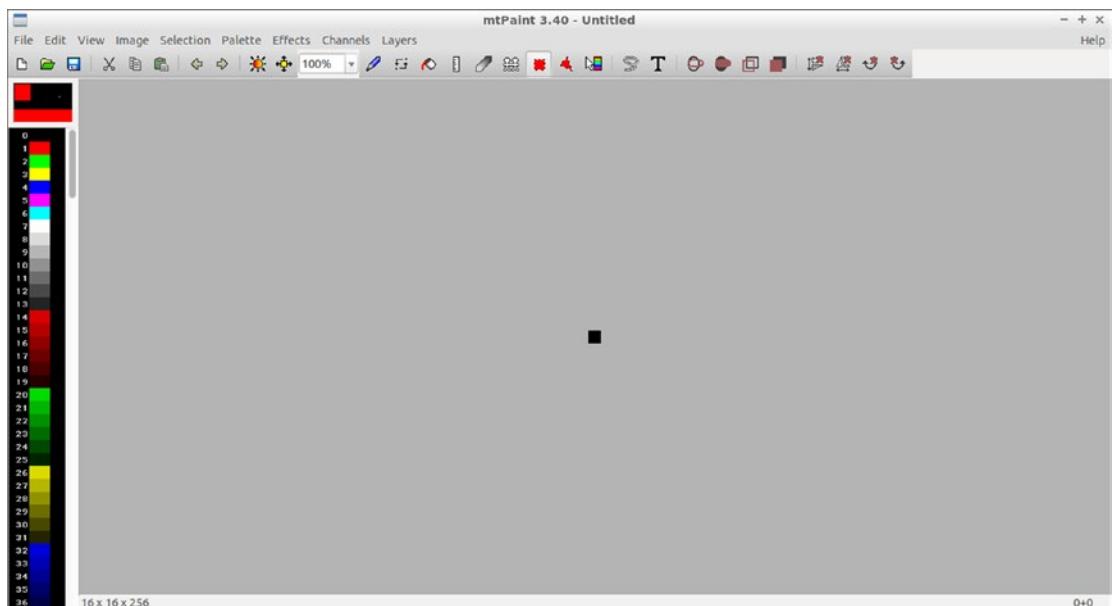


Figure 4-23. The tiny favicon image with a 16×16 pixel size

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

This is a tiny canvas to work with, so zoom in to enlarge it. To do this, in the zoom percentage drop-down, use a bigger zoom than 100%, for instance 1200% (Figure 4-24).

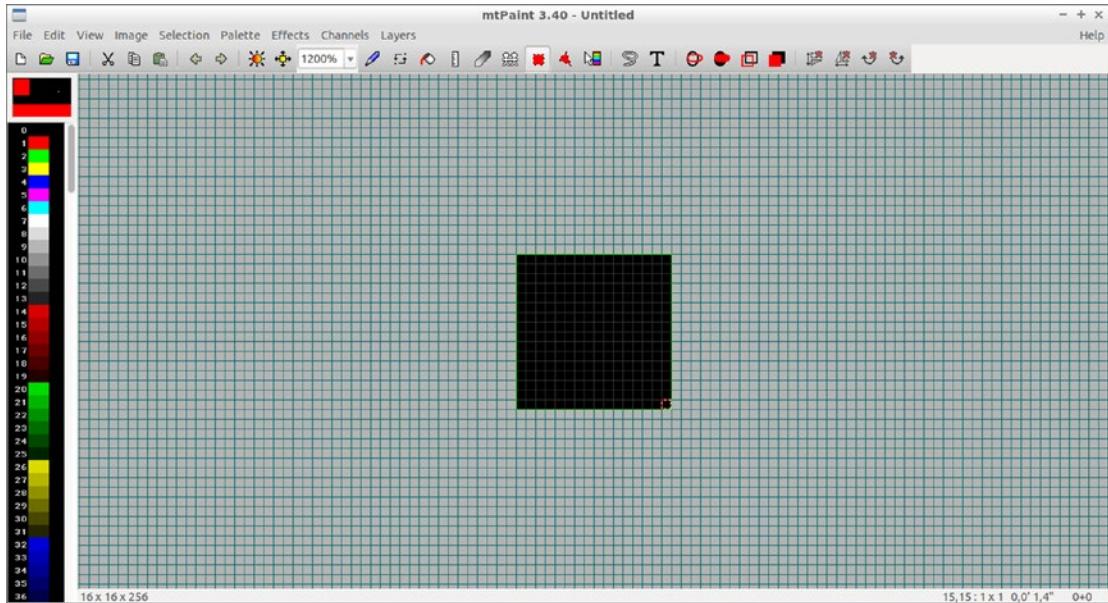


Figure 4-24. Zooming in on the image for detailed drawing

You can start by applying your ideas to the canvas. For instances, you can click the Flood Fill button (the paint bucket), select a white color (e.g., color 7 from the color list on the left), and click next any pixel of the canvas to turn the background of your image to white. Using the Paint button (the blue pencil), select a color from the list and click any pixel to color it. In Figure 4-25, the blue and red colors are being used. The image displayed includes two letters, OT, for “online tools.” The letter *T* strives to look like a hammer.

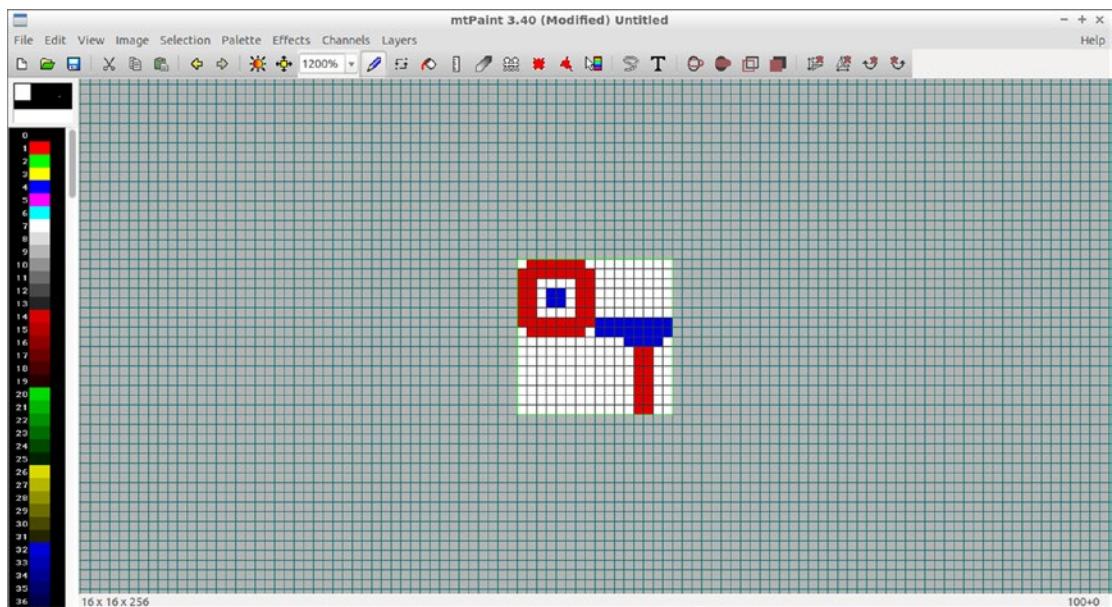


Figure 4-25. The favicon image completed

From the File menu, choose Save As to save the image to one of your directories, e.g., Pictures, and in the dialog that appears, provide the Pictures file path and favicon file name. Select also PNG as the image format in the File Format list and click the OK button.

Although you can use the PNG file type for the favicon files, the default one used for the whole site defaults to the name `favicon.ico`. The ICO file type is not supported by mtPaint. You can, however, change the file type from the terminal using the `convert` command included in the `imagemagick` package. Install `imagemagick` using the following:

```
$ sudo apt-get install imagemagick
```

Change then the file type using:

```
$ convert -background transparent "favicon.png" -define icon:auto-
resize=16,24,32,48,64,72,96,128,256 "favicon.ico"
```

Copy next the `favicon.ico` file to the document root:

```
$ sudo cp ~/Pictures/favicon.png /var/www/html
```

Test your site next. As viewed in Figure 4-26, the favicon image appears on the left of the web page tab in your browser.

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS

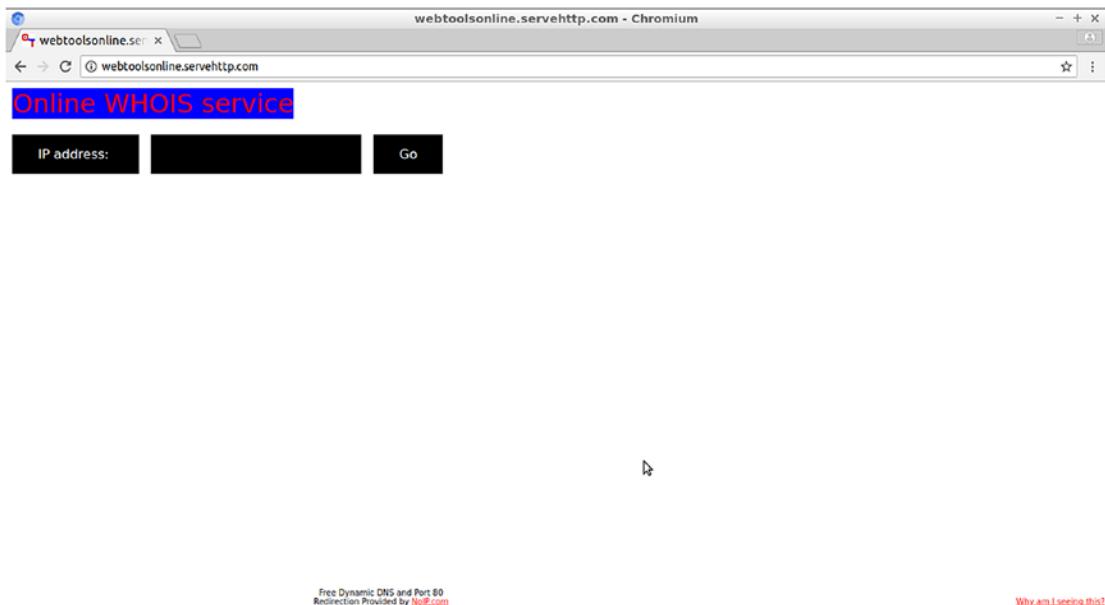


Figure 4-26. The site's home page includes a favicon image on the browser's tab

To use a different favicon image for another web page of your site, make explicit reference from the web page to another favicon image. Create a second favicon file, `favicon2.png`, that includes for simplicity just a yellow background, and copy it to the document root directory. Edit a web page, for instance the `result.php` file, and add in the head section a link reference to `favicon2.png`. Specifically, in the head section of the HTML source code, enter the following tag:

```
<link rel="icon" href="favicon2.png"/>
```

Because this is a specific reference where the file name is provided, the file type used can be of a different type than ICO; e.g., in this example PNG is used. The source code of `result.php` becomes the following:

```
<!DOCTYPE html>
<html>
<head>
<link rel="icon" href="favicon2.png"/>
</head>
<body>
```

```

<span style="color:red;text-align:left;font-size:32px;background-
color:blue;float:left">
Online WHOIS Service
</span>
<span style="color:yellow;text-align:right;font-size:32px;background-
color:blue;float:right">
<?php echo "Your IP address: " . $_SERVER['REMOTE_ADDR']; ?>
</span>
<?php
echo '<h1 style="color:blue;text-align:center"> whois ' . $_POST['user_
data'] . '</h1>';
echo '<p style="font-family:monospace;font-size:16px;background-
color:black;color:white">';
$user_data = $_POST['user_data'];
if (isset($user_data)) {
    $exec_string = "whois $user_data";
    $output = shell_exec($exec_string);
    $stream = fopen('data://text/plain,' . $output, 'r');
    while(! feof($stream)) {
        $x = fgets($stream);
        if((strpos(trim($x), '%') === 0) || (strpos(trim($x), '#') === 0))
            continue;
        echo $x;
        echo "<br>";
    }
}
echo '</p>';
?>
</body>
</html>

```

Test the `onlineservice.html` file using, for instance, the following URL:

`127.0.0.1:8080.`

As shown in Figure 4-27, the home page of the site loads with the default favicon image.

CHAPTER 4 OBTAINING A DOMAIN NAME WITH DDNS



Figure 4-27. The home page of the site loads with the default favicon image

Enter in the form textbox an IP address and click the Go button. The result.php web page is displayed in your browser. As shown in Figure 4-28, result.php includes the yellow favicon2.png icon on the left of its tab.

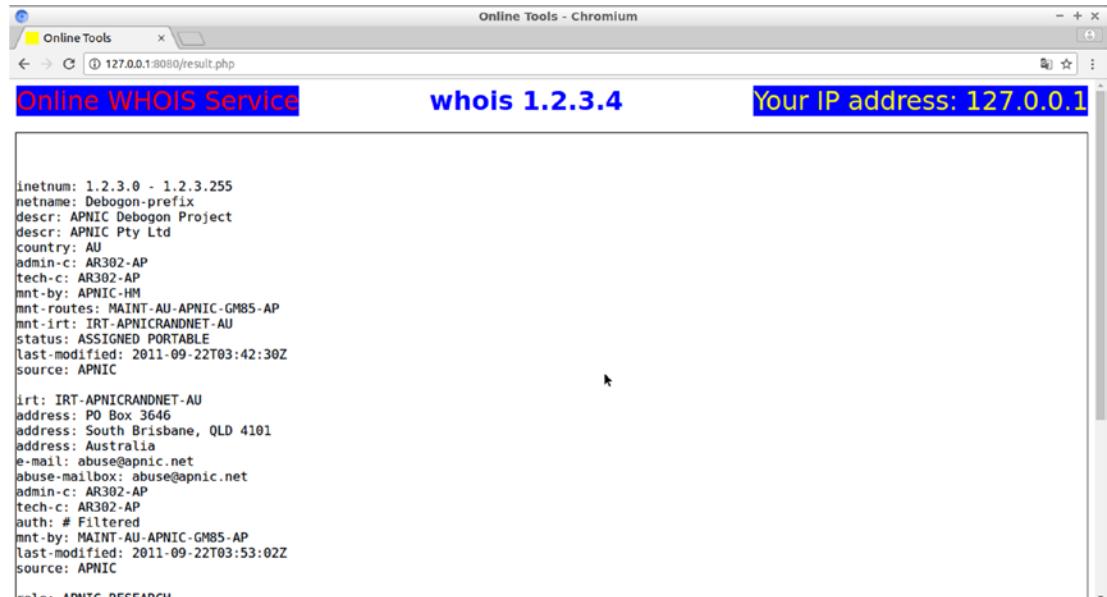


Figure 4-28. The web page result.php uses a different favicon image

You can create a favicon icon in a few steps to provide an image that personalizes your site and even become the motivation for an inspired logo for the web content of the site.

Summary

In this chapter, you did the following:

- You used the DDNS service to provide a user-friendly domain name to your site.
- You created a web project with an online web service, specifically WHOIS.
- You created a `favicon.ico` image for your site.

In the next chapter, the source code of another web service is displayed, which creates a site that allows web server administrators and web designers find how their site looks from a remote location. But first you are introduced to another popular open source web server, the Lighttpd server.

CHAPTER 5

The Lighttpd Web Server

Most of the web browsers used today have the same common features; therefore, it's easy for users to make the transition from using one browser to another. This is an advantage that it is not found on the server side. Web servers are specialized programs, not expected to be used by the average user. Therefore, the graphical user interface (GUI), if the web server is supplied with one, or the commands used at the terminal for configuring and running the server may be completely different from one server to another. In this chapter, you will have the chance to use another open source web server, Lighttpd (called "Lighty"). You will run the server and identify characteristics that you used previously in Apache, like the document root and the virtual servers. Experimenting with a second server will make it easier for you to transition to a third one.

You will then use Lighttpd with PHP to create your next online web service, which will simulate a web service like webpagetest.org that tests the appearance of your site remotely; you used webpagetest.org in Chapter 3.

Installing Lighttpd

Running two web servers at the same time is possible but not recommended because if the servers are not carefully configured (for instance, bound to different port numbers), this might lead to conflicts and unpredictable results. To stop the apache2 process and leave the field free for Lighttpd, use the following:

```
$ sudo service apache2 stop
```

This stops Apache until you shut down and reboot your computer. To avoid restarting the Apache process the next time your computer starts, use the following:

```
$ sudo systemctl disable apache2
```

You don't have to worry about your saved work so far—all the files saved in the document root directory (e.g., /var/www/html) remain there.

To install Lighttpd from the terminal, use the following commands:

```
$ sudo apt-get update  
$ sudo apt-get install lighttpd
```

Testing Lighttpd

The default document root for Lighttpd is /var/www/html, which is the same directory as with Apache's document root. Notice that for some Ubuntu distributions the default document root is /var/www. You can change the default document root from the Lighttpd configuration file, as discussed in the following section. A directory index with the name index.lighttpd.html was already created in the document root. Enter this file in your browser's address bar using the default loopback address, 127.0.0.1, or the corresponding hostname localhost, as shown here:

```
localhost/index.lighttpd.html
```

Although index.lighttpd.html is a directory index, including the file name in the URL path is required because this is the third directory index name included in the configuration file, discussed in the following section. The files index.php and index.html, included first in the list, take precedence and are displayed instead.

The web page index.lighttpd.html is the test page for the Lighttpd web server. As shown in Figure 5-1, it provides information about the basic Lighttpd configuration.

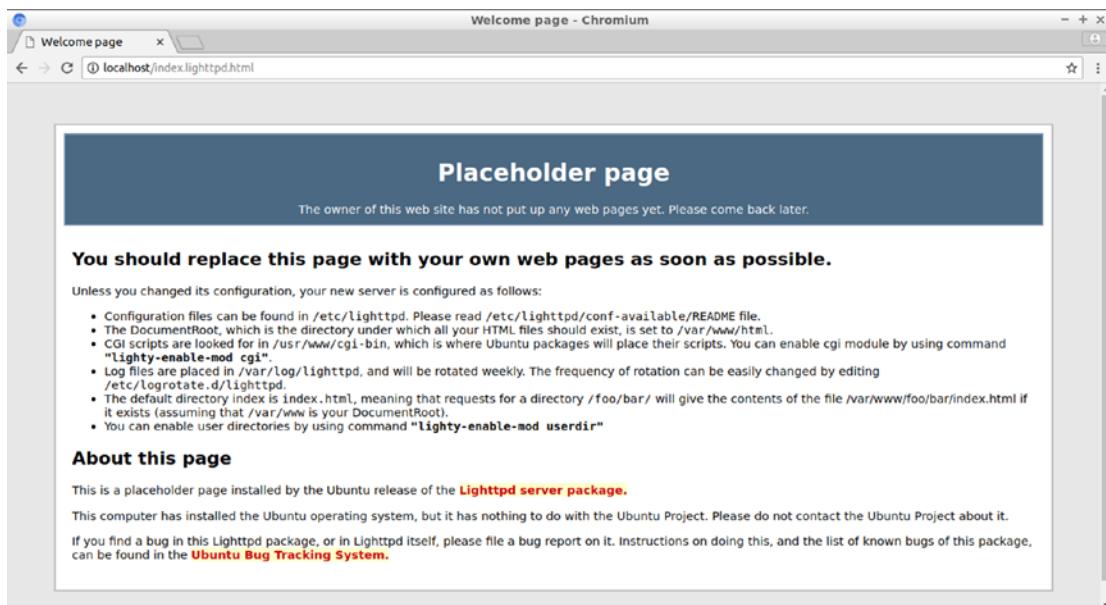


Figure 5-1. The test page of the Lighttpd web server

If you see a file with the name `index.html`, created when using Apache, in the document root, rename it as follows:

```
$ sudo mv index.html index.htmlOLD
```

Create a new directory `index` for the Lighttpd server in the document root `index.html`.

```
$ cd /var/www/html
$ sudo gedit index.html
```

Enter a simple HTML source code, like the following, and save the file:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
Hello World!
</body>
</html>
```

Then test your new directory index using the localhost hostname or the IP address 127.0.0.1 in the browser's address bar. Figure 5-2 displays the directory index dispatched by Lighttpd.

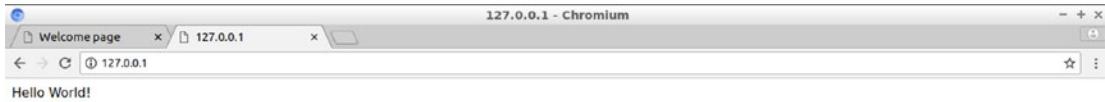


Figure 5-2. The directory index viewed from a browser locally

Try also to use in the URL the private IP address of the web server, which is 192.168.1.100 for the examples used in this text.

Figure 5-3 displays the directory index viewed from another computer in the same LAN with the web server.



Figure 5-3. The directory index viewed from a browser on another computer in the web server LAN

To view the web page from outside the local LAN, you have to use either the public IP address of the router or one of the fully qualified domain names obtained in Chapter 4. In the following section, you will make the required changes to the Lighttpd configuration file to test your web server externally from your LAN.

Working in the Lighttpd Configuration File

The Lighttpd configuration file is `lighttpd.conf`, located in the directory `/etc/lighttpd`.

Before starting to modify the configuration file, make a copy of the original one with the following commands:

```
$ cd /etc/lighttpd/
$ sudo cp lighttpd.conf lighttpd.conf.bak
```

Edit `lighttpd.conf` with a text editor, for instance `gedit`.

```
$ sudo gedit lighttpd.conf
```

The file contents are as follows:

```
server.modules = (
    "mod_access",
    "mod_alias",
    "mod_compress",
    "mod_redirect",
)

server.document-root      = "/var/www/html"
server.upload-dirs        = ( "/var/cache/lighttpd/uploads" )
server.errorlog           = "/var/log/lighttpd/error.log"
server.pid-file           = "/var/run/lighttpd.pid"
server.username            = "www-data"
server.groupname           = "www-data"
server.port                = 80

index-file.names          = ( "index.php", "index.html", "index.lighttpd.
                               html" )
url.access-deny           = ( "~", ".inc" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

compress.cache-dir         = "/var/cache/lighttpd/compress/"
compress.filetype          = ( "application/javascript", "text/css",
                               "text/html", "text/plain" )

# default listening port for IPv6 falls back to the IPv4 port
## Use ipv6 if available
#include_shell "/usr/share/lighttpd/use-ipv6.pl" + server.port
include_shell "/usr/share/lighttpd/create-mime.assign.pl"
include_shell "/usr/share/lighttpd/include-conf-enabled.pl"
```

The Lighttpd directives follow object-oriented programming principles. The first part is the object that is referred to, and then after a dot the specific attribute of this object is indicated. The attribute is then assigned a value using the equal (=) sign.

At the start of the configuration file, you will see the `server.modules` directive, which is a list with the modules of the web server to be loaded. Other directives follow. Here are some examples:

- `server.errorlog`, the location of the error log file.
- `server.pid-file`, the file that stores the process ID (PID) of the Lighttpd server. This is the PID that you can find using the command `ps xa | grep lighttpd`.
- `server.username`, the username of the user the Lighttpd server assumes.
- `server-groupname`, the group the Lighttpd user belongs to.
- `server.document-root`, which defines the document root.

Applying a Basic Configuration

If in your Lighttpd version the document root directive (`server.document-root`) is set to `/var/www`, you can change this to `/var/www/html` so you can test the book's sites from both Lighttpd and Apache without modification.

If you use the default HTTP port number (80) for your server, you can use the server's configuration file as is. If, on the other hand, your ISP blocks the inbound port 80 and you utilize another port, e.g., 8080, or if you plan to implement the HTTPS protocol (more about this in Chapter 8) that uses port 443, you can change the default `server.port` value.

```
server.port          = 80
```

In the previous section, you viewed the directory index `index.lighttpd.html`. This is included third in the directory's indexes list, indicated by the `index-file.names` directive.

```
index-file.names      = ( "index.php", "index.html", "index.lighttpd.html" )
```

This directive assigns the list of files used as directory indexes for the web server. From this directive, you can change the order of the filenames used. To enable any changes in the configuration file, click the Save button in the gedit window and reload the web server as follows:

```
$ sudo service lighttpd force-reload
```

With the configuration file updated and the Linux firewall already set from Chapter 1, you can connect to the web server from any location on the Internet. Use the public IP address of your router in your browser's address bar. This time retrieve your external IP address by using the following command:

```
$ curl ifconfig.co
```

The output in this example is:

87.202.116.97

You can also use a fully qualified domain name, like the ones created in Chapter 4, in the URL of your site. Here's an example:

webtoolsonline.servehttp.com

Hint! To ensure that your site runs as expected from any Internet-connected computer, you have to test the web server externally from your LAN, by using your mobile Internet connection or an online web service like webpagetest.org. Actually, the web project you will develop in this chapter creates a similar web service.

Binding to a Specific IP Address or Hostname

With the bind directive, you can restrict the server to a specific IP address or hostname, where the web server listens for client requests, for instance:

```
server.bind = "87.202.116.97"
```

87.202.116.97 is the public IP address of the router used in this example. To implement this directive, insert the previous directive in the configuration file, save the file, and reload the server using the following:

```
$ sudo service lighttpd force-reload
```

In the following example, it is assumed that the server listens on port 8080. Use the following IP address and port number combination in the browser's address bar:

87.202.116.97:8080

The web page is loaded as usual. Use the following IP address and port number combination:

127.0.0.1:8080

The web page does not load anymore. Comment out the previous directive in the configuration file by prepending it with the hash symbol.

```
#server.bind = "87.202.116.97"
```

Next enable the new configuration.

```
$ sudo service lighttpd force-reload
```

Use the IP address and port number combination one more time in the browser's address bar.

127.0.0.1:8080

The web page loads again normally.

You can also test the `server.bind` directive with domain names. One of the two domain names created in Chapter 4 is the following:

`christos.ddns.net`

To bind the web server to this domain name, uncomment the `server.bind` directive and replace the IP address in its value with the FQDN `christos.ddns.net`.

```
server.bind = "christos.ddns.net"
```

Save the configuration file and reload the server using this command:

```
$ sudo service lighttpd force-reload
```

Test the domain name from the browser. Using `christos.ddns.net`, it loads as usual. If you try the public IP address of the router, e.g., 87.202.116.97, in the address bar, you'll see it does not load anymore. Comment out the `server.bind` directive and reload the server to bring its configuration to the previous state.

Changing the Document Root

The default document root as set in the configuration file is `/var/www/html`. Try to set another one, for instance `/var/www/html2`. First you have to create the new `html2` directory in `www`.

```
$ sudo mkdir /var/www/html2
```

Create a new directory index called `index.html` in `html2`.

```
$ sudo gedit /var/www/html2/index.html
```

Enter the following source code and save the file:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
This is file /var/www/html2/index.html
</body>
</html>
```

Next edit the configuration file.

```
$ sudo gedit /etc/lighttpd/lighttpd.conf
```

Change the value of the `server.document-root` directive from `/var/www/html` to `/var/www/html2`. Save the file and reload the server.

```
$ sudo service lighttpd force-reload
```

Test locally the new document root using the port the server listens on, either the following:

`localhost`

or the following:

localhost:8080

The new web page is loaded.

Restore the document root to /var/www/html to set the web server to the previous state.

Enabling and Disabling the Directory Listing

To allow the user to view the contents of a directory, when no directory index is present in this directory, you can set the `dir-listing.activate` directive. By default the directory listing is disabled. To test this, create directory `dir1` under the document root, and inside `dir1` create two files, for instance `file1` and `file2`, using the following commands at the terminal:

```
$ sudo gedit mkdir /var/www/html/dir1  
$ cd /var/www/html/dir1  
$ sudo touch file1 file2
```

Enter the following URL in your browser address bar:

localhost/dir1

or if you utilize port 8080:

localhost:8080/dir1

A web page with the HTTP error message 403 appears, as displayed in Figure 5-4.



Figure 5-4. The HTTP 403 error status code displayed using the URL of a directory with no directory index and the directory listing disabled

Open the configuration file for editing.

```
$ sudo gedit /etc/lighttpd/lighttpd.conf
```

Enter the following line:

```
dir-listing.activate = "enable"
```

Save the configuration file and reload the server.

```
$ sudo service lighttpd force-reload
```

Figure 5-5 shows the directory contents, and the users can download the files included in this directory by clicking the file names.

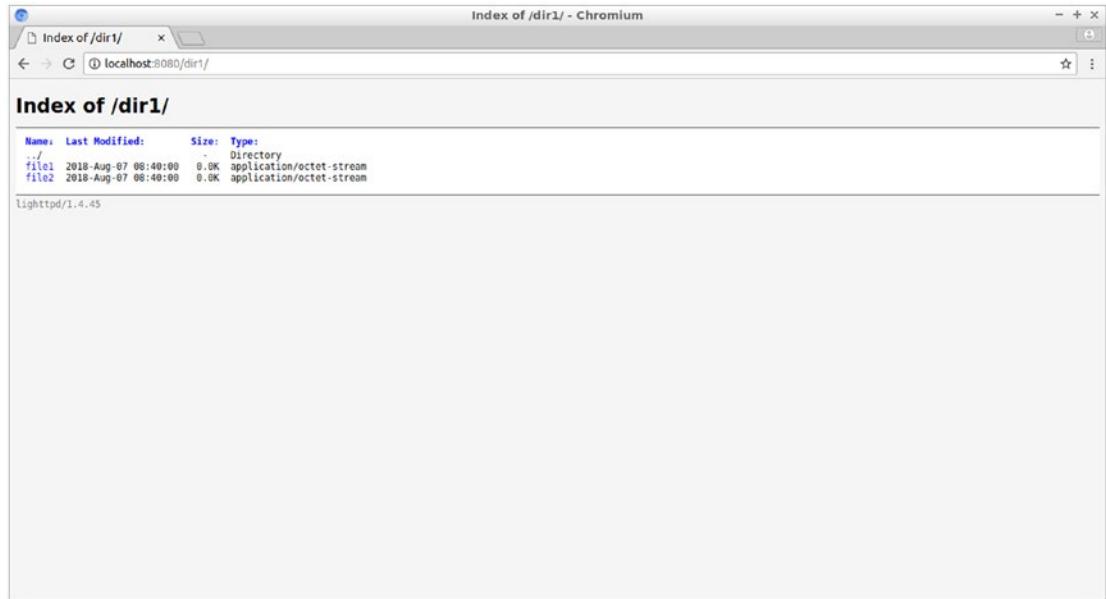


Figure 5-5. The directory contents listed using the URL of a directory lacking a directory index when the directory listing is enabled

To restore the `dir-listing.activate` directive, switch its value in the configuration file from `enable` to `disable`, save the configuration file, and reload the web server.

Sending Custom-Made Error Replies to the Client

In the previous section, the web server sends the default message for the HTTP error status code 403. This is the message:

403 – Forbidden

To customize the content of the error message, open the configuration file for editing.

```
$ sudo gedit /etc/lighttpd/lighttpd.conf
```

With the following line, set the `server.errorfile-prefix` directive and save the file:

```
server.errorfile-prefix = "/srv/www/errors/status-"
```

Reload the web server.

```
$ sudo service lighttpd force-reload
```

Create the directory `www/errors` in `/srv`.

```
$ sudo mkdir -p /srv/www/errors
```

Here, the `-p` option creates parent directories (`www` in this case).

Create next in the `errors` directory your customized status code web pages with the format:`<errorfile-prefix><status-code>.html`

`<errorfile-prefix>` is the value of the `server.errorfile-prefix` directive, and `<status-code>` is the specific error status code. For instance, for the HTTP error status code 403, create the file `/srv/www/errors/status-403.html`.

```
$ sudo gedit /srv/www/errors/status-403.html
```

Enter the following source code:

```
<!DOCTYPE html>
<html>
<head>
<style>
p{
color:red;

```

```
font-size:120px;
text-align:center;
text-decoration: underline overline;
}
img {
    display: block;
    margin-left: auto;
    margin-right: auto;
}
.top-right {
    position: absolute;
    top: 8px;
    right: 16px;
}
.top-left {
    position: absolute;
    top: 8px;
    left: 16px;
}
</style>
</head>
<body>
<div class="top-left">
<p>Error</p>
</div>
<div>

</div>
<div class="top-right">
<p>403</p>
</div>
</body>
</html>
```

With the `dir-listing.activate` directive disabled, you can test the customized error page by using the URL of the directory `dir1`, created previously. Here's an example:

`http://webtoolsonline.servehttp.com/dir1`

Figure 5-6 displays the customized error page.



Figure 5-6. The custom error page for the 403 HTTP error status code

Accessing the Lighttpd Log Files

Create and use Lighttpd log files to collect information about the visitors of your site. For instance, create a file called `access.log` in the `/var/log/lighttpd/` directory with the following command:

```
$ sudo touch /var/log/lighttpd/access.log
```

Using the `chown` command, I set the owner of the file `access.log` to the Lighttpd process using the `chown` command. At this command I use: `www-data:www-data` the first `www-data` is the username, the Lighttpd process assumes the second `www-data` is the group this user belongs.

```
$ sudo chown www-data:www-data /var/log/lighttpd/access.log
```

Edit the Lighttpd configuration file next.

```
$ sudo gedit /etc/lighttpd/lighttpd.conf
```

Include `mod_accesslog` in the group of modules to be loaded. It should appear after `mod_alias` since the order of the modules is the order they will be executed in. Include also the `accesslog.filename` directive to set the file name and path of the log file.

```
accesslog.filename      = "/var/log/lighttpd/access.log"
```

The configuration file is now as follows:

```
server.modules = (
    "mod_access",
    "mod_alias",
    "mod_accesslog",
    "mod_compress",
    "mod_redirect",
)

server.document-root      = "/var/www/html"
server.upload-dirs        = ( "/var/cache/lighttpd/uploads" )
server.errorlog            = "/var/log/lighttpd/error.log"
server.pid-file           = "/var/run/lighttpd.pid"
server.username           = "www-data"
server.groupname          = "www-data"
server.port                = 8080
#server.bind               = "webtoolsonline.servehttp.com"
server.errorfile-prefix   = "/srv/www/errors/status-"

dir-listing.activate       = "disable"

accesslog.filename         = "/var/log/lighttpd/access.log"

index-file.names           = ( "index.php", "index.html", "index.lighttpd.
                               html" )
url.access-deny             = ( "~", ".inc" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )
```

CHAPTER 5 THE LIGHTTPD WEB SERVER

```
compress.cache-dir          = "/var/cache/lighttpd/compress/"
compress.filetype           = ( "application/javascript", "text/css",
                                "text/html", "text/plain" )

# default listening port for IPv6 falls back to the IPv4 port
## Use ipv6 if available
#include_shell "/usr/share/lighttpd/use-ipv6.pl " + server.port
#include_shell "/usr/share/lighttpd/create-mime.assign.pl"
#include_shell "/usr/share/lighttpd/include-conf-enabled.pl"
```

Reload the server to enable the changes in the configuration file.

```
$ sudo service lighttpd force-reload
```

To test the `access.log` file, first do a local client request, e.g., using `localhost:8080` from a browser on the web server, and then do another request from any other computer, external to your LAN. Probably the nearest computer is your mobile phone. Figure 5-7 displays a connection to [webtoolsonline.servehttp.com/index.html](http://olsonline.servehttp.com/index.html) from a Samsung Galaxy S3 using the Google Chrome browser.

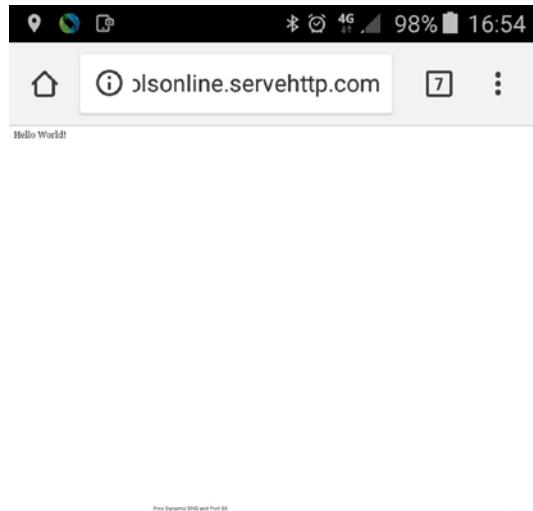


Figure 5-7. Connecting from a mobile phone to test access.log

Read the contents of `access.log` using the following commands:

```
$ sudo su
# cat /var/log/lighttpd/access.log
```

The output from the previous commands in this example is as follows:

```
127.0.0.1 localhost:8080 - [10/Aug/2018:08:29:18 +0300] "GET /favicon.ico HTTP/1.1" 404 345 "http://localhost:8080/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/67.0.3396.99 Chrome/67.0.3396.99 Safari/537.36"
66.249.81.241 christos.ddns.net:8080 - [10/Aug/2018:08:29:56 +0300]
"GET /index.html HTTP/1.1" 304 0 "http://webtoolsonline.servehttp.com/index.html" "Mozilla/5.0 (Linux; Android 5.1.1; SM-J320FN Build/LMY47V) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.70 Mobile Safari/537.36"
```

The first recording provides details for the local connection, issued from the web server computer, while the second provides details for the remote connection, issued from the mobile phone.

Using Virtual Hosts with Lighttpd

In Chapter 1 you used IP-based, port number-based, and name-based virtual hosts with Apache. Those features are also available with Lighttpd. The names used in the name-based Apache virtual host tests were hostnames, like `myserver.com`, that used only locally on the web server. In Chapter 4, you created FQDN hostnames like `christos.ddns.net` and `webtoolsonline.servehttp.com` that could be used universally, throughout the Internet. Next you can apply those DNS names to create two virtual hosts that run simultaneously on the web server, with each one dedicated to a completely different site. The names created in Chapter 4 are actually not different because with the DDNS configuration, the name `webtoolsonline.servehttp.com` redirects to `christos.ddns.net`.

This is your chance to create a third DDNS name using the steps for creating the other two. You can use the third name for the first one of the two virtual hosts. Log in to the DDNS server provider's web page, which is <https://www.noip.com> for the examples used in the book. Figure 5-8 shows the Dashboard web page.

CHAPTER 5 THE LIGHTTPD WEB SERVER

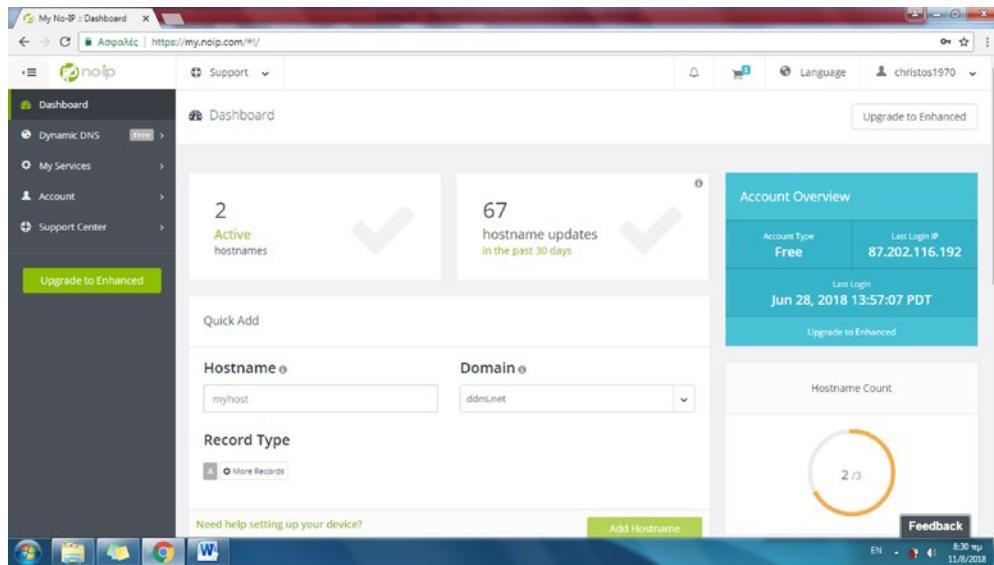


Figure 5-8. The Dashboard web page of no-IP

Enter the name of a server that is available in the Hostname field, for instance **secureserver**. In the Domain drop-down list, select a second-level domain name, e.g., **ddns.net**, as shown in Figure 5-9.

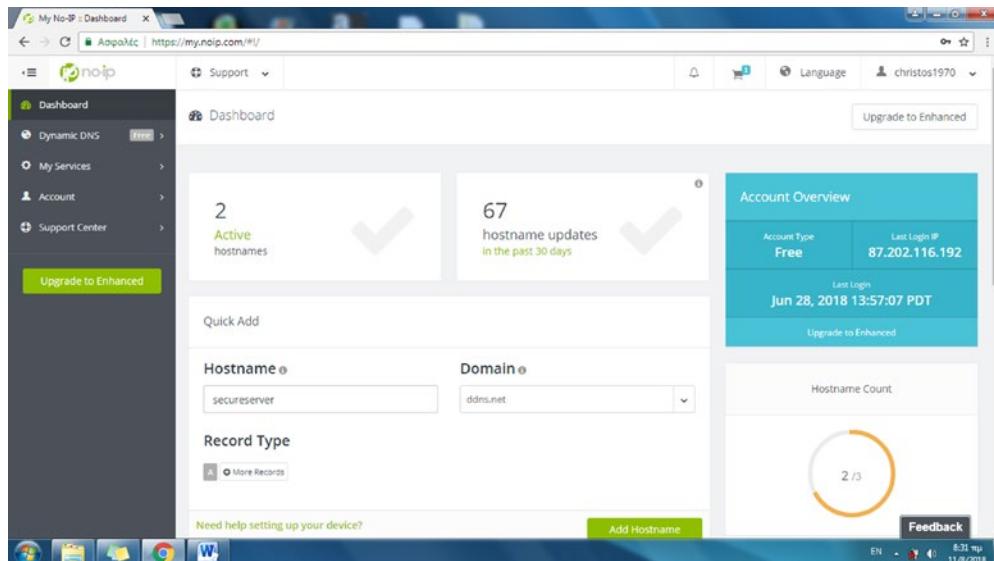


Figure 5-9. The Dashboard web page with the user settings for creating a new hostname

This name will also be used in Chapter 8, which is about cryptography and secure communication. Click the Add Hostname button to confirm your choice. Click also the Dynamic DNS menu on the left and then select the No-IP Hostnames option to view all the hostnames created so far. Figure 5-10 displays the Hostnames web page.

The screenshot shows the No-IP Hostnames web interface. On the left, a sidebar lists options like Dashboard, Dynamic DNS (selected), No-IP Hostnames, Personal Hostnames, Groups, Dynamic Update Client, Device Configuration Assistant, My Services, Account, and Support Center. A green 'Upgrade to Enhanced' button is at the bottom of the sidebar. The main area is titled 'Hostnames' and contains a table with three rows of data:

Hostname	Last Update	IP / Target	Type
christos.ddns.net Expires in 29 days	Aug 9, 2018 22:45 PDT	87.202.116.97	A
secureserver.ddns.net Expires in 29 days	Aug 10, 2018 22:31 PDT	87.202.116.97	A
webtoolsonline.servehttp.com Expires in 29 days	Aug 9, 2018 22:35 PDT	christos.ddns.net:8080	URL

To the right of the table, there's a 'Create Hostname' button, a 'Service Level' section (Free), and a 'Hostname Count' summary (3/3). A red circle highlights the '3/3' count. A 'Feedback' button is at the bottom right.

Figure 5-10. The Hostnames web page displays the hostnames already created

You can test your new hostname instantly by entering the following URL:

`secureserver.ddns.net`

or if you have configured the web server to listen on port 8080:

`secureserver.ddns.net:8080`

As displayed in Figure 5-11, the web page is loaded in your browser's window.

CHAPTER 5 THE LIGHTTPD WEB SERVER

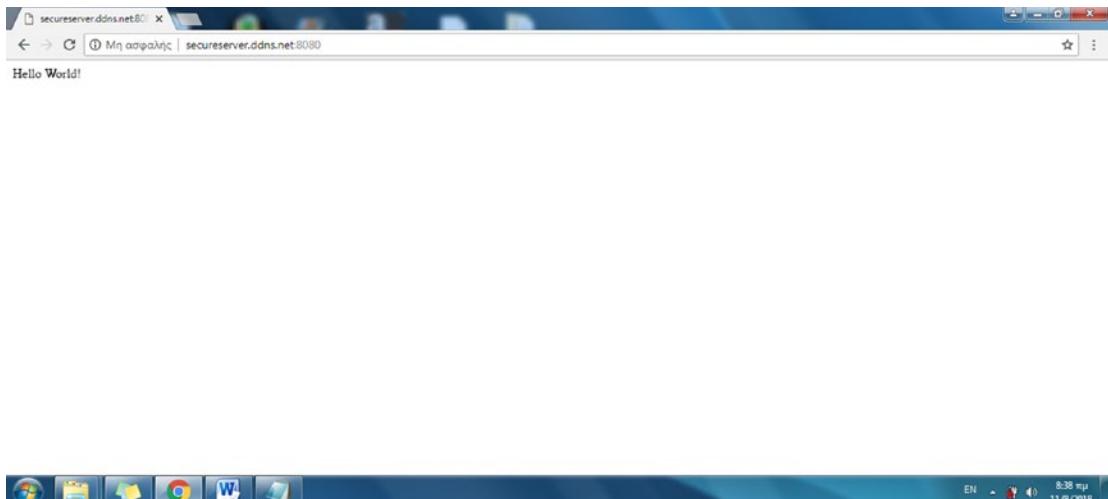


Figure 5-11. Displaying a web page using the new hostname

To create two virtual hosts on Lighttpd, with each serving a different domain name, use the following steps.

Create two document root directories, one for each virtual host. Give the corresponding name of the domain name used to the document root directory.

```
$ cd /var/www  
$ sudo mkdir secureserver.ddns.net christos.ddns.net
```

For each document root, change the owner from root (the Linux administrator is the owner because the directory was created with the `sudo` command) to `www-data`, the one Lighttpd assumes. This user belongs to the group `www-data` (`www-data:www-data`).

```
$ sudo chown www-data:www-data secureserver.ddns.net christos.ddns.net
```

Create one directory index named `index.html` for each virtual host. Use the following for the first virtual host:

```
$ sudo gedit /var/www/secureserver.ddns.net/index.html
```

Enter the following HTML source code and save the file:

```
<!DOCTYPE html>  
<html>  
<head>
```

```
<title>Welcome!</title>
<style>
body{
background-color:yellow;
}
p{
font-size:80px;
}
</style>
</head>
<body>
<p>
Hello from secureserver.ddns.net!
</p>
</body>
</html>
```

Do the same for the second virtual server.

```
$ sudo gedit /var/www/christos.ddns.net/index.html
```

Enter the following HTML source code and save the file:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<style>
body{
background-color:orange;
}
p{
font-size:80px;
}
</style>
</head>
```

```
<body>
<p>
Hello from christos.ddns.net!
</p>
</body>
</html>
```

Edit the Lighttpd configuration file.

```
sudo gedit /etc/lighttpd/lighttpd.conf
```

Append the following lines to the end of the file and save it:

```
$HTTP["host"] == "secureserver.ddns.net" {
    server.document-root = "/var/www/secureserver.ddns.net"
}
$HTTP["host"] == "christos.ddns.net" {
    server.document-root = "/var/www/christos.ddns.net"
}
```

With the previous conditional configuration, you will provide the option of using a different document root for each one of the previous domain names. The condition of the first directive checks whether the value of the Host HTTP protocol header of the client request corresponds to [secureserver.ddns.net](#), and if this matches, the document root is set to `/var/www/secureserver.ddns.net`. In the second directive, the value of the Host header is also checked against [christos.ddns.net](#), and if this matches, the document root is set to `/var/www/christos.ddns.net`.

Save `lighttpd.conf` and reload the server.

```
$ sudo service lighttpd force-reload
```

Test the two virtual servers from any computer on the Internet. For the first one, enter the following in your browser's address bar:

```
secureserver.ddns.net
or if for the if the port 8080 configuration is used:
secureserver.ddns.net:8080
```

Figure 5-12 displays the web page that loads.

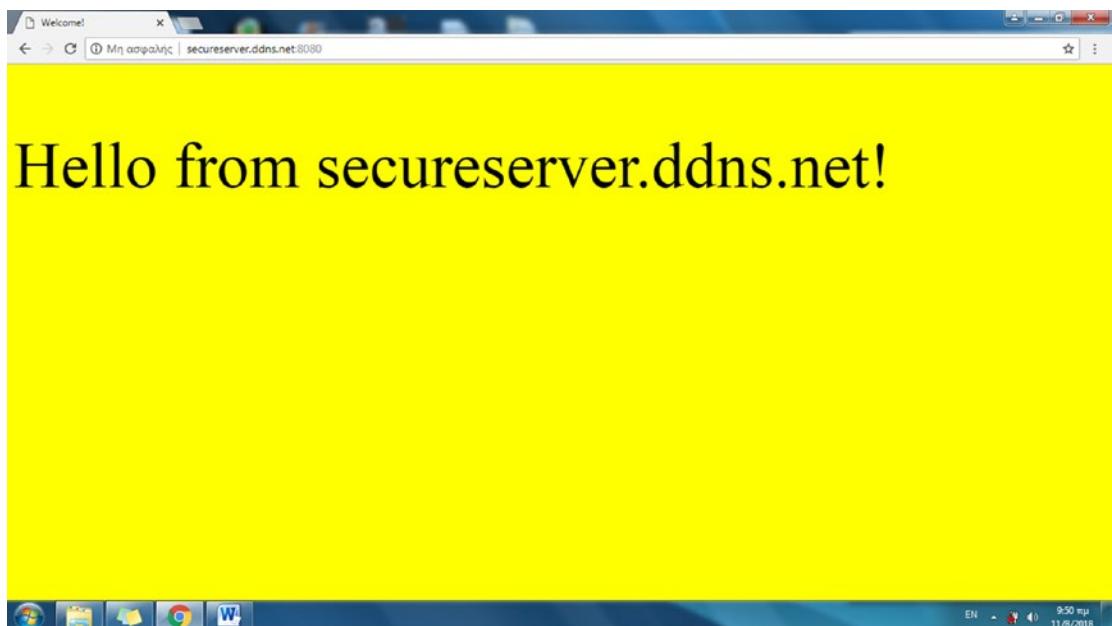


Figure 5-12. The directory index of the first virtual host

Use the URL of the second virtual host.

christos.ddns.net
or if the configuration for the 8080 port is used:
christos.ddns.net:8080

The web page displayed in Figure 5-13 loads in your browser.

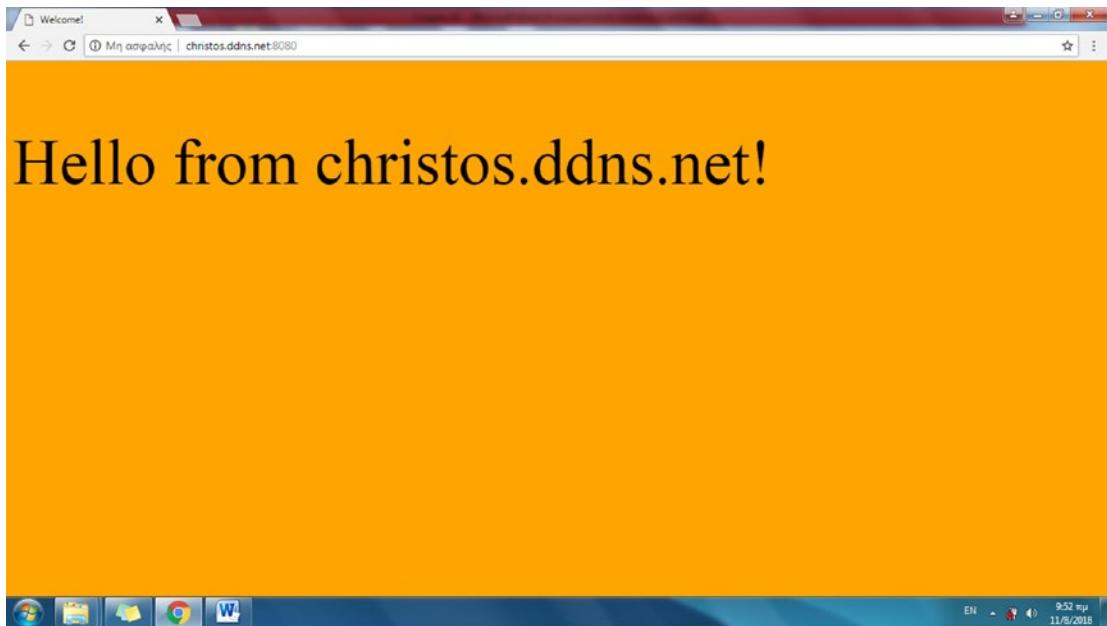


Figure 5-13. The directory index of the second virtual host

Hint! A practical way to configure multiline options in `lighttpd.conf` is to create another file in the `/etc/lighttpd` directory and move the extra configuration there. Create, for instance, a second configuration file using the following:

```
$ sudo gedit /etc/lighttpd/vhost.conf
```

Cut the following conditional directives from `lighttpd.conf` and paste them into `vhost.conf`:

```
$HTTP["host"] == "secureserver.ddns.net" {  
    server.document-root = "/var/www/secureserver.ddns.net"  
}  
$HTTP["host"] == "christos.ddns.net" {  
    server.document-root = "/var/www/christos.ddns.net"  
}
```

In `lighttpd.conf`, to create a link to `vhost.conf`, append the following directive to the end of the file:

```
include "vhost.conf"
```

Reload the server to enable the changes in the configuration files.

```
$ sudo service lighttpd force-reload
```

Using PHP with Lighttpd

To use PHP with Lighttpd, assuming that PHP 7.0 is already installed on your server, enter the following command at the Linux terminal:

```
$ sudo apt-get install php7.0-cgi
```

Test the PHP instantly using the file `/var/www/html/info.php` created in Chapter 2 for Apache. Because the original document root `/var/www/html` is different from those used for the virtual hosts, you can restore the configuration file to exclude virtual hosts. At the end of `lighttpd.conf`, comment out the `include` directive.

```
#include "vhost.conf"
```

Reload the web server to enable the changes in the configuration file.

```
$ sudo service lighttpd force-reload
```

In your browser's address bar, enter the following:

`http://christos.ddns.net/info.php`

Or, if your server listens on port 8080, enter the following:

`http://christos.ddns.net:8080/info.php`

As shown in Figure 5-14, the web page that appears displays the output of the `phpinfo()` function, indicating that PHP is up and running.

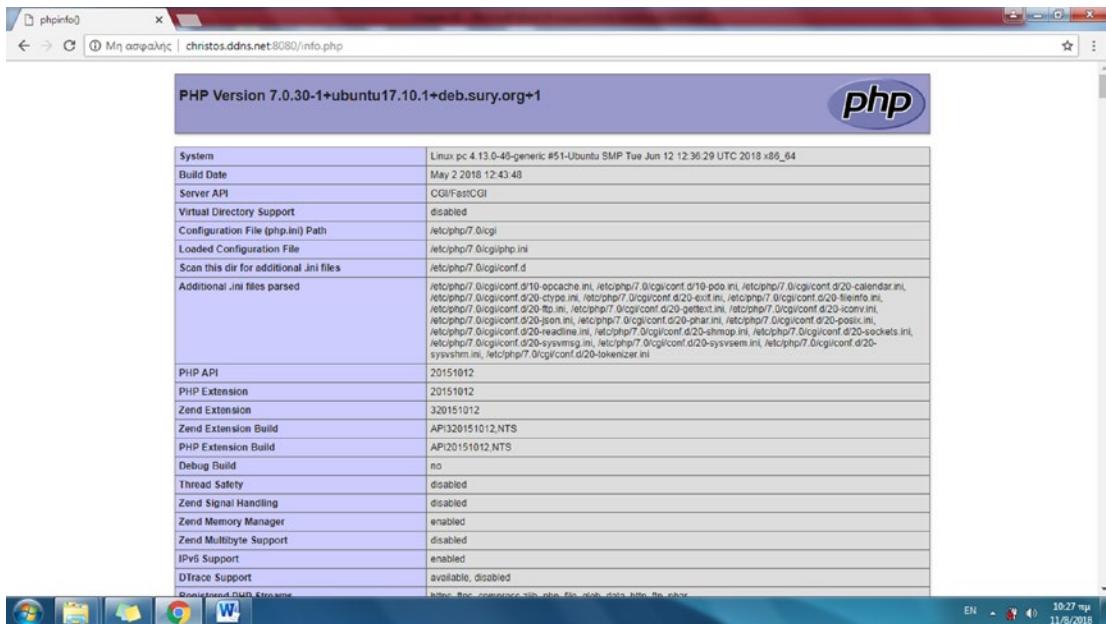


Figure 5-14. The web page with the evaluated `phpinfo()` function

With the basic Lighttpd server configuration options discussed in the previous sections, you can use Lighttpd in the following section to create your next online web service.

Creating Online Services with Lighttpd

In Chapters 3 and 4, you created online services with Apache. Here you'll use the Lighttpd server and the PHP and JavaScript languages to create another online service. The service will simulate online web services that test sites remotely like the webpagetest.org site utilized in Chapter 3. Similar to the other PHP projects, you will start by preparing two basic web pages.

- `index.php`, the web page that includes a form for submitting the URL of a site to the server.
- Screenshot `.php`, the program referred to in the `action` attribute of the `index.php` form, which processes the URL sent by the client and returns to the client's browser an image of the corresponding

web page. As shown in the Lighttpd configuration file, `lighttpd.conf`, the `index-file.names` directive has already set `index.php` as a directory index.

```
index-file.names          = ( "index.php", "index.html", "index.lighttpd.html" )
```

The file name `index.php` has precedence over the `index.html` and `index.lighttpd.html` file names. This means that by default the Lighttpd web server will serve `index.php` if this file name exists in the directory specified by the URL, for instance:

`http://webtoolsonline.servehttp.com`

When the user enters the previous URL in the browser's address bar, `index.php` appears, as shown in Figure 5-15.

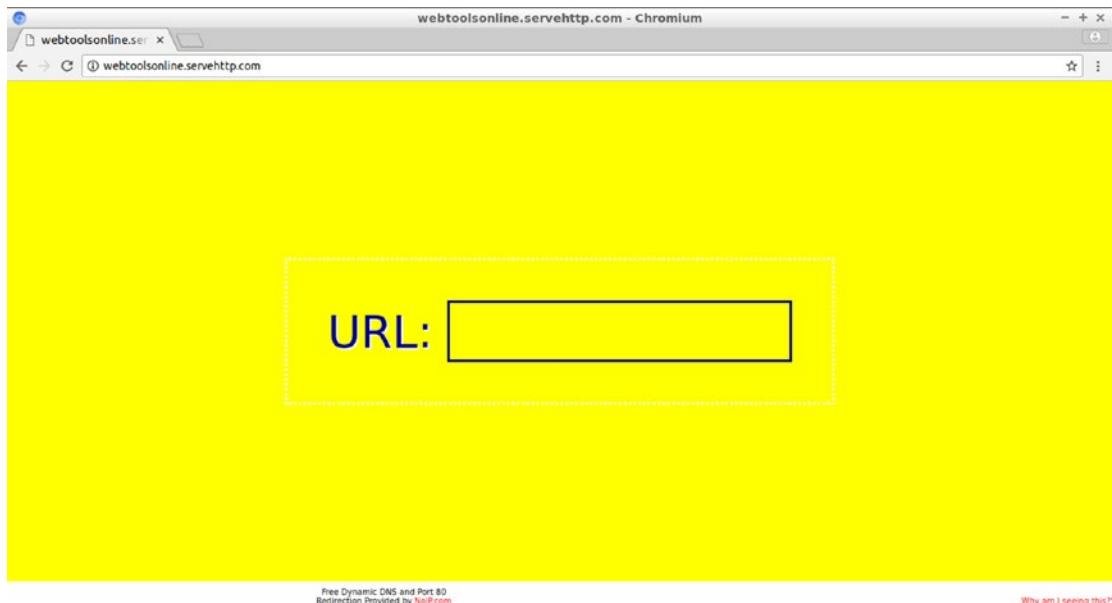


Figure 5-15. The home page of the online service

The form in `index.php` includes just one textbox, without any submit button. As shown in Figure 5-16, the user inserts the URL of a site in the textbox and presses the Enter key to submit the data.



Figure 5-16. Inserting a URL to the form's textbox

In the previous figure, the URL <https://www.amazon.com> was entered. The client request may take a few seconds to complete, so when the user presses the Enter key, the message “Please wait...” appears in the upper area of the browser’s window, as shown in Figure 5-17.



Figure 5-17. The message “Please wait...” appears when the user submits the data until the server responds

When the web server finishes processing the request, the web page replies with the evaluated code of `screenshot.php`. This is the web page that is set as the value of the `action` argument of the `form` tag. As viewed in Figure 5-18, for any valid URL, the `screenshot.php` displays a small-sized capture of the home page of the corresponding site.

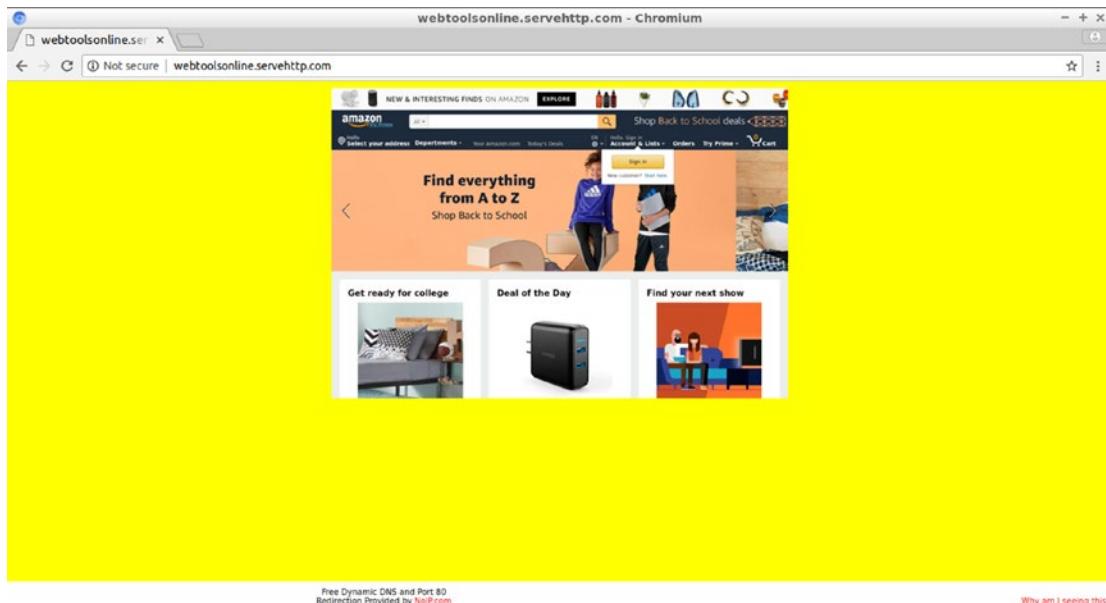


Figure 5-18. The web server responds to the client request by displaying a screenshot of the web page specified by the user

The user may insert an erroneous URL, for instance <https://123www.amazon.com>, as shown in Figure 5-19.

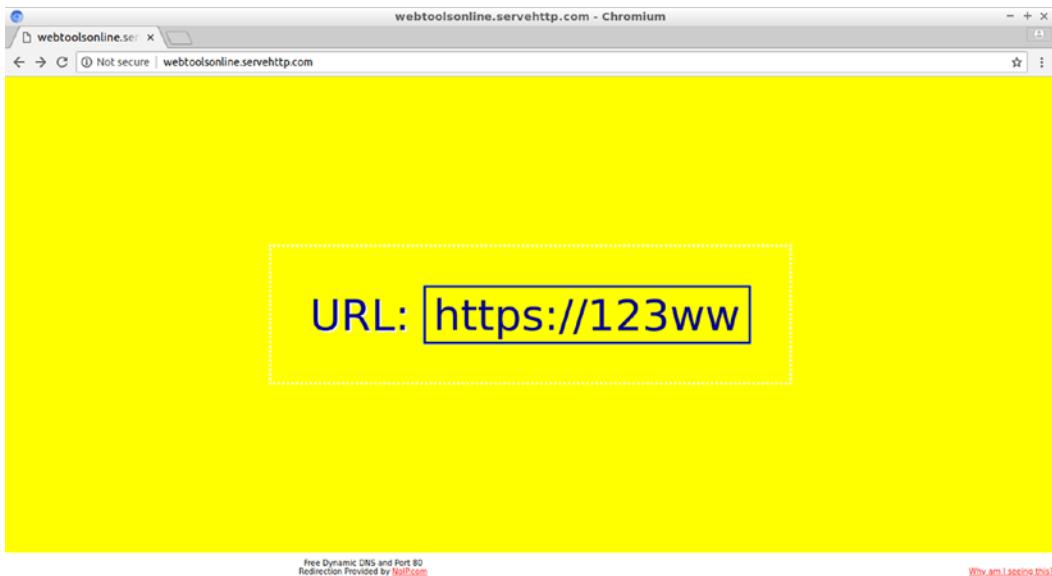


Figure 5-19. Testing an invalid URL

As shown in Figure 5-20, by pressing the Enter key, the `screenshot.php` code evaluates by displaying a message and a button instead of an image. These are used for getting back to the online service home.



Figure 5-20. The web server's response to an invalid URL

Creating the Directory Index of the Online Service

Use the following commands at the Linux terminal to create index.php:

```
$ cd /var/www/html  
$ sudo gedit index.php
```

Insert the following lines for the index.php source code and save the file:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
body{  
background-color:yellow;  
}  
.center {  
background-color:yellow;  
padding: 50px 50px;  
border: 3px dotted white;  
text-align: center;  
font-size:56px;  
color:darkblue;  
text-shadow: 2px 2px #ffffff;  
position:absolute;  
top:50%;  
left:50%;  
-ms-transform: translateX(-50%) translateY(-50%);  
-webkit-transform: translate(-50%, -50%);  
transform: translate(-50%, -50%);  
}  
input[type=text] {  
width: 400px;  
height:50px;
```

```
border: 3px solid darkblue;
font-size:56px;
color:darkblue;
background-color:yellow;
padding: 10px 10px;
}
h1{
color:darkblue;
font-size:56px;
}
</style>
</head>
<body>
<h1 id="id1"></h1>
<script>
function sub() {
document.getElementById("id1").innerHTML = "Please wait...";
}
</script>
<div class="center">
<form name="form1" method="post" action="screenshot.php" onsubmit="sub()">
    URL: <input type="text" name="url">
</form>
</div>
</body>
</html>
```

This PHP web page basically includes the form that submits a URL to the web server. It includes a CSS section for defining properties for styling the HTML elements. The basic content of this web page is a form that includes just one textbox named url. This is the object the client uses to submit the site's URL. The form's method is POST, and the action is set to the PHP file screenshot.php. Therefore, screenshot.php will take action to process the user request on the web server since no other directory path is included.

A JavaScript `onsubmit` type function, named `sub()`, is assigned to the `onsubmit` event of the form. Therefore, the function `sub()` will execute when the form is submitted. One line is included in this function, shown here:

```
document.getElementById("id1").innerHTML = "Please wait...";
```

The web page's HTML element with the value of the `id` (Identity) attribute equal to `id1` is returned by `getElementById()`. `getElementById()` is a method of the document instance, and the document represents the current web page. The element of the current web page with `id` equal to `id1` is the header with size 1 (`<h1>`). For this element, the `innerHTML` property indicates the header text, which is the text included between the start and end `<h1>` tags. When the user presses the Enter key and before the new web page loads, the message “Please wait...” will fill the text between the start/end `<h1>` tags, which is initially empty.

Creating the Action File for the Online Service

To create `screenshot.php` in the document root of the Lighttpd server, enter the following commands at the Linux terminal:

```
$ cd /var/www/html
```

```
$ sudo gedit screenshot.php
```

Enter the following lines and save the file:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
background-color:yellow;
}
</style>
</head>
```

```

<body>
<?php
if(!empty($_POST['url'])){
$url = $_POST["url"];
$command = "xvfb-run --server-args=\"-screen 0, 1366x768x24\" wkhtmltoimage
--crop-w 1366 --crop-h 768 " . $url . " /var/www/html/screenshot/php.png";
exec($command, $out, $status);
if ($status==0) {
$command2 = "convert -resize 50% /var/www/html/screenshot/php.png /var/
www/html/screenshot/php.png";
shell_exec($command2);
echo '';
} else {
echo '<div><p style="font-size:80px;color:darkblue;text-
align:center;">Please enter a valid URL.</p></div>';
echo '<div><a href="index.php">';
echo '<button style="font-size:80px;color:yellow;background-color:darkblue;
margin:auto;display:block;">Go Back</button>';
echo '</a></div>';
}
}
?>
</body>
</html>

```

The PHP code accepts as `$_POST["url"]` the value submitted by the textbox named `url` in the form of `index.php`. This is then assigned as the value of the PHP variable `$url`. This value is the URL entered by the user, which may be valid or invalid, and it is the information that the web server requires to act. The PHP engine that evaluates the `screenshot.php` source code uses the URL to form the following string:

```
xvfb-run --server-args="-screen 0, 1366x768x24" wkhtmltoimage --crop-w
1366 --crop-h 768 " . $url . " /var/www/html/screenshot/php.png
```

This string consists of the command and the command arguments that the PHP engine has to pass to the Linux terminal to create an image of the web site corresponding to the URL.

This certainly looks like a long command that requires some explanation. It uses the `wkhtmltoimage` open source command-line tool, which renders a web page into an image file. To enable the PHP engine to use the previous command, you have to download and install `wkhtmltoimage` using the following command at the Linux terminal:

```
$ sudo apt-get install wkhtmltopdf
```

An example of running `wkhtmltoimage` from the Linux terminal is as follows:

```
$ wkhtmltoimage https://www.amazon.com amazon.png
```

This creates the image `amazon.png` that depicts the home page of [amazon.com](https://www.amazon.com). This is a large image because the Amazon home page spans many computer screens, as viewed in Figure 5-21.

CHAPTER 5 THE LIGHTTPD WEB SERVER

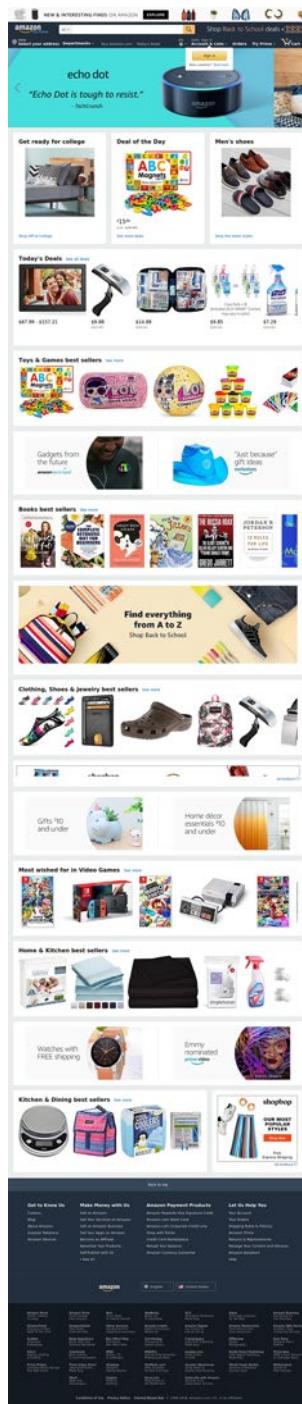


Figure 5-21. The large image created by wkhtmltoimage for Amazon.com

To include only the upper part, the image must be cropped to the screen dimensions of the computer the program was used on, e.g., 1366×768, using the crop width (`crop-w`) and the crop height (`crop-h`) arguments.

```
$ wkhtmltoimage --crop-w 1366 --crop-h 768 https://www.amazon.com amazon.png
```

Although the previous command runs if used directly at the command line, using it from the PHP engine may require an X Server specified for the graphics part. For this, you can download the Xvfb (X virtual framebuffer) server, which enables the PHP engine to run graphical applications without a display. Download and install Xvfb using the following at the Linux terminal:

```
$ sudo apt-get install xvfb
```

The following command, run from the PHP engine, starts `wkhtmltoimage` using the `Xvfb` server:

```
$command = "xvfb-run --server-args=\"-screen 0, 1366x768x24\" wkhtmltoimage --crop-w 1366 --crop-h 768 " . $url . " /var/www/html/screenshot/php.png";
```

Notice that `$command` is the variable that stores the string of the command to run, and notice that escape characters (\) were used to retain the double quotes inside the string. Using the dot (.) notation, the `$url` variable (the URL sent by the client) is concatenated with two strings, i.e., the first part of the command and the file path of the image file, created by the command. To create the directory `screenshot`, use the following command:

```
$ sudo mkdir /var/www/html/screenshot
```

Because `screenshot` was created by the user root, with the previous `sudo` command, the `screenshot`'s owner is user root. You can verify this using the `ls -l /var/www/html` command. You need to change the owner of the directory to enable Lighttpd and the PHP engine to access it in order to store the image files it creates. Use the following command:

```
$ sudo chown -R www-data:www-data /var/www/html/screenshot
```

This command sets the new owner of `/var/www/html/screenshot` to the user `www-data` that belongs to the group `www-data`. The recursive (-R) option is used for `www-data` to retain the ownership to all subdirectories of `screenshot`.

Next, the string value of the \$command variable is passed to the exec() PHP function.

```
exec($command, $out, $status);
```

In Chapter 4, you used the function shell_exec() to run terminal commands from the PHP engine. This time, the function exec() is used, which returns the return status of the command to its third argument (\$status in this example). By checking the value of \$status, you can find out whether the \$command execution succeeded or failed. You can use an if condition and replace the //SUCCESS and //FAILURE comments with the appropriate actions.

```
if ($status==0) {
//SUCCESS
} else {
//FAILURE
}
```

For the success case, you can use the following source code lines:

```
$command2 = "convert -resize 50% /var/www/html/screenshot/php.png /var/www/html/screenshot/php.png";
shell_exec($command2);
echo '';
```

A second PHP variable, \$command2, is used to store another command, convert, which is used to resize the image at a rate of 50 percent. The file path is used twice because using convert you have the option to convert the file format, such as from PNG to JPEG. For this, you would enter the following at the command line:

```
$ convert -resize 50% /var/www/html/screenshot/php.png /var/www/html/screenshot/php.jpg
```

This time the function shell_exec() is used to execute the second command.

To use convert, download and install the imagemagik package using the following:

```
$ sudo apt-get install imagemagick
```

The final command of the success case creates an HTML image tag (``) that renders the previously created `php.png` image at the center of the screen.

```
echo '';
```

For the failure case, you can use the following source code:

```
echo '<div><p style="font-size:80px; color:darkblue; text-  
align:center;">Please enter a valid URL.</p></div>';  
echo '<div><a href="index.php">';  
echo '<button style="font-size:80px; color:yellow; background-color:darkblue;  
margin:auto; display:block;">Go Back</button>';  
echo '</a></div>';
```

This returns a web page with the message “Please enter a valid URL.” to the client and also creates a button that, when clicked, returns the user to the site’s home page.

Enabling the Site to Serve Multiple Client Requests

While the source code of `screenshot.php` works well for a single client request, as tested previously, the server cannot dispatch multiple client requests when they are issued simultaneously because two or more `Xvfb` simultaneously calls fail. To verify this, open two (or more) tabs in your browser, all using the URL <http://webtoolsonline.servehttp.com>. In the textbox of the first tab, insert a URL of a site for testing and press the Enter key. Switch quickly to the second tab and do the same. Notice that while the first client request is fulfilled, the second fails. Figure 5-22 displays two tabs, with the second one including the “failure” page.



Figure 5-22. Issuing two simultaneous client requests to the site

Hint! You can also test this from the command line by calling the `xvfb` command used in `screenshot.php` twice (as fast as you can). Enter the following, for instance, at the Linux terminal:

```
$ sudo xvfb-run --server-args="-screen 0, 1366x768x24"  
wkhtmltoimage --crop-w 1366 --crop-h 768 https://www.amazon.  
com /var/www/html/screenshot/php1.png &
```

Then enter at the same terminal a command like the following. By using the ampersand (&) symbol at the end of the command, you are running the command in the background and the terminal is released to wait for another command.

```
$ sudo xvfb-run --server-args="-screen 0, 1366x768x24"  
wkhtmltoimage --crop-w 1366 --crop-h 768 https:// https://www.barnesandnoble.com /var/www/html/screenshot/php2.png &
```

The following message is output to the terminal the second time:

```
xvfb-run: error: Xvfb failed to start
```

To solve this problem, create a second version of the site that will require replacing the action file `screenshot.php` with `screenshot2.php` in `index.php`. Use the following command to edit `index.php`:

```
$ sudo gedit /var/www/html/index.php
```

In the form tag, replace `screenshot.php` with `screenshot2.php`.

```
<form name="form1" method="post" action="screenshot2.php" onsubmit="sub()">
```

Save the file and create the new action file.

```
$ sudo gedit /var/www/html/screenshot2.php
```

In the new version, the PHP file locking mechanism will be used to block all other client requests when an initial one is served. Therefore, only one request each time will be enabled to use the PHP code that creates the PNG file from a given URL. You'll create a text file, e.g., `lock.txt`, to be used just for implementing the file locking mechanism, without serving any other purpose in the source code. At the Linux terminal, use the `touch` command to create the text file.

```
$ sudo touch /var/www/html/lock.txt
```

Also, use the `chown` command to set the new owner of the file to the user assumed by the web server and PHP, `www-data`.

```
$ sudo chown www-data:www-data /var/www/html/lock.txt
```

The new text file will be used from the `screenshot2.php` source code as follows:

```
$fp = fopen("lock.txt", "w");
if (flock($fp, LOCK_EX)) {
    // PHP source Code that previously conflicted
    flock($fp, LOCK_UN);
}
fclose($fp);
```

In the previous lines, the file handle `fp` is created for writing (`w`) to `lock.txt` with a call to the function `fopen()`. Then comes a call to the function `flock()`, which locks the file exclusively (`LOCK_EX`). The PHP source code is therefore assured to be executed

only from a single process. Other instances will find the file locked and will block until the file is unlocked. The role of this file is therefore to hand over permission to use the enclosed code from one process to the other, with the assurance that all other processes are waiting to take their turn and won't return without serving the client. With the next `flock()` call in the `LOCK_UN` argument, the file unlocks when the current process is done.

The complete source code of `screenshot2.php` is as follows:

```
<html>
<head>
<style>
body {
background-color:yellow;
}
</style>
</head>
<body>
<?php
if(!empty($_POST['url'])){
$url = $_POST["url"];
$fp = fopen("lock.txt", "w");
if (flock($fp, LOCK_EX)) {
$command = "xvfb-run --server-args=\"-screen 0, 1366x768x24\" wkhtmltoimage
--crop-w 1366 --crop-h 768 " . $url . " /var/www/html/screenshot/php2.png";
exec($command, $out, $status);
if ($status==0) {
$command2 = "convert -resize 50% /var/www/html/screenshot/php2.png /var/
www/html/screenshot/php2.png";
shell_exec($command2);
echo '';
} else {
echo '<div><p style="font-size:80px;color:darkblue;text-
align:center;">Please enter a valid URL.</p></div>';
echo '<div><a href="index.php">' ;
echo '<button style="font-size:80px;color:yellow;background-color:darkblue;
margin:auto;display:block;">Go Back</button>';
```

```
echo '</a></div>';  
}  
flock($fp, LOCK_UN);  
}  
fclose($fp);  
}  
?  
</body>  
</html>
```

Test the new version of the site by opening two (or more) tabs in your browser and entering a URL in each web page's textbox. As shown in Figure 5-23, all client requests are now dispatched.

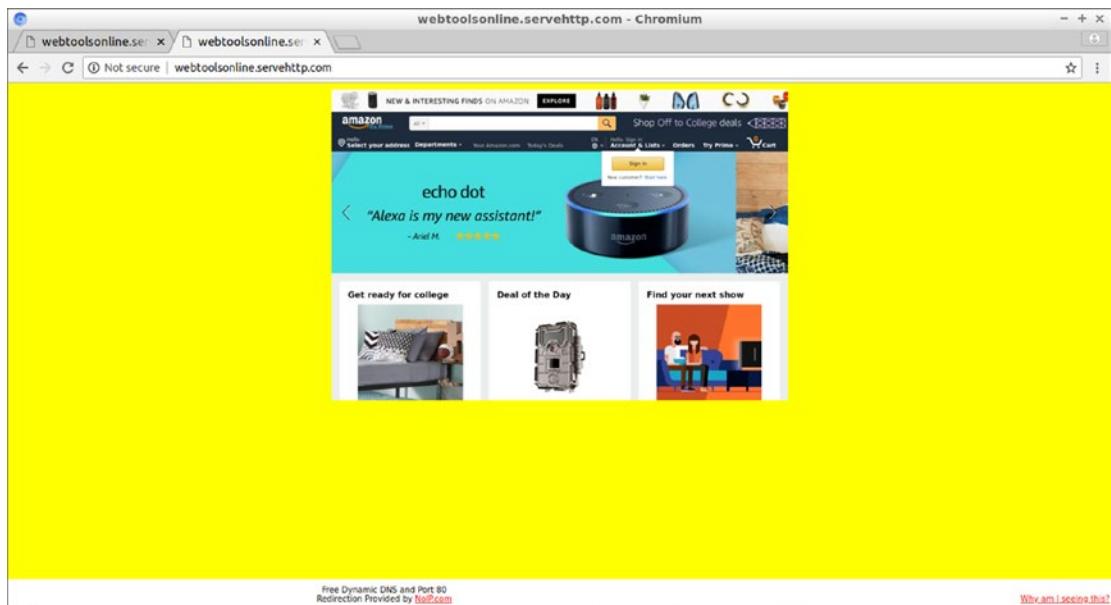


Figure 5-23. Testing the site with multiple client requests succeeds

When you have issued the second request, switch to the Linux terminal and enter the following command:

```
$ sudo lslocks
```

This command displays information about the current file locks in the system. The output of the command includes two records about `lock.txt`, which means that there are currently two processes that lock or try to exercise a lock to the file `test.txt`. In this example, the records have the following form:

php-cgi	722	FLOCK	0B	WRITE*	0	0	/var/www/ html/lock.txt
php-cgi	724	FLOCK	0B	WRITE	0	0	/var/www/ html/lock.txt

Here, 722 and 724 are the process IDs (PIDs) of `php-cgi`, the process that executes the PHP source code. Use, for instance, the following command to verify this:

```
$ ps xa | grep 724
```

The command's output is as follows:

```
724 ? S 0:00 /usr/bin/php-cgi
```

When the first client request is dispatched and the image of the test page is rendered to the web page, enter the `lslocks` command at the terminal another time:

```
$ sudo lslocks
```

As the command's output indicates, this time there is only one record about `lock.txt`, which means only one process locks `lock.txt`.

php-cgi	722	FLOCK	0B	WRITE	0	0	/var/www/ html/lock.txt
---------	-----	-------	----	-------	---	---	----------------------------

When the second client request is fulfilled, try `lslocks` again. This time, as expected, no records are output about `lock.txt`.

Creating an Animated PNG Image

In the project of the previous section, it is a good idea to include an animated image that appears while the client's request is being processed to signify that the wait is because of the request processing and not because of any network latency or other problem. This improves the user's experience and makes the waiting time less boring.

You can create animated images using software applications on your PC or using online tools. In this example, you will use the online tool <https://ezgif.com/apng-maker> to create an animated PNG image. The procedure of making the animated PNG file is similar to using other online PNG animation tools.

To create the animated PNG image file, you need to create a number of PNG files used as frames that will be displayed consecutively and create the effect of animation. There is a basic option you have to configure, which is the time that should elapse between each frame. All PNG frames, along with the time lapse information, are then packaged into a final PNG file, in other words, the animated PNG file.

Use your favorite image editor to create a PNG image like file 0.png, shown in Figure 5-24.



Figure 5-24. The pattern for making the images for the animated PNG image

This image will be used as a pattern that other PNGs will be based on. I purposely used black to paint the unused areas, because with the ezgif.com online service, it is easy to turn black to transparent. This means when the image is loaded in your web page, the black color will be replaced by the current background of the page. You can also use other colors for setting transparency, but because an RGB code is required, it is best to do this with black or with white.

Figure 5-25 displays the idea behind the animated PNG. You create six PNG files, file 1.png up to file 6.png, with each one lighting up a different square with a chosen color. For example, the first image lights up the first square, the second image lights up the second square, and so on. Rendering all images, one after the other, has the effect of moving the bright color from the first to the last square. Then just copy file 5.png as 7.png, 4.png as 8.png, 3.png as 9.png, and finally 2.png as 10.png. Therefore, after 6.png, the sequence returns backward until file 10.png, and then everything starts again from the beginning.

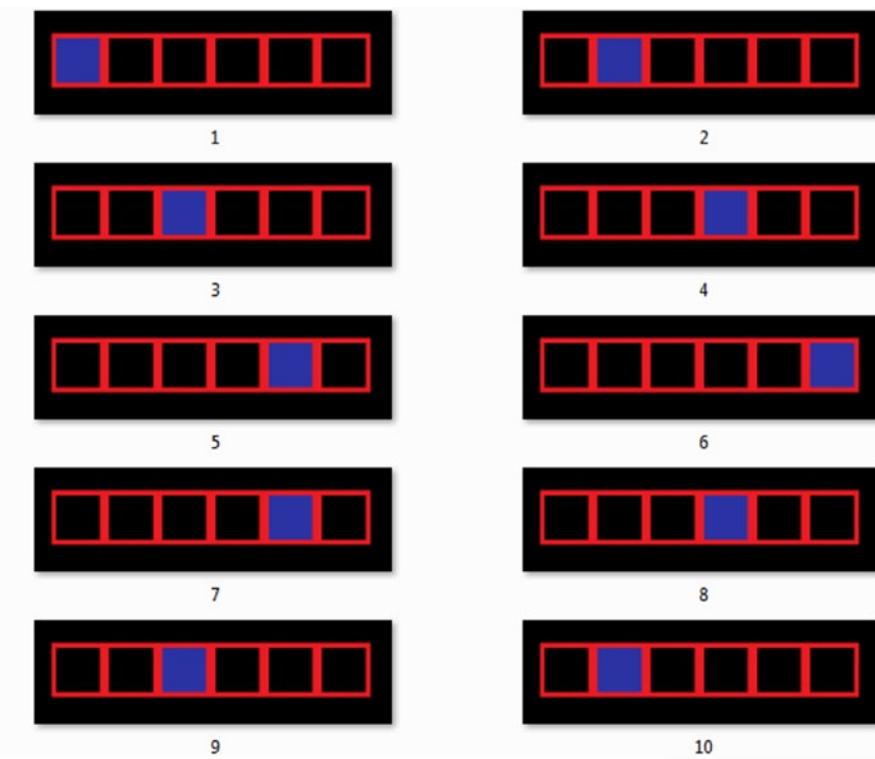


Figure 5-25. The PNG images required for creating the animated PNG file

Place files 1.png up to 10.png in the folder png and zip the files, creating a zipped file. Enter the URL <https://ezgif.com/apng-maker> in your browser's address bar to visit the web page, as shown in Figure 5-26. Click the Choose Files button and located the zipped folder with the PNG files in the directory tree of your computer. Click the Open button in the dialog and then click the Upload! button on the ezgif.com web page.

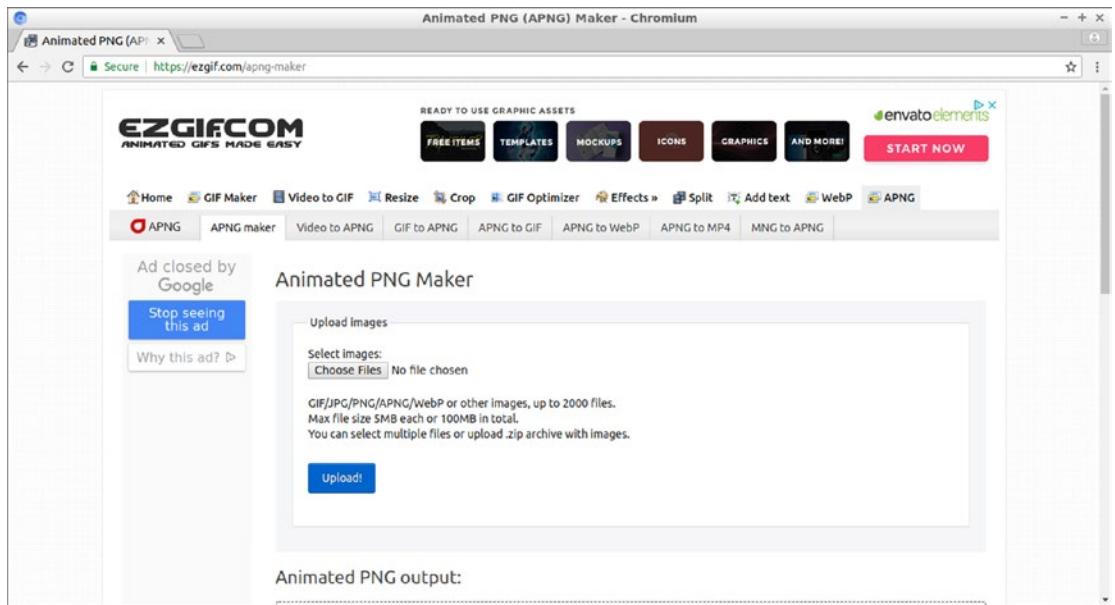


Figure 5-26. The Animated PNG Maker web page

The uploaded zipped file is extracted, and all images are visible in your browser's window, as displayed in Figure 5-27.

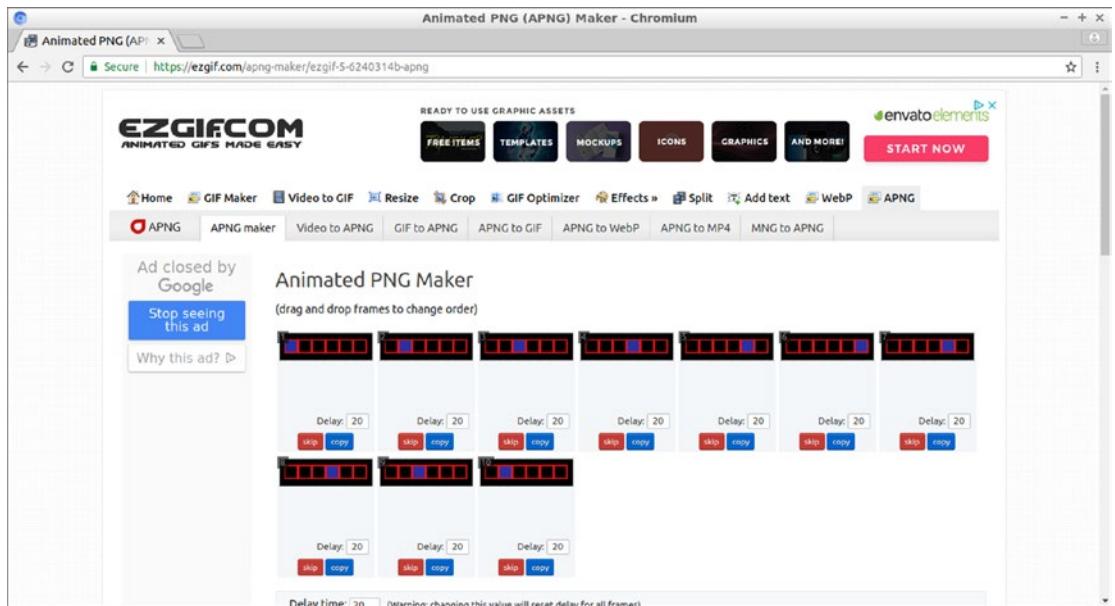


Figure 5-27. The uploaded images to the Animated PNG Maker site

CHAPTER 5 THE LIGHTTPD WEB SERVER

On this web page, you can set the delay time and also the loop count. You can leave those settings at their default values. Scroll down and click the Make APNG! button. As shown in Figure 5-28, the animated PNG file has been created under “Animated PNG output.”

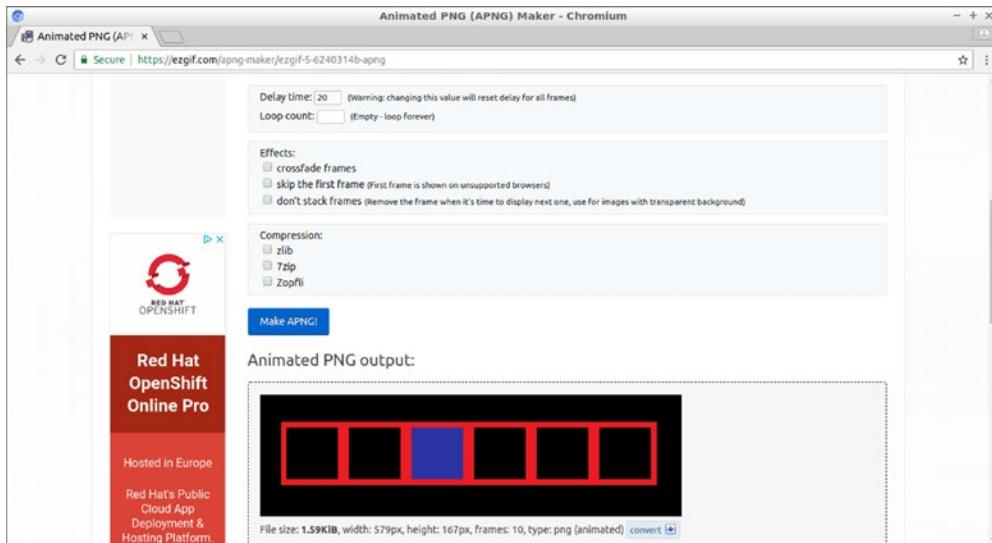


Figure 5-28. The Animated PNG file displayed on the ezgif.com web page

Scroll down to view the toolbar, as displayed in Figure 5-29. Click the effects tool.

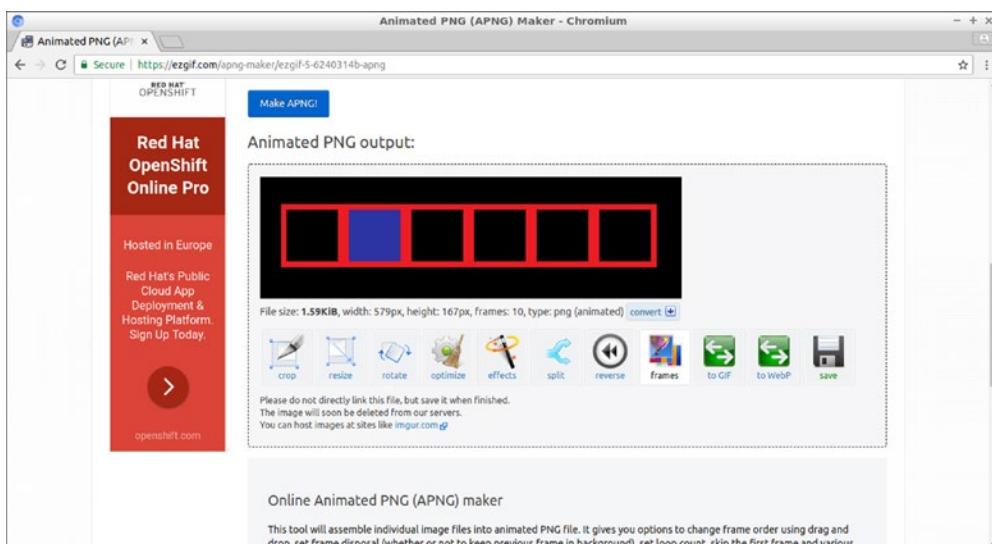


Figure 5-29. The effects tool in the ezgif.com toolbar

The web page displayed in Figure 5-30 appears. In the “Replace color with transparency” section, select the black box to turn all the black pixels of the PNG files transparent. Click the “Apply selected!” button.

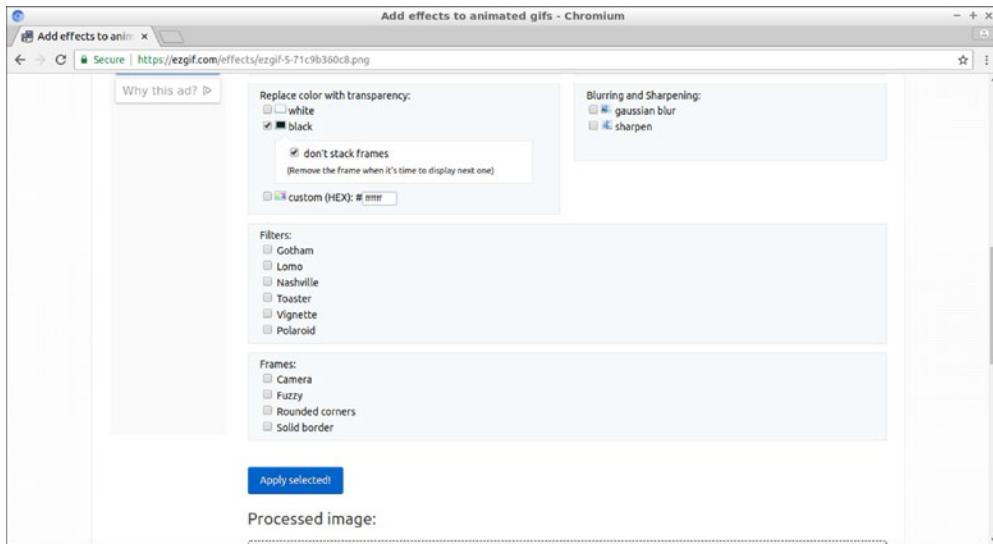


Figure 5-30. The ezgif.com web page buttons for setting color transparency

The new processed image with the transparency set renders in the “Processed image” section and starts animating, as shown in Figure 5-31.

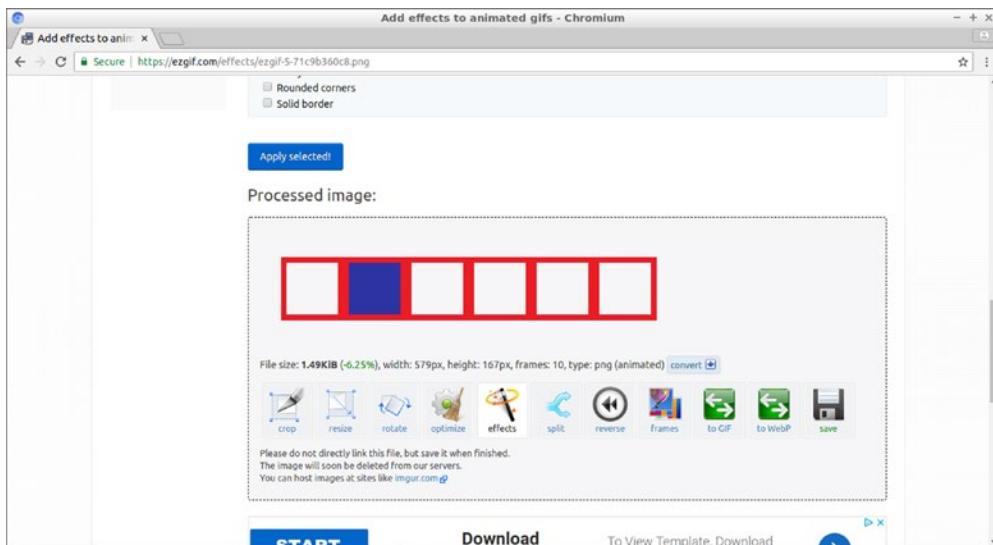


Figure 5-31. The processed image as viewed on the ezgif.com web page

Right-click the image, and in the pop-up menu that appears select “Save image as” (or your browser’s equivalent option). In the directory tree of the dialog that appears, set the destination of the home directory as `animation.png`. Use the following `sudo` command to enable copying the file to the document root of the site:

```
$ sudo mv ~/animation.png /var/www/html
```

Change the file owner (set with `sudo` to root) to `www-data`.

```
$ sudo chown www-data:www-data /var/www/html/animation.png
```

Use the animated PNG image instead of the “Please wait...” message. The source code of `index.php` is now as follows:

```
<!DOCTYPE html>
<html>
<head>
<style>

body{
background-color:yellow;
}

.center {
    background-color:yellow;
    padding: 50px 50px;
    border: 3px dotted white;
    text-align: center;
    font-size:56px;
    color:darkblue;
    text-shadow: 2px 2px #ffffff;
    position:absolute;
    top:50%;
    left:50%;
    -ms-transform: translateX(-50%) translateY(-50%);
    -webkit-transform: translate(-50%,-50%);
    transform: translate(-50%,-50%);
}
```

```
input[type=text] {
    width: 400px;
    height:50px;
    border: 3px solid darkblue;
    font-size:56px;
    color:darkblue;
    background-color:yellow;
    padding: 10px 10px;
}

#id1{
display:none;
margin-left: auto;
margin-right: auto;
}

</style>
</head>

<body>



<script>
function sub() {
document.getElementById("id1").style.display = "block";
}
</script>

<div class="center">
<form name="form1" method="post" action="screenshot2.php" onsubmit="sub()">
    URL: <input type="text" name="url">
</form>
</div>

</body>
</html>
```

The function `getElementById()` in the JavaScript source code in this version of the function `sub()` accesses the element of the current web page (`document`) with a value for the attribute `id` equal to `id1`.

```
document.getElementById("id1").style.display = "block";
```

This `id` belongs to the `image` element. Its style is then accessed with the `style` property, which represents an element's style attribute. The `display` style property is set then to `block`. In the CSS section of the source code, the `display` for the element with the `id` equal to `id1` was previously set to `none`.

```
#id1{
display:none;
margin-left: auto;
margin-right: auto;
}
```

Therefore, by submitting the form, the function `sub()` runs, and this displays the animated PNG, which is the image with the `id` property equal to `id1`.

The new directory index when the user URL is processed looks like the one in Figure 5-32.

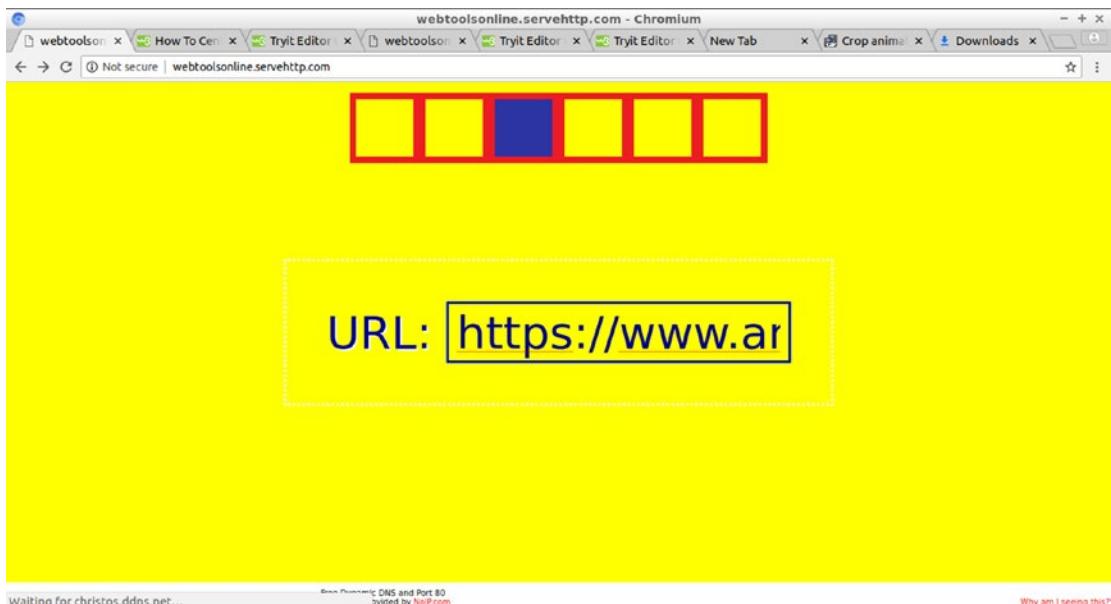


Figure 5-32. The animated PNG runs while waiting for the web server to respond

By creating an inspired animated PNG image, the waiting time is more fun.

Summary

In this chapter, you installed and set up Lighttpd, an open source web server. You had the chance to use features similar to Apache in another web server, and ideally this will make the transition to a third web server more straightforward. You mainly set the directory index, document root, listening port number, listening IP address or hostname, virtual servers, log files, and custom error reply web pages.

You also created another online web service that tests web pages remotely. In the following chapter, you will start using the MySQL database to take your online services to another level. With database queries enabled, your site's capabilities greatly increase.

CHAPTER 6

The MySQL Database Server

With PHP you can enable the web server to interface with any other program. One of the most commonly used applications for interfacing with a web server is a database server. With a database connection to the web server, your site can search the data of the database system and thus offer dynamic content. There is a large list of database systems to choose from. In this chapter, you will download and start using one of the most common ones, MySQL, which is a relational database that utilizes the Structured Query Language (SQL). SQL is the most widespread set of instructions used to set up and query a database system.

In this chapter, you will use some basic SQL commands to create and manage a MySQL database and to create a shell script to automatically feed a MySQL database with data collected from a web site, a process that is often called *web scraping*.

Installing and Testing MySQL

Use the following command at the Linux terminal to download the MySQL database system:

```
$ sudo apt-get install mysql-server mysql-client
```

The `mysql-server` package will install the MySQL database server, which you can interact with from the terminal by using the `mysql` command from the `mysql-client` package. During the installation process, the dialog shown in Figure 6-1 appears, and you are prompted to enter a password for the MySQL administrator user, named `root`. In the examples used in this book, no password was provided for accessing the MySQL server, and the password value in the source code examples is an empty string. Likewise, you can leave the password field blank and just press the Enter key.

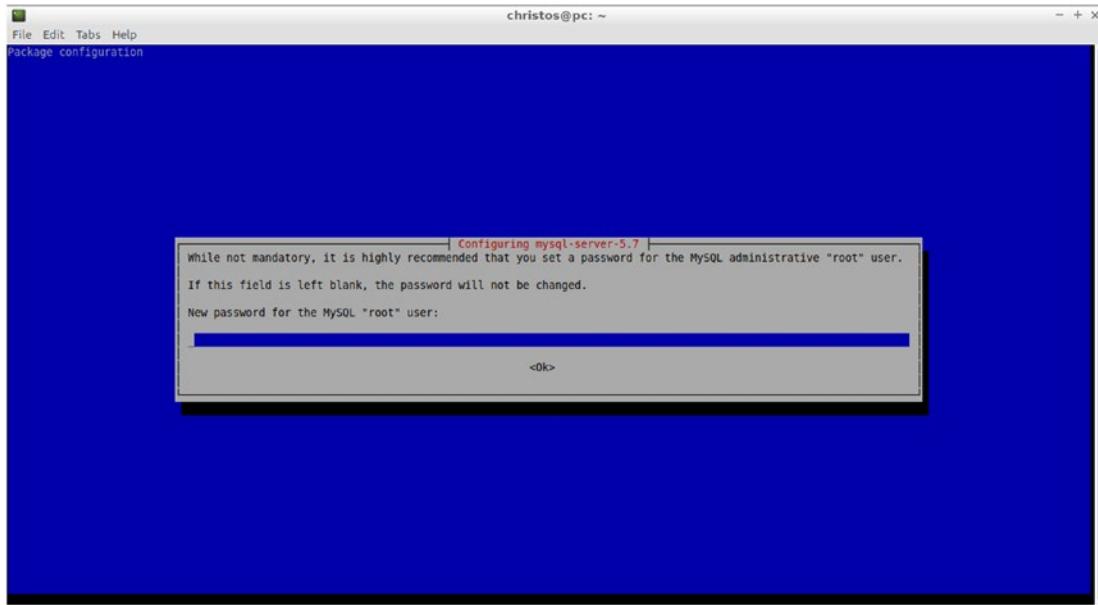


Figure 6-1. The first dialog of the MySQL installation process prompts you for a password

A second window appears for the password confirmation. If you did not enter a password previously, leave the field in this window empty and press Enter again.

The installation process continues, with information displayed in the terminal until completion.

To find out the status of the MySQL server, use the following command:

```
$ service mysql status
```

Some sample output of this command indicating that MySQL is up and running is shown here:

```
Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
Active: active (running) since Sun 2018-08-19 13:23:05 EEST; 3min 3s ago
Process: 662 ExecStartPost=/usr/share/mysql/mysql-systemd-start post
          (code=exited, status=0/SUCCESS)
Process: 608 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre
          (code=exited, status=0/SUCCESS)
```

```
Main PID: 661 (mysqld)
  Tasks: 28 (limit: 4915)
  CGroup: /system.slice/mysql.service
           └─661 /usr/sbin/mysqld
```

```
Aug 19 13:22:50 pc systemd[1]: Starting MySQL Community Server...
Aug 19 13:23:05 pc systemd[1]: Started MySQL Community Server.
```

Press Q (quit) on the keyboard to exit the command and release the terminal.

Like with Apache and Lighttpd, to stop the MySQL server and then start it again, use the following commands at the terminal:

```
$ sudo service mysql stop
$ sudo service mysql start
```

To connect to the MySQL server, you can use the `mysql` client program. At the Linux terminal, enter the following command:

```
$ mysql -u root
```

A welcome message along with some basic information is output to the terminal. Also, the MySQL prompt appears instead of the shell's prompt, indicating that you are connected to the MySQL server, ready to send SQL commands.

```
Welcome to the MySQL monitor. Commands end with ; or \g.
```

```
Your MySQL connection id is 36
```

```
Server version: 5.7.22-Ubuntu0.17.10.1 (Ubuntu)
```

```
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.
```

```
mysql>
```

This message notes that you can use the command `help` or its corresponding shortcut (`\h`) to list all the available commands. Run `help` as a first command.

```
mysql> help
```

The output of help displays the commands that mysql interprets, shown here:

List of all MySQL commands:

Note that all text commands must be first on line and end with ';'.

```
?          (\?) Synonym for `help'.
clear      (\c) Clear the current input statement.
connect    (\r) Reconnect to the server. Optional arguments are db and host.
delimiter  (\d) Set statement delimiter.
edit       (\e) Edit command with $EDITOR.
ego        (\G) Send command to mysql server, display result vertically.
exit       (\q) Exit mysql. Same as quit.
go         (\g) Send command to mysql server.
help       (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an
          argument.
status    (\s) Get status information from the server.
system   (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given outfile.
use      (\u) Use another database. Takes database name as argument.
charset  (\C) Switch to another charset. Might be needed for processing
          binlog with multi-byte charsets.
warnings (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
resetconnection(\x) Clean session context.
```

For server side help, type 'help contents'

Test, for instance, the system command, which executes shell commands, with the terminal command clear.

```
mysql> system clear
```

The terminal clears with the mysql prompt ready to accept the next command.

To end the MySQL session and return to the Linux terminal, type the exit command.

```
mysql> exit
```

With these commands, you can start and exit the MySQL client and also view and run the set of the available commands that mysql interprets. Next, you will learn about the SQL statements that mysql issues to the MySQL server to create and manage a database.

Creating Your First MySQL Database

Start the mysql client to connect to the server so you can create and use your first database. In this section, you will design the tables included in this database, you will define the relationships between the tables, and then you will use the tables to enter your data. You will then be able to issue database queries on the table data and generate results.

First connect to the MySQL server by using mysql at the Linux terminal.

```
$ mysql -u root  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 3  
Server version: 5.7.22-0ubuntu0.17.10.1 (Ubuntu)  
  
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql>
```

To create a new database named library, enter the following create database SQL statement and press Enter:

```
mysql> create database library;
```

MySQL server responds with the following message:

```
Query OK, 1 row affected (0.00 sec)
```

Hint! Each MySQL command requires a semicolon (;) to indicate the ending. Some SQL databases require the commands to be inserted in uppercase characters (e.g., CREATE DATABASE library;). The MySQL syntax used here does not require uppercase.

To view all the databases included so far, use the show databases statement.

```
mysql> show databases;
```

The command's output indicates that your new database is added in the default set of the five databases. The MySQL server already includes the following:

```
+-----+
| Database      |
+-----+
| information_schema |
| apress         |
| library        |
| mysql          |
| performance_schema |
| sys            |
+-----+
6 rows in set (0.00 sec)
```

You can delete the new database at any time, regardless of whether it is empty or filled with data, using the drop database statement.

```
mysql> drop database library;
```

If you test drop at this time, make sure to repeat the previous create command to re-create the database.

Creating and Deleting Tables of Your Database

To start working with the new database, you have to define that the commands that will follow will refer to that specific database. The use SQL statement indicates the database that will be used for the commands that follow, up to the point where another use is typed.

```
mysql> use library;
```

You'll now create your first table for the database. For each table that will be included in the database, the column (field) names and the types of data inserted to the columns must be defined at creation time. For instance, for the first table of `library`, called `author`, the author's name and the author's country will be defined as the table's columns. To create table `author`, enter the following at the `mysql` prompt:

```
mysql> CREATE TABLE author (name VARCHAR(30), country CHAR(2));
```

The MySQL server responds with an OK message.

```
Query OK, 0 rows affected (0.43 sec)
```

The first column, called `name`, is of type `VARCHAR`, which means it's a string of characters with a variable length. You allocate the maximum number of characters for the value in parentheses. In this example, the author's name can be up to 30 characters. The second column is of type `CHAR`, also a string of characters but with a fixed length. The length in this example is specified in the parentheses as 2, which means the author's country will always be inserted as two characters. This is sufficient because the country column value will be provided as the ISO 3166-1 alpha two-character code, e.g., `uk`.

To print an overview of the table's structure, enter the following:

```
mysql> describe author;
```

MySQL responds with a description of the table's columns, shown here:

Field	Type	Null	Key	Default	Extra
<code>name</code>	<code>varchar(30)</code>	YES		<code>NULL</code>	
<code>country</code>	<code>char(2)</code>	YES		<code>NULL</code>	

2 rows in set (0.01 sec)

Because it's a relational database, MySQL allows you to connect two or more tables by *relating* their entries. This allows you to store data in multiple tables, instead of just a big one. This approach makes data more manageable and less error prone.

The queries you'll execute next will combine columns from both tables. In this example, details of the books written by the authors in the author's table will make up a second table called book. To connect the two tables and allow their data to be combined, a common column must be used from both tables. This column is defined as the *primary key* for the first table and the *foreign key* for the second. A primary key (and therefore a foreign key) must have a unique value in each table's record so that it discriminates this record from the others.

So far, no primary key was used for the author table. To correct this, you have two options: delete this table and re-create it or alter the existing table's structure. You can delete a table regardless of whether the table is empty or filled with data. Since you are in an early stages of the database's design, I'll display the table deletion here and leave the table modification for later. To delete table author, use the drop table statement as follows:

```
mysql> drop table author;
```

The MySQL server responds with an OK message.

```
Query OK, 0 rows affected (0.22 sec)
```

Re-create the table authors, this time including author_id, which is the column that will be the primary key of the table. The primary key constraint is the one that assigns author_id as the primary key. The auto_increment keyword defines that for each record (row) inserted in the table, the author_id value will receive automatically the next available integer, starting from 1. The data type of author_id is set to int (integer). At the mysql prompt, enter the following:

```
mysql> create table author (author_id int auto_increment, primary key(author_id), name varchar(30), country char(2));
```

The MySQL server responds with an OK message.

```
Query OK, 0 rows affected (0.44 sec)
```

Use the describe SQL statement again to view the new table's structure.

```
mysql> describe author;
```

The MySQL server responds by displaying the following table, which describes the structure of author:

```
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key  | Default | Extra        |
+-----+-----+-----+-----+-----+
| author_id  | int(11)   | NO   | PRI  | NULL    | auto_increment |
| name        | varchar(30) | YES  |       | NULL    |               |
| country     | char(2)    | YES  |       | NULL    |               |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Notice that the `author_id` field is identified by the `PRI` value in the `Key` column as the primary key. In addition, as indicated by the `NO` value in the `Null` column field, `author_id` should not be empty for any record. Also, the `auto_increment` keyword for the primary key is shown in the `Extra` column.

Next, create `book`, the second table that will be used for the library database. Column `book_id` will be the primary key for the `book` table; column `title` will hold the book title, which will be up to 255 characters; column `language` will be the language the book is written to; and column `author_id` will be the foreign key from the `author` table, defining a connection among the two tables. The `not null` constraint is also set so that all `author_id` values are required to be filled. At the `mysql` prompt, enter the following `create table` command:

```
mysql> create table book (book_id char(13), primary key(book_id), title
varchar(255), language varchar(20), author_id int not null, foreign
key(author_id) references author(author_id));
```

The MySQL server responds with an OK message.

```
Query OK, 0 rows affected (0.57 sec)
```

Use the `describe` SQL statement for the second table to view its structure.

```
mysql> describe book;
```

The MySQL server responds with the following output:

```
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| book_id    | char(13)   | NO   | PRI | NULL    |       |
| title      | varchar(255)| YES  |     | NULL    |       |
| language   | varchar(20) | YES  |     | NULL    |       |
| author_id  | int(11)    | NO   | MUL | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

This time, two columns have a `NO` value in the `Null` column: `book_id`, which is the primary key, and `author_id`, the foreign key, which was explicitly set to not null. The `MUL` (multiple) key indicates that multiple rows may have the same value as the `author_id` column.

Inserting, Displaying, and Deleting Records

You can use the `insert into` SQL statement to enter a number of records in the `author` table. You don't have to specify the `author_id` value since it is automatically entered because of the `auto_increment` attribute. The following three commands are used, with their output shown here:

```
mysql> insert into author (name, country) values('Tolkien, John Ronald
Reuel', 'UK');
Query OK, 1 row affected (0.07 sec)
mysql> insert into author (name, country) values('Steinbeck, John Ernst
Jr.', 'US');
Query OK, 1 row affected (0.07 sec)
mysql> insert into author (name, country) values('Eco, Umberto', 'IT');
Query OK, 1 row affected (0.08 sec)
```

To view the records inserted into table `author` so far, use the `select` SQL statement, which is one of the most important statements used for SQL queries. In its simplest form, without any clause, `select` takes as argument the asterisk (*) wildcard, which corresponds to all columns, with no other condition. The following command therefore displays all elements of table `author`:

```
mysql> select * from author;
```

The command's output is as follows:

author_id	name	country
1	Tolkien, John Ronald Reuel	UK
2	Steinbeck, John Ernst Jr.	US
3	Eco, Umberto	IT

3 rows in set (0.01 sec)

Enter the last record another time.

```
mysql> insert into author (name, country) values('Eco, Umberto', 'IT');
```

Use the SQL `select` statement to view the table records again.

```
mysql> select * from author;
```

The duplicated entry for the name is displayed next. In this case, this is an unwanted result, which can be corrected in the next section.

author_id	name	country
1	Tolkien, John Ronald Reuel	UK
2	Steinbeck, John Ernst Jr.	US
3	Eco, Umberto	IT
4	Eco, Umberto	IT

4 rows in set (0.00 sec)

Use the delete SQL statement to remove the fourth entry of the author table. In the following command, the fourth record is indicated with the where clause, which specifies the records to be deleted by setting the condition: author_id (the primary key) equal to 4.

```
mysql> delete from author where author_id=4;
```

The MySQL server responds with an OK message.

```
Query OK, 1 row affected (0.18 sec)
```

Display the remaining author records using the SQL select statement.

```
mysql> select * from author;
```

The command's output is as follows:

author_id	name	country
1	Tolkien, John Ronald Reuel	UK
2	Steinbeck, John Ernst Jr.	US
3	Eco, Umberto	IT

3 rows in set (0.01 sec)

Altering the Table's Structure

Even when a table is filled with data, it is not too late to modify the table's characteristics. In this section, you will make three alterations in a table's structure. The first one solves the problem of the duplicated entry shown in the previous section. The second modification will be to the data type, and the third will be to add a column.

First, by using the alter table statement with the add clause, you'll turn the column name into a unique key. This is a third kind of key category, with the others being the primary and foreign keys.

```
mysql> alter table author add unique(name);
```

The MySQL server responds with an OK message.

```
Query OK, 0 rows affected (0.47 sec)
```

```
Records: 0  Duplicates: 0  Warnings: 0
```

To view the new table's structure and display the UNI (unique) key, use the `describe` SQL statement.

```
mysql> describe author;
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+
| author_id  | int(11)   | NO   | PRI | NULL    | auto_increment |
| name        | varchar(30) | YES  | UNI | NULL    |                 |
| country     | char(2)    | YES  |      | NULL    |                 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

To test the unique key, use the `insert` SQL statement with a value for the column name already entered.

```
mysql> insert into author(name, country) values('Eco, Umberto', 'IT');
```

The duplicated record is not allowed in the table, and the web server responds with an error message.

```
ERROR 1062 (23000): Duplicate entry 'Eco, Umberto' for key 'name'
```

As the second example of the `alter table` statement, you will change the data type of a column. For the `author` table, try first to enter a record with a name value larger than 30 characters.

```
mysql> insert into author (name, country) values('Solzhenitsyn, Aleksandr Isayevich', 'RU');
```

The MySQL server responds with an error message.

```
ERROR 1406 (22001): Data too long for column 'name' at row 1
```

Use the `alter table` SQL statement to modify the data type of the `name` column.

```
mysql> alter table author modify name varchar(255);
```

The MySQL server responds with an OK message.

```
Query OK, 0 rows affected (0.56 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

CHAPTER 6 THE MYSQL DATABASE SERVER

Try to enter the previous entry again with the long name.

```
mysql> insert into author (name, country) values(Solzhenitsyn, Aleksandr Isayevich', 'RU');
```

This time MySQL responds with an OK message.

```
Query OK, 1 row affected (0.07 sec)
```

With the select statement, you can view the previous entry of the author table.

```
mysql> select * from author;
```

MySQL displays the result.

author_id	name	country
1	Tolkien, John Ronald Reuel	UK
2	Steinbeck, John Ernst Jr.	US
3	Eco, Umberto	IT
6	Solzhenitsyn, Aleksandr Isayevich	RU

4 rows in set (0.00 sec)

Use the describe SQL statement to view the new data type, reflected in the table's structure.

```
mysql> describe author;
```

The command's output displays a data type of varchar(255) instead of varchar(30).

Field	Type	Null	Key	Default	Extra
author_id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(255)	YES	UNI	NULL	
country	char(2)	YES		NULL	

3 rows in set (0.01 sec)

For the third modification, you will add a new column to the book table, for instance pub_date, corresponding to the publication date of each book. Like most modern programming environments, MySQL supports a large set of data types. Common values such as date, time, or year are expected to have their own type. This is the case with the year type that will be used next. At the mysql prompt, enter the following:

```
mysql> alter table book add pub_date year;
```

MySQL responds with an OK message.

```
Query OK, 0 rows affected (1.16 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

To view the new book structure, use the describe statement.

```
mysql> describe book;
```

The MySQL server responds by displaying the new structure of the book table.

Field	Type	Null	Key	Default	Extra
book_id	char(13)	NO	PRI	NULL	
title	varchar(255)	YES		NULL	
language	varchar(20)	YES		NULL	
author_id	int(11)	NO	MUL	NULL	
pub_date	year(4)	YES		NULL	

5 rows in set (0.00 sec)

Testing the Table Connection

By using a foreign key, the two database tables are connected with a parent-child relationship. In this example, the parent table is author, which provides its primary key as a foreign key to the book table. You can't delete the author table if you have not already deleted the book table. Also, you can't insert a book entry with a foreign key value that does not exist.

In the following command, the nonexistent value 8 of the foreign key is used:

```
mysql> insert into book (book_id, title, language, author_id, pub_date)
values('9780743273565','The Great Gatsby', 'English',8,2004);
```

The MySQL server responds with an error message.

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails (`library`.`book`, CONSTRAINT `book_ibfk_1` FOREIGN KEY
(`author_id`) REFERENCES `author` (`author_id`))
```

Enter some records in the table book to be used in the query examples. For instance, run the following insert commands (the output of each command is also included):

```
mysql> insert into book (book_id, title, language, author_id, pub_date)
values('9780140177374','The Pearl','English',2,2000);
Query OK, 1 row affected (0.06 sec)
```

```
mysql> insert into book (book_id, title, language, author_id, pub_date)
values('9782806272836','De ratones y hombres','Spanish',2,2016);
Query OK, 1 row affected (0.07 sec)
```

```
mysql> insert into book (book_id, title, language, author_id, pub_date)
values('9780547928227','The Hobbit','English',1,2012);
Query OK, 1 row affected (0.06 sec)
```

```
mysql> insert into book (book_id, title, language, author_id, pub_date)
values('9780547928197','The Return of the King','English',1,2012);
Query OK, 1 row affected (0.07 sec)
```

```
mysql> insert into book (book_id, title, language, author_id, pub_
date) values('9780143105459','The Acts of King Arthur and His Noble
Knights','English',2,2008);
Query OK, 1 row affected (0.06 sec)
```

In the previous commands, since book_id is the primary key for book, the book_id values must be unique. For the values of the book_id column, the ISBN-13 of each book was chosen because it's unique to each book.

To view the book records, do a basic query by selecting all (*) records.

```
mysql> select * from book;
```

MySQL outputs the following result:

```
+-----+-----+-----+-----+
| book_id      | title                  | language | author_id | pub_date |
+-----+-----+-----+-----+
| 9780140177374 | The Pearl              | English   |          2 |    2000   |
| 9780143105459 | The Acts of King  
Arthur and His Noble  
Knights           | English   |          2 |    2008   |
| 9780547928197 | The Return of the King | English   |          1 |    2012   |
| 9780547928227 | The Hobbit              | English   |          1 |    2012   |
| 9782806272836 | De ratones y hombres | Spanish   |          2 |    2016   |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

The two tables, connected now with a parent-child relationship and filled with records, are used in the following section for extracting information from the database. In other words, you'll next query the database.

Performing SQL Queries with the MySQL Server

You have used the `select` statement so far in its simplest form and with the asterisk wildcard. `select` can be combined with functions such as `avg` (average), `count`, `max`, `min`, `round`, etc. The `where` clause is used to filter records by providing a condition that must be met. For instance, to count the number of book records with the language set to English, use the following:

```
mysql> select count(book_id) from book where language='English';
```

The result is four books.

```
+-----+
| count(book_id) |
+-----+
|          4 |
+-----+
1 row in set (0.00 sec)
```

You'll now use the `select` statement to execute a query that selects columns from both tables. To specify a column, the table's name must be prepended to the column's name, separated by a period. For instance, using this notation, `book.author_id` discriminates from `author.author_id`. The outcome of this combination is determined by the `join` clause. In the next example, the `inner join` clause is used, which selects for the output only records that have matching values in *both* tables.

```
mysql> select book.title, author.name, book.pub_date from author inner join book on book.author_id=author.author_id where book.language='English';
```

The `inner join` includes records of the two tables that have the same value in their `author_id` columns. The columns for the records included are the title (from `book`), the name (from `author`), and the publication date (from `author`) where the language column in the `book` table is English.

MySQL displays the following output:

title	name	pub_date
The Pearl	Steinbeck, John Ernst Jr.	2000
The Acts of King Arthur and His Noble Knights	Steinbeck, John Ernst Jr.	2008
The Return of the King	Tolkien, John Ronald Reuel	2012
The Hobbit	Tolkien, John Ronald Reuel	2012

4 rows in set (0.00 sec)

The following variation of the previous command sorts the records according to the `pub_date` column in descending order:

```
mysql> select book.title, author.name, book.pub_date from author inner join book on book.author_id=author.author_id where book.language='English' order by book.pub_date desc;
```

```
+-----+-----+
| title | name | pub_date |
+-----+-----+
| The Return of the King | Tolkien, John Ronald Reuel | 2012 |
| The Hobbit | Tolkien, John Ronald Reuel | 2012 |
| The Acts of King Arthur and His Noble Knights | Steinbeck, John Ernst Jr. | 2008 |
| The Pearl | Steinbeck, John Ernst Jr. | 2000 |
+-----+-----+
4 rows in set (0.00 sec)
```

The following query includes a count function and an `inner join` clause. The `group by` clause groups the results by the column `author.name`.

```
mysql> select count(book.title), author.name from author inner join book on book.author_id=author.author_id group by author.name;
```

As the output indicates, for the first author, three books are included in the database and for the second two books are included.

```
+-----+
| count(book.title) | name |
+-----+
| 3 | Steinbeck, John Ernst Jr. |
| 2 | Tolkien, John Ronald Reuel |
+-----+
2 rows in set (0.00 sec)
```

In the following command, two columns participate in the `group by` clause.

```
mysql> select count(book.title), author.name, pub_date from author inner join book on book.author_id=author.author_id group by author.name, pub_date;
```

The query's result is as follows:

```
+-----+-----+
| count(book.title) | name           | pub_date |
+-----+-----+
|      1 | Steinbeck, John Ernst Jr. | 2000   |
|      1 | Steinbeck, John Ernst Jr. | 2008   |
|      1 | Steinbeck, John Ernst Jr. | 2016   |
|      2 | Tolkien, John Ronald Reuel | 2012   |
+-----+-----+
4 rows in set (0.00 sec)
```

In the previous table, the output is grouped from right to left, first by publication date (two records have the value 2012) and then by author's name.

Modifying Records with the update Command

Use the following update and set statements to change an author's name:

```
mysql> update author set name='Tolkien, J.R.R.' where name='Tolkien, John
Ronald Reuel';
```

To view the changes, display the author table's contents.

```
mysql> select * from author;
```

The MySQL server displays the following result:

```
+-----+-----+
| author_id | name           | country |
+-----+-----+
|      1 | Tolkien, J.R.R. | UK      |
|      2 | Steinbeck, John Ernst Jr. | US      |
|      3 | Eco, Umberto | IT      |
|      6 | Solzhenitsyn, Aleksandr Isayevich | RU      |
+-----+-----+
4 rows in set (0.01 sec)
```

Using the SQL like Operator

A useful operator, used with the where clause, is like. You can use it to query a database when the user wants to specify a keyword but recalls only part of it. With like, you can query for a pattern that is included in the value of a column. For instance, to look for the title, author, and publication date (year) of a book that includes the word *king* in its title, you can use the like clause as shown here:

```
mysql> select book.title, author.name, book.pub_date from author inner join
book on book.author_id=author.author_id where book.title like '%king%';
```

The command's output is as follows:

title	name	pub_date
The Acts of King Arthur and His Noble Knights	Steinbeck, John Ernst Jr.	2008
The Return of the King	Tolkien, J.R.R.	2012

2 rows in set (0.00 sec)

Two wildcard characters are used with like.

- The percentage (%) wildcard matches any string of zero or more characters.
- The underscore (_) wildcard matches a single character.

You'll next use the previous command in another way. This time the title must end with the word *king*.

```
mysql> select book.title, author.name, book.pub_date from author inner join
book on book.author_id=author.author_id where book.title like '%king%';
```

CHAPTER 6 THE MYSQL DATABASE SERVER

The command's output is as follows:

```
+-----+-----+-----+
| title          | name           | pub_date |
+-----+-----+-----+
| The Return of the King | Tolkien, J.R.R. |      2012 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

You can try to delete the two tables used so far. Try first to delete the parent table author.

```
mysql> drop table author;
```

MySQL displays the following error message:

```
ERROR 1217 (23000): Cannot delete or update a parent row: a foreign key constraint fails
```

The constraint, which exists because of the parent-child relationship, does not allow the parent table to be deleted without previously deleting the child table. So, use the `drop table` statement to delete the child table, book.

```
mysql> drop table book;
```

The MySQL server responds with the following output:

```
Query OK, 0 rows affected (0.23 sec)
```

Now you can delete author.

```
mysql> drop table author;
```

The table is deleted this time with no complaints:

```
Query OK, 0 rows affected (0.17 sec)
```

Web Scraping with MySQL and the Linux Shell

In the previous examples, you manually inserted records into your database. Filling your database one record at a time is certainly tedious. For more realistic examples, you can use automated methods to transfer your data to one or more tables.

In this section, you will implement what is commonly called *web scraping*. This is the detection and collection of data from various web pages and inserting it into a file (e.g., a spreadsheet) or more appropriately into a database. The Bash shell scripting language and MySQL will be used for this project. Writing a Linux shell script that applies only to a certain web site may seem like overkill; however, this may help you process thousands of data records in some cases.

The URLs Describing the Resources

To start with web scraping, you have to identify the format in which the data is encoded for a specific portal. To search for data in specific web pages, you first have to describe their URLs in a systematic way. Consider, for instance, [amazon.com](https://www.amazon.com). By visiting any product page, you can find that what uniquely identifies a product for Amazon is the Amazon Standard Identification Number (ASIN) code. This is a 10-character alphanumeric unique identifier.

The ASIN is included in the URL of the specific product. For example, here is one of the products returned from searching using the keyword hi-fi:

```
https://www.amazon.com/Sharp-XLHF102B-HI-Component-MicroSystem/dp/  
B00XWIVTXY/ref=sr_1_2?s=amazon-devices&ie=UTF8&qid=1535449079&sr=8-  
2&keywords=hi-fi
```

Figure 6-2 displays the web page with the previous URL.

CHAPTER 6 THE MYSQL DATABASE SERVER

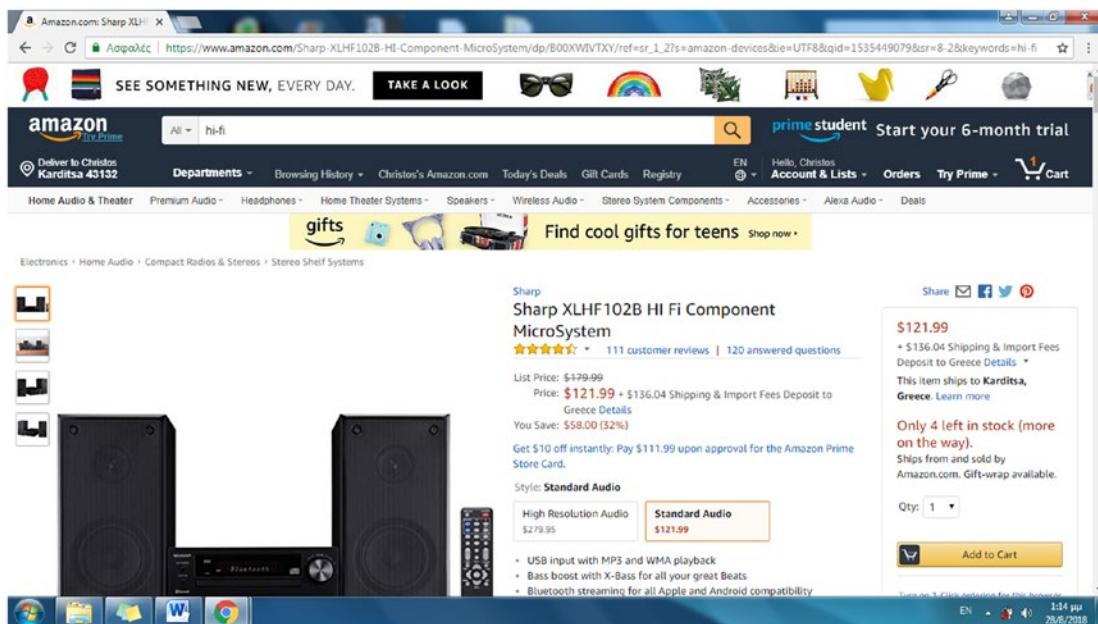


Figure 6-2. The ASIN is included in the product's URL

Usually an identifier like the ASIN in this example specifies the product's web page with a more straightforward URL. Try, for instance, the following:

<https://www.amazon.com/dp/B00XWIVTXY/>

The same web page is rendered with the simplified format, as shown in Figure 6-3.

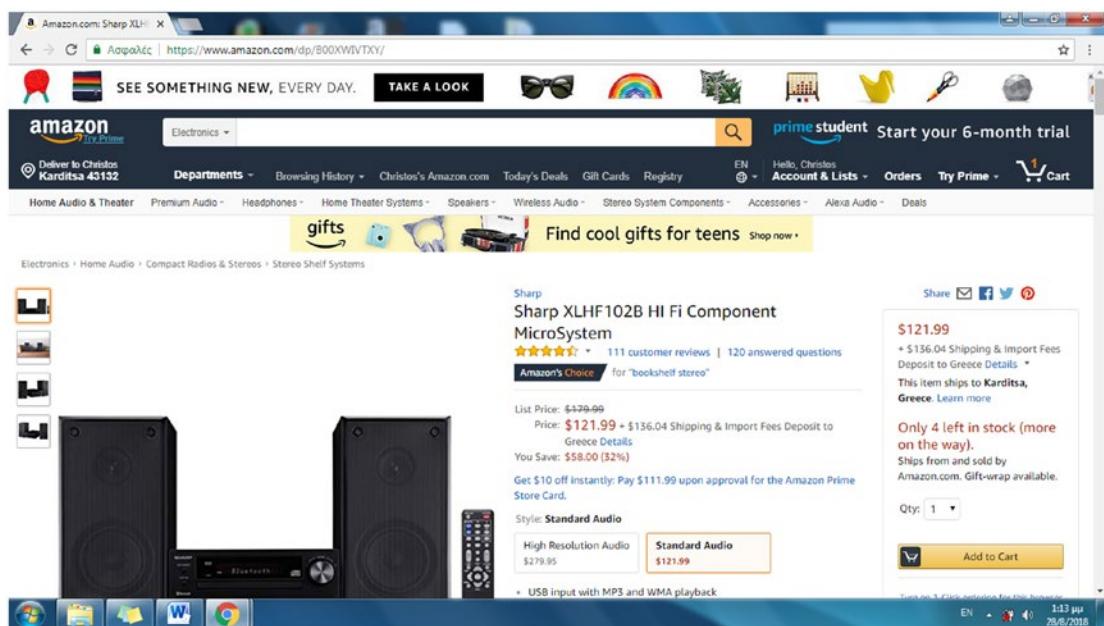


Figure 6-3. The ASIN included in the simplified URL format of a product

A unique identifier is used in most commercial sites. For web sites that deal with books, there is no need to use any other product identifier because each book is already identified in a unique way with its ISBN. ISBN stands for International Standard Book Number and currently is used in two formats.

- A 13-digit ISBN, used for books released after the January 1, 2007
- A 10-digit ISBN, used for books released before January 1, 2007

Apress.com identifies its products with the 13-digit ISBN identifier. Visit the Apress home page at apress.com, displayed in Figure 6-4, and then click any link advertising an Apress book.

CHAPTER 6 THE MYSQL DATABASE SERVER

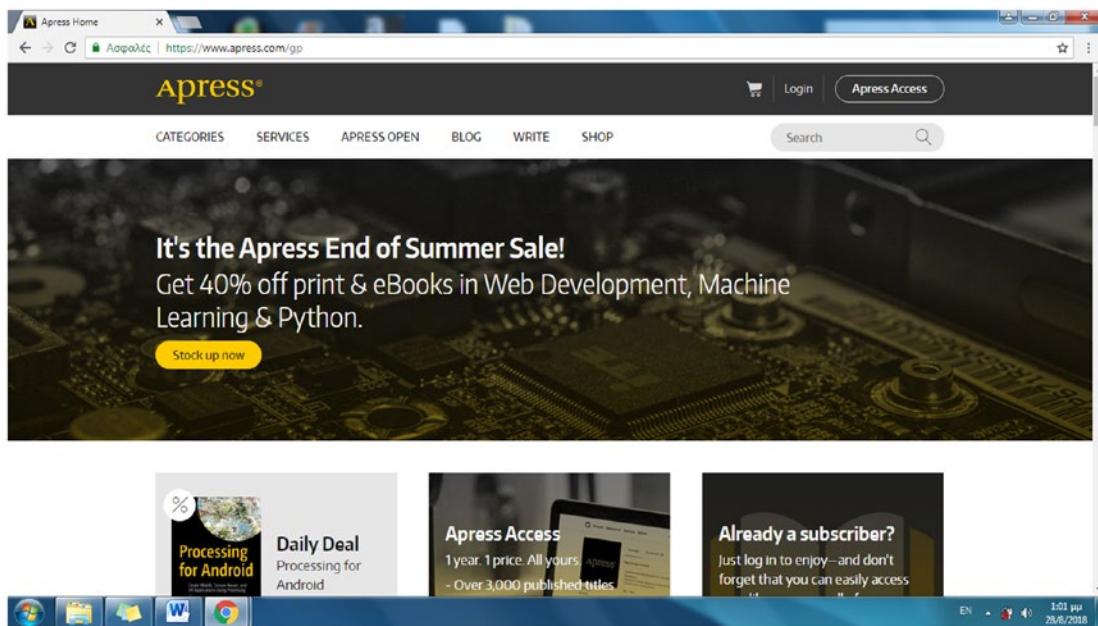


Figure 6-4. The Apress home page

As viewed in Figure 6-5, the URL in the address bar for the specific example is as follows:

```
https://www.apress.com/gp/book/9781484227183?wt_mc=Internal.Banner.3.EPR868.APR_DotD_Teaser#otherversion=9781484227190
```

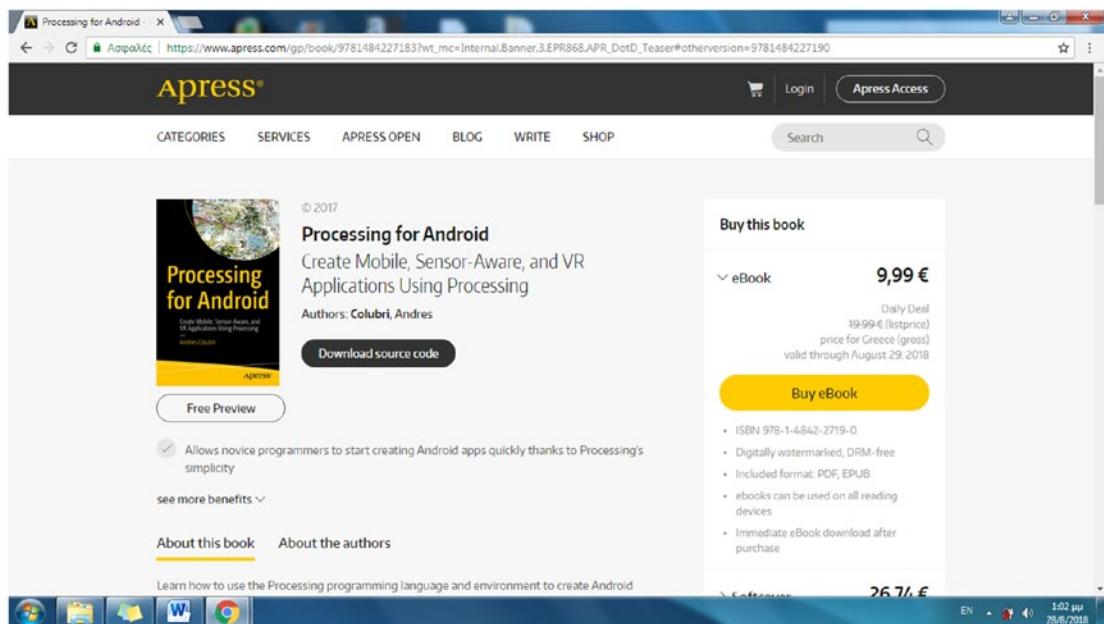


Figure 6-5. A product page on the Apress site identifies a book with the ISBN in the URL

Notice also that (similarly to the previous [Amazon.com](#) example) the URL can be simplified. For instance, the following URL leads to the same web page:

<https://www.apress.com/gp/book/9781484227183>

The next step is to locate and extract the data format used in the HTML source code of each product web page. For e-commerce sites, the data format is stored in a consistent way by implementing a data layer. The structure that defines the data layer for this example, the `appDataLayer` struct, is discussed in the following section.

Designing the Web Scraping Project

The following is the plan for creating the scrapping project: The web scraping shell script receives as an argument the URL of an Apress web page, which includes books of a certain category. It downloads this web page and then searches in the HTML source code of this web page for URLs of web pages that represent certain books. For each URL, the Bash shell script will perform an HTTPS connection to download the corresponding web page. The shell script will examine the web page's

CHAPTER 6 THE MYSQL DATABASE SERVER

data layer, found behind the scenes, in the HTML source code for values of specific book attributes such as for the title, ISBN, and price. These values will be stored in a MySQL database table that will be created for gathering the data acquired with the previous web scraping technique.

Visit first the Apress home page and hover your mouse over the Categories menu at the top of the web page. As shown in Figure 6-6, a number of menu choices appear, with each one corresponding to a specific book category.

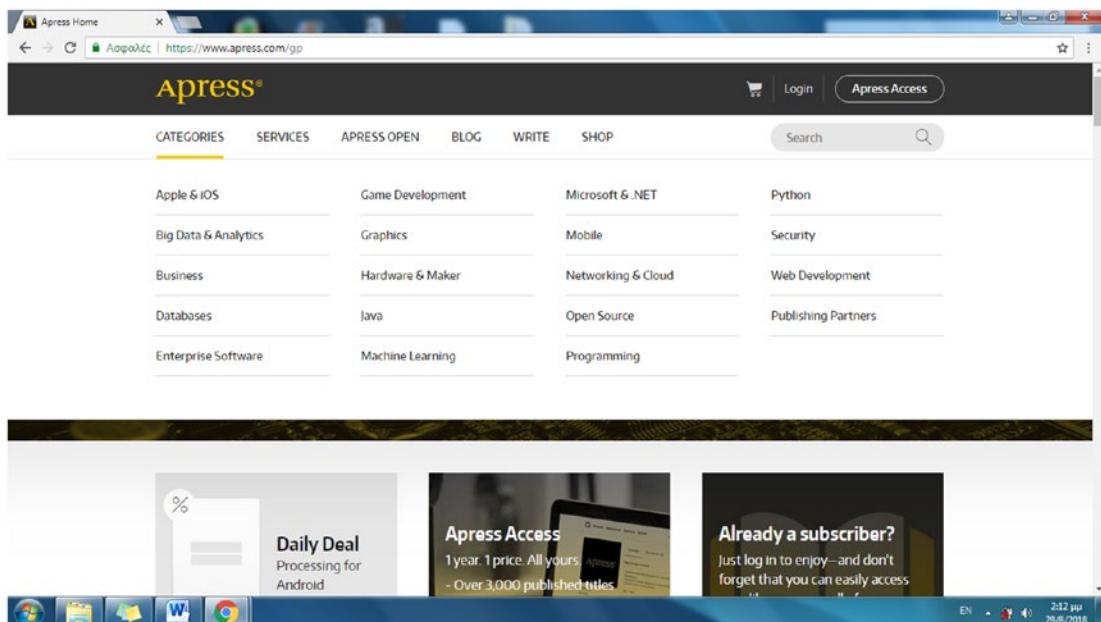


Figure 6-6. The Categories menu on the Apress home page

For this project, you can run the script for some (or even one) of these categories and collect the attributes of the books appearing in each category. Let's visit one of those categories, for instance, Web Development. By clicking this category's link, the web page displayed in Figure 6-7 appears.

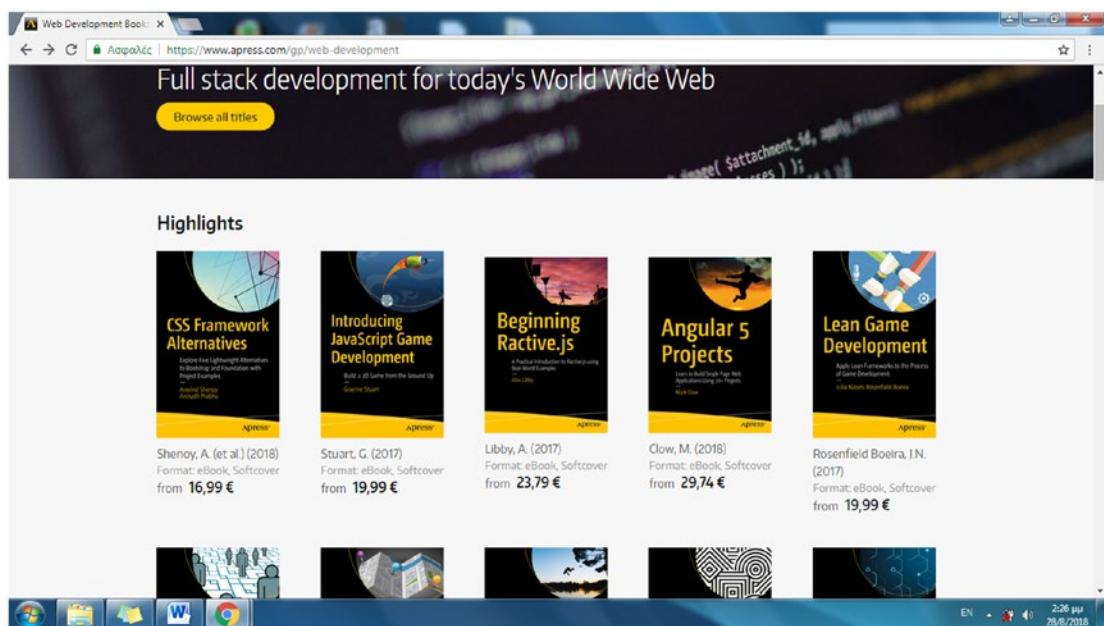


Figure 6-7. The books included in the Web Development category of Apress

On this web page, a number of images representing the books appear. Those images are links to web pages that provide descriptions to the corresponding books.

Hint! Each option in the Categories menu leads to a web page that includes only a portion of the category's books. It is, however, sufficient for examining the web scraping method here. To view all the books in each category, follow the “Browse all titles” link in each category.

Notice the URL of the category in the address bar.

<https://www.apress.com/gp/web-development>

Similarly, the URL is formed for any other category by concatenating two parts.

<https://www.apress.com/gp/>

It uses the category name, in lowercase, with the space between the words substituted by a hyphen (-).

For instance, for the Open Source category, the URL is as follows:

<https://www.apress.com/gp/open-source>

This consistency can be used to further automate the process and use the script to search multiple categories; however, to simplify the shell script for this example, one category will be used to feed the script each time it runs. The following command will be used at the Linux terminal for the web scraping process:

```
$ ./shell.sh https://www.apress.com/gp/web-development apress book  
pPriceGross fn isbn
```

The following are the parts that make up the command:

- `shell.sh` is the name of the shell script that performs the web scraping.
- `https://www.apress.com/gp/web-development` is the URL of the category web page, used to provide the URLs of the books included in the category web page, which finally will be downloaded and searched for the `pPriceGross`, `fn`, and `isbn` values of each book.
- `apress` is the database name that will be used to store the information.
- `book` is the `apress` database's table that includes the `pPriceGross`, `fn`, and `isbn` fields.
- `pPriceGross`, `fn`, and `isbn` are three fields found in the data layer of each page that provide values for the price, the name, and the ISBN, respectively. The number of fields this script supports is variable, which means that the same script can also run for two or five fields without changing any part of the source code.

The web page of the URL provided to the shell script command is the category page that includes a number of URLs leading to product web pages, each one dedicated to a single book. The product URLs have to be retrieved by the script and to be visited to extract the values of the `pPriceGross`, `fn`, and `isbn` fields of the data layer struct.

On the category page, all product pages are included between the start and ending <h3> tags. For instance, a product page link is as follows:

```
#<h3><a href="/gp/book/9781484233986" onmousedown="wt.sendinfo({linkId:  
&#039;recommendation&#039;, customClickParameter : { 2 : &#039;shoppage.  
recommendedproducts - 1&#039;}});" data-baynote-pid="978-1-4842-3398-6">CSS  
Framework Alternatives</a></h3>
```

By including product links in the heading tags of a specific size, size 3 in this case, the designer of the site separated them from other links that appear in the page. For each product, the shell script extracts the ISBN, found after the /gp/book/ directory in the URL, and then uses this ISBN to form the product's simplified URL. As you recall from the previous section, this has the following form:

<https://www.apress.com/gp/book/{ISBN}>

Here's an example:

<https://www.apress.com/gp/book/9781484233986>

Collecting the URLs of the products included in the category page is the first part of the script's functionality. The second part is visiting all those URLs and extracting from the HTML source code of the web pages the information about the specified fields (pPriceGross, fn, and isbn in this example). To locate where those fields are stored in each product page, visit a book's URL from <https://www.apress.com/gp/web-development>. For instance, visit <https://www.apress.com/gp/book/9781484233986>, which is shown in Figure 6-8.

CHAPTER 6 THE MYSQL DATABASE SERVER

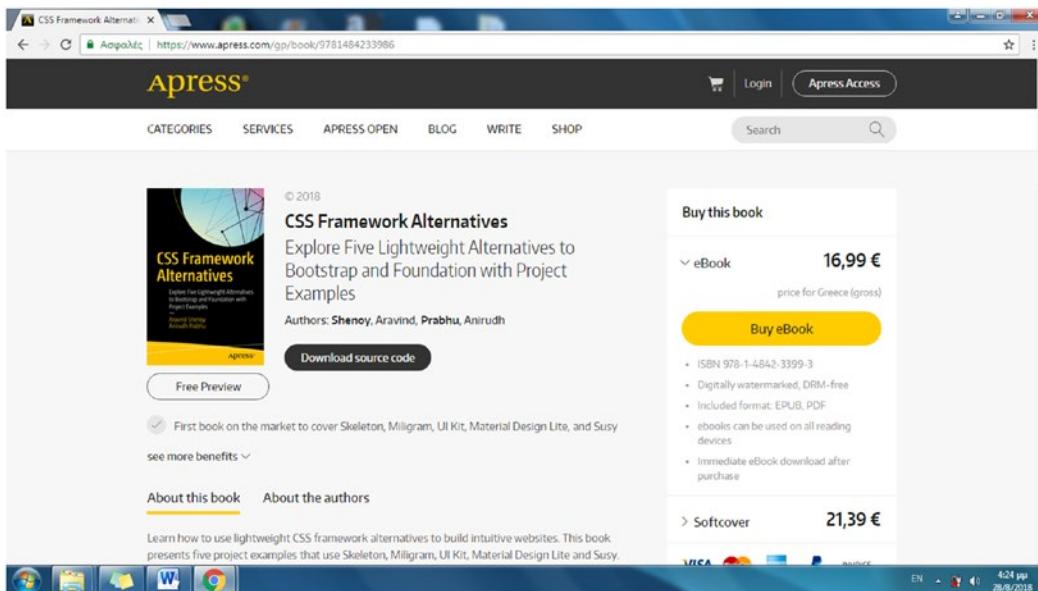


Figure 6-8. Visiting an Apress book web page to view the HTML source code and locate the fields required for the shell script

Right-click the web page and choose View Page Source from the pop-up menu. As shown in Figure 6-9, the web page's source code appears on a new tab in your browser.

The screenshot shows the raw HTML source code of the 'CSS Framework Alternatives' page. The code includes script tags containing JavaScript logic for managing a shopping cart or wishlist, and a large block of JSON-like data representing the book's details and variants. The JSON data includes fields such as 'name', 'id', 'price', 'brand', 'category', and 'variant' for both the main book and its softcover version.

```

<!-->
<script>
    current[key] = update[key];
}
return current;
}
</script>
<script type="text/javascript">
    appUpdateLayer = [
        {
            "pageType": "product",
            "topLevelCategory": "Computer",
            "productCategory": "Computer",
            "isbn": "9781484233986",
            "price": "16.99",
            "splicedPrice": "21.39",
            "ePrice": "16.99",
            "eISBN": "9781484233993",
            "pISBN": "9781484233986",
            "fn": "CSS Framework",
            "description": "CSS Framework Alternatives",
            "currency": "EUR",
            "url": "https://www.apress.com/978-1-4842-3398-6",
            "photo": "https://images.springer.com/sgw/books/medium/9781484233986.jpg",
            "ecommerce": {
                "currencyCode": "EUR",
                "details": [
                    {
                        "products": [
                            {
                                "name": "CSS Framework Alternatives",
                                "id": "978-1-4842-3399-3",
                                "price": "16.99",
                                "brand": "Apress",
                                "category": "Computer Science",
                                "variant": "paperback"
                            },
                            {
                                "name": "CSS Framework Alternatives",
                                "id": "978-1-4842-3398-6",
                                "price": "21.39",
                                "brand": "Apress",
                                "category": "Computer Science",
                                "variant": "softcover"
                            }
                        ]
                    }
                ]
            }
        }
    ];
</script>

```

Figure 6-9. The HTML source code of an Apress web page for a specific book

In the first lines of the source code, you can detect the `appDataLayer` struct. It looks like the following:

```
<script type="text/javascript">
    appDataLayer = [
        {
            "pageType" : "product",
            "topLevelCategory" : "Computer",
            "productCategory" : "Computer",
            "isbn" : "9781484233986",
            "productId" : "9781484233986",
            "pPriceGross" : "21.39",
            "ePriceGross" : "16.99",
            "eIsbn" : "9781484233993",
            "pIsbn" : "9781484233986",
            "fn" : "CSS Framework",
            "description" : "CSS Framework Alternatives",
            "currency" : "EUR",
            "url" : "//www.apress.com/978-1-4842-3398-6",
            "photo" : "https://images.springer.com/sgw/books/medium/9781484233986.jpg",
            "ecommerce" : {
                "currencyCode" : "EUR",
                "detail" : {
                    "products" : [ {
                        "name" : "CSS Framework Alternatives",
                        "id" : "978-1-4842-3399-3",
                        "price" : "16.99",
                        "brand" : "Apress",
                        "category" : "Computer Science",
                        "variant" : "ebook"
                    }, {
                        "name" : "CSS Framework Alternatives",
                        "id" : "978-1-4842-3398-6",
                        "price" : "21.39",
                        "brand" : "Apress",
                    } ]
                }
            }
        }
    ]
</script>
```

```
        "category" : "Computer Science",
        "variant" : "softcover"
    } ]
}
},
"content" : {
    "authorization" : {
        "status" : false
    }
}
}
];
};
```

The data layer is a collection of information required to pass data to other systems or software. The data layer is included in the source code of the web page, and more specifically to the JavaScript source code, usually as an object or a variable and therefore is hidden from the user. The format of the data layer can be considered as variable-value pairs (for instance, variable `description` and value `CSS Framework Alternatives`).

This data layer struct, found in each product page, is the resource for the information that the web scraping script will gather. In the current example, the values of the three fields retrieved (`pPriceGross`, `fn`, and `isbn`) will be stored in a database table, which is created in the following section.

Creating the MySQL Database Used for the Web Scraping Project

To connect to the MySQL server with `mysql`, assuming that no MySQL password was set, use the following at the Linux terminal:

```
$ mysql -u root
```

The `mysql>` prompt appears to receive MySQL commands. To create a new database called `apress`, use the following command:

```
mysql> create database apress;
```

Select the specific database to apply the commands that will follow by entering the following:

```
mysql> use apress;
```

With the following command, you create a table called book, which has three fields.

- isbn, which is of type CHAR with a length of 13 characters
- fn, which is of type VARCHAR with a length up to 40 characters
- pPriceGross, of type DECIMAL, with four digits and two decimal places

```
mysql> create table book (isbn CHAR(13), fn VARCHAR(40), pPriceGross  
DECIMAL(4,2));
```

You can view the table created previously by displaying all the database tables with the `show tables` statement.

```
mysql> show tables;
```

The command's output is as follows:

```
+-----+  
| Tables_in_apress |  
+-----+  
| book |  
+-----+  
1 row in set (0.01 sec)
```

To view details about the table's format, use the following command:

```
mysql> describe book;
```

The command's output is as follows:

```
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| isbn        | char(13)   | YES  |     | NULL    |       |
| fn          | varchar(40) | YES  |     | NULL    |       |
| pPriceGross | decimal(4,2) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

In the following section, you will create a Bash shell script that runs the web scraping project to download the web content, search for the specified fields, and fill the database.

Implementing the Web Scraping Project

`shell.sh` is the shell script, implemented with the Bash scripting language, that will be used for the web scraping project. It is roughly separated into two parts. The first finds all hyperlinks included between the `<h3>` and `</h3>` tags and extracts the ISBN, e.g., `9781484233986` in this example:

```
<h3><a href="/gp/book/9781484233986" onmousedown="wt.sendinfo({linkId:  
&#039;recommendation&#039;, customClickParameter : { 2 : &#039;shoppage.  
recommendedproducts - 1&#039;}});" data-baynote-pid="978-1-4842-3398-6">CSS  
Framework Alternatives</a></h3>
```

The ISBNs from all links are stored in the file `file.txt`. Then, in the second part of the shell code, the ISBNs are used to form the simplified product URLs like the following:

`https://www.apress.com/gp/book/9781484233986`

All URLs are then visited, and the `appDataLayer` array in each web page is used to provide the values for the `isbn`, `fn`, and `pPriceGross` fields. Finally, the fields are stored in a database table.

To create the file `shell.sh` in your home directory, enter the following in the Linux terminal:

```
$ cd ~  
$ gedit shell.sh
```

Insert the following lines in shell.sh:

```
#!/bin/bash
# USAGE:
# ./shell.sh https://www.apress.com/gp/web-development apress book
#pPriceGross fn isbn

# 1st PART
# store arguments in a special array
args=("$@");
# get number of elements
ELEMENTS=$#;

# connect to the URL indicated by the 2nd argument, argument $1 (the first,
#$0, is the script name)
# fill file.txt with ISBNs of the books
wget -q -O - $1 | \
grep '<h3><a href="/gp/book/' | \
sed 's:..*<a href="/gp/book/::' | \
awk -F'""' '{print $1}' > file.txt;

# 2nd PART
# get all book urls of a web page
for line in `cat file.txt`;
do

echo -e "\n-----\n";
a="https://www.apress.com/gp/book/";
b=$a$line;
echo $b;

# for the number of fields to be retrieved
for ((i=3;i<$ELEMENTS;i++)); do
output=$( \
wget -q -O - $b | \
grep -m 1 ${args[$i]} | \
awk -F ":" '{print $2}' | \
sed 's/,//g' | \
```

CHAPTER 6 THE MYSQL DATABASE SERVER

```
sed 's/\//g');
```

```
args2[$i]="$output;
```

```
done
```

```
mysql=${args[3]}
```

```
for ((i=3+1;i<$ELEMENTS;i++)); do
```

```
sql=${args[$i]}
```

```
mysql="$mysql, $sql";
```

```
done
```

```
echo $mysql;
```

```
mysql2='${args2[3]}';
```

```
mysql2=$(echo $mysql2 | sed 's/ //');
```

```
for ((i=3+1;i<$ELEMENTS;i++)); do
```

```
sql2=${args2[$i]}
```

```
sql2=$(echo $sql2 | sed 's/ //');
```

```
mysql2="$mysql2, '$sql2'";
```

```
done
```

```
echo -e "$mysql2\n";
```

```
sqlstring="INSERT INTO $3 ($mysql) VALUES($mysql2);";
```

```
mysql --user="root" --database="$2" -e "$sqlstring";
```

```
echo $sqlstring;
```

```
done
```

```
echo -e "\n-----\n";
```

```
rm -f file.txt;
```

To provide execute (x) file permission to yourself as the owner user (u) of the shell script so that you can execute it like any other program at the terminal, use the `chmod` (change mode) command. If your shell file is located in your home directory, enter the following at the Linux terminal:

```
$ sudo chmod u+x ~/shell.sh
```

The `shell.sh` script makes use of two classic but still powerful Unix tools used for text manipulation: `awk` and `sed`. It utilizes also the commands `grep` and `cat` and the `wget` program. A short intro to those commands follows:

- `awk` takes its name from the initials of its developers Aho, Weinberger, and Kernighan. It is a utility language used for data extraction from text. `awk` is often combined with `sed`, and the most common usage is to extract a column from text.
- `sed` stands for Stream Editor and is capable of receiving text and performing operations such as appending, inserting, deleting, or substituting text. The latter is one of the most commonly used operations of `sed` and is the one applied in the current example.
- `grep` is a command-line utility for searching text for lines that match a regular expression (or just a piece of text). Its name comes from the text editor command `g/re/p` that means to globally (g) search a regular expression (re) and print (p).
- `cat` is used to read content of files or concatenate (hence the command's name) files.
- `wget` (web get) is used to fetch content of the web servers it connects to from the command line.

The Script's First Part

The first line of `shell.sh` sets Bash as the scripting language that will be used in this shell program:

```
#!/bin/bash
```

The next lines starting with a hash (#) are just comments. The comment illustrates by example the syntax of the `shell.sh` command.

```
# USAGE:  
# ./shell.sh https://www.apress.com/gp/web-development apress book  
pPriceGross fn isbn
```

Notice that the command in the previous comment is executed from the same directory where the shell script is located (`./`). When executing a Linux program, a relative or absolute directory path must be prepended, as shown here:

```
/home/christos/shell.sh https://www.apress.com/gp/web-development apress
book pPriceGross fn isbn
```

In the next section, you will enable `shell.sh` to run from any directory at the terminal with only its name provided.

The script starts with the following lines:

```
# 1st PART
# store arguments in a special array
args=("$@");
# get number of elements
ELEMENTS=$#;
```

In Bash, variable `$0` indicates the script's name, `$1` is the first argument of the command, `$2` is the second, and so on. The special variable `$@` indicates all arguments passed to the script. The arguments are stored in the `args` array with the following instruction:

```
args=("$@");
```

The `ELEMENTS` variable is assigned the number of arguments, indicated by the special variable `$#`:

```
ELEMENTS=$#;
```

This also could be indicated as `ELEMENTS=${#args[@]}`.

The next part of the code uses the Unix pipe (`|`) symbol to apply the Unix piping technique to a number of commands. By piping the output of the command, the left side of the symbol becomes input to the command on the right side. The backslash (`\`) is also used to indicate that the command is continued to the next line.

```
wget -q -O - $1 | \
grep '<h3><a href="/gp/book/' | \
sed 's:.*<a href="/gp/book/::' | \
awk -F''' '{print $1}' > file.txt;
```

In the first command, `wget` runs in quiet (flag `-q`) mode, outputs the content retrieved to `stdout` (flag `-O`), and connects to the URL indicated with the first argument (`$1`) of the shell program (the URL of the category web page).

The source code of the web page downloaded with `wget` is passed therefore to the `grep` command to extract the line including the string `<h3><a href="/gp/book/`, the fixed part of the hyperlink inserted in the `<h3>` header, which is the format of the links for products included in this web page.

The previous line is passed next as input to `sed`. The default symbol used as a delimiter in the `sed` command is the forward slash (/), but since this character is included in the string, any other symbol can be used instead, for instance a colon (:). The `sed` substitution (s) command has, with the colon delimiter, the following syntax:

```
sed 's:text_to_be_substituted:text_that_replaces_the_original:'
```

In this example, this applies as follows:

```
sed 's:.*<a href="/gp/book/::' | \
```

In the output provided by `grep`, the text `<a href="/gp/book/` with zero or more characters to the left (`.*`) is therefore substituted with empty text. Therefore, this text is deleted. Consider, for instance, the following line with the three dots indicating extra text to the left:

```
. . .<h3><a href="/gp/book/9781484233986" onmousedown="wt.sendinfo({linkId:  
&#039;recommendation&#039;, customClickParameter : { 2 : &#039;shoppage.  
recommendedproducts - 1&#039;}});" data-baynote-pid="978-1-4842-3398-6">CSS  
Framework Alternatives</a></h3>
```

When the previous `sed` command is fed this line, it outputs the following:

```
9781484233986" onmousedown="wt.sendinfo({linkId:  
&#039;recommendation&#039;, customClickParameter : { 2 : &#039;shoppage.  
recommendedproducts - 1&#039;}});" data-baynote-pid="978-1-4842-3398-6">CSS  
Framework Alternatives</a></h3>
```

This output is then passed as input to `awk`, which with the `-F` option defines the double quotes ("") that follow `-F` as the column separator. It then prints the first column, and the output is redirected with the greater-than symbol (>) from standard output to the file `file.txt`. The first column separated from the second with double quotes is the ISBN.

```
9781484233986
```

Therefore, when the script runs, `file.txt` includes a number of ISBNs, with each one placed on its own line.

The Script's Second Part

The second part of the script reads each line of `file.txt` using a `for` loop and assigns it during each iteration to the variable `line`. The variable `line` holds therefore an ISBN identifier.

```
# 2nd PART
# get all book urls of a web page
for line in `cat file.txt`;
do
```

In the `for` loop, each ISBN value is concatenated to the fixed string <https://www.apress.com/gp/book/> to form the simplified URL of the products pages. Here's an example:

<https://www.apress.com/gp/book/9781484233986>

In the following lines of the Bash code, variable `a`, which holds the fixed part of the URL, is concatenated with the variable `line`, which holds the specific ISBN to form the value of variable `b`, which is the product's URL:

```
a="https://www.apress.com/gp/book/";
b=$a$line;
echo $b;
```

Notice that the value of `b` is output to the terminal with the `echo` Bash command. The `echo` commands in the program are used to display the connections performed by the shell script while the script runs.

The code snippet that follows uses a `for` loop to make one iteration for any field entered as a shell argument:

```
for ((i=3;i<$ELEMENTS;i++)); do
output=$( \
wget -q -O - $b | \
grep -m 1 ${args[$i]} | \
awk -F ":" '{print $2}' | \
```

```

sed 's/,//g' | \
sed 's/\\"//g');

args2[$i]="$output;

done

```

Consider, for instance, the case that the `shell.sh` runs with the following command:

```
# ./shell.sh https://www.apress.com/gp/web-development apress book
pPriceGross fn isbn
```

In this case, the number of fields to be filled in the database is three (`pPriceGross`, `fn`, and `isbn`), and the loop iterates three times. The loop starts from number 3 because array `args[]` includes the following elements for the current example:

```

args[0] : https://www.apress.com/gp/web-development
args[1] : apress
args[2] : book
args[3] : pPriceGross
args[4] : fn
args[5] : isbn

```

For each field (arguments 3 to 5), a `wget` connection is created to the product's URL, and the source code of the web page fetched is searched with `grep` for the specific field (e.g., `pPriceGross`). The lines returned by `grep` are passed next to `awk`, which uses a colon (:) as the delimiter symbol to print the second column. Recall that the fields are found in the `appDataLayer[]` array of each page, which has the following format:

```

<script type="text/javascript">
    appDataLayer = [
        {
            "pageType" : "product",
            "topLevelCategory" : "Computer",
            "productCategory" : "Computer",
            "isbn" : "9781484233986",
            "productId" : "9781484233986",
            "pPriceGross" : "21.39",
            "ePriceGross" : "16.99",

```

```
"eIsbn" : "9781484233993",
"pIsbn" : "9781484233986",
"fn" : "CSS Framework",
. . .
```

The second column therefore consists of the values of the fields, with the field names consisting of the first column and with the two columns separated by colons. The values are included in double quotes and are suffixed with commas. For instance, when grep searches for the field fn, the second column of the line returned is as follows:

```
"CSS Framework",
```

The following two sed commands strip off the double quotes and the commas:

```
sed 's/,//g' | \
sed 's/\"//g');
```

The output is assigned to variable output. In this example, this is as follows:

```
CSS Framework
```

The value of output fills the corresponding element of a new array called args2[].

```
args2[$i]="$output;
```

Therefore, if args[4] currently has the value fn, args2[4] has the value of the field fn, in this example CSS Framework.

For each line of file.txt, the args2[3], args2[4], and args2[5] elements are filled.

The following code snippet concatenates the args[] elements with the field names args[3], args[4], and args[5] using commas:

```
mysql=${args[3]};
for ((i=3+1;i<$ELEMENTS;i++)); do
sql=${args[$i]};
mysql="$mysql, $sql";
done
```

shell.sh was used here with three fields, but it works equally for any number of fields. For example, if four fields were used, the previous code snippet would concatenate the elements args[3], args[4], args[5], and args[6] (the new field) using commas.

The code runs recursively and assigns the first field name to variable `mysql`, finds the next and concatenates the two fields with commas as `mysql` again, and repeats until all fields are included.

For the current example, the elements are `pPriceGross`, `fn`, and `isbn`. The variable `mysql` after the concatenation has the following value:

```
pPriceGross, fn, isbn
```

This string will be used to form the `sqlstring`, the 'INSERT INTO' SQL command, that will be used on the MySQL server to insert the data into the database's book table. An example of `sqlstring` is as follows:

```
INSERT INTO book (pPriceGross, fn, isbn) VALUES('35.3', 'Mastering Zoho Creator', '9781484229064');
```

To form the values part of the fields for the `sqlstring`, the values of the fields, previously placed in `args2[3]`, `args2[4]`, and `args2[5]` elements, must be enclosed in single quotes and be separated by commas. Here's an example:

```
'35.3', 'Mastering Zoho Creator', '9781484229064'
```

The following code snippet does this:

```
mysql2+"'${args2[3]}'";
mysql2=$(echo $mysql2 | sed 's/ //');
for ((i=3+1;i<$ELEMENTS;i++)); do
sql2=${args2[$i]};
sql2=$(echo $sql2 | sed 's/ //');
mysql2="$mysql2, '$sql2'";
done
```

In the previous source code, `mysql2` forms the string of all field values separated by commas with a recursive process. The reason the spaces need to be removed from the `mysql2` and `sql2` variables is that Bash currently prepends a space character to elements, which are read using a `for` loop from an array.

With the following code line, the MySQL query string, held in variable `sqlstring`, is finally formed:

```
sqlstring="INSERT INTO $3 ($mysql) VALUES($mysql2);";
```

For instance, the `sqlstring` may hold this value:

```
INSERT INTO book (pPriceGross, fn, isbn) VALUES('35.3', 'Mastering Zoho Creator', '9781484229064');
```

For this example, statement `INSERT INTO` inserts a record into the `book` table by entering value `35.3` in the `pPriceGross` field, value `Mastering Zoho Creator` in the `fn` field, and value `9781484229064` in the `isbn` field.

To issue the previous command, a connection is created with the `mysql` server. The value `root` is provided to the argument `--user`, and the value `apress` (the value of variable `$2`) is provided as the `--database` argument. The execution command (`-e`) is the value of the `sqlstring`.

```
mysql --user="root" --database="$2" -e "$sqlstring";
```

The previous command runs a number of times equal to the number of books included in the web page source of the URL provided as the first argument of the shell (`$1`).

The text file used in this shell program is deleted at the end of the script.

```
rm -f file.txt;
```

Testing the Web Scraping Shell Program

After creating the `shell.sh` script and the `book` table in the `apress` database, you can use your web scraping shell script to complete the table. At the Linux terminal, change the working directory to the one that includes the shell script, in this example the home directory, and enter the `shell.sh` command with the required arguments.

```
$ cd ~  
$ ./shell.sh https://www.apress.com/gp/web-development apress book  
pPriceGross fn isbn
```

Hint! To run the shell script from any working directory, copy the shell.sh file to /usr/local/bin.

```
$ sudo cp ~/shell.sh /usr/local/sbin
```

Execute it then from any working directory as follows:

```
$ sudo shell.sh https://www.apress.com/gp/web-development  
apress book pPriceGross fn isbn
```

In shell.sh, all programs run in silent mode; therefore, the only messages printed to the output are the echo command messages printed at the terminal. The message displayed in each iteration has the following format:

```
-----  
  
https://www.apress.com/gp/book/9781484237144  
pPriceGross, fn, isbn  
'35.3', 'RESTAPI Development with Node.js', '9781484237144'  
  
INSERT INTO book (pPriceGross, fn, isbn) VALUES('35.3', 'RESTAPI  
Development with Node.js', '9781484237144');
```

When the shell.sh script ends, open the apress database and print the book table contents. Use Ctrl+Alt+T to open a second terminal window. At the command line, enter the following:

```
$ mysql -u root
```

The MySQL server connects, and the mysql> prompt appears.

Enter the following commands to select the apress database and also view all the records of the table book.

```
mysql>use apress;  
mysql>select * from book;
```

The following is the output of the `select` command:

isbn	fn	pPriceGross
9781484233986	CSSFramework	21.39
9781484232514	IntroducingJavaScript Game	26.74
9781484230923	Beginning	32.09
9781484232781	Angular5	39.58
9781484232156	LeanGame Development	26.74
9781484232668	Discussionsin User Experience	29.95
9781484232811	IntroducingArcGIS API 4 for	32.09
9781484230268	Beginning	37.44
9781484229361	TheDefinitive Guide to	40.65
9781484228258	Electron	37.44
9781484239216	UsingYour Web Skills To Make	26.74
9781484238639	HTML5and JavaScript	40.65
9781484226094	WebApplications with	24.60
9781484210741	Scalability	24.56
9781484237144	RESTAPI Development with Node.js	35.30
9781484238639	HTML5and JavaScript	40.65
9781484236963	Programming	35.30

17 rows in set (0.00 sec)

Run the script a second time, using another URL from the ones included in the Categories link of the www.apress.com home page. For instance, use the following:

```
$ ./shell.sh https://www.apress.com/gp/open-source apress book pPriceGross
fn isbn
```

At the second terminal, enter again the last used command.

```
mysql>select * from book;
```

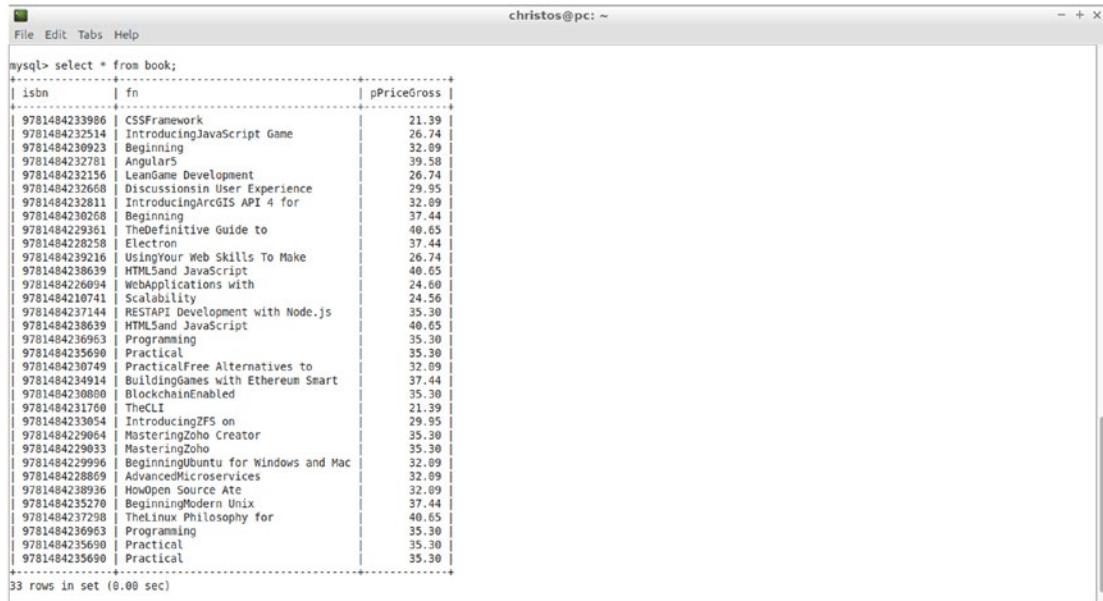
As indicated by the following output, more records are added to the database table (see Figure 6-10):

isbn	fn	pPriceGross
9781484233986	CSSFramework	21.39
9781484232514	IntroducingJavaScript Game	26.74
9781484230923	Beginning	32.09
9781484232781	Angular5	39.58
9781484232156	LeanGame Development	26.74
9781484232668	Discussionsin User Experience	29.95
9781484232811	IntroducingArcGIS API 4 for	32.09
9781484230268	Beginning	37.44
9781484229361	TheDefinitive Guide to	40.65
9781484228258	Electron	37.44
9781484239216	UsingYour Web Skills To Make	26.74
9781484238639	HTML5and JavaScript	40.65
9781484226094	WebApplications with	24.60
9781484210741	Scalability	24.56
9781484237144	RESTAPI Development with Node.js	35.30
9781484238639	HTML5and JavaScript	40.65
9781484236963	Programming	35.30
9781484235690	Practical	35.30
9781484230749	PracticalFree Alternatives to	32.09
9781484234914	BuildingGames with Ethereum Smart	37.44
9781484230800	BlockchainEnabled	35.30
9781484231760	TheCLI	21.39
9781484233054	IntroducingZFS on	29.95
9781484229064	MasteringZoho Creator	35.30
9781484229033	MasteringZoho	35.30
9781484229996	BeginningUbuntu for Windows and Mac	32.09
9781484228869	AdvancedMicroservices	32.09
9781484238936	HowOpen Source Ate	32.09
9781484235270	BeginningModern Unix	37.44

CHAPTER 6 THE MYSQL DATABASE SERVER

9781484237298 TheLinux Philosophy for	40.65
9781484236963 Programming	35.30
9781484235690 Practical	35.30
9781484235690 Practical	35.30
+-----+-----+-----+	

33 rows in set (0.01 sec)



```
christos@pc: ~
File Edit Tabs Help
mysql> select * from book;
+-----+-----+-----+
| isbn   | fn          | pPriceGross |
+-----+-----+-----+
| 9781484233986 | CSSFramework           | 21.39 |
| 9781484232514 | IntroducingJavaScript Game | 26.74 |
| 9781484230923 | BeginningAngular5        | 32.09 |
| 9781484232781 | Angular5               | 39.58 |
| 9781484232156 | LeanGame Development     | 26.74 |
| 9781484232668 | Discussionsin User Experience | 29.95 |
| 9781484232811 | IntroducingArcGIS API 4 for | 32.09 |
| 9781484230268 | BeginningElectron         | 37.44 |
| 9781484229361 | TheDefinitive Guide to Electron | 40.65 |
| 9781484228258 | UsingYour Web Skills To Make | 37.44 |
| 9781484229216 | HTML5and JavaScript       | 26.74 |
| 97814842238639 | HTML5and JavaScript       | 40.65 |
| 9781484226094 | MobileApplications with Scalability | 24.60 |
| 9781484227144 | RESTfulDevelopment with Node.js | 24.56 |
| 9781484237144 | RESTfulDevelopment with Node.js | 35.30 |
| 9781484238639 | HTML5and JavaScript       | 40.65 |
| 9781484236963 | Programming               | 35.30 |
| 9781484235690 | Practical                | 35.30 |
| 9781484230749 | PracticalFree Alternatives to BlockchainEnabled | 32.09 |
| 9781484234914 | BuildingGames with Ethereum Smart | 37.44 |
| 9781484230800 | BlockchainEnabled          | 35.30 |
| 9781484231769 | TheCLI                   | 21.39 |
| 9781484233054 | IntroducingZFS on ZFS      | 29.95 |
| 9781484229064 | MasteringZoho Creator     | 35.30 |
| 9781484223983 | MasteringZoho              | 35.30 |
| 9781484223996 | BeginningUbuntu for Windows and Mac | 32.09 |
| 9781484228869 | AdvancedMicroservices       | 32.09 |
| 97814842238936 | HowOpen Source Ate          | 32.09 |
| 9781484235270 | BeginningModern Unix       | 37.44 |
| 9781484237298 | TheLinux Philosophy for | 40.65 |
| 9781484236963 | Programming               | 35.30 |
| 9781484235690 | Practical                | 35.30 |
| 9781484235690 | Practical                | 35.30 |
+-----+
33 rows in set (0.00 sec)
```

Figure 6-10. The output

You can add even more records by using the rest of the Categories menu links.

Summary

In this chapter, you used the MySQL server to create, manage, and query databases using SQL commands. You inserted records into the database tables manually and also used the web scraping technique to fill the database automatically by collecting data provided by a site.

In the following chapter, you will run PHP programs that connect to the MySQL server, enabling the web server to provide a search capability and therefore offer dynamic web content to its sites.

CHAPTER 7

Creating a Dynamic Content Web Site

In this chapter, you'll create a site that provides a form for the user to submit keywords to the web server, query a database, and view the results. The results page will list hyperlinks that lead to web pages related to the keywords entered.

To do this, you will utilize the PHP language to make the connection to the MySQL server via the MySQL-PHP interface routines.

Instead of having the query results appear on a single page, you'll implement pagination to group the results on separate web pages, with images (buttons) used to switch back or forth from one web page to another.

Also, you'll implement an automated procedure, based on the cron daemon, that executes scheduled commands to update the MySQL database with the contents of the web pages included in the site.

Search-Enabled Site Overview

As discussed, in this project you'll create a search-enabled site. Allowing a user to perform queries remotely in a web environment is a feature commonly found on e-commerce sites, search engines, wikis, and online academic databases. You will create next a site that provides similar services as the rfc-editor.org site and also you can borrow RFCs for your site's content. Requests for Comments (RFCs) are the content offered from this site that the user will query. (RFCs are papers describing Internet protocols and technologies.)

The following is the URL to search RFCs at rfc-editor.org:

https://www.rfc-editor.org/search/rfc_search.php

CHAPTER 7 CREATING A DYNAMIC CONTENT WEB SITE

Figure 7-1 displays results for using the *HTTP* keyword in the RFC Editor. The results are links that lead to the corresponding RFCs. Pagination is also implemented to group them into 25 results per web page.

The screenshot shows a Microsoft Internet Explorer window with the title bar "RFC Search Detail" and the URL "rfc-editor.org/search/rfc_search_detail.php?title=HTTP&pubstatus%5B%5D=Any&pub_date_type=any". The main content area is titled "RFC Editor" and contains a search form with fields for "RFC Number (or Subseries Number)" and "Title/Keyword" (set to "HTTP"). There are checkboxes for "Show Abstract" and "Show Keywords", and a dropdown menu for "Additional Criteria". Below the form is a message "131 results (Show 25 | All)". A table lists 131 RFCs, showing columns for Number, Files, Title, Authors, Date, More Info, and Status. The first few entries are:

Number	Files	Title	Authors	Date	More Info	Status
RFC 1945	ASCII PDF	Hypertext Transfer Protocol -- HTTP/1.0	T. Berners-Lee, R. Fielding, H. Frystyk	May 1996		Informational
RFC 2002	ASCII PDF	Applicability of Standards Track MIBs to Management of World Wide Web Servers	C. Kalbfleisch	November 1996		Informational
RFC 2048	ASCII PDF	Hypertext Transfer Protocol -- HTTP/1.1	R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee	January 1997	Obsoleted by RFC 2616	Proposed Standard
RFC 2069	ASCII PDF	An Extension to HTTP : Digest Access Authentication	J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Simek, L. Stewart	January 1997	Errata: Obsoleted by RFC 2617	Proposed Standard
RFC 2109	ASCII PDF	HTTP State Management Mechanism	D. Kristol, L. Montulli	February 1997	Obsoleted by RFC 2965	Historic (changed from Proposed Standard to Obsolete)

Figure 7-1. Querying the RFC Editor for HTTP

Figure 7-2 displays the home page of the project's site.

The screenshot shows a Microsoft Internet Explorer window with the title bar "Mr. oospaMjč | webtoolsonline.servrhttp.com". The main content area has a large green background with a white search form in the center. The form contains the placeholder text "Search the Papers:" followed by a text input field and a "Go" button. At the bottom of the page, there is footer text: "Free Dynamic DNS and Port 80 Redirection Provided by [tiny.cc](#)" and "Who am I seeing this?". The status bar at the bottom shows "EN", "823 pix", and the date "8/9/2013".

Figure 7-2. The home page of the project's site

As displayed in Figure 7-3, the results appear as links to RFCs, and pagination is used to show five results per page.

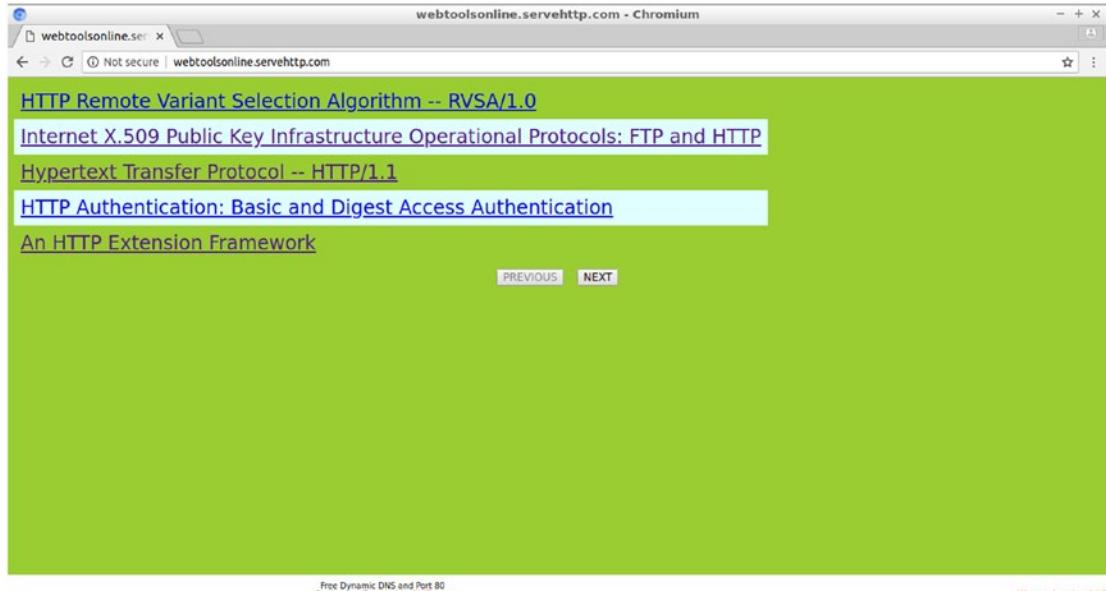


Figure 7-3. The results of querying the project's site for HTTP

Designing the Project

The site you'll implement requires a number of web pages that will be the content that the site offers; these pages are placed in a separate directory called `info` in this example. A Bash shell script running in regular intervals from `cron` will update a database with all the web page files currently included in the directory `info`. The title included in each “content” web page is what the user will remotely query in the database. The query will be issued from the home page where the user will enter keywords in an HTML form. A PHP program, after receiving the keywords, will connect to and query the database and return the results as links, with the title as the link text and the relative URL of the content web page file as the link's URL.

Creating the First Web Content Samples

You can place this project's content web pages in the new directory in the document root so that they are isolated from the directory index that provides the form and the action PHP program that processes the keywords. In this way, only the content web pages will be automatically inserted into the database. To create a new directory named `info`, enter the following command at the Linux terminal:

```
$ sudo mkdir /var/www/html/info
```

To create the first web page of this site, continue with the following commands:

```
$ cd info  
$ sudo gedit page1.html
```

Enter the following HTML source code:

```
<!DOCTYPE html>  
<html>  
<head>  
<title></title>  
</head>  
<body>  
<h1>Hypertext Markup Language - 2.0</h1>  
</body>  
</html>
```

This is an empty web page that includes a heading (`<h1>`) that is the RFC's title. The heading size here must be the same that the Bash shell will use for locating the title of each content web page. You can borrow the titles from the previous RFC Editor search. You can also enter some content in the body section of your page, which for simplicity is empty here other than the heading. What is required for this project is to create a large number of similar web pages with different titles to display how pagination is implemented. For the moment, just create a second content web page by using the following at the Linux terminal:

```
$ sudo cp page1.html page2.html  
$ sudo gedit page2.html
```

In the gedit window, change the title to another one from the RFC Editor and save the file. Use, for instance, the title **Form-based File Upload in HTML** in the **h1** element.

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body>
<h1>Form-based File Upload in HTML</h1>
</body>
</html>
```

You can continue this process to create lots of web pages for the site. The aim is to have many pages so that you can have a lot of query results.

Creating and Updating the Project's Database

Next, you will create the database that includes all the titles of the content web pages placed in the **info** directory. As described in the following section, the database update will be an automated process.

To create **info**, the database used in this project, and also **content**, the table included in the **info** database, connect to the MySQL server with **mysql**, the MySQL client, using the following command:

```
$ mysql -u root
```

At the **mysql>** prompt that appears, create the **info** database.

```
mysql> create database info;
```

The MySQL server responds with an OK command.

```
Query OK, 1 row affected (0.00 sec)
```

The database is included with the other databases of the MySQL server. You can view this with the following command:

```
mysql> show databases;
```

The MySQL server responds with the following table, with the names of all the databases included so far:

```
+-----+
| Database      |
+-----+
| information_schema |
| apress         |
| info           |
| library        |
| mysql          |
| performance_schema |
| sys            |
+-----+
7 rows in set (0.00 sec)
```

You can locate database `info` and database `apress`, used in the previous chapter, as well as all the preexisting MySQL databases. Select `info` as the database that the commands entered from this point on will target.

```
mysql> use info;
```

The MySQL server responds with the following message:

```
Database changed
```

Create a table called `content`. This table will include two columns: `url`, which is a `varchar` up to 255 characters, and `title`, which also is type `varchar`. The `url` column will store the root-relative URL path of a specific web page. For instance, the URL for `page1.html` will be `/info/page1.html`, where the first slash is the document root. The column `url` is a nice candidate for a primary key because it includes unique values; no files are allowed by the operating system to have the same name in the same directory. The `title` column corresponds to the title of the web page, included in the `<h1>` tags. For instance, for `page1.html`, the title is `Hypertext Markup Language - 2.0`. To create the table `content`, use the `create table` SQL statement.

```
mysql> create table content (url varchar(255), primary key(url), title varchar(255));
```

The MySQL server responds with an OK message.

```
Query OK, 0 rows affected (0.40 sec)
```

You don't have to manually enter any records because, as explained previously, this will be an automated process. Exit the MySQL client using the following command:

```
mysql> exit
```

MySQL responds with a good-bye message.

```
Bye
```

Writing the Shell Script That Updates the Database

Who will be responsible for updating the database so that each time a new web page is added to the site its title is included in the database? For this, many approaches can be used. For instance, it could be the responsibility of the person who places a new web page in the `info` directory to open MySQL and use an `insert into` command to store the title of the new page in the database. This sounds like it creates a little overhead, and if a person forgets to do this every time they add a page, the system will lose its reliability. Another approach is to create a shell script that hides the overhead from the administrator; however, forgetting to run the script would result in the same problem of unreliability. Another solution would be to automate the process and have the script run at regular intervals to update the database. Because this approach doesn't require anyone to intervene, it is less error prone.

To create `db.sh`, the Bash shell script that updates the database, change the working directory to your home directory, e.g., `/home/christos/`, with the following command:

```
$ cd ~
```

Use a text editor like gedit to create a file called `db.sh`.

```
$ gedit db.sh
```

In the new window that appears, enter the following lines of Bash code and save the file:

```

#!/bin/bash

FILES=/var/www/html/info/*

db="info";
table="content";
sqlstring="truncate table $table;";
mysql --user="root" --database="$db" -e "$sqlstring";

for file in $FILES
do
    title=`grep -o '<h1>.*</h1>' $file | sed 's/\\(<h1>\\|</h1>\\)//g'`;
    filename=$(basename "$file");
    path="/info/";
    url=$path$filename;
    records="url, title";
    my_values="'$url', '$title'";
    sqlstring="insert into $table ($records) values($my_values);";
    mysql --user="root" --database="$db" -e "$sqlstring";
done

```

The first line is a directive stating that the Bash language will be used for this script. Next comes a number of variable definitions. For instance, FILES means all (*) files found in /var/www/html/info. The variable db is assigned to info, which is the name of the database in this example. Also, table, the variable that holds the name of the database's tables, is assigned to the value content. The variable sqlstring temporarily holds the value:

```
"truncate table $table;"
```

That evaluates to the following string:

```
"truncate table content;"
```

One approach for the script is to enter only the titles of the HTML files that are not already included in the table. However, if the title of a specific file changes, the database will still include the previous one. The other approach is to delete all the database entries and add all the titles from HTML files located in info to the database. This script implements the second approach, and for this, the truncate table SQL statement is used to remove all the records from a table.

The db.sh script will run either manually, invoked by the user, or automatically in predefined time intervals. Each time the script runs, it truncates the table and then inserts the titles and file paths of the HTML files in the info directory.

The following is the command used by the shell script to connect to the MySQL server:

```
mysql --user="root" --database="$db" -e "$sqlstring";
```

It evaluates to the following command:

```
mysql --user="root" --database="info" -e "truncate table content;";
```

Hint! This is a command run from inside a shell program, but you can run this as a single command from the Linux terminal. After all, a shell program is a collection of shell commands.

After all the records in the table are deleted, you can use a for loop to insert the title and root-relative path of each HTML file in the info directory to the info database. The loop is included between the do...done commands.

```
for file in $FILES
do
...
done
```

For each file in the directory info, the line that includes the h1 element—whose text is the web page's title—is extracted with grep:

```
title=`grep -o '<h1>.*</h1>' $file | sed 's/\(<h1>\|</h1>\)//g'`;
```

By using the Unix pipe operator (|), the line output by grep is passed as input to sed. Then sed removes the start and end tags of the h1 element by replacing them with an empty character. The sed output, which is the title of the HTML page, is assigned as a value to the variable title.

The web page file name is returned to the variable filename when the command basename runs.

```
filename=$(basename "$file");
```

The file name is then concatenated to string /info/ to form url, the root-relative path.

```
path="/info/";
url=$path$filename;
```

For instance, for the file name page1.html, the path becomes /info/page1.html.

The title and url values (e.g., info/page1.html and Hypertext Markup Language - 2.0) will be entered in the corresponding table columns (url, title) with the sqlstring, which forms the SQL insert command.

```
records="url, title";
my_values="'$url', '$title'";
sqlstring="insert into $table ($records) values($my_values);";
```

The next command uses the mysql client to connect to the MySQL server and submit the command included in sqlstring, shown here:

```
mysql --user="root" --database="$db" -e "$sqlstring";
```

The following is an example of the command sent to the server after evaluating the variables:

```
mysql --user="root" --database="info" -e "insert into content (url, title)
values('/info/page1.html', 'Hypertext Markup Language - 2.0');";
```

To run the shell script, you need to grant executable (x) file permission at least to yourself, that is, to the user (u). The other options are to grant this right to your group (g) and all others (o). Use the chmod command from your home directory as follows:

```
$ chmod u+x db.sh
```

Run manually the shell script from your home directory as follows:

```
$ ./db.sh
```

The period before the slash denotes the current directory. This is used because when running executable Linux files, their relevant or absolute path needs to be provided except if the script is stored on one of the directories assigned to the PATH environment variable. To view the directories included in the PATH list, use the following:

```
$ echo $PATH
```

Next, connect to the `info` database to view the contents of the table called `content`. Use the command:

```
$ mysql -u root
```

The MySQL client waits for the next command with the `mysql>` prompt. Select the database to work with.

```
mysql> use info;
```

Read all the records of the table `content`.

```
mysql> select * from content;
```

After using this command, the MySQL server responds by outputting the following table (given that so far you have created two web pages in the directory `info`: `page1.html` and `page2.html`):

url	title
/info/page1.html	Hypertext Markup Language - 2.0
/info/page2.html	Form-based File Upload in HTML

2 rows in set (0.00 sec)

Close the database with the `exit` command.

```
mysql> exit
```

The MySQL server responds with a good-bye message.

Bye

Automating the Database Updates with cron

To schedule `db.sh` to execute automatically, use Linux `cron` to schedule tasks. This daemon process checks a `crontab` (`cron` table) file containing instructions for `cron` to execute. `cron` as a daemon process runs in the background and executes commands at specific dates and times; for instance, it can be used hourly to take backups from a database.

The `crontab` command opens the `crontab` for editing and lets you add, remove, or modify scheduled tasks. Before using `crontab`, you may want to select the text editor to work with. For instance, to use gedit, enter the following at the Linux terminal:

```
$ export VISUAL=gedit
```

Run the command `crontab` as follows:

```
$ crontab -e
```

As shown in Figure 7-4, `crontab` opens in the gedit environment. Enter the following entry:

```
*/20 * * * * /home/christos/db.sh
```

Each crontab entry consists of six fields, specified in the following order:

minute(s) hour(s) day(s) month(s) weekday(s) command(s)

An asterisk (*) indicates “every,” a forward slash (/) means repeat at a specific interval, and a hyphen (-) indicates a range. Therefore, the previous entry indicates the following: “Execute `/home/christos/db.sh` every 20 minutes (`*/20`), every hour, every day, every month, every weekday.”

```

# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
*/20 * * * * /home/christos/db.sh

```

Plain Text ▾ Tab Width: 8 ▾ Ln 12, Col 3 ▾ INS

Figure 7-4. The crontab entry used in the example

Hint! To experiment with the syntax, use the URL <https://crontab.guru/> in the address bar of your browser.

This site, displayed in Figure 7-5, provides a textbox to insert a crontab entry and get a description, based on the entry, about how often a specific job will be run.

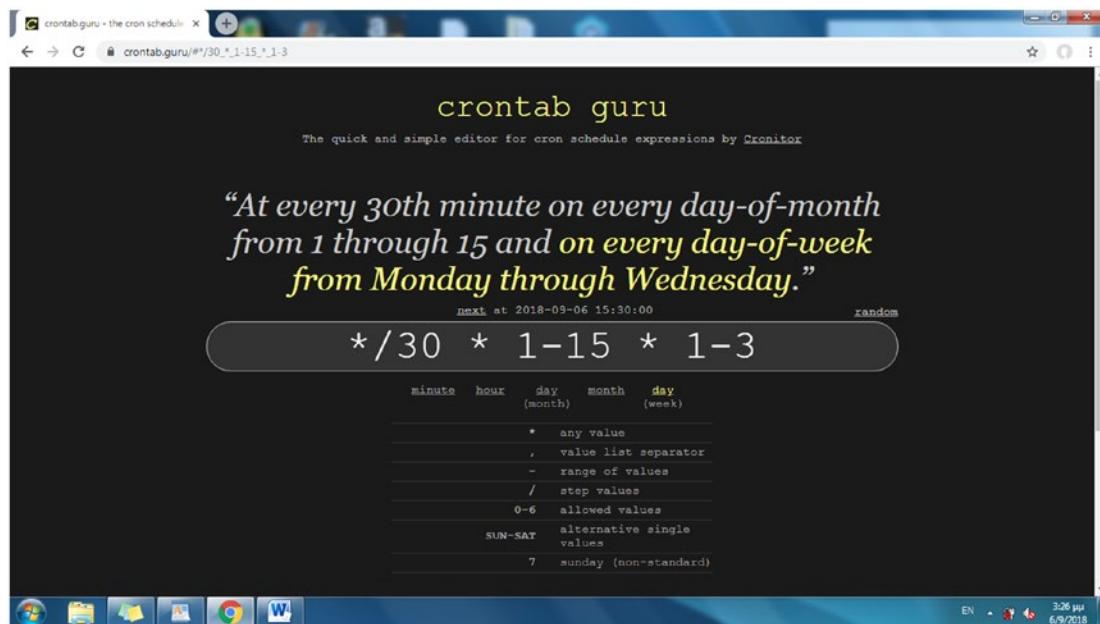


Figure 7-5. The crontab test page from crontab.guru

Do not click the Save button yet in the gedit window.

To test crontab, press Ctrl+Alt+T to open a new terminal, connect with mysql to the MySQL server, and use the info database. Run the select statement, the same used in the previous section.

```
mysql> select * from content;
```

The MySQL server responds with the same output as last time.

```
+-----+-----+
| url           | title          |
+-----+-----+
| /info/page1.html | Hypertext Markup Language - 2.0 |
| /info/page2.html | Form-based File Upload in HTML |
+-----+-----+
2 rows in set (0.00 sec)
```

Press Ctrl+Alt+T to open a new terminal window. Make info the working directory.

```
$ cd /var/www/html/info
```

Create another web page.

```
$ sudo cp page1.html page3.html
```

Change the title of `page3.html` to be a different one from the one used in `page1.html`.

```
$ sudo gedit page3.html
```

Replace the original title with another. For instance, use the following:

```
<h1>Hypertext Transfer Protocol -- HTTP/1.1</h1>
```

Save the file and close the gedit window.

Move to the gedit window of `crontab` and change the time interval of the `crontab` entry from 20 to 2 minutes so that you won't wait long for the daemon process to run. You can restore it to 20 minutes later. You can now click the Save button in the gedit window of `crontab`. Wait for two minutes, and at the terminal of the `mysql` client press the up arrow on the keyboard to execute the previous command.

```
mysql> select * from content;
```

In the output table, a new record is included with the columns corresponding to the `page3.html` filepath and title.

url	title
/info/page1.html	Hypertext Markup Language - 2.0
/info/page2.html	Form-based File Upload in HTML
/info/page3.html	Hypertext Transfer Protocol -- HTTP/1.1

3 rows in set (0.00 sec)

This entry was added by the `db.sh` shell script that was run by cron. You can restore now the entry interval from 2 to 20 minutes.

At this point, the database used for searching all the web page titles has been created, and the Bash script scheduled by cron constantly updates the database to reflect the added or deleted HTML files of the `info` directory.

Now is a good opportunity to create more web pages and therefore more entries in the `info` database so you can have a larger number of results, which will be required when we implement pagination. The site will show five results per web page, instead of putting all the results on a single web page. For the example used in this chapter, the HTML files `page1.html` up to `page30.html` were created, as indicated by the `ls -l` output for the `info` directory, displayed in Figure 7-6.

```
christos@pc:/var/www/html/info$ ls -l
total 120
-rw-r--r-- 1 root root 139 Sep  8 18:20 page10.html
-rw-r--r-- 1 root root 164 Sep  8 18:21 page11.html
-rw-r--r-- 1 root root 127 Sep  8 18:22 page12.html
-rw-r--r-- 1 root root 147 Sep  8 18:22 page13.html
-rw-r--r-- 1 root root 115 Sep  8 18:24 page14.html
-rw-r--r-- 1 root root 101 Sep  8 18:25 page15.html
-rw-r--r-- 1 root root 141 Sep  8 18:25 page16.html
-rw-r--r-- 1 root root 126 Sep  8 18:26 page17.html
-rw-r--r-- 1 root root 116 Sep  8 18:26 page18.html
-rw-r--r-- 1 root root 119 Sep  8 18:27 page19.html
-rw-r--r-- 1 root root 118 Sep  8 18:38 page1.html
-rw-r--r-- 1 root root 199 Sep  8 18:28 page20.html
-rw-r--r-- 1 root root 109 Sep  8 19:05 page21.html
-rw-r--r-- 1 root root 125 Sep  8 19:08 page22.html
-rw-r--r-- 1 root root 176 Sep  8 19:09 page23.html
-rw-r--r-- 1 root root 150 Sep  8 19:10 page24.html
-rw-r--r-- 1 root root 124 Sep  8 19:11 page25.html
-rw-r--r-- 1 root root 136 Sep  8 19:12 page26.html
-rw-r--r-- 1 root root 121 Sep  8 19:13 page27.html
-rw-r--r-- 1 root root 146 Sep  8 19:14 page28.html
-rw-r--r-- 1 root root 134 Sep  8 19:14 page29.html
-rw-r--r-- 1 root root 118 Sep  8 20:36 page2.html
-rw-r--r-- 1 root root 127 Sep  8 19:15 page30.html
-rw-r--r-- 1 root root 127 Sep  5 17:43 page3.html
-rw-r--r-- 1 root root 139 Sep  8 18:16 page4.html
-rw-r--r-- 1 root root 119 Sep  8 18:17 page5.html
-rw-r--r-- 1 root root 134 Sep  8 18:18 page6.html
-rw-r--r-- 1 root root 141 Sep  8 18:18 page7.html
-rw-r--r-- 1 root root 135 Sep  8 18:19 page8.html
-rw-r--r-- 1 root root 127 Sep  8 18:19 page9.html
```

Figure 7-6. Web pages `page1.html` up to `page30.html` are the content web pages used for the example's site

After the time indicated in the `crontab` file, `db.sh` updates the database with the 30 entries shown in Figure 7-7.

```
christos@pc: ~
File Edit Tabs Help
mysql: select * from content;
+ url | title
+-----+
| /info/page1.html | Hypertext Markup Language - 2.0
| /info/page10.html | HTTP Remote Variant Selection Algorithm -- RVSA/1.0
| /info/page11.html | Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP
| /info/page12.html | Hypertext Transfer Protocol -- HTTP/1.1
| /info/page13.html | HTTP Authentication: Basic and Digest Access Authentication
| /info/page14.html | An HTTP Extension Framework
| /info/page15.html | HTTP Over TLS
| /info/page16.html | Internet Open Trading Protocol (IOTP) HTTP Supplement
| /info/page17.html | HTTP MIME Type Handler Detection
| /info/page18.html | Use of HTTP State Management
| /info/page19.html | HTTP State Management Mechanism
| /info/page2.html | Form-based File Upload in HTML
| /info/page20.html | Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement (AKA) Version-2
| /info/page21.html | PATCH Method for HTTP
| /info/page22.html | HTTP-Enabled Location Delivery (HELD)
| /info/page23.html | Using POST to Add Members to Web Distributed Authoring and Versioning (WebDAV) Collections
| /info/page24.html | Transport of Real-time Inter-network Defense (RID) Messages over HTTP/TLS
| /info/page25.html | HTTP Strict Transport Security (HSTS)
| /info/page27.html | HTTP Header Field X-Frame-Options
| /info/page28.html | HTTP Usage in the Registration Data Access Protocol (RDAP)
| /info/page29.html | Hypertext Transfer Protocol Version 2 (HTTP/2)
| /info/page3.html | Hypertext Transfer Protocol - HTTP/1.1
| /info/page3.html | Manual Authentication Protocol for HTTP
| /info/page4.html | An Extension to HTTP : Digest Access Authentication
| /info/page5.html | HTTP State Management Mechanism
| /info/page6.html | Use and Interpretation of HTTP Version Numbers
| /info/page7.html | A Trivial Convention for using HTTP in URN Resolution
| /info/page8.html | Simple Hit-Metering and Usage-Limiting for HTTP
| /info/page9.html | Transparent Content Negotiation in HTTP
+-----+
29 rows in set (0.00 sec)

mysql>
```

Figure 7-7. Cron updates the database and creates one record for each web page found in the info directory

With the content of the site ready, you can proceed to designing `index.php`, the home page of the site that provides the search form, and also `search.php`, the action program that receives the form's keywords, connects to the `info` database, and returns the results to the client.

Designing the Home Page of the Site

Like with the previous web-based projects, two basic HTML files are required for this site. One web page provides the HTML form for submitting the keywords to the web server, usually included in the home page of the site. Another web page, indicated in the `action` attribute of the `form` element, is the one that receives the keywords, queries the database, and dispatches the query results to the user. The database is automatically updated at regular intervals with the titles and the filepaths of all the content web pages of the site.

In the previous chapter, you learned how to connect to the MySQL server with the Bash shell scripting language. Here you will connect to the MySQL server from a PHP program. The PHP-MySQL interface routines will be used to issue the query and include the results in an HTML table, in the form of hyperlinks that lead to the corresponding content web pages.

Creating the Directory Index of the Site

For the home page of the site, you will use the `index.php` file. You probably already have a directory index with the name `index.php` in the document root directory from a project of a previous chapter. If the command `ls /var/www/html` returns an `index.php` file, rename it by using the following commands:

```
$ cd /var/www/html  
$ sudo cp index.php indexOLD.php
```

Create next a new `index.php` file with gedit.

```
$ sudo gedit index.php
```

Enter the following source code and save the file:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
body{  
background-color:greenyellow;  
}  
.center {  
margin: auto;  
width: 80%;  
border: 3px solid #74AD23;  
padding: 10px;  
}  
p{  
text-align:center;  
font-size:32px;
```

```

color:green;
font-weight:bold;
}
input{
border-color:#74AD23;
font-size:32px;
color:green;
padding:10px;
background-color:greencyellow;
}
</style>
</head>
<body>
<br><br><br><br><br><br>
<div class="center">
<form name="form1" method="post" action="search.php">
<p>Search the Papers:
    <input type="text" name="keywords">&nbsp;
    <input type="submit" value="Go">
</p>
</form>
</div>
</body>
</html>

```

The previous source code creates a web page with a form. You center the form horizontally by setting the CSS `margin` property to `auto`.

```
margin: auto;
```

The form's method is POST, and the form's action is set to `search.php`. The code of `search.php` will therefore receive the keywords the user enters in the textbox and will use the PHP-MySQL interface routines to forward them to the MySQL server to query the `info` database. The query results will be converted to links with text that is the title of the web page. An HTML table with the results will be displayed to the user. As indicated by the value in the `action` attribute file, `search.php` is located in the same directory as `index.php` since no other relevant or absolute path is provided.

Figure 7-8 shows the home page of the site (`index.php`).

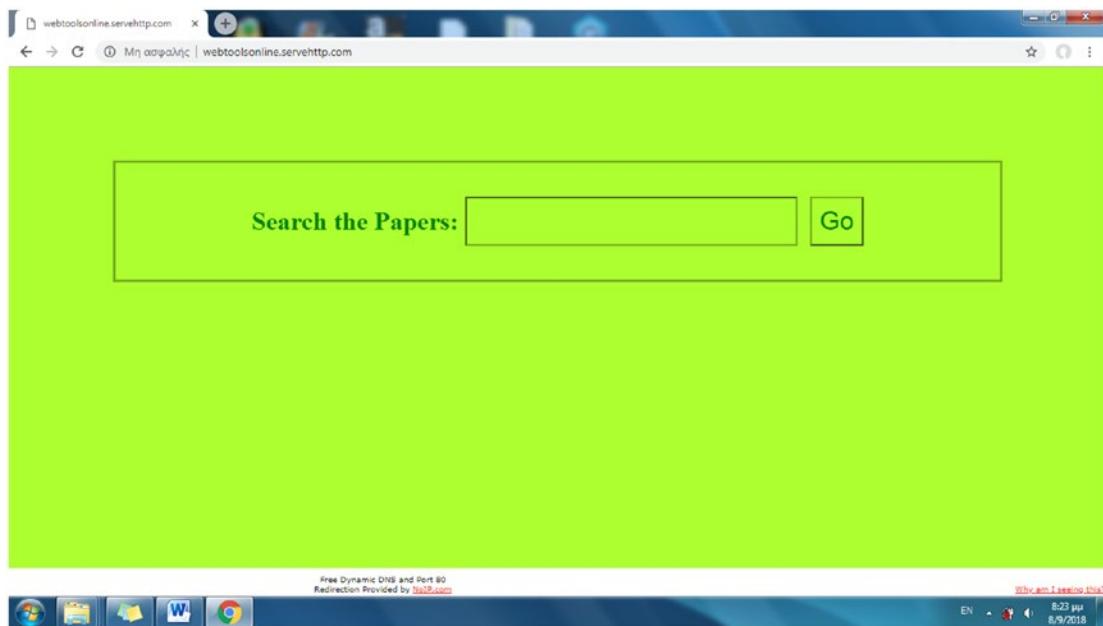


Figure 7-8. The home page of the site

There are two main methods to connect PHP and MySQL.

- MySQLi extension (MySQL improved)
- PDO (PHP Data Objects)

In this project, you will use the first one. To be able to use the `mysqli` PHP command, you have to install the MySQL extension for PHP. At the command line, enter the following:

```
$ sudo apt-get install php-mysql
```

Creating the Action PHP Program

Create `search.php` in the document root directory. `search.php` is used as the value of the form's `action` attribute, the one that defines the file that processes the user's keywords.

```
$ cd /var/www/html  
$ sudo gedit search.php
```

Enter the following lines and click the Save button in the gedit window to save the file:

```
<!DOCTYPE html>

<html>
<head>
<style>
body {
background-color:yellowgreen;
}
a {
font-size:24px;
}
</style>
</head>

<body>

<?php

if(!empty($_POST["keywords"])){
$keywords = $_POST["keywords"];
}

$array = preg_split('/\s+/', trim($keywords));

$cnt=0;
$item="";
foreach ($array as $item) {
    ++$cnt;
    $item = " title like '%{$item}%'";
    if ($cnt != count($array)) {
        $item.="$item or ";
    }
    $items .= $item;
}
}
```

```

$servername = "localhost";
$username = "root";
$password = "";
$dbname = "info";

$mysqli = new mysqli($servername, $username, $password, $dbname);

if ($mysqli->connect_error) {
    die("Connection failed: " . $mysqli->connect_error);
}

$sqlquery = "select title, url from content where ".$items." ;";

if ($result = mysqli_query($mysqli, $sqlquery)) {
    echo "<table>";
    while ($row = mysqli_fetch_assoc($result)) {
        echo "<tr><td>";
        echo "<a href=\"".$row['url']."'>".$row['title']."</a>";
        echo "</td></tr>";
    }
    echo "</table>";
    mysqli_free_result($result);
}

$mysqli->close();

?>
</body>
</html>

```

The PHP code starts with the following command, which sets the PHP variable `$keywords` to the value sent with the POST HTTP method, from the textbox named `keywords` of the form found in `index.php`:

```

if(!empty($_POST["keywords"])){
$keywords = $_POST["keywords"];
}

```

In the next code line, the PHP function `trim()` removes spaces before and after the `$keyword` value, which is a string.

At a result, the `preg_split()` function applies. PREG stands for PCRE Regular Expression, where PCRE means Perl Compatible Regular Expressions. `preg_split()` splits a string into a regular expression and returns the parts to an array, `$array` in this example. The regular expression used is as follows:

```
/\s+/
```

This indicates one or more (+) spaces (\s) from the start to the end of the string (/ /).

```
$array = preg_split('/\s+/' , trim($keywords));
```

The next code snippet starts by setting the variable `$cnt` to zero. This variable will increase by one at any `foreach()` iteration, which extracts each item of `$array` and assigns it to the variable `$item`. `$array` at this point is an array of all the keywords inserted into the form's textbox. On each iteration, the current `$item` is used to form the string `title like '%{$item}%'`, which for instance for the keyword `HTML` evaluates to the string `title like '%HTML%'`. This string will be used as part of the SQL query command for the `info` database that will be submitted to the MySQL server.

```
select title, url from content where title like '%HTML%';
```

Hint! The curly braces (`{}`) provide another way to enter a PHP variable into a string.

As you recall from the previous chapter, the `like` operator is used with the `where` SQL clause to return select columns that include the keyword that `like` applies to. By including the keyword between the `like` wildcard symbol (%), which represents zero or more characters, the keyword may be included in any position of the column's value.

The `$cnt` variable is used therefore to determine the last element of the `$array`. For any item of the array except the last one, an `or` (the OR logical operator) is appended at the end of the item. For a multikeyword search, for instance if both the `HTML` and `HTTP` keywords are used, the values of the variable `$items` will be concatenated with `or` (and for the last keyword an `or` operator should not be used).

```
title like '%HTML%' or title like '%HTTP%'
```

The concatenation of all \$item values takes place recursively and is accumulated to the \$items variable. The concatenating assignment operator (.=) is used to append the argument on the right side to the argument on the left side.

```
$cnt=0;
$items="";
foreach ($array as $item) {
    ++$cnt;
    $item = " title like '%{$item}%'";
    if ($cnt != count($array)) {
        $item="{$item} or ";
    }
    $items .= $item;
}
```

The following variables are used for the mysqli() function in the PHP code to make a connection to the MySQL server:

- \$servername indicates the hostname of the computer, and it can simply be assigned to the localhost value.
- \$username is assigned to root, the username used for the MySQL server connection. This corresponds to the -u argument of the mysql command.
- \$password is the password used for the MySQL connection, provided with the -p argument when running mysql at the command line. The empty string (no password) is used in this example.
- \$dbname is assigned to info, the name of the database used in the MySQL server.

The previous variables hold therefore the following values for this connection:

```
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "info";
```

Those variables are passed to the `mysqli()` object constructor function to return on connection success `$mysqli`, the newly created `mysqli` object. The following code makes and also tests the connection from the PHP code to the MySQL server:

```
$mysqli = new mysqli($servername, $username, $password, $dbname);

if ($mysqli->connect_error) {
    die("Connection failed: " . $mysqli->connect_error);
}
```

With the arrow (`->`) operator and the name of the object, you can access object properties and methods.

Next, the `$sqlquery` variable is assigned the command of the query to be executed. The following value after evaluation selects the columns `title` and `url` from the table `content` where the title includes one or more of the keywords entered in the form by the user.

```
$sqlquery = "select title, url from content where ".$items." ;";
```

The query is submitted to the server with the `mysqli_query()` function, and the results are retrieved with the `$result` variable. The function `mysqli_fetch_assoc()` returns the next record of the query results in the variable `$row`. The while loop iterates until all result rows are read. For each `$row` value, the two table columns `url` and `title` are indicated as `$row['url']` and `$row['title']`, respectively. An HTML link will be formed with the title as the link text and the URL as the value of the `href` attribute. By using the PHP concatenation operator `(.)` to concatenate strings like `<a href=\\"` and string variable values like `$row['url']`, a working HTML link is formed. For example, with the title `Hypertext Markup Language - 2.0` and URL `/info/page1.html`, the HTML link element becomes the following:

```
<a href="/info/page1.html">Hypertext Markup Language - 2.0</a>
```

Notice that the PHP escape `\` character is used with a double quote `\\"` to allow the double quote be used as a normal character so that it does not delimit the start or end of a string.

All links will be included in a table data (`<td>`) element, and this element will be included in a table row (`<tr>`) element of a table. The table is used to align the results. For example, center alignment can also be used for better visual appearance. The table is delimited with the start table tag and the end table tag, both entered in the evaluated PHP code with the PHP echo command.

```
if ($result = mysqli_query($mysqli, $sqlquery)) {  
    echo "<table>";  
    while ($row = mysqli_fetch_assoc($result)) {  
        echo "<tr><td>";  
        echo "<a href=\"".$row['url']."'\">".$row['title']."</a>";  
        echo "</td></tr>";  
    }  
    echo "</table>";  
    mysqli_free_result($result);  
}  
  
$mysqli->close();
```

The function `mysqli_free_result()` is used to free the memory occupied by `$result` when it is no longer needed. Also, the database connection is closed by invoking the `close` method of `$mysqli` as follows:

```
$mysqli->close();
```

The result of the previous code snippet is to evaluate the PHP code to an HTML table without borders. Each row includes the title of each result that the MySQL query returns. This title is a link that leads to the web page that discusses the topic included in the title.

Testing the Dynamic Content Site

In the address bar of your browser, use the URL domain name you have registered to display the directory index (`index.php`) of your site. In this example, the following domain name is used:

`http://webtoolsonline.servehttp.com/`

Use some keywords and feel free to include unnecessary spaces before and after the words, as displayed in Figure 7-9.



Figure 7-9. Entering keywords in the home page's form

Click the Go button. The query results (in the form of links) appear, as shown in Figure 7-10.

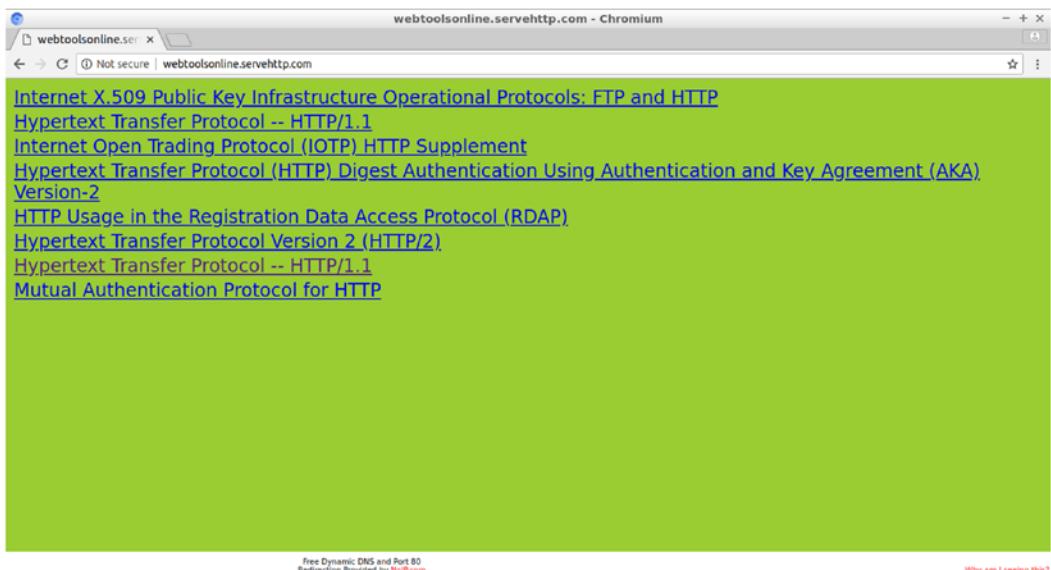


Figure 7-10. The query results appear as links, each one leading to one of the content web pages

Click any of the links included in the results. The corresponding web page appears in your browser, as shown in Figure 7-11 (for simplicity it is not filled with any usable content yet).

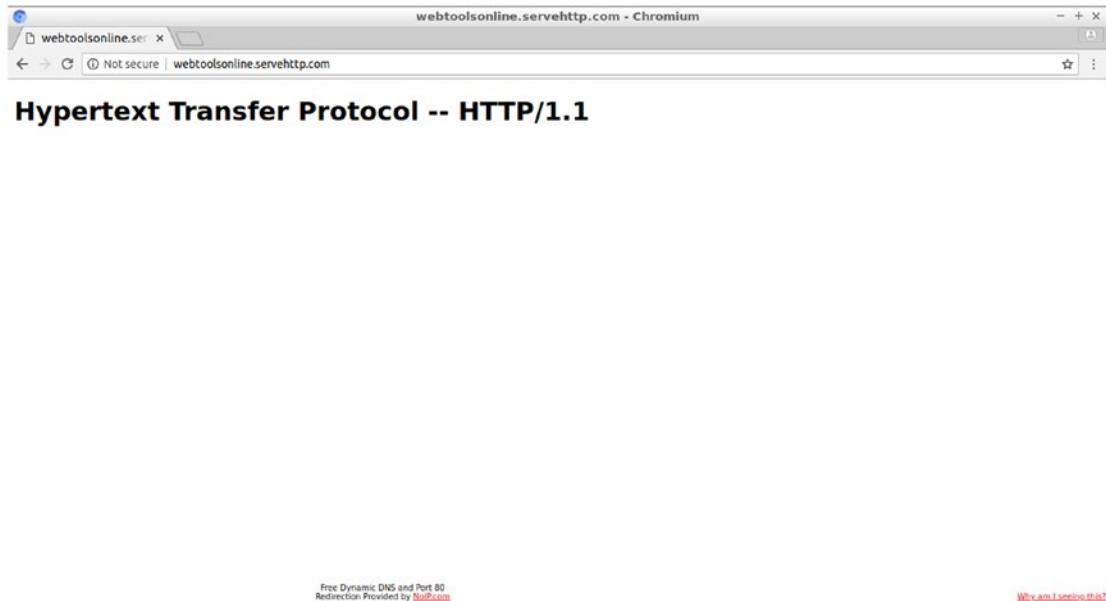


Figure 7-11. The web page of one of the query results

Next, enter a query that will return more results. You can use even more keywords than the ones used here. Figure 7-12 displays, for instance, using *HTML*, *HTTP*, and *protocol* as keywords.

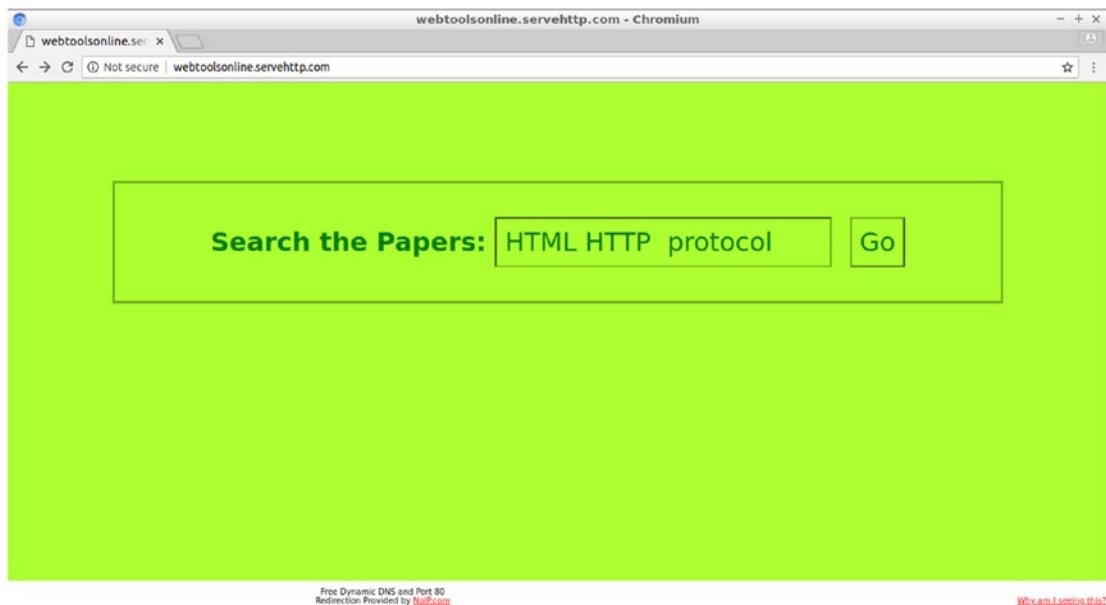


Figure 7-12. Using multiple keywords to view more results than a computer screen accommodates

Click Go again. The web page shown in Figure 7-13 appears.

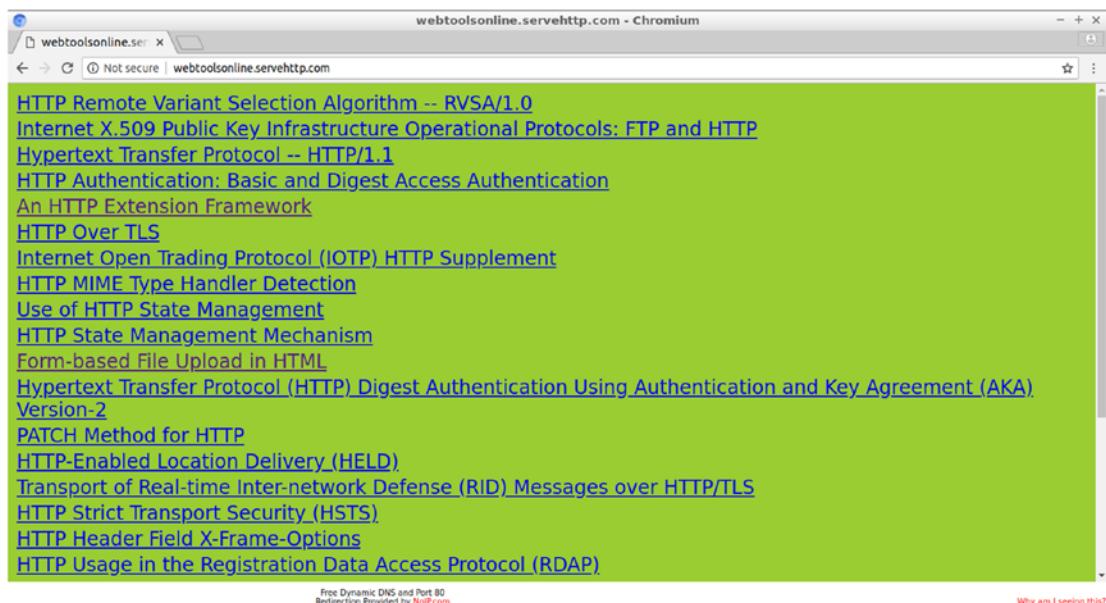


Figure 7-13. Viewing multiple results requires scrolling

To view the query results, scrolling is required. Another approach is to use pagination to group the results onto separate pages. This is what you'll do next.

Making Modifications

To make some modifications and inspect in detail the inner workings of the previous code, create a backup copy of `search.php` called `search2.php`.

```
$ cd /var/www/html
```

```
$ sudo cp search.php search2.php
```

You can modify `search.php` and restore it when you finish experimenting. In the following steps, you will change the source code to examine how the exclusion of `trim()` and `preg_split()` can lead to serious problems.

Enter at the end of the following code snippet the `echo $items` command for inspecting the variable's value:

```
<?php
if(!empty($_POST["keywords"])){
$keywords = $_POST["keywords"];
}

// trim() spaces left right, preg_split() between
$array = preg_split('/\s+/', trim($keywords));

$cnt=0;
$item="";
foreach ($array as $item) {
++$cnt;
$item = " title like '%{$item}%'";
if ($cnt != count($array)) {
$item.=$item" or ";
}
$item .= $item;
}
echo $items;
```

Use your browser to make a query from `index.php` and view the results. Use the following keywords (including some spaces before or after the words):

working with hypertext

Click the Go button. As displayed in Figure 7-14, the `$items` value is displaying in the `search.php` window.

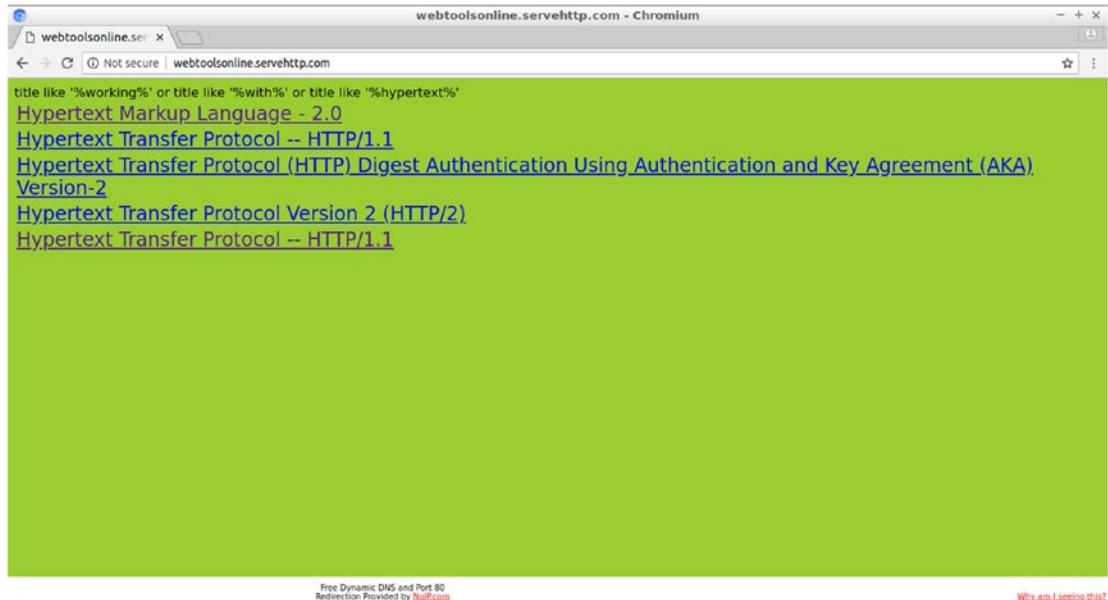


Figure 7-14. Printing echo messages for relating the source code to the query results

The `$items` value is as follows:

```
title like '%working%' or title like '%with%' or title like '%hypertext%'
```

Modify the following line:

```
$array = preg_split('/\s+/', trim($keywords));
```

Remove the `trim()` function:

```
$array = preg_split('/\s+/', $keywords);
```

CHAPTER 7 CREATING A DYNAMIC CONTENT WEB SITE

Run the previous query again. Without `trim()` in the query string, there are now empty keywords (%) at the start and the end, as the following echo output displays:

```
title like '%%' or title like '%working%' or title like '%with%' or title  
like '%hypertext%' or title like '%%'
```

Figure 7-15 shows the evaluated search.php web page.

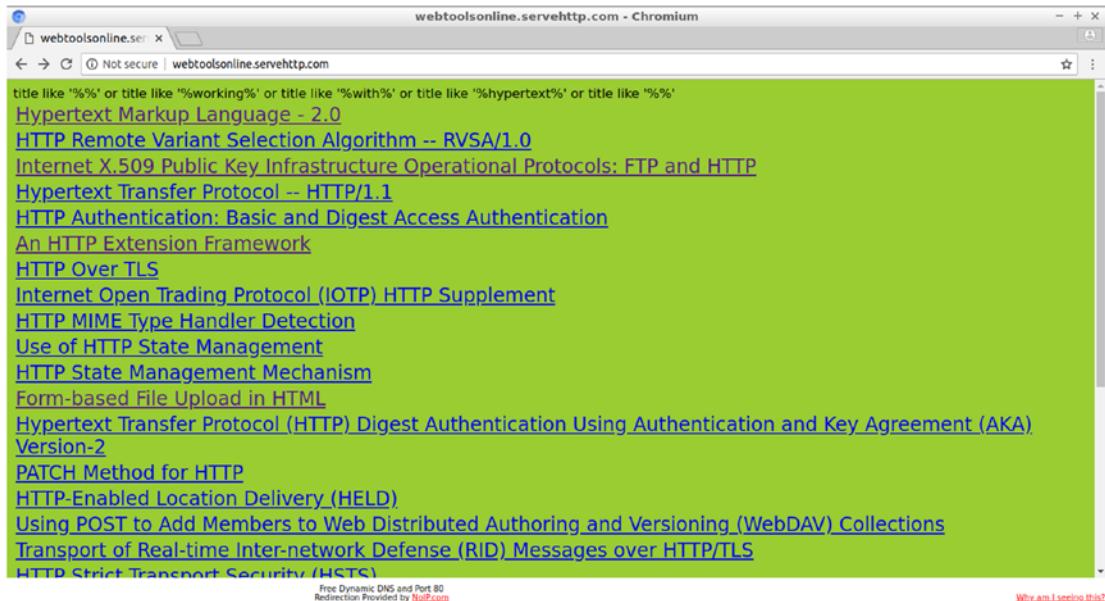


Figure 7-15. By excluding `trim()`, all database records are selected

As shown in Figure 7-15, including `like '%%'` inserts all the available database records in the query results.

Make a final modification to `search.php` by removing the function `trim()` and the function `preg_split()` from the command line. Insert `$array` using a simple function like `explode()` that uses a single space as a delimiter.

```
$array = explode(" ", $keywords);
```

The \$items may look like the following:

```
title like '%' or title like '%' or title like '%' or title like '%' or
title like '%working%' or title like '%' or title like '%' or title like
'%with%' or title like '%' or title like '%' or title like '%' or title
like '%' or title like '%' or title like '%hypertext%' or title like '%'
or title like '%' or title like '%' or title like '%' or title like '%'
or title like '%' or title like '%' or title like '%' or title like '%'
or title like '%' or title like '%' or title like '%' or title like '%'
```

As shown in Figure 7-16, by excluding preg_split(), all the database query records are included in the query results.

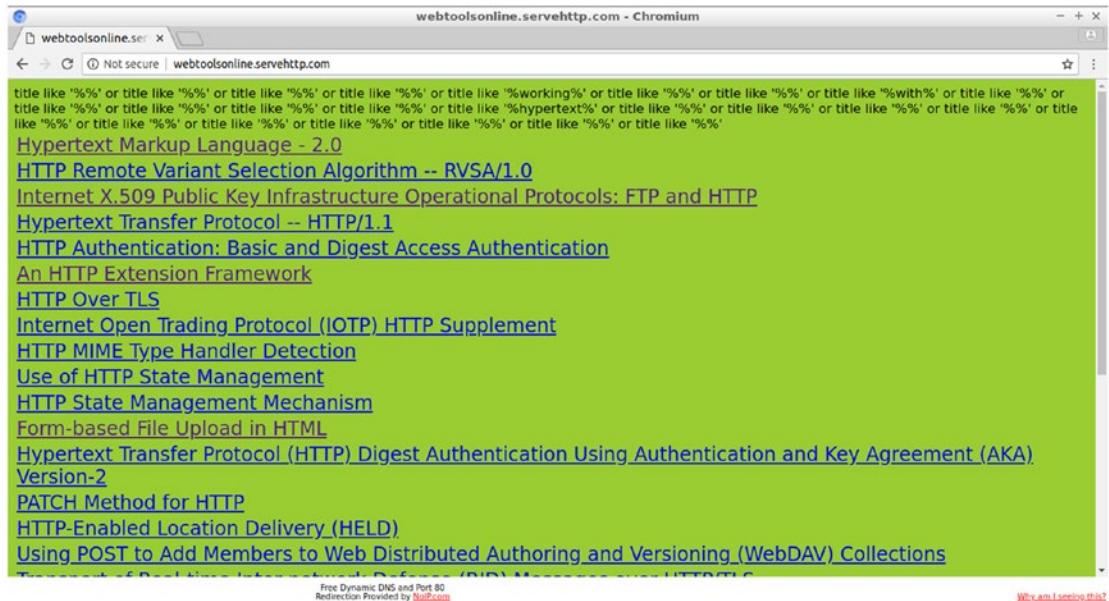


Figure 7-16. By excluding preg_split(), all database records are selected

Finally, restore the original search.php file.

```
$ sudo rm search.php
$ sudo mv search2.php search.php
```

Improving the Query Results Appearance with a Two-Colored Table

Next, you'll use CSS properties to provide a different background color to the even-numbered rows than the odd-numbered rows. Also, with the :hover CSS pseudoclass, you can provide a different background color to the row the mouse is resting on. Change the style element of search.php to look like the following:

```
<style>

body {
background-color:yellowgreen;
}

a {
font-size:24px;
}

table.center {
    margin-left:auto;
    margin-right:auto;
}

table {
    border-collapse: collapse;
    /* border: 1px solid blue; */
}

th {
    text-align: center;
    padding: 8px;
    color: goldenrod;
    font-size:24px;
}

td {
    text-align: left;
    padding: 8px;
```

```

    color: green;
    font-size:18px;
}

tr:nth-child(even){
background-color: lightcyan;
}

tr:hover {
background-color:silver;
}

tr.no_hover:hover {
background-color:floralwhite;
}

</style>

```

Use the form in `index.php` to make a new query, e.g., for the keyword *protocol*.

Figure 7-17 displays the results displayed in a two-colored table and also shows the row highlighted that the mouse pointer is on.

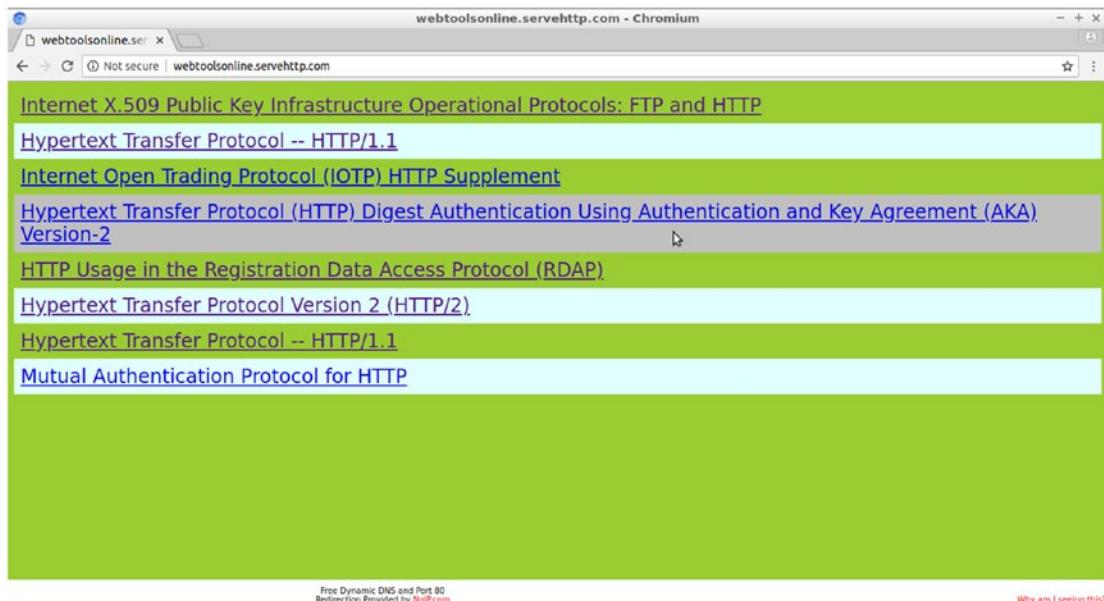


Figure 7-17. Using a two-colored table

Implementing Pagination

Instead of having all the results appear on a single page, you can group them into sets of a specific number per page, e.g., 20, and present them using several web pages that are linked with buttons or images. You can use at least two buttons: one for the next available results and one for the previous results (if any exist). This is especially useful when the query returns lots of results like a real-world search engine query usually does.

Each web page will provide part of the total results. The `limit` MySQL clause is the key to return the results starting at a specific offset and to include a count of the number of results to return. To experiment with the `limit` option, run a SQL query from the MySQL client. For example, use the following:

```
$ mysql -u root  
mysql> use info;  
mysql> select * from content where title like '%http%';
```

MySQL responds with the following results:

url	title
/info/page10.html	HTTP Remote Variant Selection Algorithm -- RVSA/1.0
/info/page11.html	Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP
/info/page12.html	Hypertext Transfer Protocol -- HTTP/1.1
/info/page13.html	HTTP Authentication: Basic and Digest Access Authentication
/info/page14.html	An HTTP Extension Framework
/info/page15.html	HTTP Over TLS
/info/page16.html	Internet Open Trading Protocol (IOTP) HTTP Supplement
/info/page17.html	HTTP MIME Type Handler Detection
/info/page18.html	Use of HTTP State Management

```

| /info/page19.html | HTTP State Management Mechanism
| /info/page20.html | Hypertext Transfer Protocol (HTTP)
|                   Digest Authentication Using Authentication and
|                   Key Agreement (AKA) Version-2
| /info/page21.html | PATCH Method for HTTP
| /info/page22.html | HTTP-Enabled Location Delivery (HELD)
| /info/page24.html | Transport of Real-time Inter-network Defense (RID)
|                   Messages over HTTP/TLS
| /info/page25.html | HTTP Strict Transport Security (HSTS)
| /info/page27.html | HTTP Header Field X-Frame-Options
| /info/page28.html | HTTP Usage in the Registration Data
|                   Access Protocol (RDAP)
| /info/page29.html | Hypertext Transfer Protocol Version 2 (HTTP/2)
| /info/page3.html | Hypertext Transfer Protocol -- HTTP/1.1
| /info/page30.html | Mutual Authentication Protocol for HTTP
| /info/page4.html | An Extension to HTTP : Digest Access Authentication
| /info/page5.html | HTTP State Management Mechanism
| /info/page6.html | Use and Interpretation of HTTP Version Numbers
| /info/page7.html | A Trivial Convention for using HTTP in
|                   URN Resolution
| /info/page8.html | Simple Hit-Metering and Usage-Limiting for HTTP
| /info/page9.html | Transparent Content Negotiation in HTTP
+-----+
26 rows in set (0.01 sec)

```

Next use the same command with the `limit` clause, where `limit 5, 3` indicates limit is 5 and count is 3:

```
mysql> select * from content where title like '%http%' limit 5, 3;
```

The results are restricted therefore to only three records, starting from the record that is indexed as five. Because the first record is indexed as zero, the following table includes the sixth, seventh, and eighth records:

url	title
/info/page15.html	HTTP Over TLS
/info/page16.html	Internet Open Trading Protocol (IOTP) HTTP Supplement
/info/page17.html	HTTP MIME Type Handler Detection

3 rows in set (0.00 sec)

In the new version of the previous site, the number of records per page will be set for simplicity to five. The initial query from `index.php` will set the `limit` and `count` values to 0 and 5, respectively, to display the first five records. Two buttons with the captions Previous and Next will be used on the evaluated `search.php` page for browsing the rest of the records. Initially, the Previous button will be disabled since the first results page does not include previous results. By clicking the Next button, the following five (or fewer if the remaining result number is less than five) records will be fetched. The Previous button will move you to the five records before the first one currently displayed.

Each button is used as a submit-type button in a separate HTML form, requesting a query that moves the offset backward for the Previous button and forward for the Next button to the result list.

Where is the current offset saved, though, when swapping from one page to another? The PHP engine extracts the offset sent along with the form data, updates it (either increasing or decreasing it), and includes it on the reply page so that it can be available for the next form submission. The offset placement is not visible to the user since it is included as the value of the form's input element of type `Hidden`. Other than the offset, the keywords entered by the user need also to be retained at each pagination transition when clicking the buttons. The initial web page used by the client includes a textbox for entering the keywords; however, the Previous and Next buttons can perform a keyword search on the same keywords by carrying them also in a hidden field.

Figure 7-18 displays the HTML form of the new version of search.php used to query the database for *HTTP*. This is the same query executed previously from the command line.

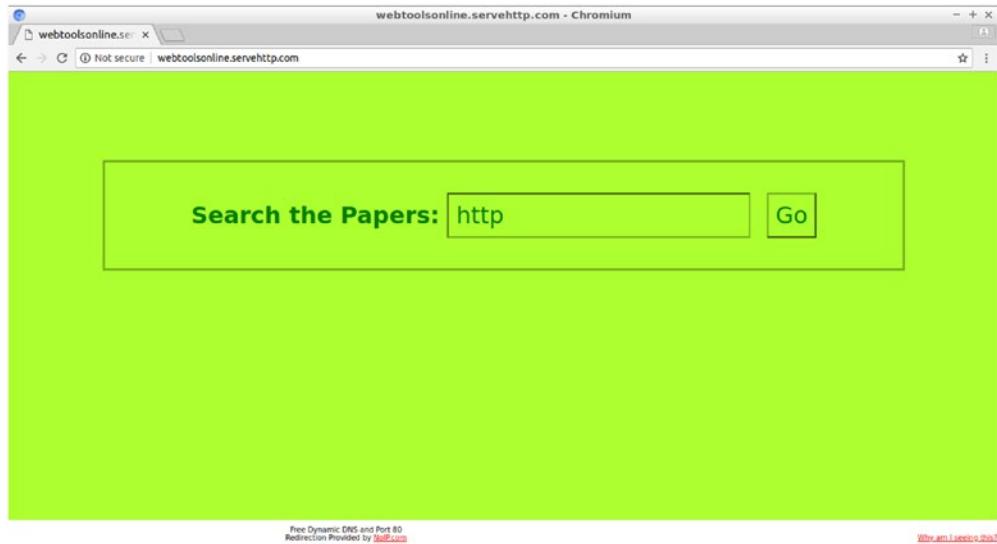


Figure 7-18. Using the pagination version of index.php

By clicking the Go button, the first set of records appear, as shown in Figure 7-19.

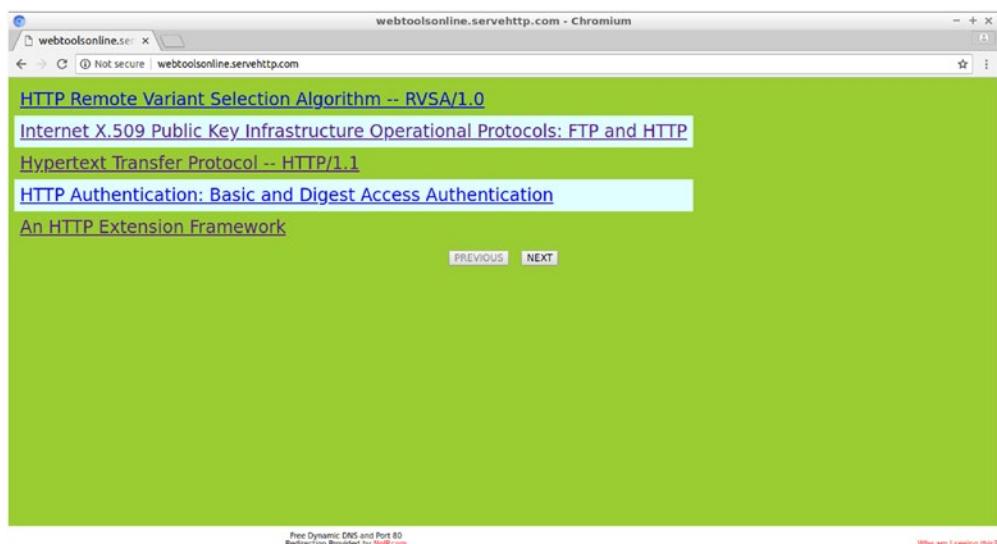


Figure 7-19. Displaying the query results with the new version of search.php

So far, no previous results exist, and therefore the Previous button is disabled. Click the Next button. The web page displayed in Figure 7-20 with the next five records appears.

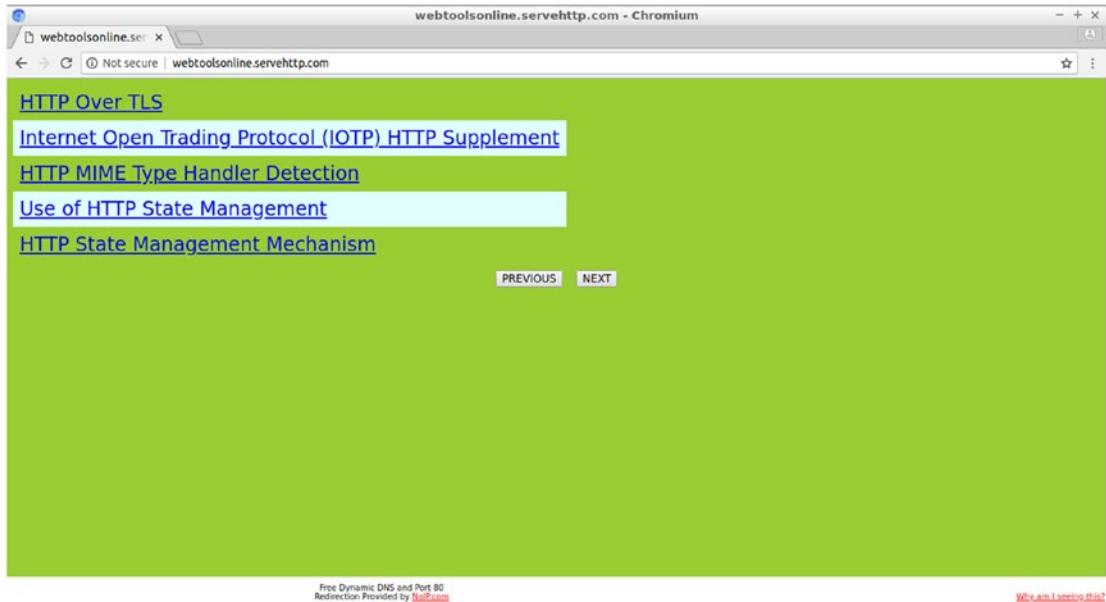


Figure 7-20. By clicking the Next button, the Previous button becomes enabled

This time, both buttons are enabled, and you can proceed forward or return backward. Click the Next button four times until you reach the final web page, the sixth one, displayed in Figure 7-21.

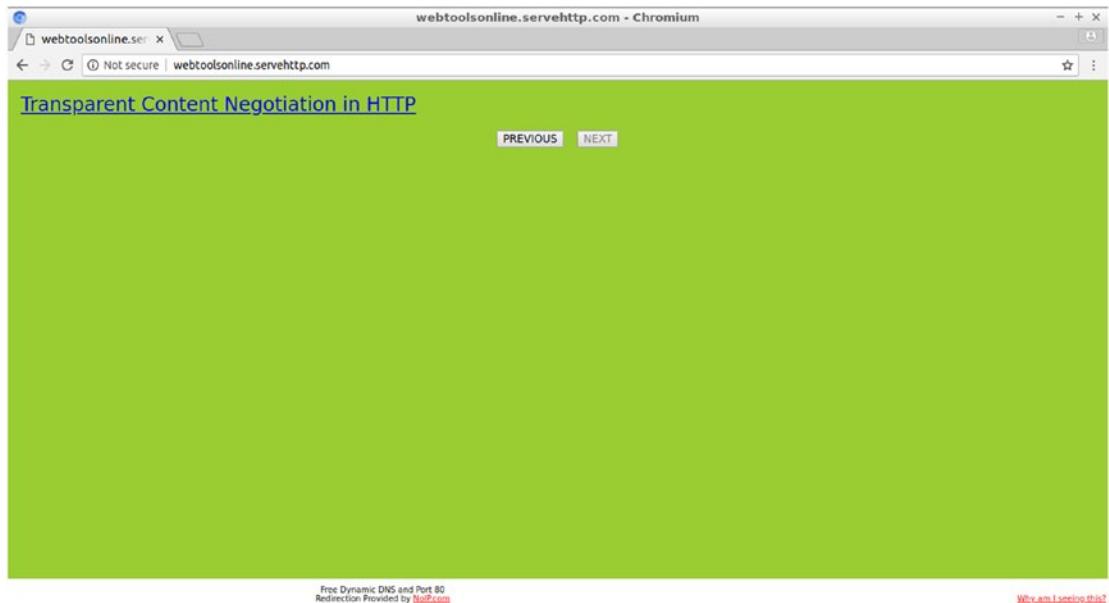


Figure 7-21. When you reach the last web page of results, the Next button becomes deactivated

The Next button as expected is disabled, and you can move only backward. You can also provide on each page a Home link to get to the home page of your site. For now, you can click the Refresh button of your browser to move to the home page.

The Pagination-Enabled Version of index.php

To implement the pagination feature on the site, changes to both the `index.php` and `search.php` source code are required. In these files, the new code added will be included between comment lines of forward slashes. The HTML comments in those files will look therefore like the following:

```
<!-- ////////////////// -->
```

CSS comments will look like the following:

```
/* ////////////////// */
```

PHP comments will look like the following:

```
///////////////////////
```

The source code for the new version of `index.php` is as follows:

```
<!DOCTYPE html>
<html>
<head>
<style>
body{
background-color:greenyellow;
}
.center {
    margin: auto;
    width: 80%;
    border: 3px solid #74AD23;
    padding: 10px;
}
p{
text-align:center;
font-size:32px;
color:green;
font-weight:bold;
}
input{
border-color:#74AD23;
font-size:32px;
color:green;
padding:10px;
background-color:greenyellow;
}
</style>
</head>
<body>
<br><br><br><br><br><br>
<div class="center">
```

```

<form name="form1" method="post" action="search.php">
  <p>Search the Papers:
    <input type="text" name="keywords">&nbsp;
    <input type="submit" value="Go">
  </p>
  <!-- ////////////////////////////// -->
  <input type="hidden" name="offset" value="0">
  <!-- ////////////////////////////// -->

</form>
</div>
</body>
</html>

```

The only line added is the hidden input type element required to provide zero as the initial offset value for the first set of result records. The original index.php version did not use a limit clause for the select statement, and therefore no offset variable was used.

The Pagination-Enabled Version of search.php

The following is the source code of the new version of search.php:

```

<!DOCTYPE html>

<html>
<head>
<style>
body {
background-color:yellowgreen;
}
a {
font-size:24px;
}

table.center {
  margin-left:auto;
  margin-right:auto;
}

```

CHAPTER 7 CREATING A DYNAMIC CONTENT WEB SITE

```
table {  
    border-collapse: collapse;  
    /* border: 1px solid blue; */  
}  
  
th {  
    text-align: center;  
    padding: 8px;  
    color: goldenrod;  
    font-size:24px;  
}  
  
td {  
    text-align: left;  
    padding: 8px;  
    color: green;  
    font-size:18px;  
}  
  
tr:nth-child(even){  
background-color: lightcyan;  
}  
  
tr:hover {  
background-color:silver;  
}  
tr.no_hover:hover {  
background-color:floralwhite;  
}  
  
/* ////////////////////////////// */  
/* class center is used just for the second table */  
.center {  
    margin-left:auto;  
    margin-right:auto;  
}  
/* ////////////////////////////// */
```

```
</style>
</head>

<body>
<?php

//if(!empty($_POST["keywords"])){
$keywords = $_POST["keywords"];
//}

// trim() spaces left right, preg_split() between
$array = preg_split('/\s+/', trim($keywords));

$cnt=0;
$item="";
foreach ($array as $item) {
    ++$cnt;
    $item = " title like '%{$item}%'";
    if ($cnt != count($array)) {
        $item = "{$item} or ";
    }
    $items .= $item;
}

$servername = "localhost";
$username = "root";
$password = "";
$dbname = "info";

$mysqli = new mysqli($servername, $username, $password, $dbname);

if ($mysqli->connect_error) {
    die("Connection failed: " . $mysqli->connect_error);
}

///////////////////////////////
// count: number results per page
$count = 5;
$offset = isset($_POST["offset"])?$_POST["offset"]:"";
```

CHAPTER 7 CREATING A DYNAMIC CONTENT WEB SITE

```
$sqlquery = "select title, url from content where ".$items." limit $offset, $count ;";
///////////////////////////////
if ($result = mysqli_query($mysqli, $sqlquery)) {
    echo "<table>";
    while ($row = mysqli_fetch_assoc($result)) {
        echo "<tr><td>";
        echo "<a href=\"".$row['url']."'\">".$row['title']."</a>";
        echo "</td></tr>";
    }
    echo "</table>";
    mysqli_free_result($result);
}

/////////////////////////////
$sqlquery2 = "select count(title) as c from content where ".$items." ;";

if ($result2 = mysqli_query($mysqli, $sqlquery2)) {
    $row = mysqli_fetch_assoc($result2);
    $c = $row['c'];
    //echo $c;
    //echo ceil($c/$count);
}
/////////////////////////////

$mysqli->close();
?>

<!-- ////////////////// -->
<table class="center">
<tr><td>
<form method="POST" action=<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
<?php $sub=$offset-$count; echo $sub>=0? '<input type="submit" value=
"PREVIOUS">'.'<input type="hidden" name="keywords" value="'.$keywords.'">'.'<
input type="hidden" name="offset" value="'.$sub.'">':'<input type="submit"
value="PREVIOUS" disabled>'; ?>
</form>
</td><td>
```

```

<form method="POST" action=<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
<?php $sum=$offset+$count; echo $sum<$c? '<input type="submit"
value="NEXT">'.'<input type="hidden" name="keywords"
value="'.\$keywords.'">'.'<input type="hidden" name="offset"
value="'.\$sum.'">':'<input type="submit" value="NEXT" disabled>'; ?>
</form>
</td></tr>
</table>
<!-- ////////////////////////////// -->
</body>
</html>

```

In the CSS section, the class center was used for the second HTML table, which includes only the Previous and Next buttons. A distinct class is required that does not implement multicoloring, as the plain table selector does for the first table, used for the query results.

The following PHP code sets \$offset and \$count, the two arguments of limit. The variable \$count is set to five. \$offset is set by \$_POST["offset"], and for the query issued from the form's textbox, the value sent is always zero. The form of index.php is not, however, the only form included in the new version of this client-server application. As shown in the last part of the new version of search.php, each one of the two buttons (Previous and Next) is used in their own form, which updates the offset value.

With the \$count set to a constant value and \$offset set either by the index.php form or by the forms used with the buttons, the query can be executed in the new version with the limit option.

```

////////////////////////////
// count: number results per page
$count = 5;
$offset = isset($_POST["offset"])?$_POST["offset"]:"";
$sqlquery = "select title, url from content where ".$items." limit $offset,
$count ;";
////////////////////////////

```

With the following code, the variable \$c holds the number of the results of the previous query (the limit option is not used for this query to return the result number). The variable \$c will be used later to disable the Previous and Next buttons when at the start or at the end of the query results, respectively. Notice that the echo ceil(\$c/\$count); command is commented out but can be also used for debugging purposes to return the total number of result pages. The PHP function ceil() returns the next highest integer value by rounding up if necessary.

```
///////////
$sqlquery2 = "select count(title) as c from content where ".$items." ;";
if ($result2 = mysqli_query($mysqli, $sqlquery2)) {
$row = mysqli_fetch_assoc($result2);
$c = $row['c'];
//echo $c;
//echo ceil($c/$count);
}
///////////
```

The last added source code creates the second HTML table for this web page with the two buttons Previous and Next that are used as the submit buttons of their forms. The value of the action attribute in each form is filled by PHP code, specifically by the echo command. It is set to \$_SERVER['PHP_SELF'], the variable that returns the current PHP script executed.

```
<!-- ////////////////////////////////////////////////// -->
<table class="center">
<tr><td>
<form method="POST" action=<?php echo htmlspecialchars($_SERVER['PHP_
SELF']); ?>">
<?php $sub=$offset-$count; echo $sub>=0? '<input type="submit"
value="PREVIOUS">'.'<input type="hidden" name="keywords"
value="'.$.keywords.'">'.'<input type="hidden" name="offset"
value="'.$.sub.'">':<input type="submit" value="PREVIOUS" disabled>'; ?>
</form>
</td><td>
<form method="POST" action=<?php echo htmlspecialchars($_SERVER['PHP_
SELF']); ?>">
```

```
<?php $sum=$offset+$count; echo $sum<$c? '<input type="submit"
value="NEXT">'.'<input type="hidden" name="keywords"
value="'.\$keywords.'">'.'<input type="hidden" name="offset"
value="'.\$sum.'">':'<input type="submit" value="NEXT" disabled>'; ?>
</form>
</td></tr>
</table>
<!-- //////////////////////////////// -->
```

The PHP code for the Previous button updates the offset, used hence as \$sub, by decreasing it by five as follows:

```
$sub=$offset-$count;
```

Then a conditional echo runs, with the condition `$sub>=0`. If this is true, which means there are more previous pages, the following HTML tags are printed with the conditional echo command to the evaluated web page received by the client.

```
'<input type="submit" value="PREVIOUS">'.'<input type="hidden"
name="keywords" value="'.\$keywords.'">'.'<input type="hidden" name="offset"
value="'.\$sub.'">'
```

This prints to the evaluated web page the tags that create three form elements: a submit button with the Previous caption and also two hidden objects, one for sending the keywords used by the client and the other for sending the updated offset (\$sub).

If the echo condition is false, just one form element is formed: a submit button. It is disabled, though.

```
'<input type="submit" value="PREVIOUS" disabled>';
```

Similar is the code for the Next button, with the difference being that the update to the offset is done by adding the following count:

```
$sum=$offset+$count;
```

Also, the choice to provide an enabled or disable Next button is determined by the following condition:

```
$sum<$c
```

Here, \$c is the total number of query results.

By using each one of the Previous and Next buttons as the submit buttons of two forms, whose action attribute indicates the same PHP code, and using hidden input type fields for updating the offset value and carrying the keywords, with a button click the query slide moves one position backward or forward.

Using Images Instead of Submit Buttons

For a stylish appearance, you can replace the submit buttons with images by providing the appropriate CSS properties to switch them from enabled to disabled. In the following example, two images were borrowed from Wikimedia Commons.

<https://commons.wikimedia.org/wiki/File:Minimal-next-icon.png>

<https://commons.wikimedia.org/wiki/File:Minimal-prev-icon.png>

Download the files, rename them to simpler names such as next.png and prev.png, and copy them to the document root directory.

Change the source code for the second table of search.php to the following:

```
<!-- ////////////////////////////// -->
<table class="center">
<tr><td>
<form method="POST" action=<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
<?php $sub=$offset-$count; echo $sub>=0? '<input type="image" src="prev.png" border=0>'.'<input type="hidden" name="keywords" value="'.$.keywords.'">'.<input type="hidden" name="offset" value="'.$sub.'">':<input type="image" src="prev.png" border=0 disabled>'; ?>
</form>
</td><td>
<form method="POST" action=<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
<?php $sum=$offset+$count; echo $sum<$c? '<input type="image" src="next.png" border=0>'.'<input type="hidden" name="keywords" value="'.$.keywords.'">'.<input type="hidden" name="offset" value="'.$sum.'">':<input type="image" src="next.png" border=0 disabled>'; ?>
</form>
</td></tr>
</table>
<!-- ////////////////////////////// -->
```

The previous code replaces the buttons created from the input elements of type submit with image buttons using a submit property. The following tag is used for the prev.png image file:

```
<input type="image" src="prev.png" border=0>
```

The following tag is used for the next.png image file:

```
<input type="image" src="next.png" border=0>
```

The disabled attribute can also be used in the previous code with the images.

```
<input type="image" src="prev.png" border=0 disabled>
```

Test the new site using the same query as previously. The two images appear in Figure 7-22.



Figure 7-22. Using images instead of buttons in search.php

Notice that there is no characteristic to visually indicate that the left image is disabled, although it still behaves as disabled when you click it. You can change that by utilizing CSS properties. In the CSS section of search.php, enter the following CSS disabled selector that applies to the input elements of type image:

```
input[type=image]:disabled  
{  
    opacity:0.5;  
}
```

When the image is disabled, its opacity property level becomes 0.5, with 1 being not transparent and 0 being completely transparent.

Run the previous query again. The transparency of the left image is set to 50 percent, as shown in Figure 7-23. This indicates that the image is not active.



Figure 7-23. Changing the opacity CSS property for the disabled images

Implementing the Site with the GET Method

Next, you'll change the request method for this site from POST to GET.

The method POST stores the form data in the body of the request message of the HTTP request, while GET attaches the data to the URL as the query string (name-value pairs) in the request line of the message. By using pagination and the GET method, the URL changes in each page to reflect the offset, count, and keywords carried in the query string.

Change all POST references to GET for both files used in this client-server application. In `index.php`, change the value of the `form` method attribute to `get`.

```
<form name="form1" method="get" action="search.php">
```

In `search.php`, replace all the `$_POST` variables with `$_GET`. Use, therefore, `$_GET["keywords"]` instead of `$_POST["keywords"]` and `$_GET["offset"]` instead of `$_POST["offset"]`.

Replace also POST with GET in the form elements, created for the buttons, so that you finally have the following:

```
<form method="GET" action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
```

To test the GET method, do not use a domain name like <http://webtoolsonline.servehttp.com/>, which was used previously with POST because of the masking DDNS feature, discussed in Chapter 4, which alters the visible URL in the user's browser. Use, for instance, the address from the masking service, which in the examples of Chapter 4 was as follows:

`http://christos.ddns.net:8080/index.php`

Or, as shown in Figure 7-24, simply use the following:

`localhost/index.php`



Figure 7-24. Viewing the GET version of the site locally

Run the previous query for the keyword *HTTP*. The query string is now visible in the address bar of your browser. In this example, as shown in Figure 7-25, it is as follows:

keywords=http&offset=0

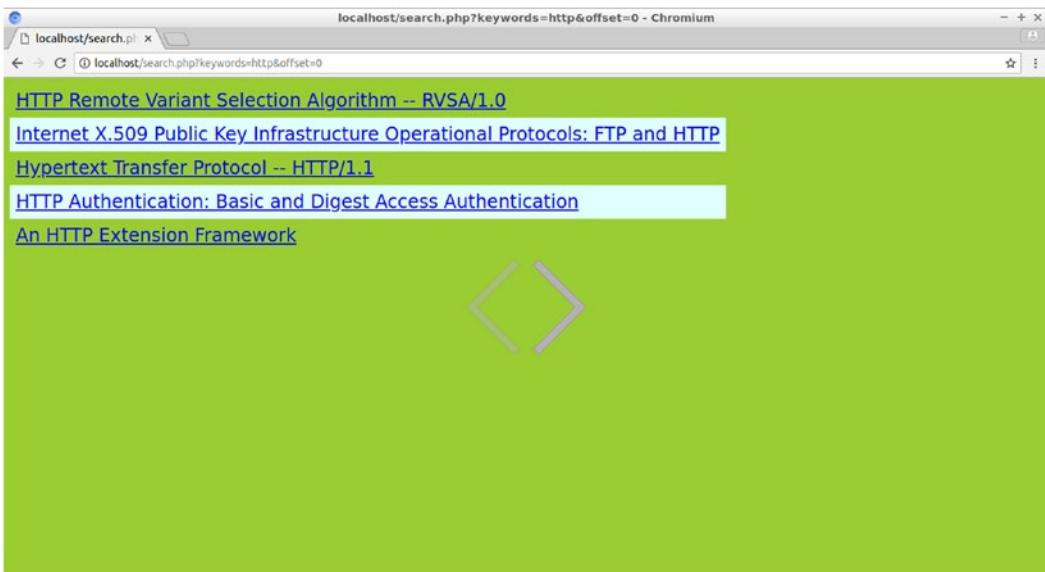


Figure 7-25. With GET, the query string appears in the URL of the site

Click the right-facing arrow image. The query string, as shown in Figure 7-26, becomes as follows:

```
x=3&y=3&keywords=http&offset=5
```

As expected, the keywords value remains the same, but the other values change. The offset is updated to 5, and you can also see x with a value of 3 and y with a value of 3. The explanation for those two mysterious names comes is that because an image was used as a submit button, it attaches to the query string the x and y coordinates of the mouse pointer when the mouse clicks the image. The value pair (3, 3) means that the mouse pointer was placed near to the origin (0, 0) of the coordinates, which is the upper-left corner of the image.

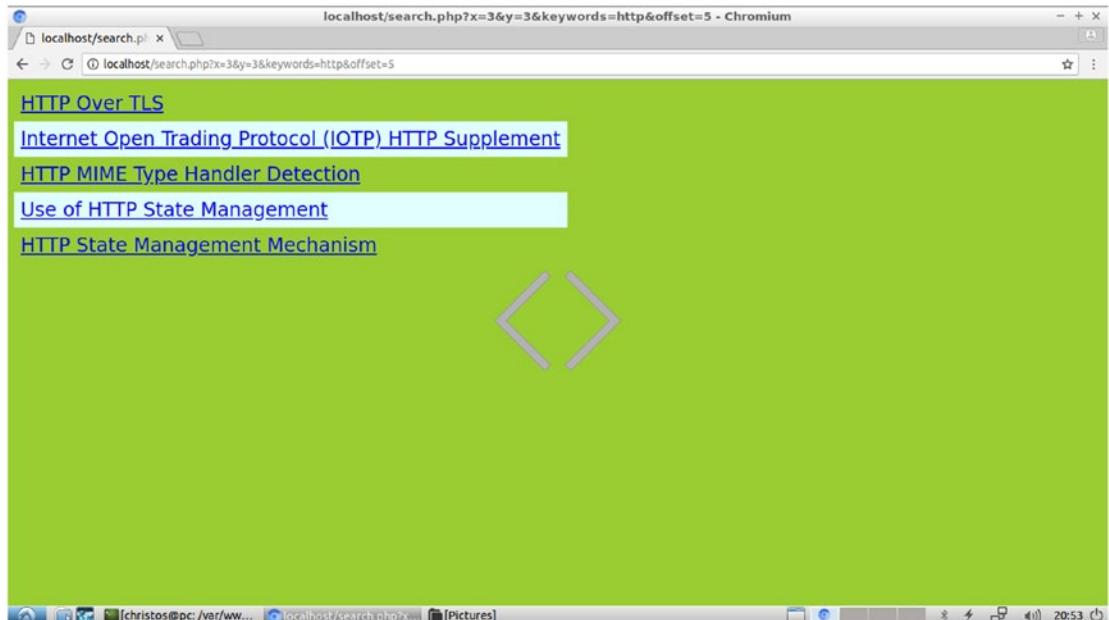


Figure 7-26. Viewing the mouse coordinates in the query string

In the browser's address bar, change the offset value manually, for instance to 17:

```
localhost/search.php?x=3&y=3&keywords=http&offset=17
```

Press the Enter key. As shown in Figure 7-27, the page content gets updated with the five next records, starting at offset 17.

CHAPTER 7 CREATING A DYNAMIC CONTENT WEB SITE

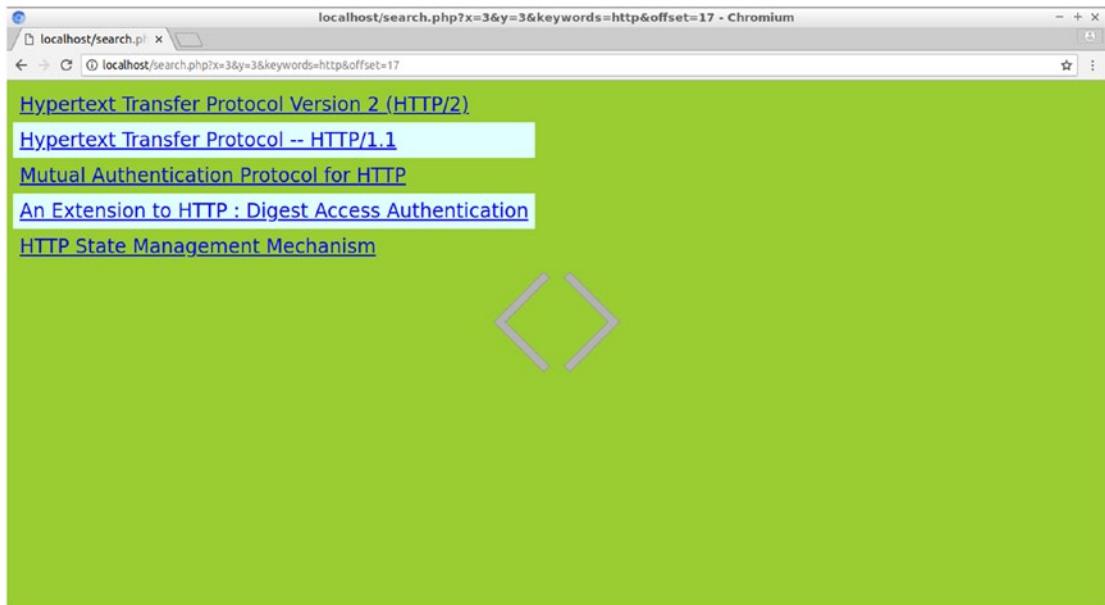


Figure 7-27. Manually changing a value in the query string

You can use this shortcut in many commercial sites when you want to move to the results at a position that is not included by the web page buttons. For instance, at the amazon.co.uk site, displayed in Figure 7-28, the current web page is page 4 of the query results as indicated with the page number at the bottom of the page and also in the web page's URL.

`https://www.amazon.co.uk/gp/search/ref=sr_pg_4?rh=n%3A266239%2Ck%3Aapress&page=4&keywords=apress&ie=UTF8&qid=1536775199`

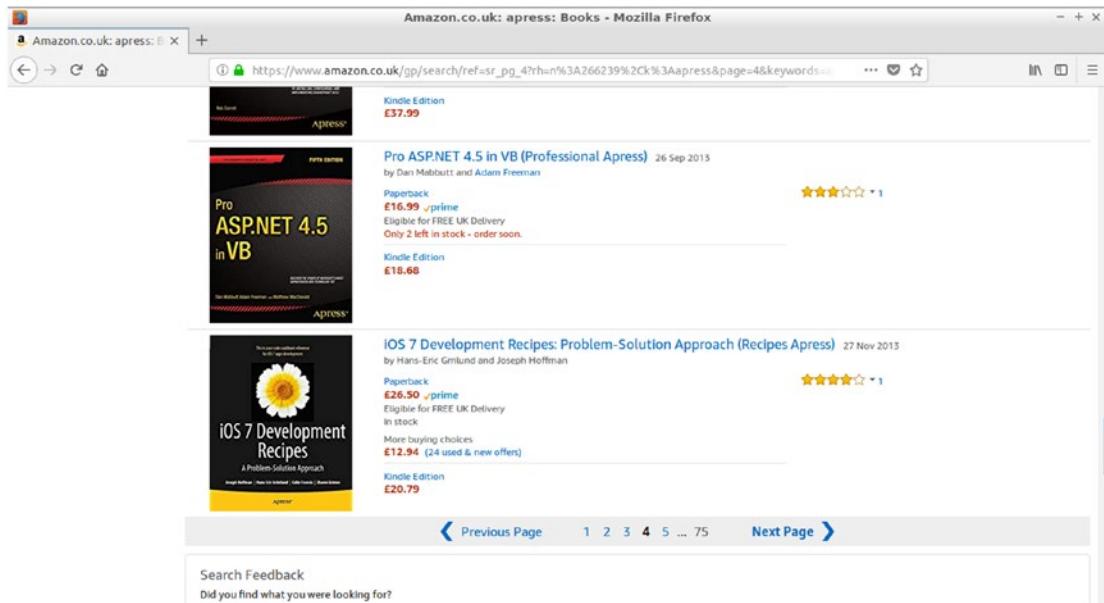


Figure 7-28. Amazon.co.uk showing page 4

Change the value 4 in `page=4` to 40 and press the Enter key to instantly move to page 40 of the results (Figure 7-29).

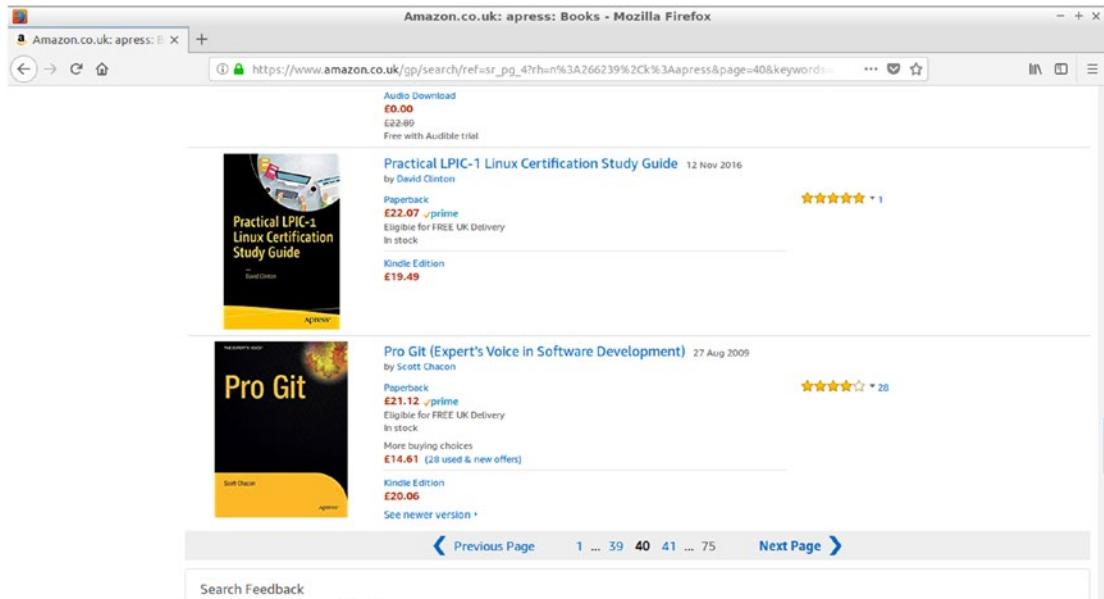


Figure 7-29. Amazon.co.uk showing page 40

Summary

In this chapter, you created a project that includes a multiword search feature that is found in a large number of web sites. The query searches the contents of the site and specifically the title of each web page entered in a database. The query results are returned as links to web pages that include the keyword.

You also automated the process of updating the database with new web pages entered into the site by using cron to schedule the updates with a Bash shell script.

In the following chapter, you'll implement Secure Sockets Layer (SSL) for your site so that cryptography is enabled and a secure communication between the web server and the clients is ensured.

CHAPTER 8

Implementing Secure Sockets Layer on Your Site

In this chapter, you'll move from using the HTTP protocol to using HTTPS, and you'll apply the SSL/TLS protocols to implement cryptography for a secure communication between the clients and the web server of your site. The messages exchanged after this between the client and the web server will be encrypted, and it will be extremely difficult for "any man in the middle" to decrypt them. The HTTPS protocol implementation will be indicated in the protocol part of the URLs as `https://` instead of `http://`, which was used in the previous chapters for simple HTTP connections. For this to work, the public key infrastructure (PKI) of the web server (Apache or Lighttpd) has to be configured correctly.

You'll create two more projects in this chapter. The first one uses HTTP cookies to allow the otherwise stateless HTTP protocol to retain some user-specific data for each visitor, such as the user's name, which is stored on the client side. The second project utilizes HTTP cookies to implement PHP sessions and allow users to switch from one page to another in your site while maintaining the user-defined customization settings, which are stored on the server side.

Implementing SSL/TLS

PKI is considered one of the previous century's most important inventions. The ability to transfer sensitive data, such as account numbers in bank transactions or credit card details when online shopping, is essential for the reliable operation of all web services.

Web security relies on the protocols of public key cryptography, also referred as *asymmetric cryptography*. The adjective *asymmetric* means two different cryptographic keys are used for the encryption and the decryption of a message—the private key and the public key. A *key* in cryptography is a string of bits of a specific length, and it used by a cryptographic algorithm to transform plain text into cipher text, or vice versa. A message encrypted with the public key, which is usually publicly available for the clients, can be decrypted only by its corresponding private key, which is kept secret on the server side. While a PKI session between a client and a server initiates with the public/private key pair, at the end of the session a symmetric key is established to be used for the client-server communication.

Secure HTTP (HTTPS) is implemented by including Secure Sockets Layer (SSL)—or its successor the Transport Layer Security (TLS) protocol—in the TCP/IP protocol stack just below the application protocol, which is HTTP for the World Wide Web.

SSL Certificates

To establish a secure communication with SSL/TLS, the client and the server exchange the information required to create a common symmetric key. The server sends to the client its public key in the form of a digital certificate. A digital certificate, commonly called an *SSL certificate*, is a data file that binds the public key of the web server with other attributes that identify the specific server. A valid certificate adds authentication to the communication by ensuring that an imposter, or the *man in the middle* of the communication path, will not assume the identity of the other communication's end.

Hint! While the TLS protocol tends to replace SSL, the name SSL is used broadly when referring to the cryptographic protocol used, even when the underlying mechanism is actually TLS. Because of this, certificates are generally called SSL certificates even though the right term is SSL/TLS certificates.

Digital certificates are granted by root certificate authorities (CAs), which are trusted third parties that issue SSL certificates after verifying the identity of their clients. The certificates of root CAs, called *root certificates*, are self-signed. That is, in a way, they guarantee for themselves and also are pre-installed in the browsers so that they can readily be used to verify the applicant's domain certificate. To enhance the security of the

root certificate, other intermediate certificates may be utilized, with one signing another and thus creating a “chain of trust” with the domain certificate of the web server at the one end and the root CA certificate at the other.

The public key of a certain site is retrieved by the browser through the site’s domain SSL certificate. To create a digital certificate for a domain, the applicant has to create a certificate-signing request (CSR) file and submit it to a CA or otherwise can self-sign it. A self-signed certificate enables a site with HTTPS; however, it will not be trusted by certain browsers, and users will be warned by some browsers against visiting the potentially insecure site. The CSR is a digital document including the public key and other identifying information and is usually created along with the matching private key. The CA may request identifying information. For instance, the applicant may be asked to prove ownership of a domain, e.g., by responding to an e-mail sent to the address of the specific domain. Next, the CA generates the certificate, signs it using its private key, and issues it to the applicant, who then imports the CA signed certificate to their own web server.

Hint! What does digital signing mean?

The CA signs a certificate by implementing a hash algorithm on the certificate to generate a message digest, which is a fixed-length string that is typically shorter than the algorithm’s input and—theoretically—a unique output string. A hash algorithm is a one-way algorithm. You cannot derive the algorithm’s input from its output. The digest is then encrypted with the CA’s private key. The signed certificate, which is the certificate attached to the digest, is sent then to the recipient.

The recipient can verify that the certificate is received unaltered by decrypting its digest with the CA’s public key. Also, it generates another digest from the received certificate using the same hashing algorithm as the sender. The two digests are compared, and if found to be identical, the certificate is considered a valid one.

After receiving a copy of the certificate, the client side (e.g., the browser) checks which CA has signed the certificate. When the validation process succeeds, the public key of the server is used on the client side, and the private key of the server is used on

the server side to encrypt the messages of the SSL handshake process. This process involves exchanging a number of messages between the client and the server that result in creating a symmetric key used for the client-server communication.

Creating Self-Signed Certificates with OpenSSL

As mentioned in the previous section, an inexpensive option for your site, especially when you are just experimenting with cryptography and do not have to run an absolutely trustworthy commercial site, is to self-sign an SSL certificate instead of using a CA. Obtaining a certificate from a CA costs money and usually requires purchasing a non-DDNS domain name. On the other hand, this cost is not prohibitive, and you certainly avoid the browser warning messages viewed by your visitors about entering an insecure site.

Hint! In this chapter, you will create and use a self-signed certificate and configure the web server to use it. In the following chapters, you will learn how to obtain and use one from a certificate authority. However, you don't have to obtain a CA certificate to run the projects in the next chapters.

To generate a self-signed certificate and also your private key, use the `openssl req` (request) command in your home directory as follows:

```
$ cd ~  
$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout server.  
key -out server.crt
```

The `openssl req` command creates a certificate request in the PKCS#10 format, which is the common format used with CAs. It can additionally create self-signed certificates, which is how it will be used here.

The command's arguments are as follows:

- `-nodes` (no DES) creates a certificate that does not require a Data Encryption Standard (DES) passphrase.
- `-newkey rsa:2048` creates a 2048-bit RSA (Rivest-Shamir-Adleman) key for use with the certificate.

- `-x509` creates a self-signed certificate instead of a certificate request.
- `-keyout` specifies `server.key` as the private key file that will be created.
- `-out` specifies `server.crt` as the certificate file.
- `-days` determines the length of time in days that the certificate will be issued for.

You will be asked to enter the following information; you can press Enter for the default values or insert a period for a blank value:

Country Name (2 letter code) [AU]:

State or Province Name (full name) [Some-State]:

Locality Name (eg, city) []:

Organization Name (eg, company) [Internet Widgits Pty Ltd]:

Organizational Unit Name (eg, section) []:

Common Name (e.g. server FQDN or YOUR name) []:

Email Address []:

As the common name, you can specify your name or the fully qualified domain name (FQDN), e.g., [secureserver.ddns.net](#).

Two files, `server.key` and `server.crt`, are created in your home directory. Restrict the file's access rights to be read-only by the root. Use the following:

```
$ sudo chmod 400 server.key server.crt
```

Hint! To read the file contents of `server.key` or `server.crt`, use `cat` as follows:

```
$ sudo cat server.key
```

The command's output will look like the following:

```
-----BEGIN PRIVATE KEY-----
```

```
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQCwgxSlj6d+eCtw
xWFNhbMlMN4fhhk7vmMd43JjHB00JyJzQ0k+5ziBHF9MRT9KHbDlVlcfil3oiv0m
I6AvV17vEK6zDXJ87WxhIzSdY4H9YhT7dkZ2a8YhwBLiQDD0mIf/QE2bV9Ghgdek
xj+qqcpxAsh2Q4Nc0gSax6rxFv3o+rxk1S7IxsjSyTnWTWQQFzEtRCV1C/sGV3PM
```

Ws2waf9B9tEn7H0Ubq0teFXS14P1v/iMSWddGj12kBb8tzQ1ZIps/cPefY+Tx7MO
U7vLSv+RBRf06Mr6iNeZnCEQ0MxIxEgxpewGtEzNou8q+bRcDf9XyAL5KMVcOsE9
FIILA6AFAgMBAAECggEAFr010xn/099Gzw1D24FqKPVhDDLmGe4Bt31iX+bjjd8e
qi4mEaYReWGZzCh33GN3Nff1BKJkbAhXIHSijJQzLJI7teG74N90egXaJYf/1wP/
aNwscdyorfTMTBKVrf9fdHaaWlhF+GoR5QL6jpORDx+5L8ILRt1LicSEFIBtC4c
rgqAiJzD1qHhQzbpT9E+0cR+Ue/JK4h3Y0mPF7I9wcfq0GyMo0Th1c+NIRgAgybs
1k6Kpe4pN9tujmWuHolnj0a0pX/ZtbEX9GoogW0LUwPjw7PbzKou1UyUt3r2ufQe
ZbWGqmktd1+tSDFserfQepZwpW80rv8bBh8yGZYoQKBgQDkD1b16sNb7WVy5AcE
+bTB74X4s/xgt3BXcMpz1s5WwfzLTlpiU2Uom080FtDldX1TgXrDWjzpGD8DIZjr
oSplFZ7MyWtHPLsxGxccT7oBzcx5PADDmuk3ZSez8/fvbde1xzt+eIGUD6R6XhmW
RXeduZVigtcb7Xu1lh4zJLKxGQKBgQDGIwpt3JtzWm0ByblPmzY5kxtxSghPdqt+
mPI81nB1PmxmuWJULDuEcAxs6zY+6vEJiK9URago2tHHWxDIb42SwoxSxYtOVkup
DZBb8rJJLDoe/wouakjPNIDBqiTvx24vp0B9pYfuqozR1tHRaC2QhIu/BIhtUD81
Mpy/L5knzQKBgH7nWkh1Xkg1DbKk2JMYMFFKa448+U8IRGcjyEQ1X5QFdvnam8jj
BwNUNpHseEl220pAXoOeDw7WAxpG88UKZYDiSv9BhYSacr+ch3ulkae3UPSVQweV
h/jfPPYR4YFF6iaoup5hiB1PqwK8ohhQh4Mo5ctvayuLNq+Q3TUwUo7hAoGAcLtv
K6LhL3i2NRo5PXnqGEgCzSp6H/w9BwKukL7Rrw0e+bNwps0j+w5nI3GQo6u5CNuk
JiabzuLxiKPfoKsXufDHnjD/WcrvsXfuMuKbXda1z/T8Lfx7AKm2uHm+Gk81+hch
MnYEKV8QUDQRnTvQ8PD5Me26Ubfevr3VQVIrqeECgYAdKGp29sE4JxV9yCk5pECN
T40HtHbZTTmENzNm8UaMGnFiQntBW0a/JEuejeV80anKHCgxowl792Pync+hfE6D
3lzgc94vfsdthgVztpV1JHtJ5lwVq0+k8RqTHMP0c+Eg21ESghFojagoAVFSsV1F
MLWnqVLzTBD8QPHgYfHhIA==
-----END PRIVATE KEY-----

Unlike with your public key, you should never display your private key (the previous key has been exposed and therefore was replaced by another one). From the private key created previously, you can extract the public key. To include the public key in another file, e.g., file `public.key`, use the following in your home directory:

```
$ sudo openssl rsa -in server.key -pubout -out public.key
```

The server responds with the following output:

writing RSA key

Use `cat` to read `public.key`.

```
$ sudo cat public.key
```

The `public.key` contents are displayed next.

-----BEGIN PUBLIC KEY-----

```
MIIBIjANBgkqhkiG9wOBAQEAAQCAQ8AMIIIBCgKCAQEAsIMUpY+nfngrcMVhTYQT
JTDeH4YZ075jHeNyYxwTtCcic0DpPuc4gRxfTEU/Sh2w5VZQnyC99Ir0Ji0gL1de
7xCusw1yf01sYSM0nWOB/WIU+3ZGdmvGIcAS4kAwzpiH/0BNm1fRoYHXpMY/qqnK
cQEodkODXDoEmseq8Rb96Pq8ZNUuyMbI0sk51k1kEBcxLUQldQv7BldzzFrNsGn/
QfbRJ+x9FG6tLXhVopeD9b/4jElnXRo5dpAW/LcONWSKbP3D3n2Pk8ezDl07y0r/
kQUXzujK+ojXmZwhENDMSMRIMaXsBrRMzaLvKvm0XA3/V8gC+SjFXDrBPRSCCw0g
BQIDAQAB
```

-----END PUBLIC KEY-----

Concatenate the two files created, `server.key` and `server.crt`, as a `.pem` file. PEM stands for Privacy Enhanced Mail and is the de facto file format for storing and sending cryptographic keys and certificates.

```
$ sudo cat server.key server.crt > server.pem
```

Change the access right of `server.pem`.

```
$ sudo chmod 400 server.pem
```

Set root as the owner of `server.pem`.

```
$ sudo chown root:root server.pem
```

You need to import the `server.pem` file into the configuration of both the Lighttpd and Apache web servers to enable the HTTPS protocol. In the following two sections, you will first configure SSL for Lighttpd and then do the same for Apache.

Configuring SSL for Lighttpd

To use the PEM file created in the previous section to configure SSL with Lighttpd, use the steps in this section.

Copy the file `server.pem` in `/etc/lighttpd`.

```
$ sudo cp server.pem /etc/lighttpd
```

Edit `lighttpd.conf`, the Lighttpd configuration file. Use any text editor like gedit.

```
$ sudo gedit /etc/lighttpd/lighttpd.conf
```

Replace the `server.port` value currently used (port 80 or 8080) with port number 443, which is the one used by HTTPS.

```
server.port = 443
```

Also, set the value of the `ssl.engine` directive to `enable`, and set the `ssl.pemfile` directive to the filepath of your `.pem` file.

```
ssl.engine = "enable"
ssl.pemfile = "/etc/lighttpd/server.pem"
```

After saving `lighttpd.conf`, it should look like the following:

```
server.modules = (
    "mod_access",
    "mod_alias",
    "mod_accesslog",
    "mod_compress",
    "mod_redirect",
)
```

```

server.document-root      = "/var/www/html"
server.upload-dirs        = ( "/var/cache/lighttpd/uploads" )
server.errorlog           = "/var/log/lighttpd/error.log"
server.pid-file           = "/var/run/lighttpd.pid"
server.username            = "www-data"
server.groupname           = "www-data"
#####
server.port               = 443
#####
#server.port                = 8080
#server.bind                 = "webtoolsonline.servehttp.com"
server.errorfile-prefix     = "/srv/www/errors/status-"

dir-listing.activate       = "disable"

accesslog.filename          = "/var/log/lighttpd/access.log"
#accesslog.format             = "%V %h %l %u %t \"%r\" %>s %b \"%{Referer}i\""
                           \ \"%{User-Agent}i\""

index-file.names            = ( "index.php", "index.html", "index.lighttpd.
                               html" )
url.access-deny              = ( "~", ".inc" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

compress.cache-dir           = "/var/cache/lighttpd/compress/"
compress.filetype            = ( "application/javascript", "text/css",
                               "text/html", "text/plain" )

#####
ssl.engine      = "enable"
ssl.pemfile     = "/etc/lighttpd/server.pem"
#####

# default listening port for IPv6 falls back to the IPv4 port
## Use ipv6 if available
#include_shell "/usr/share/lighttpd/use-ipv6.pl" + server.port
include_shell "/usr/share/lighttpd/create-mime.assign.pl"
include_shell "/usr/share/lighttpd/include-conf-enabled.pl"

```

```
#include "vhost.conf"
```

At this point it is assumed that the Lighttpd server runs. If Apache is currently used stop this server and start lighttpd:

```
$ sudo service apache2 stop
$ sudo service lighttpd start
```

To enable the changes in the configuration file, enter the following at the Linux terminal:

```
$ sudo service lighttpd force-reload
```

Configuring SSL for Apache

The same PEM file as used previously for the Lighttpd server can be used for Apache. To test Apache, stop the Lighttpd process. At the Linux terminal, enter the following:

```
$ sudo service lighttpd stop
$ sudo service apache2 start
```

You can start the Lighttpd server using this:

```
$ sudo service lighttpd start
```

Copy the PEM file used for Lighttpd to the /etc/apache2 directory.

```
$ sudo cp /etc/lighttpd/server.pem /etc/apache2
```

You can find the default-ssl.conf configuration file in the /etc/apache2/sites-available directory. Make a backup of this file to use it as the configuration file for the Apache HTTPS server.

```
$ sudo cp /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-available/default-ssl.conf.bak
```

Edit the default-ssl.conf with a text editor.

```
$ sudo gedit /etc/apache2/sites-available/default-ssl.conf
```

You just need to add the following directive:

```
SSLCertificateFile /etc/apache2/server.pem
```

The default-ssl.conf file (with most comments omitted) will look like the following:

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        # SSL Engine Switch:
        # Enable/Disable SSL for this virtual host.
        SSLEngine on

        SSLCertificateFile      /etc/apache2/server.pem

        <FilesMatch "\.(cgi|shtml|phtml|php)$">
            SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
            SSLOptions +StdEnvVars
        </Directory>

    </VirtualHost>
</IfModule>
```

Next, you need to enable the ssl module. Use the following:

```
$ sudo a2enmod ssl
```

The command's output is as follows:

```
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
```

Enabling module `ssl`.

See `/usr/share/doc/apache2/README.Debian.gz` on how to configure SSL and create self-signed certificates.

To activate the new configuration, you need to run:

```
$ systemctl restart apache2
```

To enable the site with the configuration defined in `default-ssl.conf`, use the following:

```
$ sudo a2ensite default-ssl
```

The command's output is as follows:

Enabling site `default-ssl`.

The last action is to reload Apache with the new configuration. Use the command suggested by the `a2enmod` output.

```
$ sudo systemctl restart apache2
```

Or use the following:

```
$ sudo service apache2 force-reload
```

Test your site using either Apache or Lighttpd with the steps included in the following section.

Testing the Self-Signed Certificate

Test your site locally first using the following URL:

`https://localhost`

Because the HTTPS connection to this site is not verified by a certificate authority, the user will see a warning message, like the one shown in Figure 8-1, to urge the user to skip visiting an untrusted site.

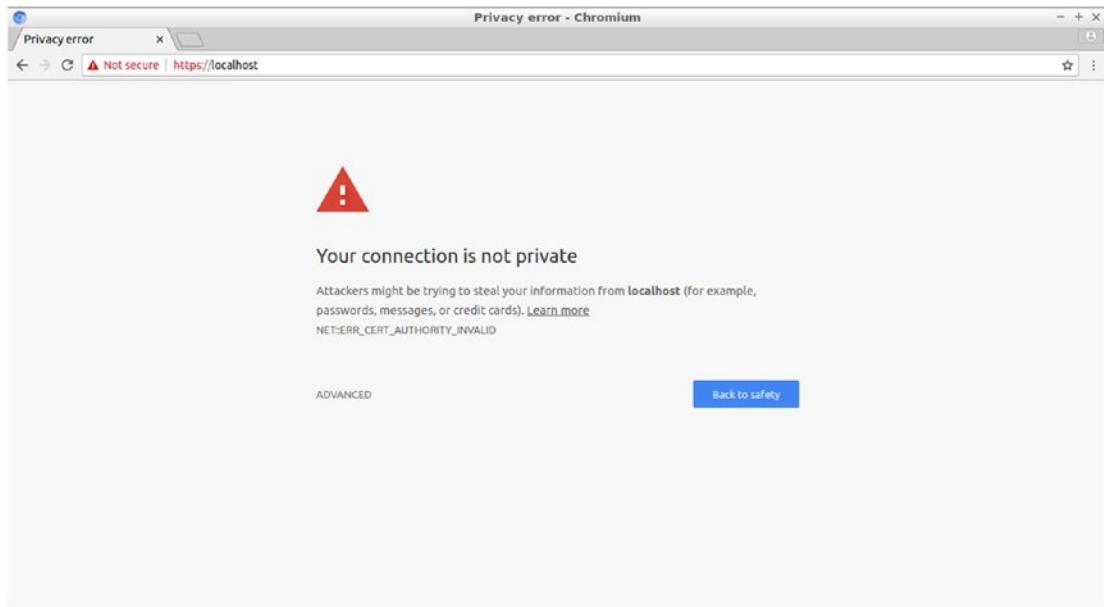


Figure 8-1. Chromium warning against visiting an untrusted site

Since you know that it is your own site, proceed by clicking the Advanced link. The following message appears (for the Chromium browser), displayed in Figure 8-2:

"This server could not prove that it is localhost; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection."

Click the "Proceed to localhost (unsafe)" link.

CHAPTER 8 IMPLEMENTING SECURE SOCKETS LAYER ON YOUR SITE

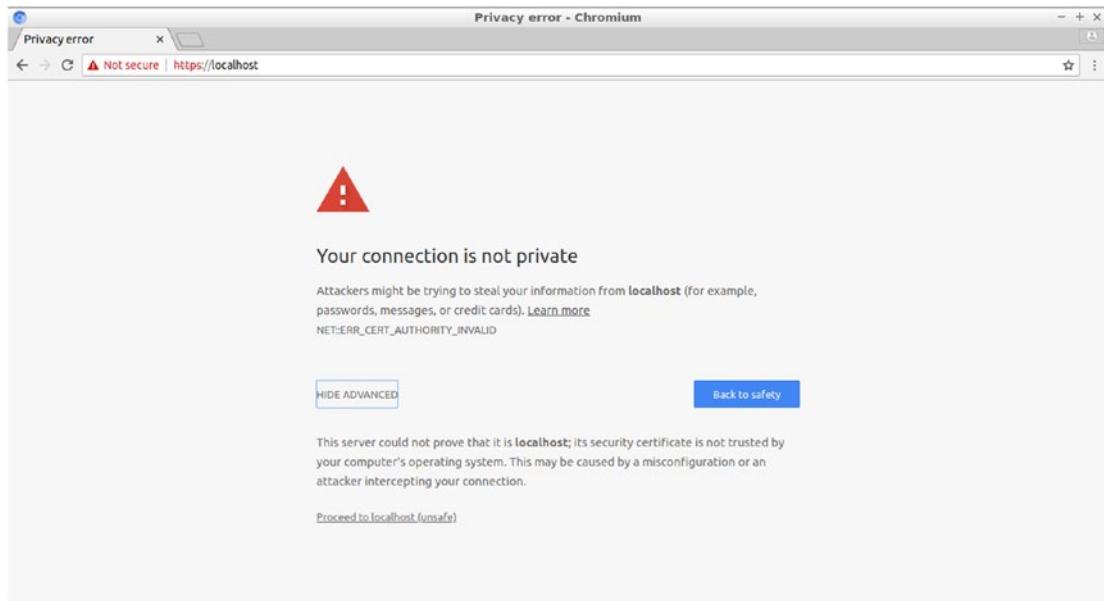


Figure 8-2. By clicking the Advanced link, you have the option to proceed to a potentially “untrusted” site

The directory index of the site appears, as displayed in Figure 8-3.

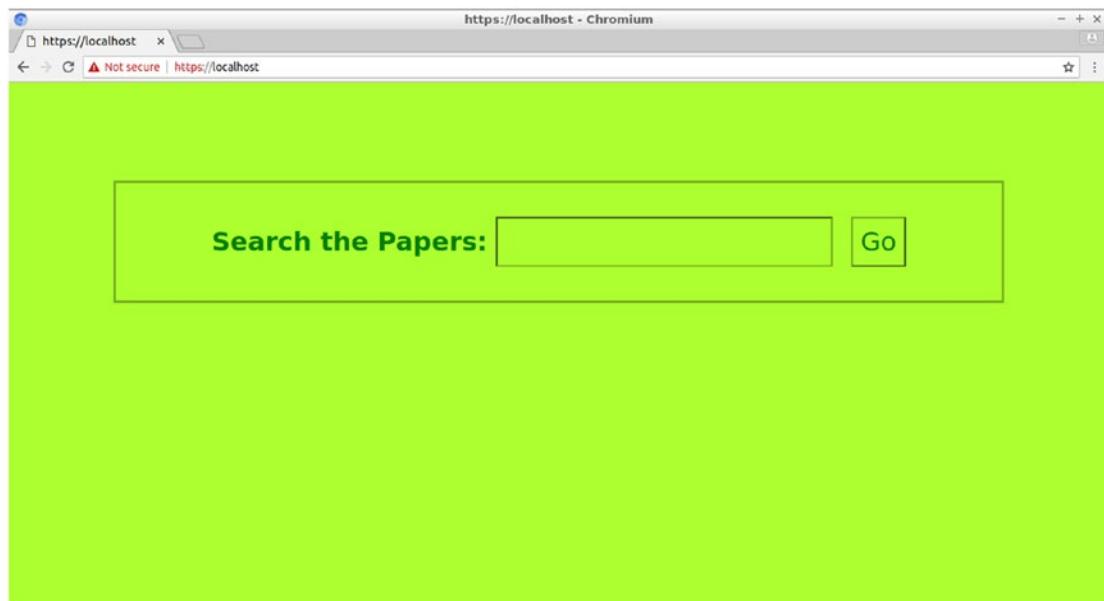


Figure 8-3. The directory index of your site, as viewed with Chromium, when a self-signed certificate is implemented

As you'll notice, the Chromium browser strikes out the protocol part of the URL in the address bar with a red line. Firefox, on the other hand, is more tactful. Using the same URL in Firefox, you get the message displayed in Figure 8-4.

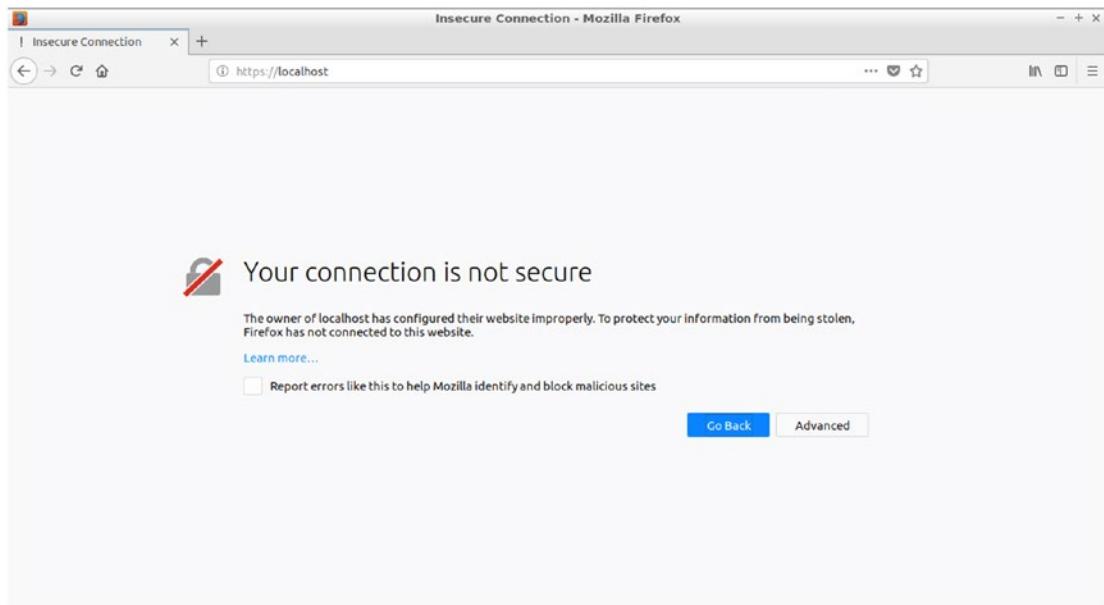


Figure 8-4. A warning message from Firefox about an insecure connection

Click the Advanced button. A frame that includes the Add Exception button appears, as shown in Figure 8-5.

CHAPTER 8 IMPLEMENTING SECURE SOCKETS LAYER ON YOUR SITE

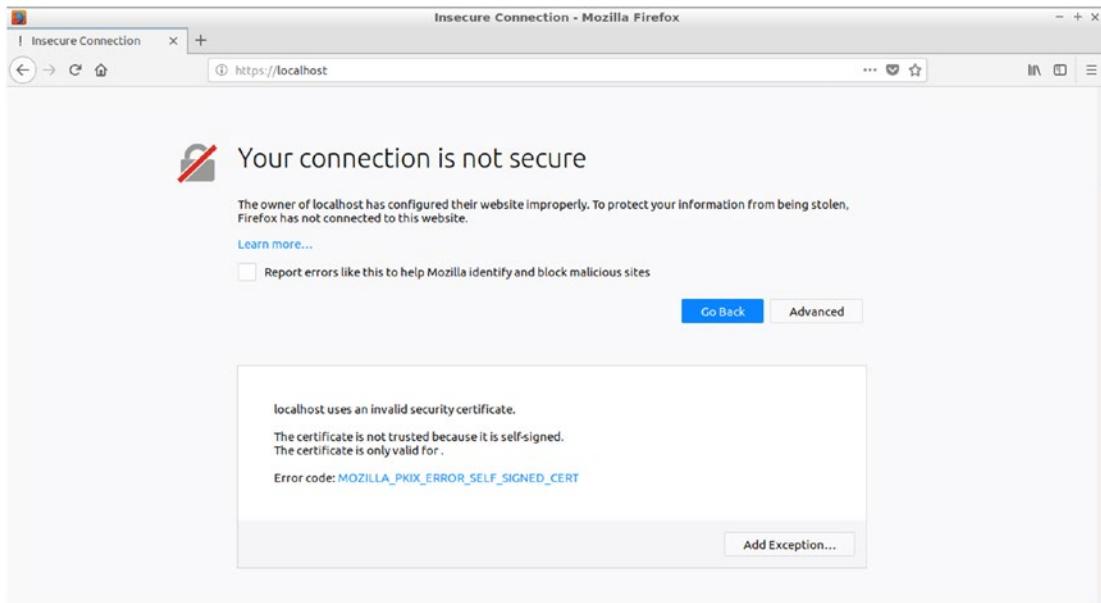


Figure 8-5. The Add Exception button in Firefox enables the user to visit the untrusted site

Click the Add Exception button. A new dialog, displayed in Figure 8-6, requests that you add a security exception for `https://localhost/`. Click the Confirm Security Exception button.

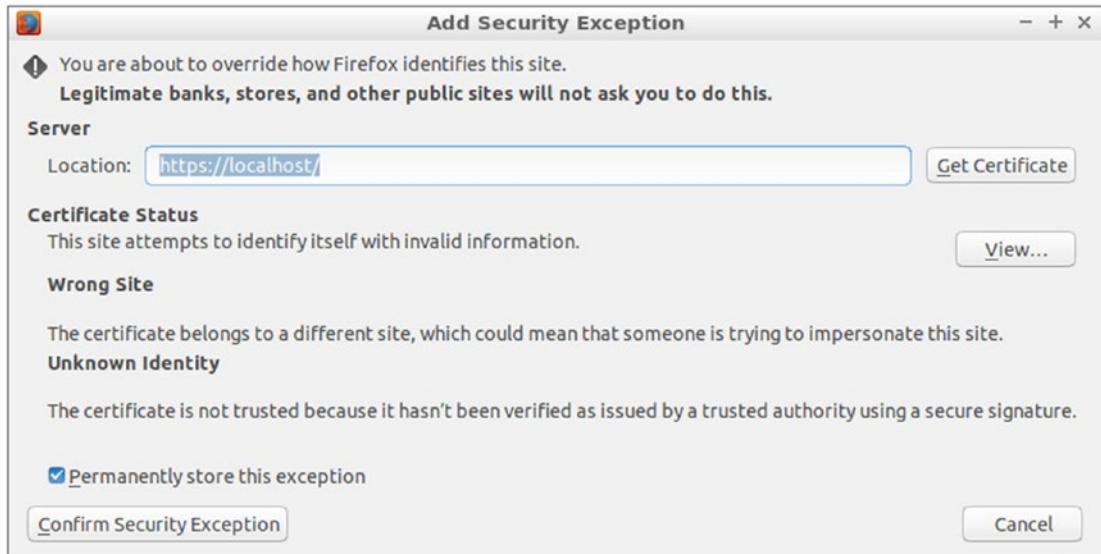


Figure 8-6. The dialog requesting a confirmation for the security exception

The directory index of the site appears, as shown in Figure 8-7.

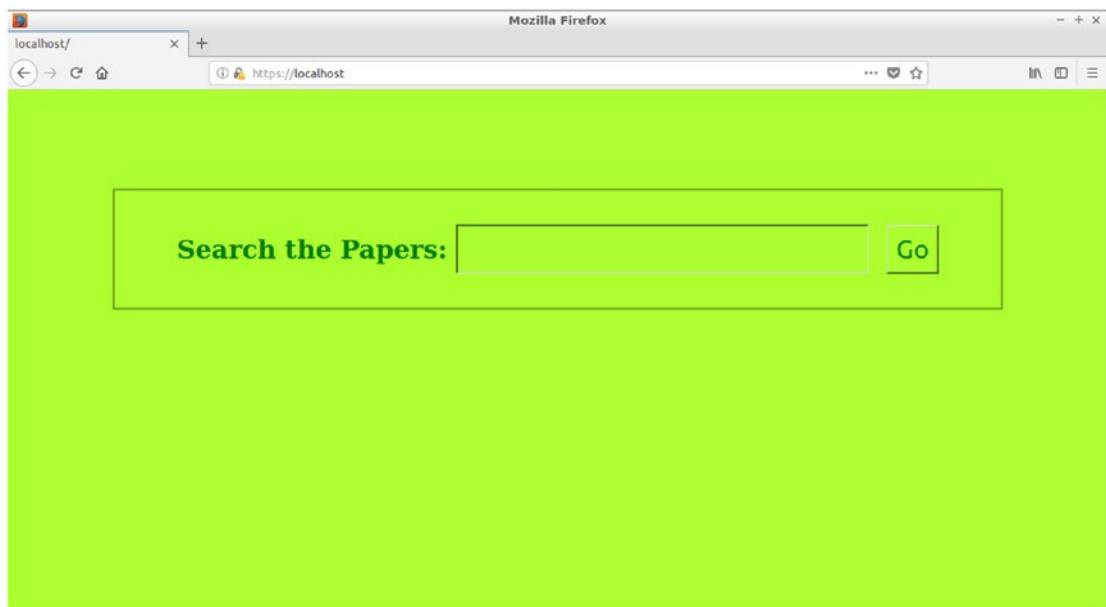


Figure 8-7. The directory index of your site, as viewed in Firefox, when a self-signed certificate is implemented

Test the site as usual. In the search textbox, enter, for instance, the *Internet protocol* keywords and click the Go button. The results appear in the user's browser, as shown in Figure 8-8. Click any link in the results. As expected, the corresponding web page is served with the HTTPS protocol.

CHAPTER 8 IMPLEMENTING SECURE SOCKETS LAYER ON YOUR SITE

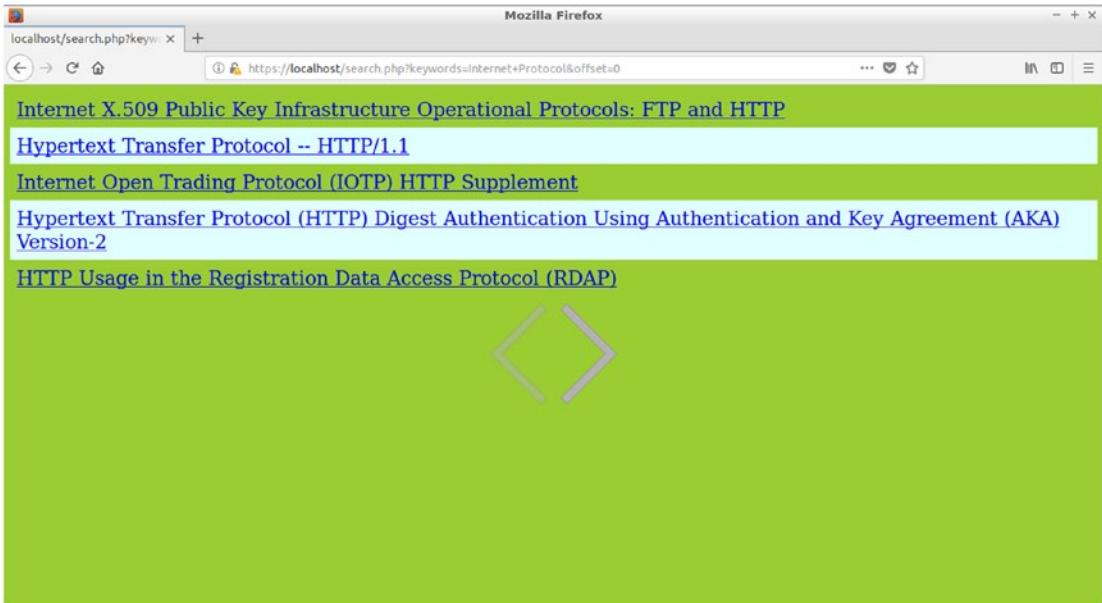


Figure 8-8. The online search for the web site is carried out with HTTPS

You can alternatively test your site using the private IP address of your server, as displayed in Figure 8-9.

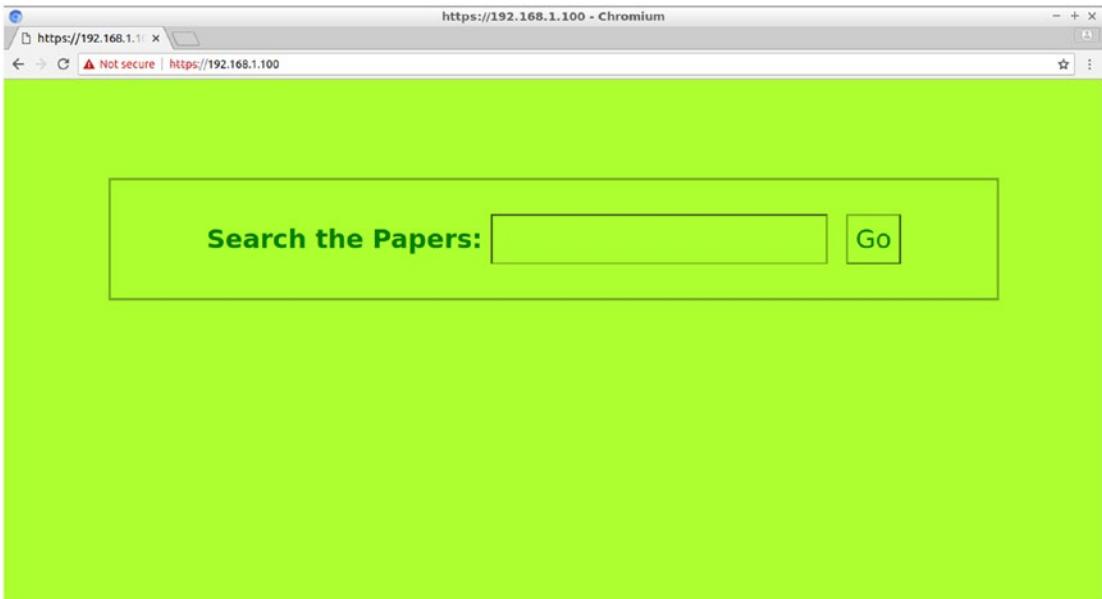


Figure 8-9. Testing the site with the private IP address of your server

Enabling Your Site to Be Viewed Outside of Your LAN

As with port 80 for the HTTP protocol in the previous chapters (or port 8080 if your ISP bans inbound port 80 connections), you have to enable port forwarding for port 443, the default HTTPS port number. Connect using the web interface of your router by entering its private IP address in the address bar of your browser. As displayed in Figure 8-10, the router's private IP address is 192.168.1.1 in this example.

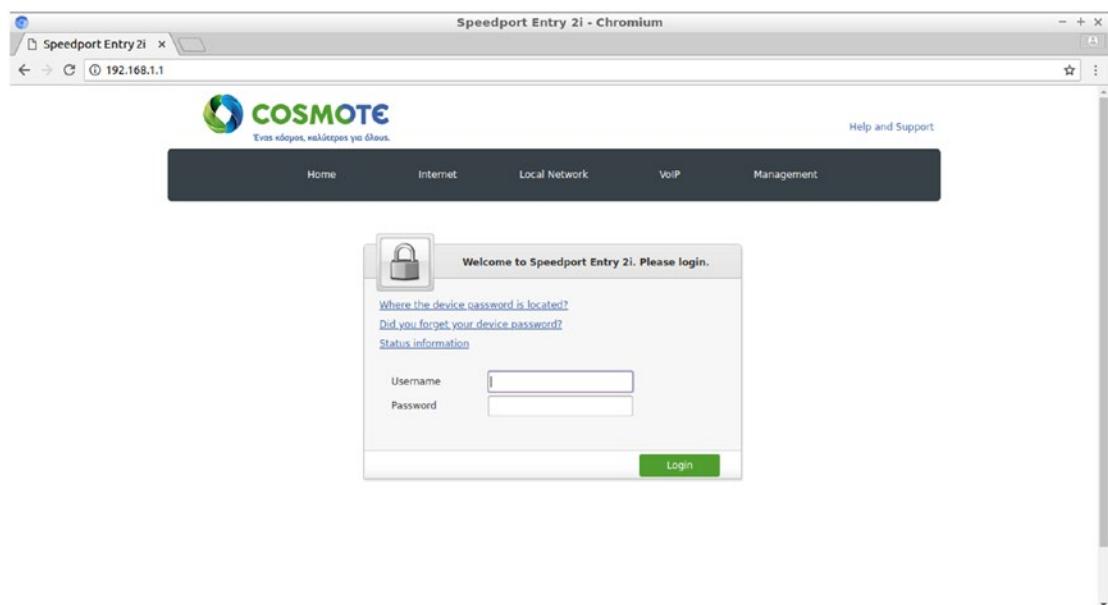


Figure 8-10. Connecting to a router's web interface

Locate the Port Forwarding (or similar name) tab, as shown in Figure 8-11.

CHAPTER 8 IMPLEMENTING SECURE SOCKETS LAYER ON YOUR SITE

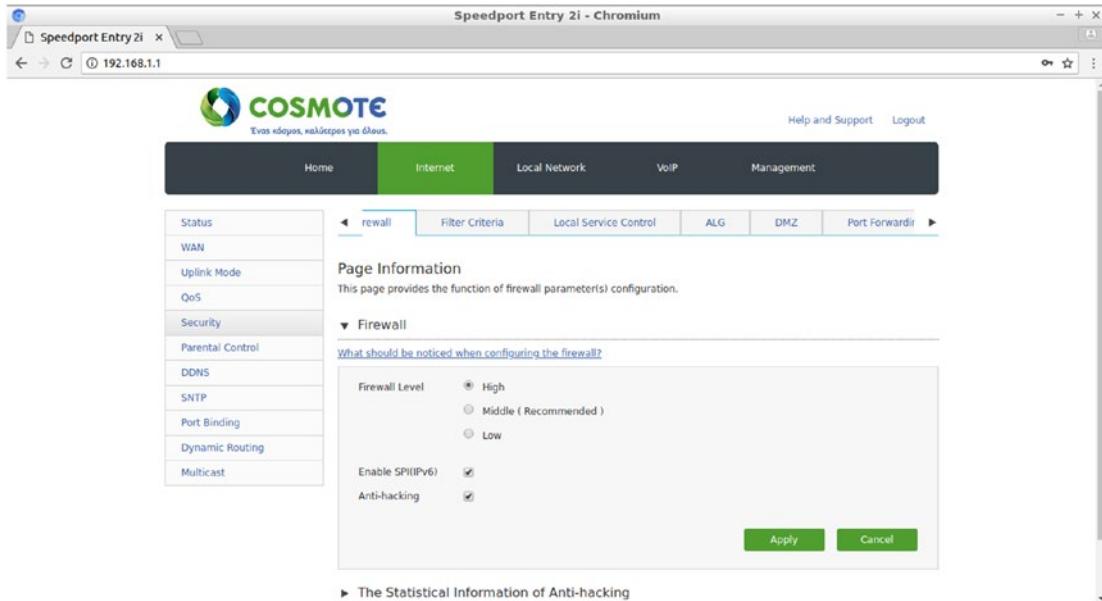


Figure 8-11. Locating the port forwarding service in the router's web interface

Create a new entry for port 443, as displayed in Figure 8-12, and save the new configuration.

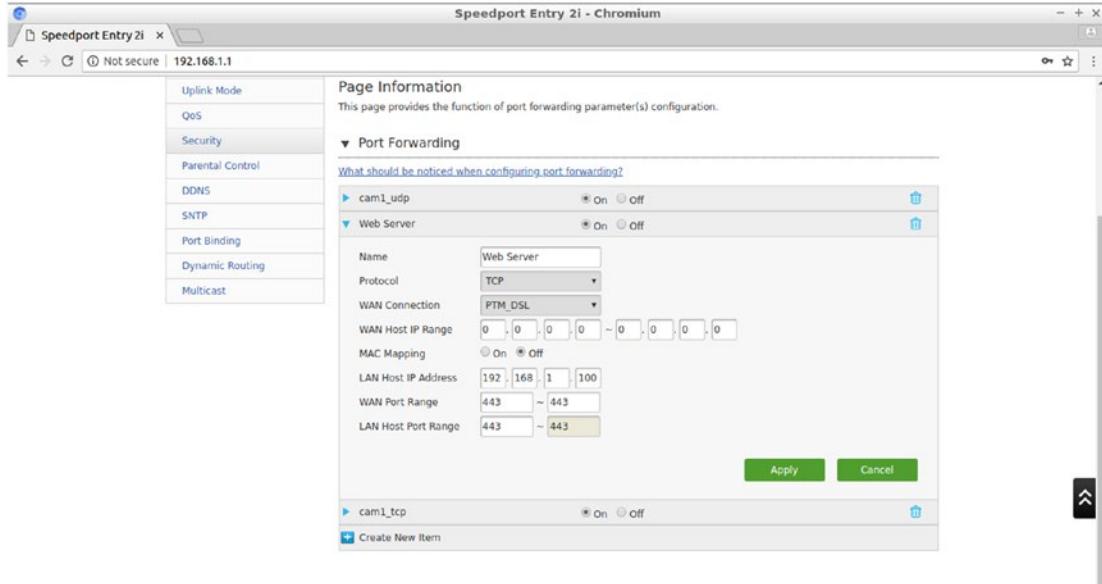


Figure 8-12. Creating a new entry in the router's Port Forwarding section for port 443

Your site is now available from outside your LAN. You can access it with the public IP address of your router or with the domain name of your web server. Test this using the router’s public IP address in the address bar of your browser. To find the public IP address of your router, use a “find my IP” online service or enter the following in the Linux terminal:

```
$ curl ifconfig.me
```

The command’s output consists just of the public IP address of your router. Here’s an example:

94.69.57.219

Enter the following URL in your browser’s address bar:

<https://94.69.57.219>

The web browser displays the “not secure” warnings discussed in the previous section, and if you proceed to download the web page, the directory index of your site is displayed, as shown in Figure 8-13.

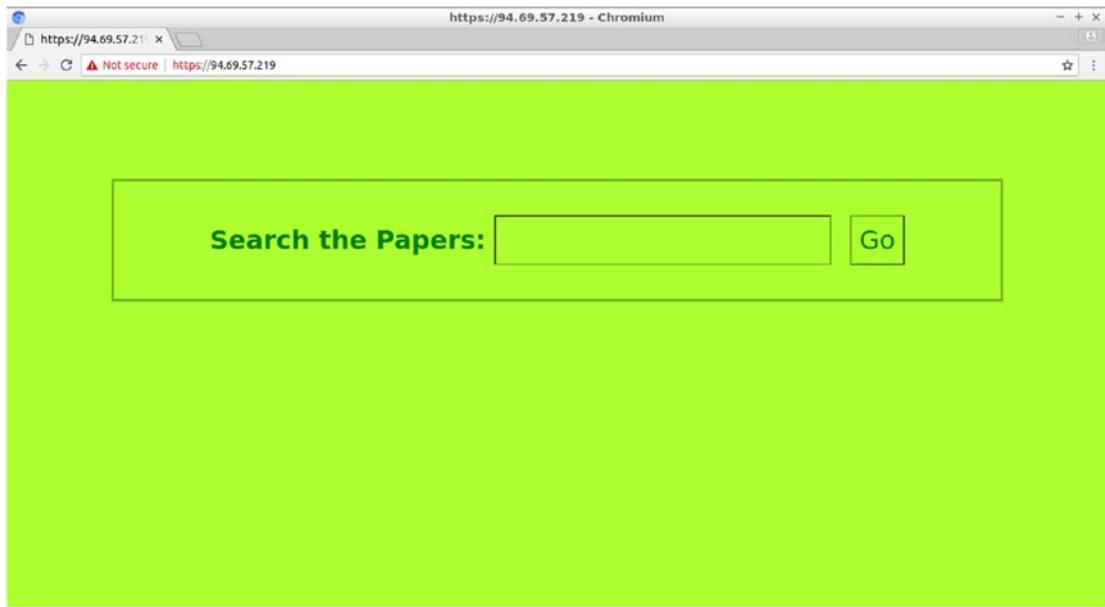


Figure 8-13. Testing the HTTPS connection to your site using the router's public IP address

You can also use the one of the FQDN DNS names you registered in Chapter 4. For instance, you can use the following:

`https://christos.ddns.net`

Or, you can use the one registered last with a more appropriate name.

`https://secureserver.ddns.net`

The directory index of the site also appears, as displayed in Figure 8-14.

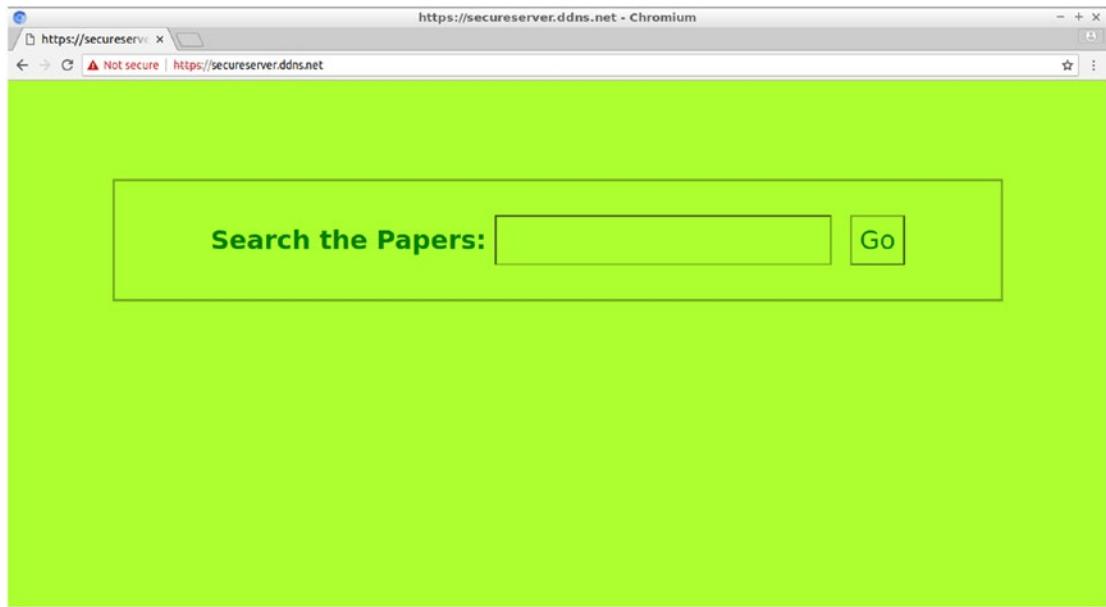


Figure 8-14. Testing the HTTPS connections to your site using an FQDN

At this point, you might expect that everything works fine. It doesn't, at least until you check the connection from outside your LAN. Using a online network tool, like webpagetest.org or geoscreenshot.com, you may discover that the web page test fails. At this point, you have to check the following:

- Your router (port forwarding)
- Your firewall
- Your ISP's inbound port 443 policy

Having set up the port forwarding feature on the router as described previously, the next immediate step is to check your firewall. You can use the ufw as follows:

```
$ sudo ufw enable  
$ sudo ufw allow 443  
$ sudo ufw status verbose
```

The previous command's output for this example, shown next, indicates that the ports used in the previous chapters are all opened:

```
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip
```

To	Action	From
--	-----	-----
10000/tcp	ALLOW IN	Anywhere
80,443/tcp (Apache Full)	ALLOW IN	Anywhere
443	ALLOW IN	Anywhere
8080	ALLOW IN	Anywhere
8181	ALLOW IN	Anywhere
10000/tcp (v6)	ALLOW IN	Anywhere (v6)
80,443/tcp (Apache Full (v6))	ALLOW IN	Anywhere (v6)
443 (v6)	ALLOW IN	Anywhere (v6)
8080 (v6)	ALLOW IN	Anywhere (v6)
8181 (v6)	ALLOW IN	Anywhere (v6)

If this still doesn't work, you have to contact your ISP and request unblocking the ports if they were indeed blocked. The ISP used for the examples of the book was blocking all inbound ports from 0 up to 1023 (the primary ports). While a connection to the site was achieved internally with the FQDN, the external test with the online service webpagetest.org failed. After a request to the ISP, the primary ports were opened. Online services like www.canyouseeme.org and www.yougetsignal.com can verify whether a specific port number of the computer whose IP address is entered in the corresponding textbox is open. Figure 8-15 displays the test run by yougetsignal.com, which shows that port 443 used for the connection to the server's address was closed.

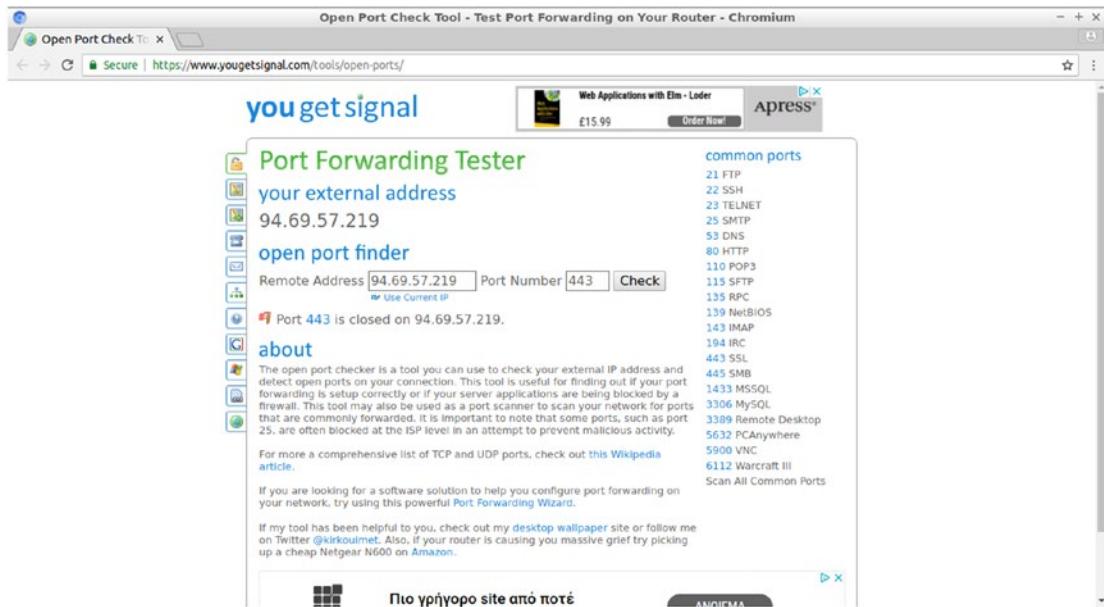


Figure 8-15. The online service [yougetsignal.com](https://www.yougetsignal.com) verifies that port 443 is closed

After the primary ports were opened by the ISP, the same online service ascertains that port 443 is open, as shown in Figure 8-16.

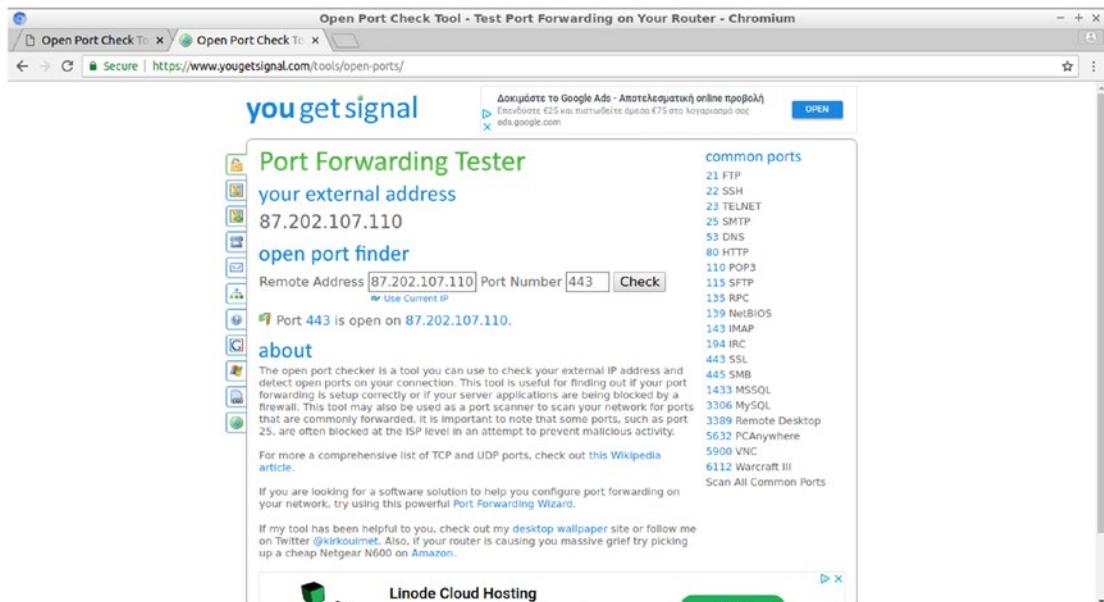


Figure 8-16. The online service [yougetsignal.com](https://www.yougetsignal.com) verifies that port 443 is open

Hint! In Chapter 5, you created an online web service like [webpagetest.org](https://www.webpagetest.org). In Chapter 10, you'll actually implement a service similar to <https://www.yougetsignal.com/tools/open-ports/>.

With port 443 released, you can make the final test for the connections to your site issued externally to your LAN. Use an online service like [webpagetest](https://www.webpagetest.org) or [geoscreenshot.com](https://www.geoscreenshot.com), as displayed in Figure 8-17, that displays your site from various locations in the world, or you can simply test the site using the mobile Internet connection from your cell phone.

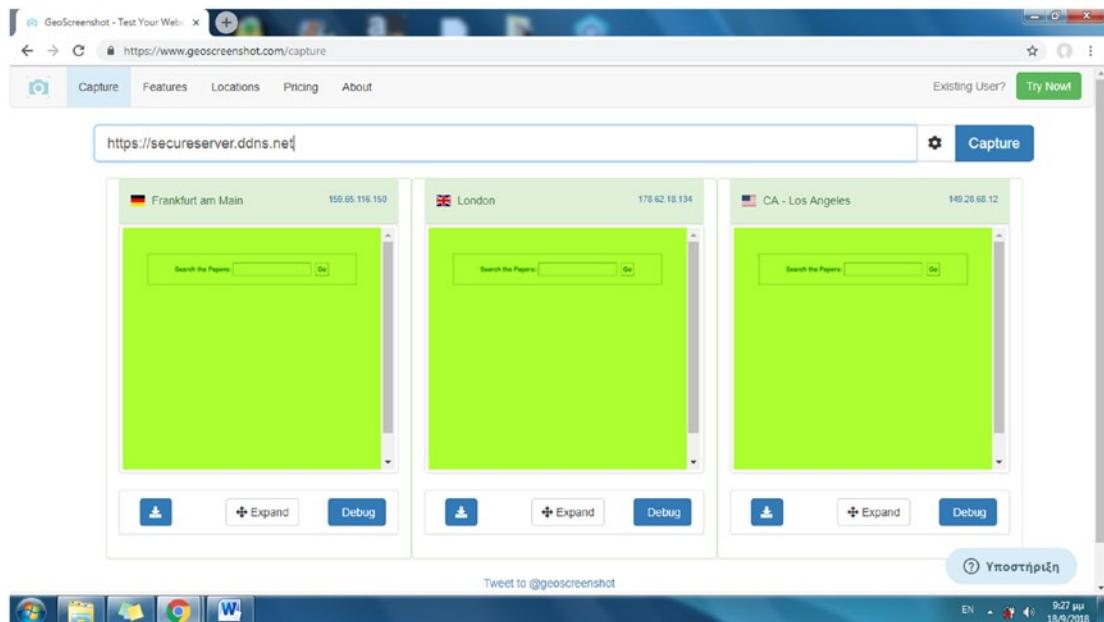


Figure 8-17. Checking the HTTPS connection to your site from an online service

HTTP Cookies and PHP Sessions

The HTTP protocol is *stateless*, which means there is no recording of any previous client request activity by default. Therefore, when a client logs in to an e-commerce site and adds an item to the shopping basket, a second login would be required to make a second purchase. To overcome this situation, commercial sites implement mainly two methods to maintain user-specific data: HTTP cookies and PHP sessions. A third method, using

input elements of type `hidden`, was examined in the previous chapter. However, its use is temporary. When a user exits the browser, the information contained in a hidden field is lost forever.

In the project for this chapter, PHP sessions will be used. A PHP session is a technique that is based on HTTP cookies, which are small pieces of data that are sent by the web server to the browser and are stored in text files as name-value pairs on the client's file system. With any further requests to the web server, the information stored in the specific client's cookies is attached to the HTTP request message headers. Therefore, cookies are carried back and forth between the client and the server with their values updated by the web server programs.

While with cookies the data is stored on the client side, with PHP sessions, all the data is stored on the server side, except the session ID, which is stored on the client side, usually in the form of a cookie. With PHP sessions, users or spyware programs do not have access to the stored information, and for this reason, PHP sessions are considered a more secure method than cookies.

Because PHP sessions are based on the cookies in the current chapter, you'll first create a cookies-based project and then a PHP sessions project. Also, in the following chapter, when you implement SSL certificates issued from a certificate authority, you'll create a project that utilizes PHP sessions to allow a client to log in to the site and while connected view the user-specific data.

Setting a Cookie with PHP

There are three basic reasons to use cookies for your site.

- Session maintenance, for allowing the user to log in to the site and stay connected, e.g., to add new items to a shopping card
- Customization, for recalling user preferences set by the user in previous visits to the site, e.g., maintaining a specific background color when the user e-mails are displayed in a webmail service
- User tracking, for recording and analyzing user behavior, e.g., e-commerce sites providing advertisements with products relevant to the ones included in web pages visited in the past by the user

The `setcookie()` PHP function defines a cookie to be sent along with the rest of the HTTP headers. This function has the following syntax:

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

Except name, the parameters are optional. The parameters of `setcookie()` are described next:

- **name:** Specifies the name of the cookie.
- **value:** Specifies the value of the cookie.
- **expires:** Specifies the time, set in Unix timestamp format, after which the cookie will become invalid.
- **path:** Specifies the URL path on the server for which the cookie will be available. If path is set to / (root), the cookie will be available within the entire domain.
- **domain:** Specifies the domain for which the cookie is available, e.g., webtoolsonline.servehttp.com.
- **secure:** Indicates that the cookie should be sent only in secure HTTPS connections.
- **httponly:** Indicates that the cookie will be available only through the HTTP protocol, without being accessible from scripting languages, e.g., JavaScript.

The following PHP code creates a cookie named `username`, assigns the value `tyler` to it, and specifies that the cookie expires in 30 days (`30 days * 24 hours * 60 minutes * 60 seconds`) from now. The function `time()` returns an integer containing the current time as a Unix timestamp.

```
<?php  
setcookie("username", "tyler", time()+30*24*60*60);  
?>
```

This function `setcookie()` calls should be placed before sending any output because cookies are sent in the headers of HTTP requests, and the headers of `http` requests come before the content of a web document.

Retrieving a Cookie Value from PHP

The PHP `$_COOKIE` superglobal variable is used to retrieve a cookie previously set. The following PHP code snippet types the value of the `username` cookie:

```
<?php
echo $_COOKIE["username"];
?>
```

Removing Cookies with PHP

To remove a cookie, you just call `setcookie()` for the cookie you want to delete with an expiration date that refers to the past. For instance to delete the `username` cookie, you could call the following:

```
<?php
setcookie("username", "", time()-3600);
?>
```

Creating a Site That Uses Cookies

You can create a PHP site that asks the visitor's first name and recalls it every time the user returns to the site until the cookie is expired.

In the document root of your web server (`/var/www/html` for both Apache and Lighttpd), create `cookies`, a new directory for this project, at the Linux terminal.

```
$ sudo mkdir /var/www/html/cookies
```

Create `index.php`, the directory index for this directory.

```
$ sudo gedit /var/www/html/cookies/index.php
```

Enter the following HTML and PHP source code:

```
<?php
if(isset($_POST)){
if (!isset($_COOKIE["username"])) && (isset($_POST['t1']))){
setcookie("username", $_POST['t1'], time()+3600); //1 hour
}
}
?>
```

CHAPTER 8 IMPLEMENTING SECURE SOCKETS LAYER ON YOUR SITE

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Cookies</title>
<style>
.p1{
color:blue;
font-size:32px;
text-align:center;
}
input{
color:blue;
font-size:32px;
}
.smiley{
color:orange;
font-size:160px;
text-align:center;
}
.center{
margin:auto;
}
</style>
</head>

<body>

<p class="p1">
<?php
if(isset($_COOKIE["username"])){
    echo "Hi " . htmlspecialchars($_COOKIE["username"]) . " I can see it is
you!";
    echo '<p class="smiley">&#x1f603;</p>';
}
```

```

} else{
    echo "Welcome to the site!";
    echo '<p class="smiley">&#x1f604;</p>';
}
?>
<p>

<div class="center">
<form method="post" action=<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
<p class="p1">Please enter your first name: <input type="text" name="t1">
</p>
</form>
</div>

</body>
</html>

```

Since no data is submitted with a POST method to index.php, the first time the PHP code evaluates, the following block of code does not run:

```

<?php
if(isset($_POST)){
if (!isset($_COOKIE["username"])) && (isset($_POST['t1']))){
setcookie("username", $_POST['t1'], time() + 3600); //1 hour
}
}
?>

```

The first time the PHP code evaluates, the PHP global variable `$_COOKIE["username"]` is not set yet. Therefore, on the first visit of the user at the site, the second block of PHP code runs the else part, which prints a general welcome message and also a smiley Unicode character in a large size, displayed in Figure 8-18.

The smiley chosen is a smiling face with open mouth and smiling eyes. This is Unicode character `😄`.



Figure 8-18. The smiley with the closed eyes appears before the user submits the first name

```
<p class="p1">
<?php
if(isset($_COOKIE["username"])){
    echo "Hi " . htmlspecialchars( $_COOKIE["username"] ) . " I can see it
    is you!";
    echo '<p class="smiley">&#x1f603;</p>';
} else{
    echo "Welcome to the site!";
    echo '<p class="smiley">&#x1f604;</p>';
}
?>
<p>
```

Any time the visitor returns to the site, the user will remain anonymous until deciding to enter their first name. In that case, the name, e.g., Christos, is entered in the textbox, as shown in Figure 8-18, and submitted by pressing the Enter key. The action attribute of the form, as evaluated in the following PHP block, specifies the same file, index.php, as the one that will receive the form's data:

```
<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>
```

When the form's data is received by index.php, the first block of PHP code runs.

```
<?php  
if(isset($_POST)){  
if (!(isset($_COOKIE["username"])) && (isset($_POST['t1']))){  
setcookie("username", $_POST['t1'], time() + 3600); //1 hour  
}  
}  
?>
```

In this code block, because the username cookie has not been set yet and also because a value (the user's name) was sent via the POST method by the t1 textbox, the username cookie is assigned the value sent by textbox t1.

For the next hour (3,600 seconds), when someone connects to the site, using the same browser at the same computer, the if part of the second PHP block runs and they receive a personalized welcoming with the specific name previously sent. Also, the smiley changes from the one with the smiling eyes to the one with the open eyes (Unicode character 😃). See Figure 8-19.

Hint! Each browser maintains its own cookies. To display this, visit with Chromium and with Firefox and provide different names in the textbox. Then visit the site another time using both Chromium and Firefox. Each browser recalls you with the specific name you submitted in the form of each browser.

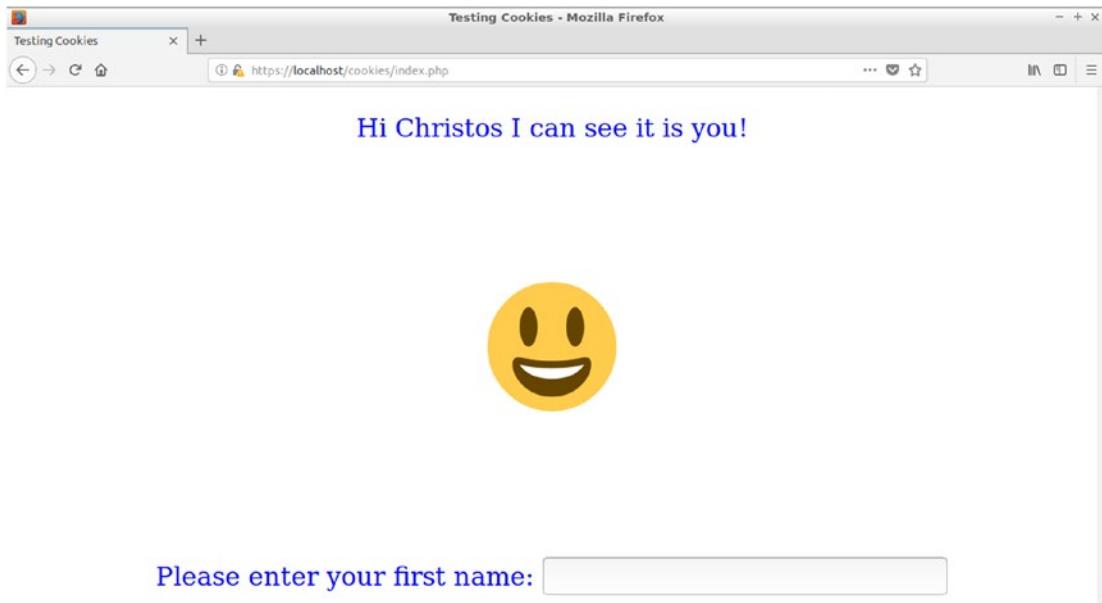


Figure 8-19. The smiley with the opened eyes along with a personalized welcome message appears when the user submits their first name

Hint! To run the previous project, cookies must be enabled on your browser. For instance, for Google Chromium, use the following steps:

1. Click the Customize and Control Chromium button (the control button with the three dots in the upper-right corner of the browser window) and select Settings.
 2. On the new tab with the settings, click the Advanced link at the bottom to expand the page contents.
 3. Under “Privacy and security,” click “Content settings.”
 4. Click Cookies and ensure that the “Allow sites to save and read cookie data (recommended)” option is on.
-

Viewing the Cookie Details in Your Browser

For the next hour for which the username cookie is valid, you can view the cookie's details in your browser. For instance, for Google Chromium, use the following steps.

Click the Customize and Control Chromium button and select Settings. On the new tab with the settings, click the Advanced link at the bottom to expand the page contents. Under "Privacy and security," click "Content settings." Click Cookies and then "See all cookies and site data." In the list that appears, locate the URL used for the site, e.g., localhost. Click the URL and locate the cookie's name, e.g., username. Click the arrow at the right to view details about the cookie, as displayed in Figure 8-20.

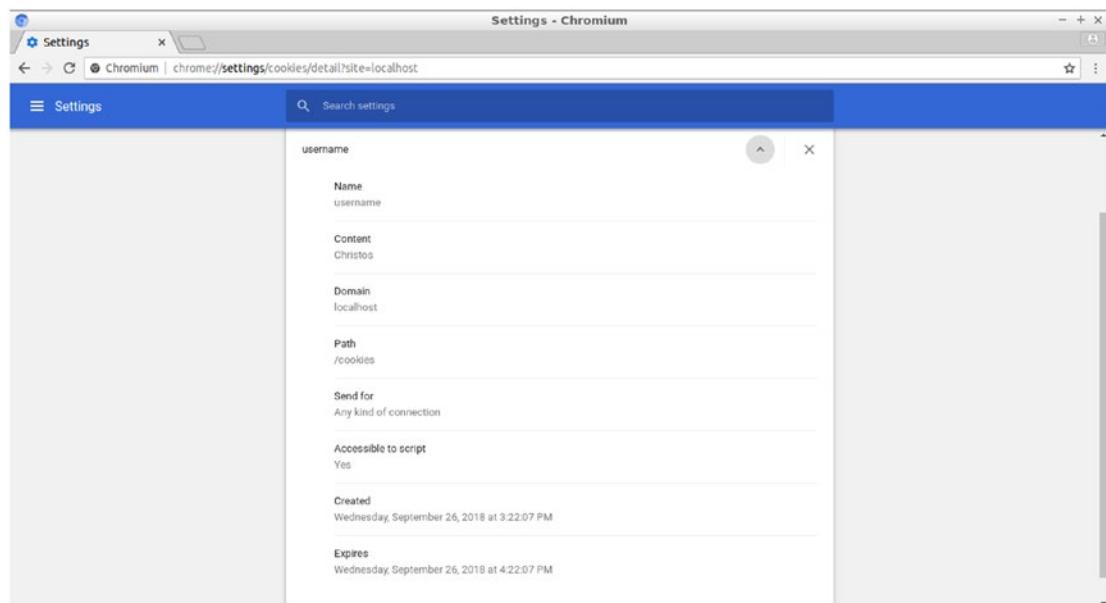


Figure 8-20. Displaying cookies details in the browser

Using Wireshark to View the HTTP Cookie Header

The cookies are transferred between the client (the browser) and the web server in the Cookie field in the HTTP protocol headers. You can use the Wireshark packet analyzer to view a cookie's fields. To install Wireshark, use the following command:

```
$ sudo apt-get install wireshark
```

To invoke Wireshark, use the following at the terminal:

```
$ sudo wireshark
```

The Wireshark program loads in a new window. As indicated in Figure 8-21, you can choose the interface from the interface list, for instance the wired Ethernet interface denoted as eno1 or with a similar name. If you are not sure about the interface name, you can run ifconfig at the terminal. Click the “Edit/apply display filter” button (the second one that depicts a funnel icon) to display only the packets that carry HTTP protocol messages. In the filter string textbox of the dialog that appears, enter **http**. Click the Apply and OK buttons. The text *http* appears in the Filter toolbar.

For this chapter, it is assumed that the ISP has released ports 80 and 443. In the previous chapters, the web servers were listening either to port 80 or to port 8080, and in a few examples to port 8181. Because the emphasis in those examples was to cover the case where the ISP bans inbound connections to port 80, examples using port 8080 were mostly used. Because HTTPS encrypts the HTTP headers, it is more straightforward with Wireshark to examine unencrypted headers. Therefore, an HTTP connection will be used next. If you have not used port 80 so far with Lighttpd, follow the next steps.

Change the Lighttpd configuration file for the server to accept also HTTP (except HTTPS) by using the following:

```
$ sudo gedit /etc/lighttpd/lighttpd.conf
```

In the configuration file, add the following conditional directive for the web server to utilize port 80 (except port 443):

```
$SERVER["socket"] == ":80" {
    server.document-root      = "/var/www/html"
    server.upload-dirs        = ( "/var/cache/lighttpd/uploads" )
    server.errorlog           = "/var/log/lighttpd/error.log"
    server.pid-file           = "/var/run/lighttpd.pid"
    server.username            = "www-data"
    server.groupname           = "www-data"
}
```

Save the file and reload the server to enable the new configuration by using the following:

```
$ sudo service lighttpd force-reload
```

For Apache to use HTTP port 80, the default configuration file `000-default.conf` should contain at a minimum the following directives:

```
<VirtualHost 192.168.1.100:80>
    DocumentRoot "/var/www/html"
</VirtualHost>
```

If you have altered the file `000-default.conf`, enable the new configuration by using the following:

```
$ sudo service apache2 force-reload
```

Next, open Wireshark by entering the following at the Linux terminal:

```
$ sudo wireshark
```

In the Wireshark main toolbar, click the “Start capturing packets” button, which is the blue fin button, and from another computer connect to the web server via port 80, using the following URL:

`http://192.168.1.100/cookies`

In the textbox of the directory index enter a name (**Christos** was used in this example) and press Enter to set the `username` cookie. The requests from this browser for the specific URL will carry the cookie in the HTTP headers of the packets sent to the web server.

On the web server, terminate the packet capturing process using the “Stop capturing packets” button, which is the red square button. You can also enter `http` in the “Apply a display filter” textbox to list only HTTP packets. In Figure 8-21, an HTTP packet is selected destined for the web server (IP address 192.168.1.100).

CHAPTER 8 IMPLEMENTING SECURE SOCKETS LAYER ON YOUR SITE

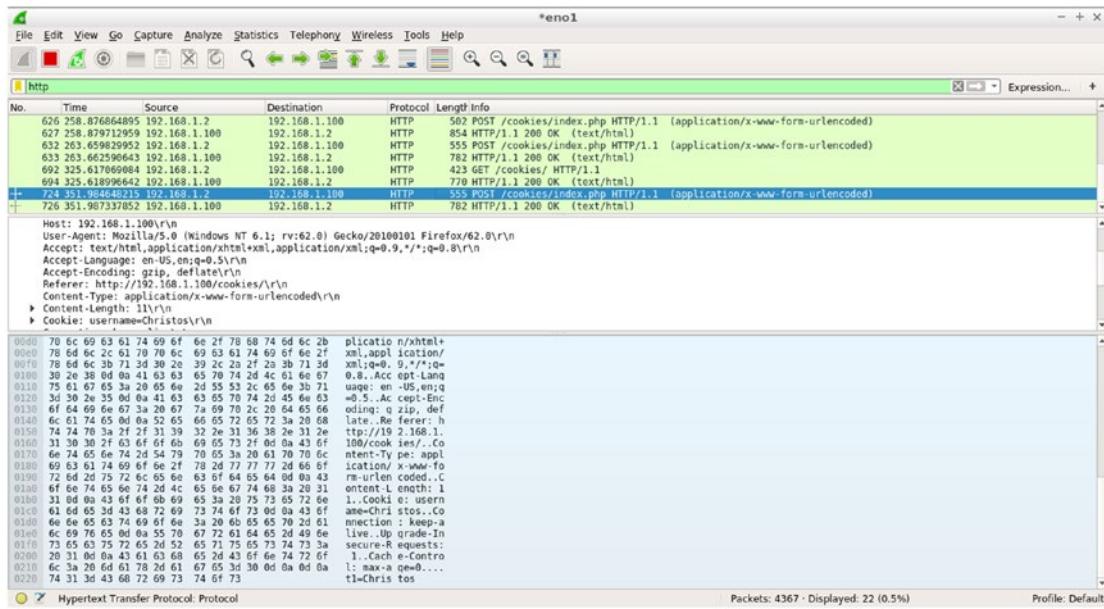


Figure 8-21. The Wireshark window displays the client request, carrying the cookie

The following are the HTTP headers for the specific packet:

```

POST /cookies/index.php HTTP/1.1\r\n
Host: 192.168.1.100\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:62.0) Gecko/20100101
Firefox/62.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;
q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Referer: http://192.168.1.100/cookies/\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 11\r\n
Cookie: username=Christos\r\n

```

The cookie with the name `username` and the value `Christos` is included in the cookie header in this and any other packet sent to the web server from the specific browser.

Using Browser Tools to View the HTTP Cookie Header

Many web browsers like Chromium and Firefox enable the user to view many details about the web page content and the connections, and this feature can be used to display the cookie header. In this section, you'll use Chromium to display the cookies set from another site, e.g., from minix3.org, using an HTTPS connection.

Visit the site and respond positively to the message about accepting cookies to the site (if such a message is used). Click the Customize and Control Chromium button (the three dots in the upper right of the window) and select “More tools” and then “Developer tools.” A new pane appears on the right of the window with the Network tab selected by default. Reload the page to view the client request messages and the web server replies. Click the web page request called, e.g., minix3.org. The HTTP header fields appear, as displayed in Figure 8-22. Scroll down to locate the cookie header.

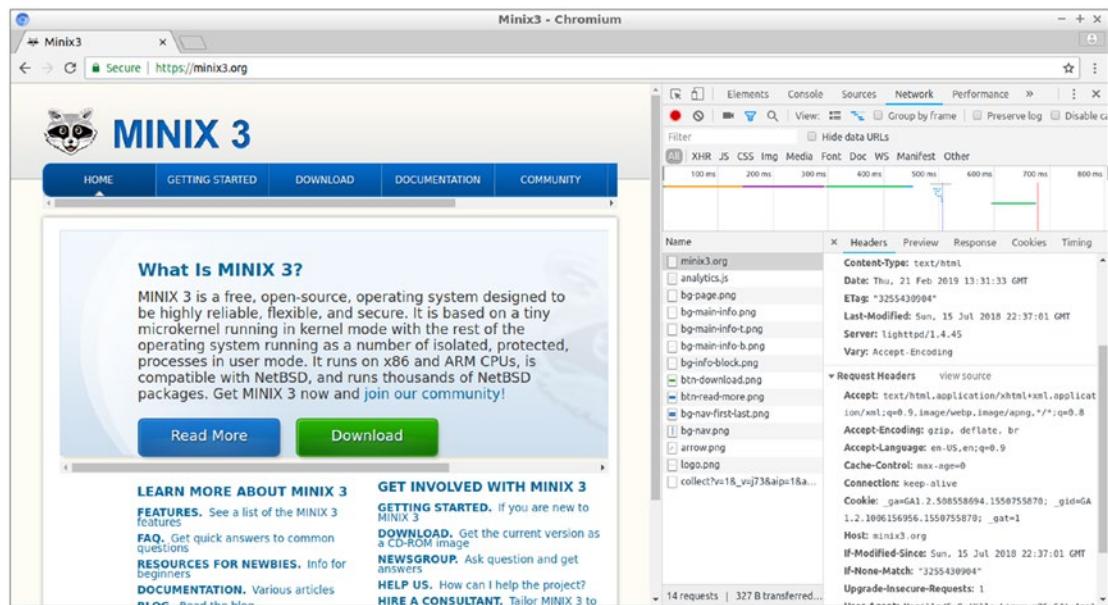


Figure 8-22. Viewing the HTTP headers from the Chromium browser

Similarly, you can view the cookie header from Firefox by clicking the Open menu button (the three stripes on the upper right of the window) and then selecting Web Developer and then Network. Reload the web page to view the request messages and the web server replies. Click the GET request of type html. The HTTP protocol headers, among them the cookie header, appear, as shown in Figure 8-23 after scrolling.

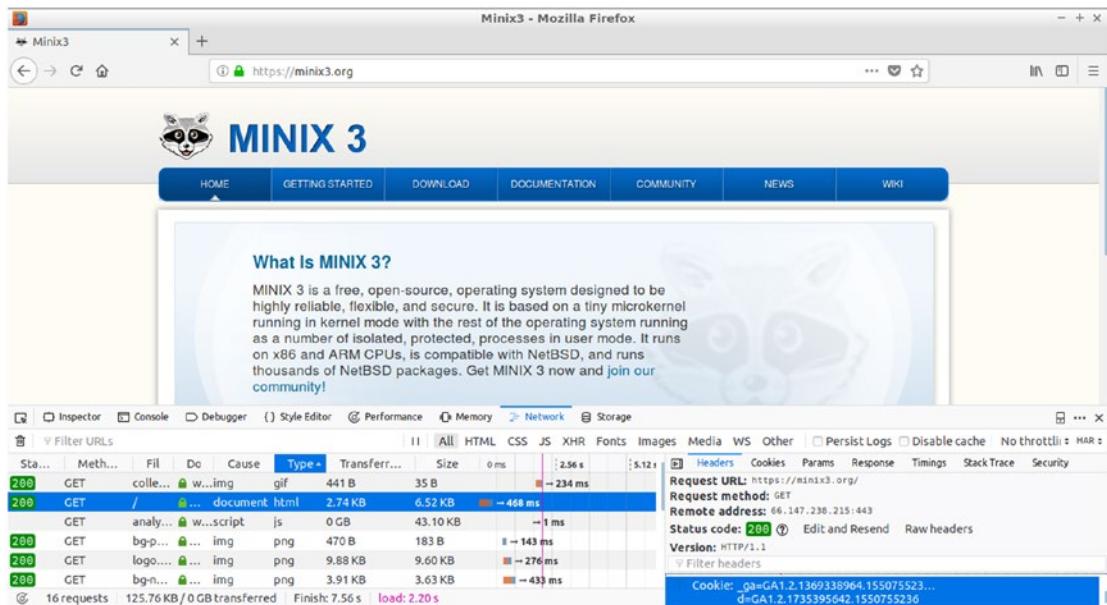


Figure 8-23. Viewing the HTTP headers from the Firefox browser

Using PHP Sessions

Retaining data from the client and server interaction using cookies has some security drawbacks. First, because they are stored on the user's system, they are less secure against potential attacks that could modify the cookie contents. With PHP sessions, sensitive data is safely stored in a centralized way on the web server. There is also another disadvantage. With cookies, each time the client makes a request to the server for a specific URL, all the cookies for this site are carried to the server along with the request. For ten cookies, each one with a size of 4KB, the browser will upload 40KB of extra data for each page being viewed, which will decrease the performance of your site.

Data with PHP sessions, on the other hand, is stored to temporary text files locally on the web server. Only the session ID, which is the unique identity of the session, is saved on the client side, usually as a cookie. Therefore, all sensitive information is stored in a centralized place under the authority of the site administrator.

To begin a new session, use the PHP `session_start()` function, for instance:

```
<?php  
session_start();  
?>
```

The function `session_start()` checks whether the session is already established by searching for a session ID on the client computer with a value of an alphanumeric string. The session ID is the only piece of data that is stored on the client side. If a session ID is found, i.e., the session is already started, session variables stored on the server computer for the specific client can be set and retrieved. Otherwise, a new session is started by creating a new session ID and storing it on the client as a cookie with the name `PHPSESSID`. The file that stores the session variables is automatically created on the server in a designated directory and with a file name consisting of the ID value prefixed by `sess_`, e.g., `sess_hg7egmhroh4vbk9925v8vu8op4`.

The PHP function `session_start()` (for the same reasons as `setcookie()`) should be called at the beginning, before the HTML tags, of each page of your site that will participate in the given session. All session data, stored as variable-value pairs, are then shared among the current page and the other web pages of the site for the specific session. This session is identified by the `PHPSESSID` cookie, which is valid for a specific browser on the client's system for the specified period of time.

The global PHP variable `$_SESSION` is the array used to store the session variables. The following PHP code creates a session and stores the values `Henrik` and `Ibsen` in the session variables `firstname` and `lastname`, respectively:

```
<?php  
session_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
<?php  
$_SESSION["firstname"] = "Henrik";
```

```
$_SESSION["lastname"] = "Ibsen";
?>
</body>
</html>
```

The session is terminated with the `session_unset()` and `session_destroy()` PHP functions. The function `session_unset()` removes all session variables, and `session_destroy()` deletes the session.

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
session_unset();
session_destroy();
?>
</body>
</html>
```

Running a PHP Session Example

In the document root of your server (usually `/var/www/html` for both Apache and Lighttpd), create `sessions`, which is the new directory for this project.

```
$ sudo mkdir /var/www/html/sessions
```

Create a site consisting of three web pages: `index.php` used as the directory index and `page1.php` and `page2.php`, which are two web pages linked from the directory index.

```
$ cd /var/www/html/sessions
$ sudo touch index.php page1.php page2.php
```

On `index.php`, the site's home page, the user has the option to personalize the web pages and choose one of three available background colors (orange, light blue, and lime) by clicking the corresponding colored button. The user can also choose the white

button to reset the background color to the initial one. By clicking a colored button, the corresponding color applies to not only the home page but also to any other web page of the site. In this example, the home page includes two links called Books and Magazines, leading to `page1.html` and `page2.html`, respectively. As displayed in Figure 8-24, light blue is selected after the user clicks the light blue button (the second one).

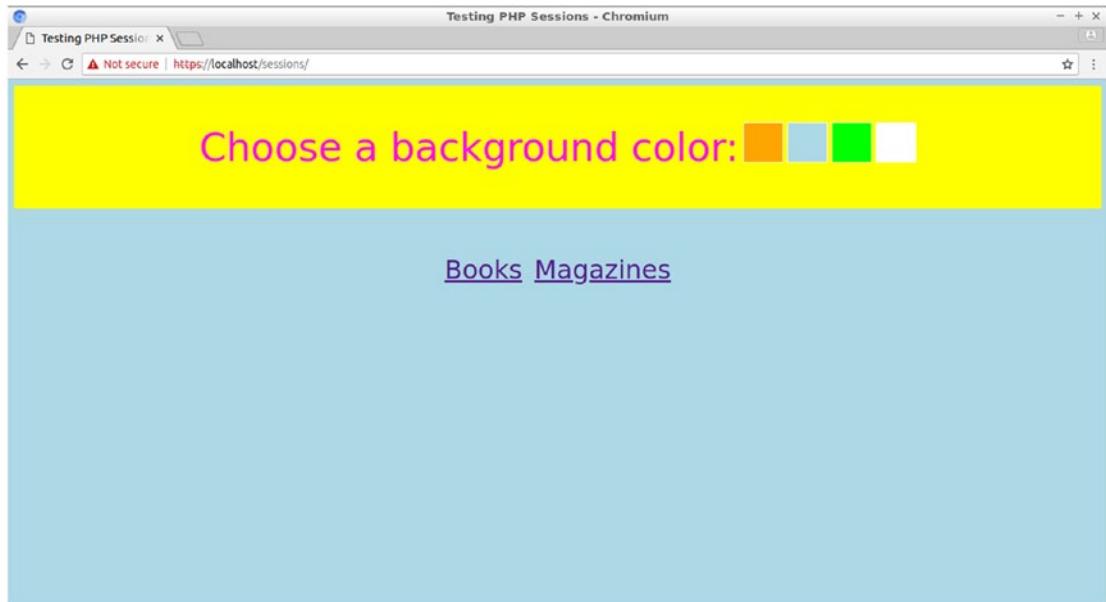


Figure 8-24. The home page of the PHP sessions site with the light blue background color selected

The user can click a link to see the background color of the corresponding web page. Figure 8-25 displays `page1.php` after the Books link is clicked.

CHAPTER 8 IMPLEMENTING SECURE SOCKETS LAYER ON YOUR SITE

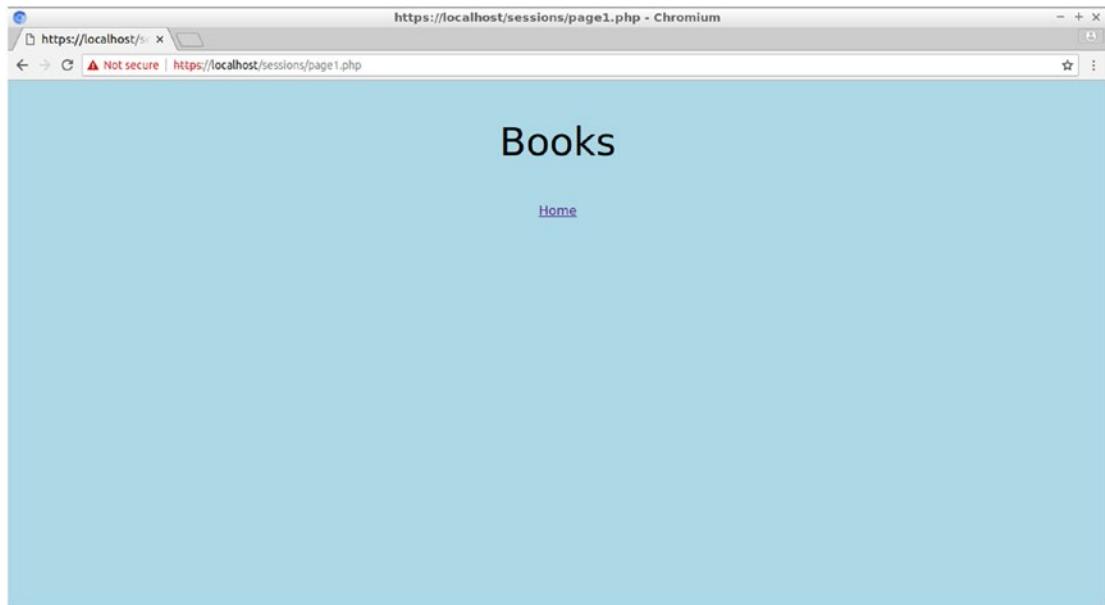


Figure 8-25. The Books web page retains the user's color selection

Figure 8-26 displays page2.php after the Magazines link is clicked.

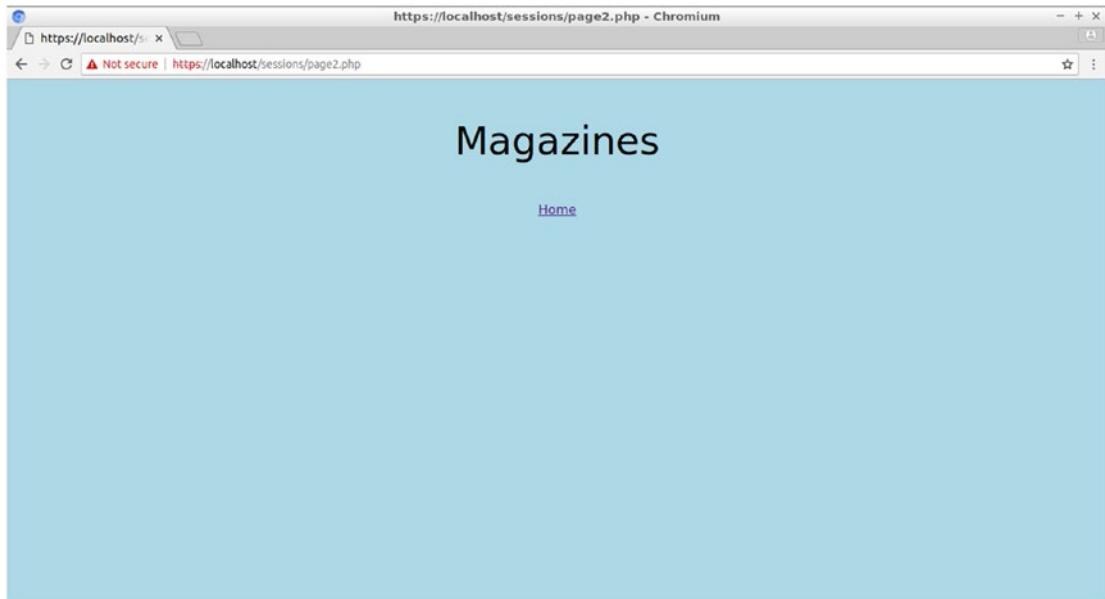


Figure 8-26. The Magazines web page retains the user's color selection

The PHP session mechanism is implemented in this project to store the user preferences as PHP session variables that are automatically available to any PHP web page of the site that calls the function `session_start()` in a PHP block before the web page's HTML source code. The session variable's value is used to change the background color of the current page but is also available to all other web pages of the site and is used as the `background-color` property of the body selector in the CSS section. Next you'll create the home page of the site.

The Source Code for index.php

Use the `gedit` command to edit `index.php`.

```
$ sudo gedit /var/www/html/sessions/index.php
```

Enter the following source code in the `gedit` window and click the Save button:

```
<?php  
session_start();  
?>  
  
<!DOCTYPE html>  
<html>  
<head>  
<title>Testing PHP Sessions</title>  
<style>  
a{  
font-size:32px;  
padding:5px;  
}  
body{  
background-color:<?php  
if(isset($_SESSION["favcolor"])){
echo $_SESSION["favcolor"] . " ";
} else {
echo "white";
}
?>
}
```

```
p{  
font-size:48px;  
color:fuchsia;  
}  
.center{  
text-align:center;  
background-color:yellow;  
}  
.center2{  
text-align:center;  
}  
.div1 {  
    display: inline-block;  
}  
</style>  
</head>  
<body>  
  
<?php  
if(isset($_POST['s1'])){  
$_SESSION["favcolor"] = "orange";  
echo '  
<script>  
document.body.style.background = "orange";  
</script>  
';  
}  
} else if(isset($_POST['s2'])){  
$_SESSION["favcolor"] = "lightblue";  
echo '  
<script>  
document.body.style.background = "lightblue";  
</script>  
';  
}  
} else if(isset($_POST['s3'])){  
$_SESSION["favcolor"] = "lime";  
echo '
```

```
<script>
document.body.style.background = "lime";
</script>
';
} else if(isset($_POST['s4'])){
session_unset();
session_destroy();
echo '
<script>
document.body.style.background = "white";
</script>
';
}
?>

<div class="center">
<div class="div1">
<p>Choose a background color:</p>
</div>
<div class="div1">
<form method="POST" action=<?php echo htmlentities($_SERVER['PHP_SELF']); ?>">
<input type="submit" name="s1" value="" style="width:50px;height:50px;
border:0;background-color:orange;border:1px solid white;">
</div>
<div class="div1">
<input type="submit" name="s2" value="" style="width:50px;height:50px;
border:0;background-color:lightblue;border:1px solid white;">
</div>
<div class="div1">
<input type="submit" name="s3" value="" style="width:50px;height:50px;
border:0;background-color:lime;border:1px solid white;">
</div>
```

CHAPTER 8 IMPLEMENTING SECURE SOCKETS LAYER ON YOUR SITE

```
<div class="div1">
<input type="submit" name="s4" value="" style="width:50px;height:50px;
border:0;background-color:white;border:1px solid white;">
</form>
</div>
</div>

<br><br><br>
<div class="center2">
<div class="div1">
<a href="page1.php">Books</a>
</div>
<div class="div1">
<a href="page2.php">Magazines</a>
</div>
</div>
</body>
</html>
```

The first block of the PHP code starts a new session (if not already started).

```
<?php
session_start();
?>
```

Next comes the CSS part inside the HTML head element. Certain styles are being applied, such as the position of the buttons and the links and the color and size of the text, but the noticeable part is the one that sets the background color of the page.

```
body{
background-color:<?php
if(isset($_SESSION["favcolor"])){
echo $_SESSION["favcolor"] . ";";
} else {
echo "white;";
}
?>
}
```

The first time the user views the home page—and until the user clicks a colored button—the web page's background color is white. The four colored buttons for selecting the background color are actually input elements of the `submit` type of the same form that submits the user selection to the same PHP page, as indicated by the global variable `$_SERVER['PHP_SELF']`. Each submit button is differentiated by a distinct name. The names used are `s1`, `s2`, `s3`, and `s4`. For instance, the second button is created with the following input element:

```
<input type="submit" name="s2" value="" style="width:50px;height:50px;
border:0;background-color:lightblue;border:1px solid white;">
```

Then, according to the button's name used for the data submission, the PHP and JavaScript source code runs.

With PHP, the PHP session variable `favcolor` is set. For instance, if `s2` (the light blue button) was the one submitted, the data `favcolor` is assigned to `lightblue`.

```
} else if(isset($_POST['s2'])){
$_SESSION["favcolor"] = "lightblue";
echo '
<script>
document.body.style.background = "lightblue";
</script>
```

Also, JavaScript runs locally in the web browser and sets instantly the background of the current web page to light blue.

Hint! As is the case with any other CSS property, `background-color` changes its name when accessed by JavaScript. For the `background-color` property, the name in JavaScript becomes `background`.

All other web pages of the site when visited retain the light blue background with the `favcolor` value setting in the CSS section of the `background-color` property of the `body` selector. The following code snippet can be used in any other web page of the site:

```
body{
background-color:<?php
if(isset($_SESSION["favcolor"])){
echo $_SESSION["favcolor"] . ";";
```

```

} else {
echo "white;";
}
?>
}

```

With the white button, there is an exception. By clicking this button, JavaScript is used locally on the client to set instantly the background color of the current web page, but the PHP code does not set favcolor; rather, it destroys the session. When favcolor is not set for any other web page visited, the else part of the previous source code runs, and the CSS background-color property for the body selector defaults to white.

?>

The last part of code in index.php creates two links to page1.php and page2.php called Books and Magazines, respectively.

```

<div class="center2">
<div class="div1">
<a href="page1.php">Books</a>
</div>
<div class="div1">
<a href="page2.php">Magazines</a>
</div>
</div>

```

You'll create the two web pages page1.html and page2.html next.

The Source Code for page1.html and page2.html

With the user selections, the favcolor PHP session variable obtains a value, and this value will be available to the other two web pages of the site, e.g., page1.php and page2.php, given that they also implement PHP sessions. Here you'll create page1.html, the Books page, and page2.html, the Magazines page.

At the Linux terminal, use gedit to insert the source code to page1.php.

```
$ sudo gedit /var/www/html/sessions/page1.php
```

Enter the following source code and save the file:

```
<?php
session_start();
?>

<!DOCTYPE html>
<html>
<head>
<style>
body{
background-color:<?php
if(isset($_SESSION["favcolor"])){
echo $_SESSION["favcolor"] . " ";
} else {
echo "white;";
}
?>
}
p{
text-align:center;
font-size:48px;
}
.center{
text-align:center;
}
</style>
</head>
<body>
<div>
<p>Books</p>
</div>
<div class="center">
<a href="index.php">Home</a>
</div>
</body>
</html>
```

The first PHP block connects to the session started from `index.php` using the same function call with `index.php, session_start()`.

```
<?php
session_start();
?>
```

After the function `session_start()` is called, if the PHP session variable `favcolor` was previously set in `index.php`, it becomes also available to `page1.php`. Its value is used to set the background color of `page1.php`. If the user decides to click the Books link and navigate to `page1.php` without first selecting one of the three colors, the `favcolor` variable is not set, and the default color (white) applies for the background color of the page.

```
body{
background-color:<?php
if(isset($_SESSION["favcolor"])){
echo $_SESSION["favcolor"] . ";";
} else {
echo "white;";
}
?>
```

In the body of `page1.php`, there is some text (Books) and a link for returning to the home page.

```
<body>
<div>
<p>Books</p>
</div>
<div class="center">
<a href="index.php">Home</a>
</div>
</body>
```

Create the web page `page2.php`.

```
$ sudo gedit /var/www/html/sessions/page2.php
```

Enter the following for the page1.html source code and save the file:

```
<?php
session_start();
?>

<!DOCTYPE html>
<html>
<head>
<style>
body{
background-color:<?php
if(isset($_SESSION["favcolor"])){
echo $_SESSION["favcolor"] . ";";
} else {
echo "white;";
}
?>
}
p{
text-align:center;
font-size:48px;
}
.center{
text-align:center;
}
</style>
</head>

<body>
<div>
<p>Magazines</p>
</div>
```

```
<div class="center">
<a href="index.php">Home</a>
</div>
</body>
</html>
```

From `index.html`, select one of the available background colors, and you'll see they also apply on `page1.html` and `page2.html` by visiting the corresponding links, Books and Magazines.

Experimenting with the Sessions Project

Comment out the `session_start()` function for `page2.php` to temporarily deactivate it.

```
<?php
//session_start();
?>
```

Click one of the three colored buttons on the home page to select a color. Visit then the Books and Magazines links. As expected, only `page1.php` has the same color background as the one selected. Uncomment the `session_start` function to enable again the session for `page2.php`.

Visit the home page of the site and select a color, e.g., orange. Locate the file where the session data is stored and view its contents. You can find this information in the `php.ini` file, which is the PHP configuration file. Load the `info.php` web page created in Chapter 2 that evaluates the `phpinfo()` function, which displays lots of information about the PHP configuration. Use, for instance, the following in your browser's address bar:

`https://localhost/info.php`

Figure 8-27 shows the web page.

System	
Build Date	May 2 2018 12:43:48
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/cgi
Loaded Configuration File	/etc/php/7.0/cgi/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/cgi/conf.d
Additional .ini files parsed	/etc/php/7.0/mysqli.ini, /etc/php/7.0/cgi/conf.d/10-opcache.ini, /etc/php/7.0/cgi/conf.d/10-pdo.ini, /etc/php/7.0/cgi/conf.d/20-curl.ini, /etc/php/7.0/cgi/conf.d/20-ctype.ini, /etc/php/7.0/cgi/conf.d/20-dom.ini, /etc/php/7.0/cgi/conf.d/20-fpm.ini, /etc/php/7.0/cgi/conf.d/20-ftp.ini, /etc/php/7.0/cgi/conf.d/20-gettext.ini, /etc/php/7.0/cgi/conf.d/20-iconv.ini, /etc/php/7.0/cgi/conf.d/20-json.ini, /etc/php/7.0/cgi/conf.d/20-phar.ini, /etc/php/7.0/cgi/conf.d/20-pdo_mysql.ini, /etc/php/7.0/cgi/conf.d/20-phar.ini, /etc/php/7.0/cgi/conf.d/20-posix.ini, /etc/php/7.0/cgi/conf.d/20-readline.ini, /etc/php/7.0/cgi/conf.d/20-shmop.ini, /etc/php/7.0/cgi/conf.d/20-sockets.ini, /etc/php/7.0/cgi/conf.d/20-sysvmsg.ini, /etc/php/7.0/cgi/conf.d/20-sysvshm.ini, /etc/php/7.0/cgi/conf.d/20-tokenizer.ini
PHP API	
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API20151012.NTS
PHP Extension Build	API20151012.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled

Figure 8-27. The info.php web page source code calls function phpinfo()

As derived from this page in the system used for the project, the php.ini configuration file is as follows:

/etc/php7/cgi/php.ini

The file php.ini is a long one, so pipe its output to grep to search for keyword save_path.

```
$ sudo cat /etc/php7/cgi/php.ini | grep save_path
```

The command's output for this example is as follows:

```
; session.save_path = "N;/path"
; session.save_path = "N;MODE;/path"
;session.save_path = "/var/lib/php/sessions"
; (see session.save_path above), then garbage collection does *not*
```

Run next ls to view the contents of the /var/lib/php/sessions directory.

```
$ sudo ls -l /var/lib/php/sessions
```

The command's output is as follows:

```
total 4
-rw----- 1 www-data www-data 22 Sep 25 16:38 sess_
hg7egmhroh4vbk9925v8vu8op4
```

The file name consists of the prefix `sess_` and the session ID:

```
hg7egmhroh4vbk9925v8vu8op4
```

To view the file's data, enter the following:

```
$ sudo cat /var/lib/php/sessions/sess_hg7egmhroh4vbk9925v8vu8op4
```

The output for the favcolor icon set to orange is as follows:

```
favcolor|s:6:"orange";
```

Switch to your browser and click the white button. The PHP session is destroyed, as revealed from the `ls -l` command:

```
$ sudo ls -l /var/lib/php/sessions
```

The command's output is as follows:

```
total 0
```

While all the information about the session is stored on the server side, the session ID is stored as a cookie on the client's computer. Use the settings of your browser to locate the specific cookie. For the Chromium browser, click the Customize and Control Chromium button (the one with the three dots in the upper right of the window) and select Settings. On the new Chromium tab that opens with the available settings, click the Advanced link at the bottom to expand the page contents. Under "Privacy and security," click "Content settings." Click Cookies and then "See all cookies and site data." In the list that appears, locate the URL used for the site, e.g., localhost. Click the URL and locate the cookie's name, which is PHPSESSID. Click the arrow at the right to view details about the cookie. Figure 8-28 displays the cookie properties.

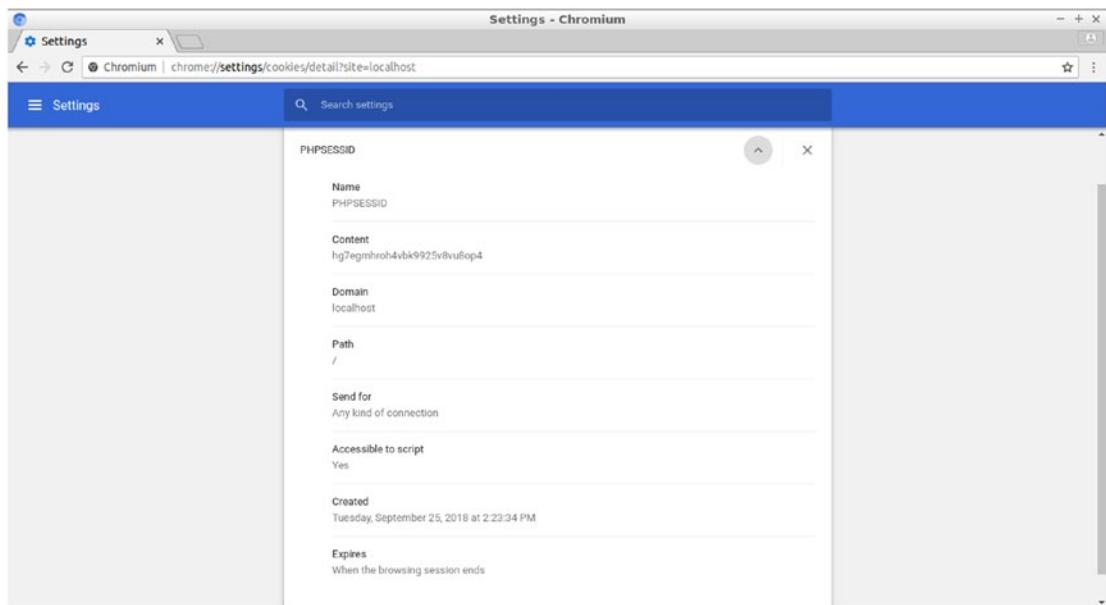


Figure 8-28. Displaying the properties of the PHPSESSID cookie

As expected, the value of the cookie is hg7egmhroh4vbk9925v8vu8op4.

This cookie for the default session remains until the time indicated in the Expires section under “When the browsing session ends.” Close the browser and open again the Settings web page. The cookie is now deleted.

Summary

In this chapter, you did the following:

- Implemented SSL for your site using a self-signed certificate
- Used HTTP cookies to maintain the state of the otherwise stateless HTTP protocol by storing data across web pages on the client side
- Used PHP sessions to store sensitive data across web pages on the web server side

In the next chapter, you’ll enhance your site’s credibility by enabling HTTPS connections with an SSL certificate from a certificate authority.

CHAPTER 9

Running Your Site with a Certificate from a Certificate Authority

In this chapter, you will upgrade your site, which so far implements HTTPS with a self-signed certificate, by obtaining an SSL certificate from a certificate authority (CA). As a result, the web browser warnings about an insecure site will not appear, and instead a padlock icon, indicating secure communication, will appear on the left of the URL in the address bar of your browser. To obtain an SSL certificate, you usually need to own a second-level domain (SLD), like `httpsserver` in `httpsserver.eu`, as opposed to a name with the second-level domain owned by a DDNS company (like the names used in the previous chapters with the SLD `ddns.net`). For this reason, in this chapter, you'll learn how to obtain a domain name, and I'll discuss the process to obtain both the domain name and the SSL certificate. The cost at the time of this writing is about \$8/year for the domain name and \$19/year for the SSL certificate. However, it is not required that you register to run the projects. The source code for the projects used in this and the following chapter can run with the old server configuration; you will just continue to get the browser warnings.

By obtaining an SSL certificate from a CA, you will create a site that provides a login connection. For a system requiring a user login, implementing SSL is an indispensable option because the password and all other sensitive data is transmitted encrypted. In this project, you will also learn how to encrypt the user password when stored in the database.

The user enters the username and the password, and by utilizing PHP sessions, the user will stay connected and can view user-specific data while browsing all the web pages of this site until a logout is issued.

Obtaining Your Own Domain Name

Plenty of companies offer domain name registration. You first have to search the company's site to see whether the name you want is available. An SLD name such as `httpsserver` may be available for certain top-level domains (TLDs) such as `.eu` but be unavailable for another TLD like `.com`. Usually the prices for the domain names are different for the various TLDs.

Visit the Domains link at [Dynu.com](https://www.dynu.com), the provider used in this example. In the textbox, enter your preferred domain name, e.g., `httpsserver.eu`, and click the Search button. Figure 9-1 displays the results indicating that `httpsserver.eu` is available for purchase.

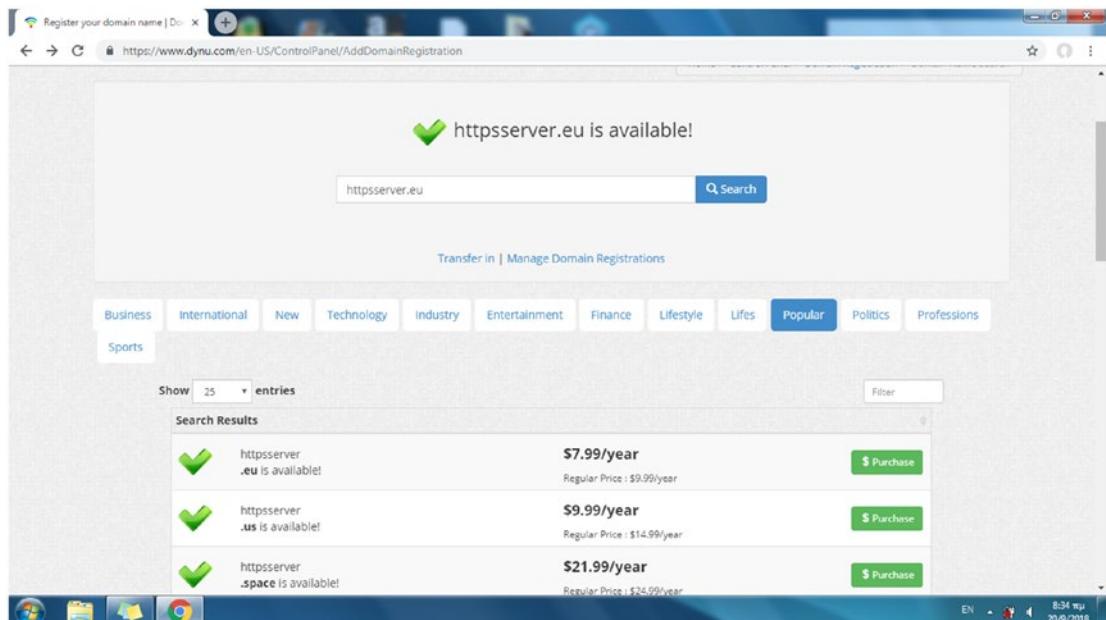


Figure 9-1. Searching the availability of a domain name

By clicking the Purchase button corresponding to the TLD of your choice, e.g., `.eu`, you can start the registration process at [Dynu.com](https://www.dynu.com), which includes the payment process for obtaining the domain name. When you're finished, the domain is included in the list of your domain names as a link, as displayed in Figure 9-2. You can further manage your domain name by clicking this link.

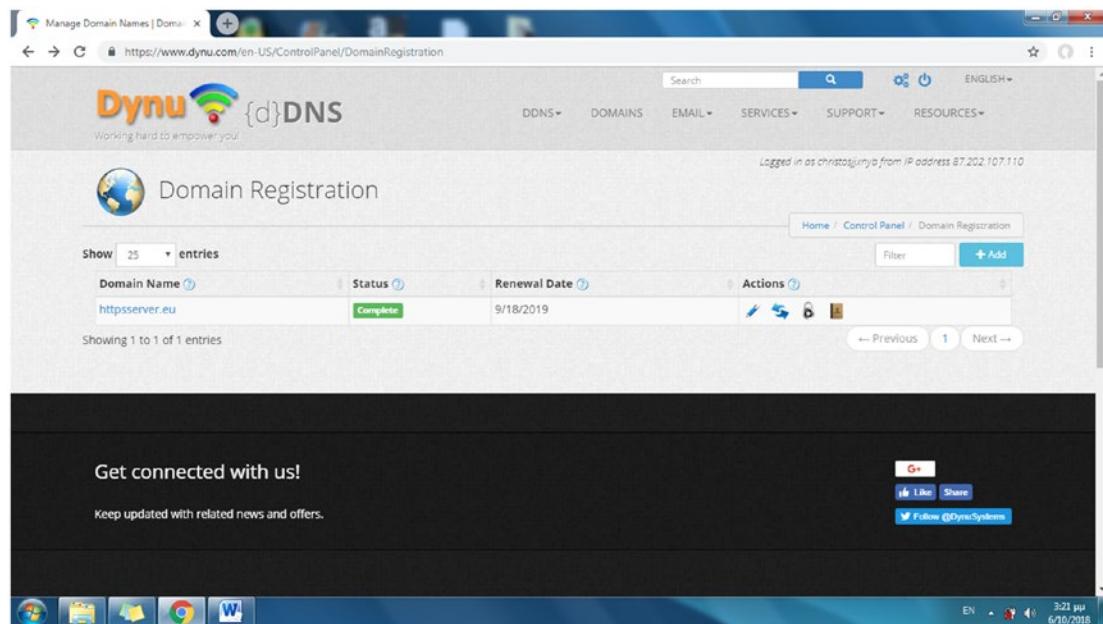


Figure 9-2. The newly registered domain name is added to the list of your domain names

The next step for running a secure site is to acquire an SSL certificate from a certificate authority for your new domain name.

Obtaining a CA SSL Certificate for Your Domain Name

To go to the SSL certificates web page at [Dynu.com](https://www.dynu.com), click the gears icon and then click the SSL Certificates link. Make a choice from the available SSL certificate offers and proceed with the payment. Figure 9-3 shows an SSL certificate from Comodo being used. The status of the SSL certificate is Awaiting CSR.

CHAPTER 9 RUNNING YOUR SITE WITH A CERTIFICATE FROM A CERTIFICATE AUTHORITY

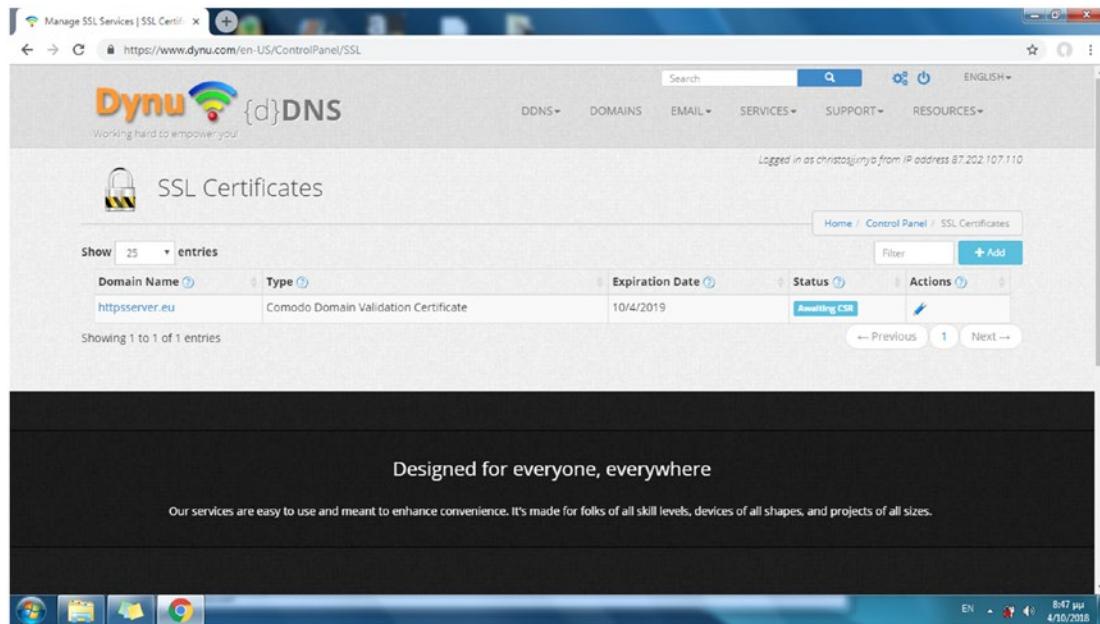


Figure 9-3. By purchasing an SSL certificate, your domain name status turns to Awaiting CSR

A certificate-signing request (CSR) is an encoded file with your application form for the CA. It is usually generated on the web server and includes information such as your location, your e-mail, your organization, etc., and basically contains your public key that will be included in the SSL certificate. You can generate your key pair, consisting of the CSR file (that includes your public key) and also your private key, by using the following `openssl req` command at the Linux terminal:

```
$ sudo openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out server.csr
```

In this command, the following options are used:

- `new`: Generates a new certificate request
- `newkey rsa:2048`: Generates a new private key, an RSA key that is 2,048 bits in size, as its argument indicates
- `nodes`: Specifies that the private key created will not be encrypted

- `keyout server.key`: Specifies `server.key` as the file name to write the newly created private key to
- `out server.csr`: Specifies `server.csr` as the file name of the CSR file

OpenSSL asks about the following information:

- Country name (two-letter code)
- State or province name (full name)
- Locality name (e.g., city)
- Organization name (e.g., company)
- Organizational unit name (e.g., section)
- Common name (e.g., server FQDN or your name)
- E-mail address
- A challenge password
- An optional company name

Most of the previous information is self-explanatory. In the Common Name field, enter your domain name, e.g., **httpsserver.eu**. If you don't represent any company or organization, just enter your name because many CAs require that all the fields be completed. When you fill in all the fields, the `openssl` command terminates, and the two files `server.key` and `server.csr` are created in your working directory. Use the following command to view them:

```
$ ls -l
```

The command's output is as follows:

```
total 8
-rw-r--r-- 1 root root 1058 Feb 27 19:49 server.csr
-rw----- 1 root root 1704 Feb 27 19:49 server.key
```

Change the `server.csr` file user rights so that only root has access to this file. File `server.key` has the user rights already set.

```
$ sudo chmod 0600 server.csr
```

Use the `ls -l` another time:

```
$ ls -l
```

CHAPTER 9 RUNNING YOUR SITE WITH A CERTIFICATE FROM A CERTIFICATE AUTHORITY

The command's output is as follows:

```
total 8
-rw----- 1 root root 1058 Feb 27 19:49 server.csr
-rw----- 1 root root 1704 Feb 27 19:49 server.key
```

Of the two files generated, your private key (`server.key`) is the one that you keep secret. The other one (`server.csr`) is the CSR file that includes your public key, and it can be available to anyone. This is the file you have to submit to the CA. Open the CSR file with a text editor, copy the contents, and switch back to your browser. In the Actions column of the [Dynu.com](#) web page accessed previously, click the Manage button (the blue pencil) to submit your CSR file. Paste your CSR text in the Certificate Signing Request (CSR) window, as viewed in Figure 9-4, and click the Save button.

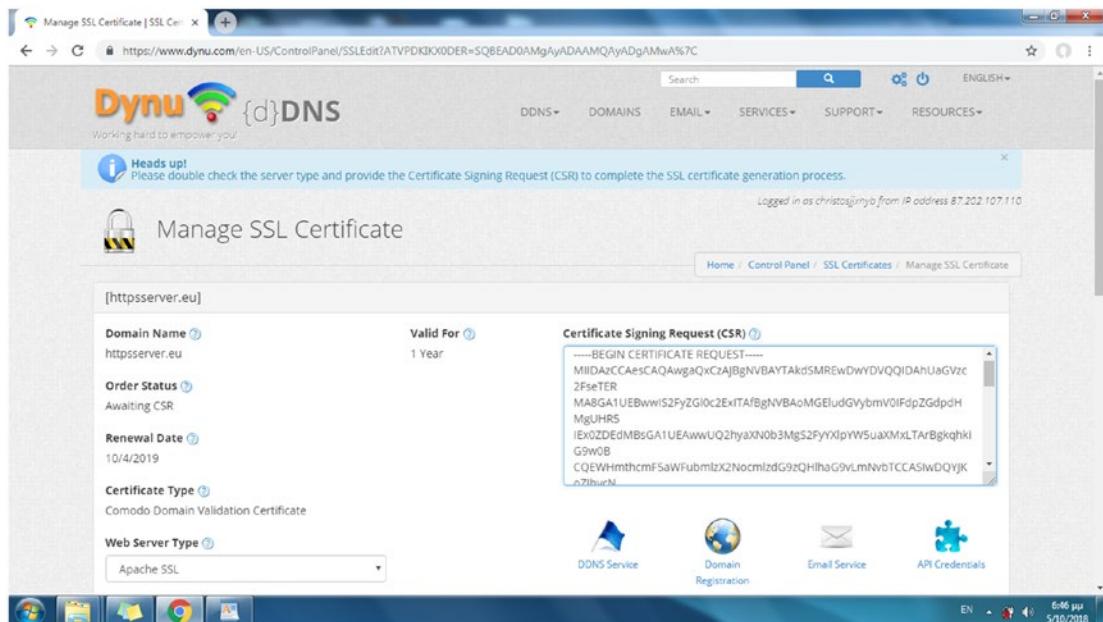


Figure 9-4. Providing the CSR file contents to the CA

In the Web Server Type list, select Apache SSL. Also, in the Approved Email list, select `admin@httpsserver.eu`. Unless you run a mail server, you probably do not have an e-mail address that uses your domain name. Such an e-mail address is, however, required by CAs for the verification process. SSL certification providers like [Dynu.com](#) create a temporary e-mail account for you to receive e-mail from the CA,

via the [Dynu.com](#) webmail interface. The username and the password for the [Dynu.com](#) webmail site are provided by [Dynu.com](#) so that you can reply to the CA's e-mail and thus complete the SSL certificate registration process.

You'll receive an e-mail from [Dynu.com](#) that includes your certificate (e.g., `httpsserver.eu_2018.cer`) and a bundle file (e.g., `httpsserver-eu.ca-bundle`) with all the intermediate certificates. For security reasons, the CA uses a *chain of trust*, where one intermediate certificate signs the next. At one end of the chain is the root certificate, which is the CA identity, and at the other end is your domain certificate, which is the certificate you purchase from a CA.

Hint! Sometimes the SSL certificate provider sends the root and intermediate certificates instead of a bundle file. You have to copy and paste the contents of the files in the order suggested by the SSL certificate provider into the `.ca-bundle` file. You can skip this and ask your SSL provider to send you a ready-to-use bundle file instead.

Since the SSL certificate is already provided, the status of the SSL Certificates web page of [Dynu.com](#) changes to Complete, as displayed in Figure 9-5.

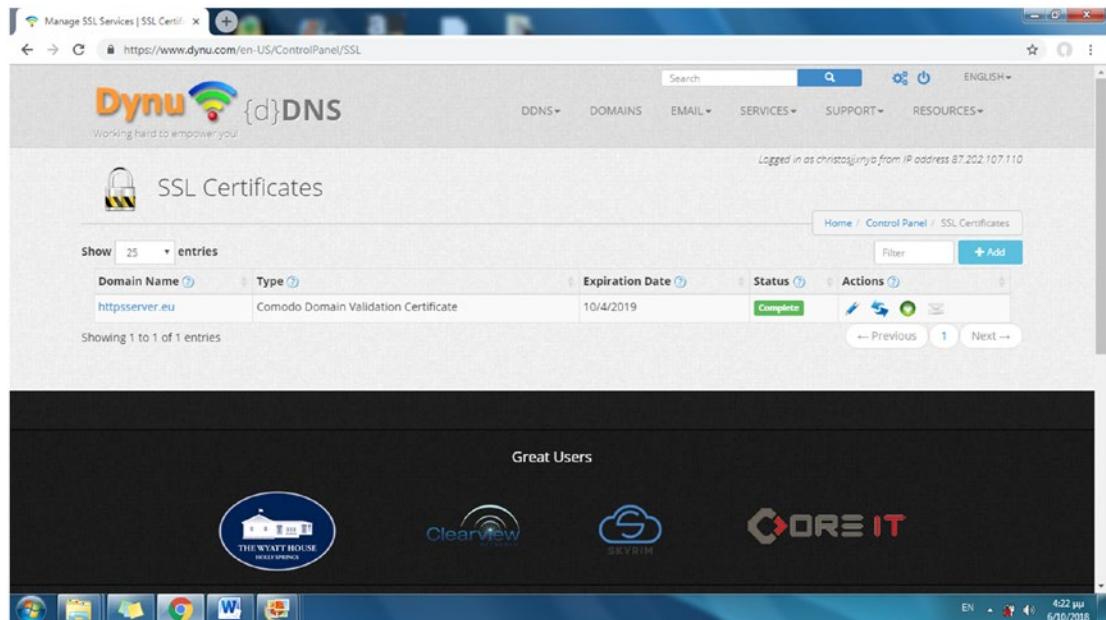


Figure 9-5. With the SSL certificate granted, the status of the domain name on the SSL Certificates web page becomes Complete

In the following section, you'll install the SSL certificate on the Apache and Lighttpd web servers so you can have a secure communication for the client login project.

Configuring SSL on the Web Servers

For both Apache and Lighttpd, three files are required for using HTTPS.

- The server's private key
- The SSL certificate for the domain name
- The bundle file that includes the root certificate and the intermediate certificates

Download the files sent with the e-mail from the SSL certificate provider and rename the certificate file using the .crt file extension.

```
$ cp httpsserver.eu_2018.cer ssl.crt
```

Next the installation process for Apache and then for the Lighttpd web servers are discussed.

Installing the CA Certificate on the Apache Web Server

Create a new directory named ca in /etc/apache2 and copy the certificates and your private key into it. In the following cp commands, the files are assumed to be copied from the home directory. Use sudo su to avoid including sudo in the rest of the commands.

```
$ sudo su  
# mkdir /etc/apache2/ca  
# cp ~/server.key /etc/apache2/ca  
# cp ~/ssl.crt /etc/apache2/ca  
  
# cp ~/httpsserver_eu.ca-bundle /etc/apache2/ca
```

Edit the default Apache SSL configuration file.

```
# gedit default-ssl.conf
```

In the configuration file, make sure the following directives are added:

```
ServerName httpsserver.eu
SSLCertificateFile      /etc/apache2/ca/ssl.crt
SSLCertificateKeyFile   /etc/apache2/ca/server.key
SSLCertificateChainFile /etc/apache2/ca/httpsserver_eu.ca-bundle
```

The whole file should look like the following:

```
<IfModule mod_ssl.c>
  <VirtualHost _default_:443>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html/login

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf

    # SSL Engine Switch:
    # Enable/Disable SSL for this virtual host.
    SSLEngine on

    # A self-signed (snakeoil) certificate can be created by installing
    # the ssl-cert package. See
    # /usr/share/doc/apache2/README.Debian.gz for more info.
```

```
# If both key and certificate are stored in the same file, only the
# SSLCertificateFile directive is needed.
#SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
#SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
#####
ServerName httpsserver.eu
SSLCertificateFile /etc/apache2/ca/ssl.crt
SSLCertificateKeyFile /etc/apache2/ca/server.key
SSLCertificateChainFile /etc/apache2/ca/httpsserver_eu.ca-bundle
#####
# Server Certificate Chain:
# Point SSLCertificateChainFile at a file containing the
# concatenation of PEM encoded CA certificates which form the
# certificate chain for the server certificate. Alternatively
# the referenced file can be the same as SSLCertificateFile
# when the CA certificates are directly appended to the server
# certificate for convinience.
#SSLCertificateChainFile /etc/apache2/ssl.crt/server-ca.crt

# Certificate Authority (CA):
# Set the CA certificate verification path where to find CA
# certificates for client authentication or alternatively one
# huge file containing all of them (file must be PEM encoded)
# Note: Inside SSLCACertificatePath you need hash symlinks
#       to point to the certificate files. Use the provided
#       Makefile to update the hash symlinks after changes.
#SSLCACertificatePath /etc/ssl/certs/
#SSLCACertificateFile /etc/apache2/ssl.crt/ca-bundle.crt

# Certificate Revocation Lists (CRL):
# Set the CA revocation path where to find CA CRLs for client
# authentication or alternatively one huge file containing all
# of them (file must be PEM encoded)
# Note: Inside SSLCARevocationPath you need hash symlinks
#       to point to the certificate files. Use the provided
#       Makefile to update the hash symlinks after changes.
```

```
#SSLCARevocationPath /etc/apache2/ssl.crl/
#SSLCARevocationFile /etc/apache2/ssl.crl/ca-bundle.crl

# Client Authentication (Type):
# Client certificate verification type and depth. Types are
# none, optional, require and optional_no_ca. Depth is a
# number which specifies how deeply to verify the certificate
# issuer chain before deciding the certificate is not valid.
#SSLVerifyClient require
#SSLVerifyDepth 10

# SSL Engine Options:
# Set various options for the SSL engine.
# o FakeBasicAuth:
#     Translate the client X.509 into a Basic Authorisation. This
#     means that
#     the standard Auth/DBMAuth methods can be used for access
#     control. The
#     user name is the `one line' version of the client's X.509
#     certificate.
#     Note that no password is obtained from the user. Every entry
#     in the user
#     file needs this password: `xxj31ZMTZzkVA'.
# o ExportCertData:
#     This exports two additional environment variables: SSL_
#     CLIENT_CERT and
#     SSL_SERVER_CERT. These contain the PEM-encoded certificates
#     of the server (always existing) and the client (only existing
#     when client
#     authentication is used). This can be used to import the
#     certificates
#     into CGI scripts.
# o StdEnvVars:
#     This exports the standard SSL/TLS related `SSL_*' environment
#     variables.
```

```
#      Per default this exportation is switched off for performance
#      reasons, because the extraction step is an expensive
#      operation and is usually useless for serving static content.
#      So one usually enables the exportation for CGI and SSI
#      requests only.
# o OptRenegotiate:
#      This enables optimized SSL connection renegotiation handling
#      when SSL
#      directives are used in per-directory context.
#SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
<FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
</FilesMatch>
<Directory /usr/lib/cgi-bin>
    SSLOptions +StdEnvVars
</Directory>

#  SSL Protocol Adjustments:
#  The safe and default but still SSL/TLS standard compliant shutdown
#  approach is that mod_ssl sends the close notify alert but
#  doesn't wait for
#  the close notify alert from client. When you need a different
#  shutdown
#  approach you can use one of the following variables:
# o ssl-unclean-shutdown:
#      This forces an unclean shutdown when the connection is
#      closed, i.e. no SSL close notify alert is send or allowed to
#      received. This violates the SSL/TLS standard but is needed
#      for some brain-dead browsers. Use this when you receive I/O
#      errors because of the standard approach where
#      mod_ssl sends the close notify alert.
# o ssl-accurate-shutdown:
#      This forces an accurate shutdown when the connection is
#      closed, i.e. a
#      SSL close notify alert is send and mod_ssl waits for the
#      close notify
```

```

# alert of the client. This is 100% SSL/TLS standard compliant,
# but in
# practice often causes hanging connections with brain-dead
# browsers. Use
# this only for browsers where you know that their SSL
# implementation
# works correctly.
# Notice: Most problems of broken clients are also related to the HTTP
# keep-alive facility, so you usually additionally want to
# disable
# keep-alive for those clients, too. Use variable "nokeepalive"
# for this.
# Similarly, one has to force some clients to use HTTP/1.0 to
# workaround
# their broken HTTP/1.1 implementation. Use variables
# "downgrade-1.0" and
# "force-response-1.0" for this.
# BrowserMatch "MSIE [2-6]" \
#     nokeepalive ssl-unclean-shutdown \
#     downgrade-1.0 force-response-1.0

</VirtualHost>
</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

```

For the Apache configuration, it is assumed that the `SSLEngine` directive in the configuration file is set to `on` and also that the `ssl` module is enabled, as described in the previous chapter. If it is not enabled, enable it now using the following:

```

$ sudo a2enmod ssl
$ service apache2 force-reload

```

The `DocumentRoot` directive was also set to `/var/www/html/login`, which will be the root directory used next in this chapter's project.

To enable redirection to the HTTPS protocol (`https://`) when a user connects using the HTTP protocol (`http://`), edit the corresponding configuration file, e.g., `000-default.conf`, and add the following directive:

```
Redirect / https://httpsserver.eu
```

The file should look like the following:

```
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html/login
        Redirect / https://httpsserver.eu
    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

With the redirection directive, the client request shown here:

```
http://httpsserver.eu
```

is redirected by the server to the following request:

```
https://httpsserver.eu
```

When a URL is entered with the `http://` protocol (or no protocol at all), the HTTP status code returned directs the client to a different URL (the connection in this case) with the `https://` protocol.

Reload Apache to validate the new configuration.

```
# service apache2 force-reload
```

Installing the CA Certificate on the Lighttpd Web Server

Use the certificates and the server's private key to enable SSL on the Lighttpd web server. If you didn't previously, enter `sudo su` to become the root user and avoid prepending `sudo` in the commands that follow.

```
$ sudo su
```

Stop Apache from running so you can work next with Lighttpd.

```
# service apache2 stop
```

Next, start the Lighttpd web server.

```
# service lighttpd start
```

Create a new directory for storing the private key and the certificates.

```
# mkdir /etc/lighttpd/ca
```

Copy the files of the `/etc/apache2/ca` directory to the new `ca` directory.

```
# cp /etc/apache2/ca /etc/lighttpd/ca
```

Create the PEM file with the private key and the domain certificate.

```
# cd /etc/lighttpd/ca  
# cat server.key ssl.crt > server.pem
```

Edit the Lighttpd configuration file.

```
# gedit /etc/lighttpd/lighttpd.conf
```

Enter the following lines and save the file:

```
$SERVER["socket"] == ":443" {
    server.document-root      = "/var/www/html/login"
    ssl.engine                = "enable"
    server.username           = "www-data"
    server.groupname          = "www-data"
    ssl.pemfile               = "/etc/lighttpd/ca/server.pem"
    ssl.ca-file               = "/etc/lighttpd/ca/httpsserver_eu.ca-bundle"
    server.name               = "httpsserver.eu"
}
```

Also, to redirect the HTTP requests to HTTPS, include the following lines in the conditional configuration for port 80:

```
$HTTP["host"] =~ "(.*)" {
    url.redirect = ( "^/(.*)" => "https://%1/$1" )
}
```

Set the document root from `/var/www/html/` to `/var/www/html/login` in the conditional configurations for ports 80 and 443, which will be the one used next in the project.

The file `lighttpd.conf` should look like the following:

```
server.modules = (
    "mod_access",
    "mod_alias",
    "mod_accesslog",
    "mod_compress",
    "mod_redirect",
)

server.document-root      = "/var/www/html"
server.upload-dirs         = ( "/var/cache/lighttpd/uploads" )
server.errorlog            = "/var/log/lighttpd/error.log"
```

```

server.pid-file          = "/var/run/lighttpd.pid"
server.username          = "www-data"
server.groupname         = "www-data"
#server.port              = 8080
#server.port              = 443
#server.bind              = "webtoolsonline.servehttp.com"
server.errorfile-prefix  = "/srv/www/errors/status-"

dir-listing.activate     = "disable"

accesslog.filename        = "/var/log/lighttpd/access.log"
#accesslog.format          = "%V %h %l %u %t \"%r\" %>s %b \"%{Referer}\n"
                           i\" \"%{User-Agent}i\""

index-file.names          = ( "index.php", "index.html", "index.lighttpd."
                               html" )
url.access-deny           = ( "~", ".inc" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

compress.cache-dir         = "/var/cache/lighttpd/compress/"
compress.filetype          = ( "application/javascript", "text/css",
                               "text/html", "text/plain" )

$SERVER["socket"] == ":443" {
    server.document-root      = "/var/www/html/login"
    ssl.engine                = "enable"
    server.username            = "www-data"
    server.groupname           = "www-data"

#####
    ssl.pemfile = "/etc/lighttpd/ca/server.pem"
    ssl.ca-file = "/etc/lighttpd/ca/httpsserver_eu.ca-bundle"
    server.name = "httpsserver.eu"
#####

}

```

```
$SERVER["socket"] == ":80" {
    server.document-root      = "/var/www/html/login"
    server.upload-dirs        = ( "/var/cache/lighttpd/uploads" )
    server.errorlog           = "/var/log/lighttpd/error.log"
    server.pid-file           = "/var/run/lighttpd.pid"
    server.username            = "www-data"
    server.groupname           = "www-data"

#####
$HTTP["host"] =~ "(.*)" {
    url.redirect = ( "^/(.*)" => "https://%1/$1" )
}
#####

}

# default listening port for IPv6 falls back to the IPv4 port
## Use ipv6 if available
#include_shell "/usr/share/lighttpd/use-ipv6.pl " + server.port
#include_shell "/usr/share/lighttpd/create-mime.assign.pl"
#include_shell "/usr/share/lighttpd/include-conf-enabled.pl"

#include "vhost.conf"

Enable the new configuration by reloading Lighttpd.

# service lighttpd force-reload
```

Testing the SSL CA Certificate

Figure 9-6 displays the home page of `httpsserver.eu`. This is the home page of the project that you'll create next in this chapter.

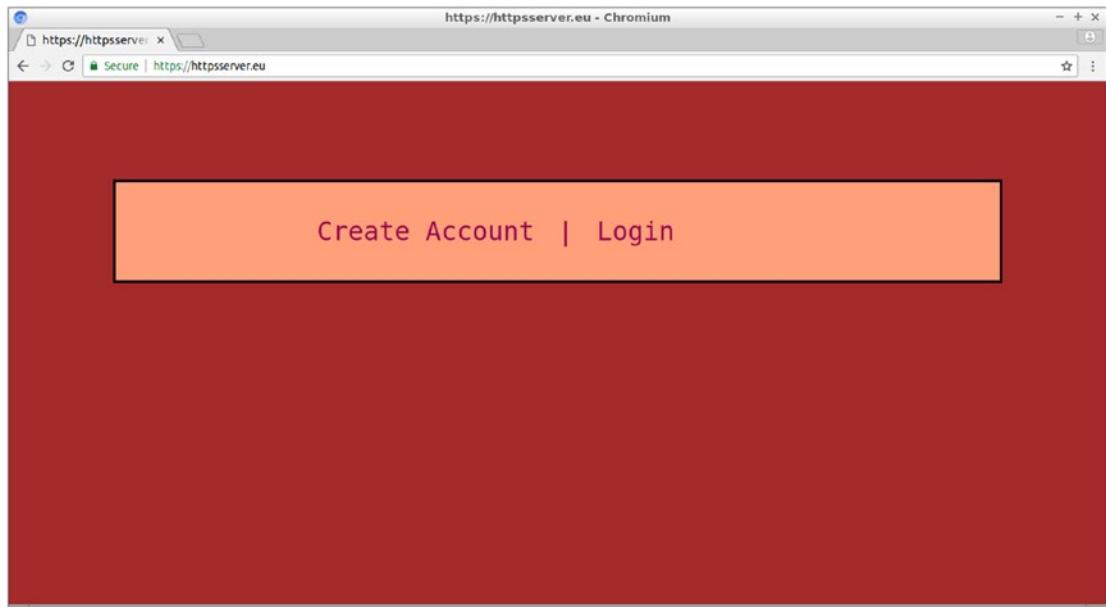


Figure 9-6. The home web page of the project's site

The browser warnings about entering an insecure site do not appear anymore. A padlock icon, indicating a secure connection, appears in the address bar of the browser, and the `https://` protocol is not struck through as previously, with the self-signed certificate. Click the padlock icon to find out some details about the current secure connection. For the Chromium browser, the menu options appear as displayed in Figure 9-7.

CHAPTER 9 RUNNING YOUR SITE WITH A CERTIFICATE FROM A CERTIFICATE AUTHORITY

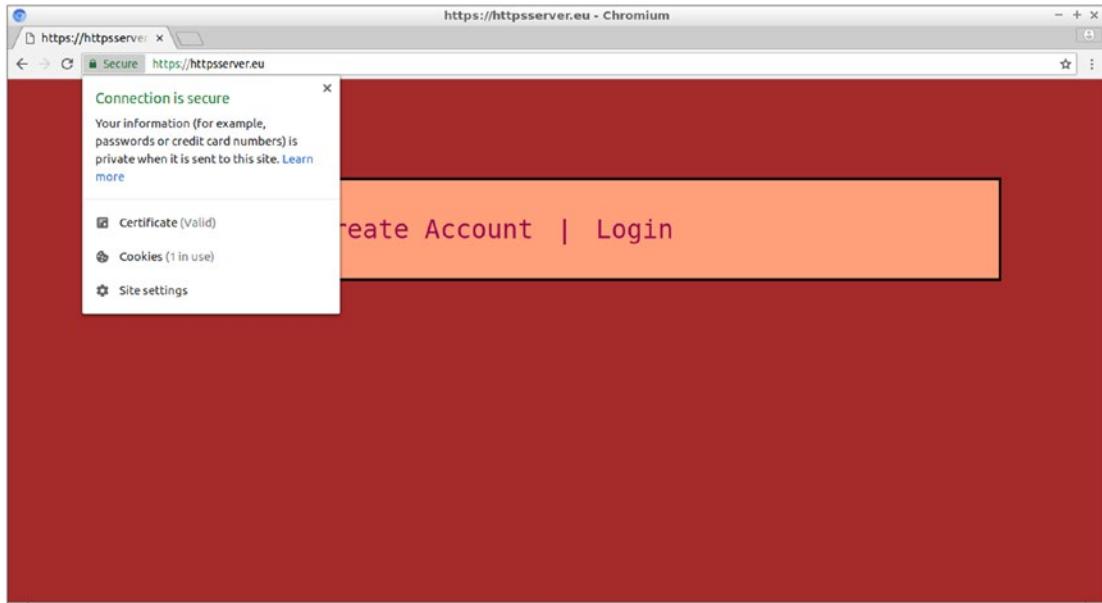


Figure 9-7. Clicking the green padlock allows you to get information about the secure connection

Click the Certificate (Valid) option, as displayed in Figure 9-8, to open the certificate viewer and find details about the current certificate.

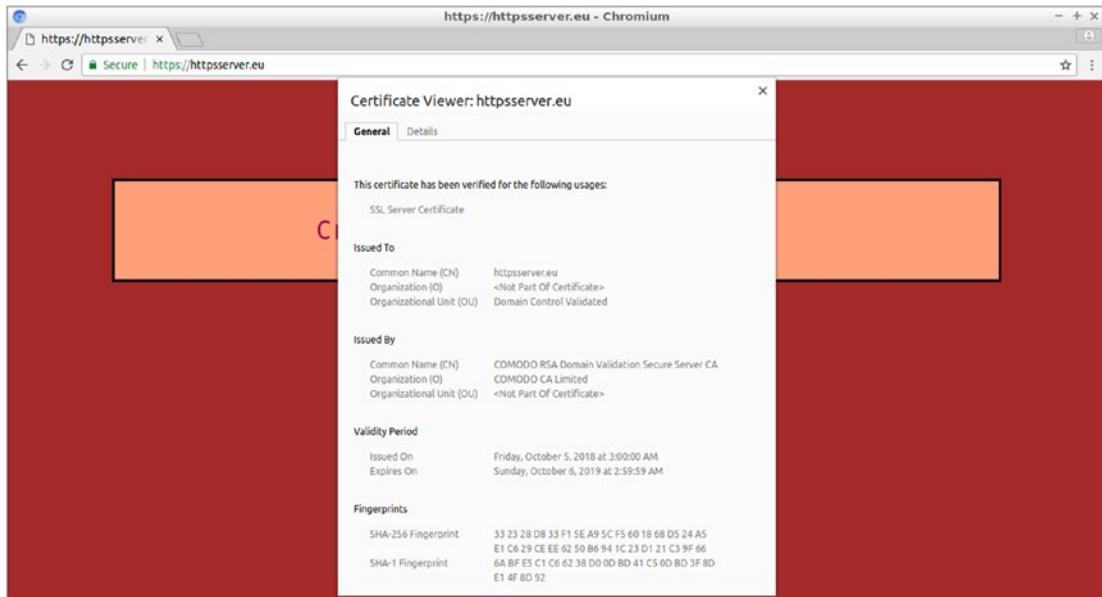


Figure 9-8. Using the Certificate Viewer in Chromium

Next use an online tool that tests and analyzes the web server certificates, for instance SSL Checker (www.sslshopper.com). In the Server Hostname textbox, enter your domain name, e.g., `httpsserver.eu`, and click the Check SSL button. Figure 9-9 displays the results, indicating no problems were diagnosed.

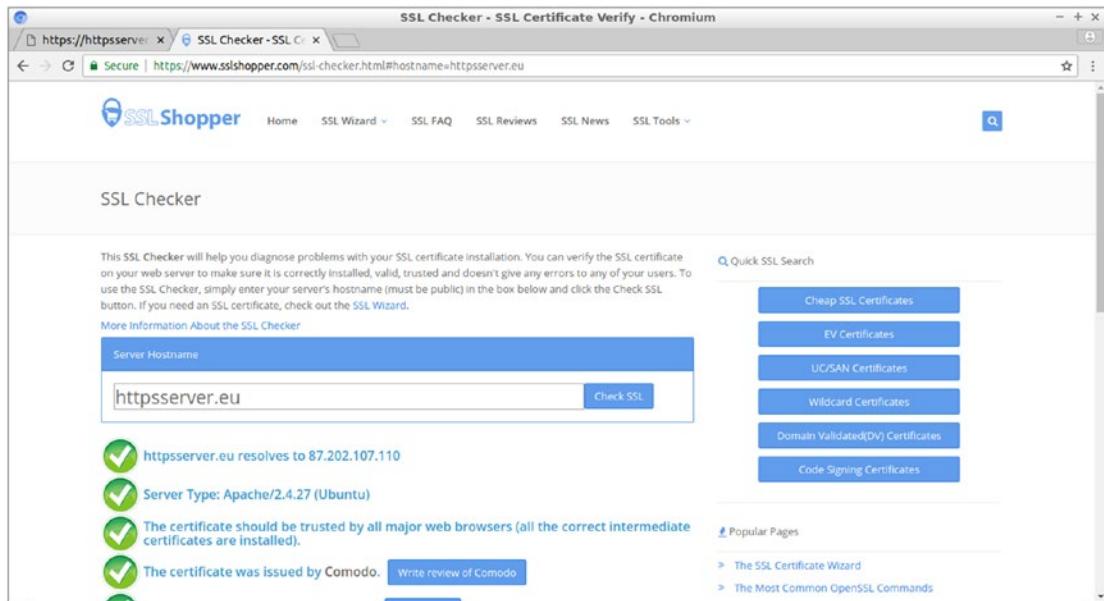


Figure 9-9. Using an online tool to check your SSL certificate

Scroll down to view the chain of trust for your site, as displayed in Figure 9-10.

CHAPTER 9 RUNNING YOUR SITE WITH A CERTIFICATE FROM A CERTIFICATE AUTHORITY

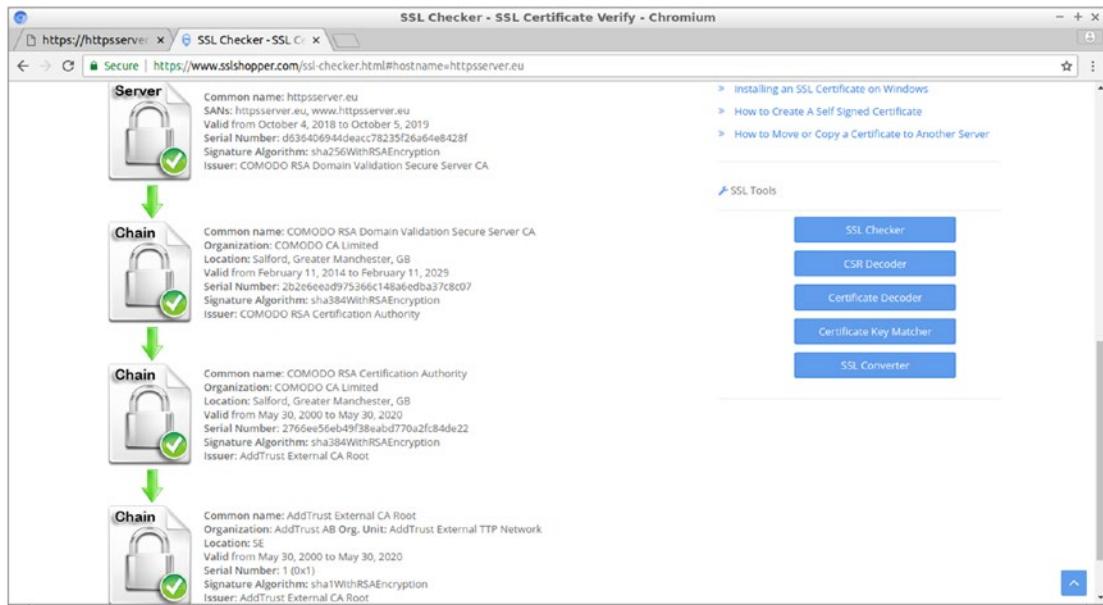


Figure 9-10. Displaying the chain of trust for your SSL certificate

The chain from your domain name certificate leads, in this example, through two intermediate certificates to the CA root certificate.

You will test your SSL certificate next with the project in the following section.

Project: Securely Logging In to a Site

By enhancing your site with an SSL certificate, obtained from a CA, you can now create a site that securely allows the user to create an account and log in to view some personal information. You do this by implementing a MySQL database that stores all the user accounts. The user can remain connected while browsing the site's pages until finally logging out. For simplicity, this example will just have one page, but you could expand it to more web pages. By implementing PHP sessions, you'll allow PHP session variables to be shared between the PHP source code of the site's web pages. In this project, the `session_user` variable set by the username when the user logs in successfully is shared among the web pages. In web pages like `profile.php`, which appears when the user

logs in, the `session_user` value is the ticket to querying a database and displaying user-specific information. This can be extended to other web pages and thus allow the user to access more information while connected.

The project requires the following five PHP files:

- `index.php`, the home web page that allows the user to either create an account or connect to the site using this account
- `account.php`, the web page that creates the account
- `login.php`, the web page that logs the user in to the site
- `profile.php`, the web page the user is transferred to after a successful login to view user-specific information
- `logout.php`, the file that uses the PHP code for the logout process, which also transfers the user from `profile.php` to `login.php`

Designing the Project's Site

Figure 9-11 displays the four web pages and the way they are linked together to form the site.

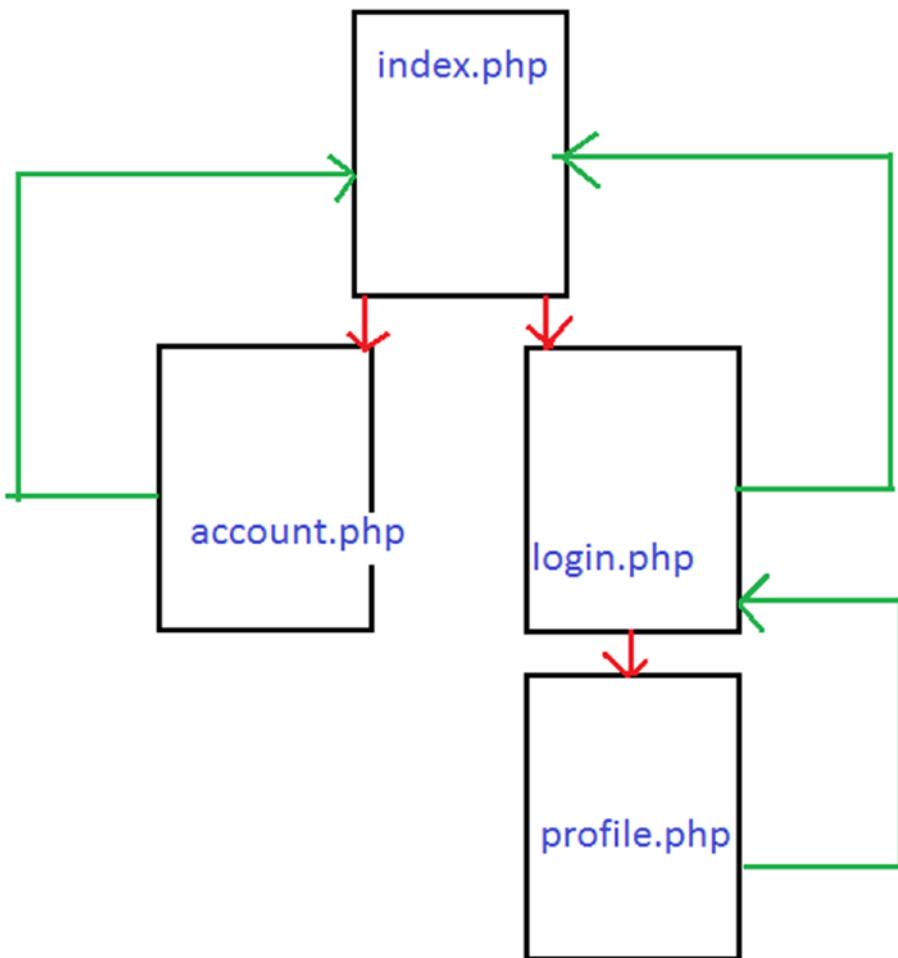


Figure 9-11. The design of the project's site

The login project site will be placed in a new directory, called `login`, in the document root. Create this directory with the following command:

```
$ sudo mkdir /var/www/html/login
```

You can use either web server you've been using in this book, but these instructions will use Apache. The directory `/var/www/html/login` is already set as the document root of the site for the Apache web server in the `default-ssl.conf` configuration file with the `DocumentRoot` directive.

```
DocumentRoot /var/www/html/login
```

If Apache is not running currently, enable it now using this:

```
$ sudo service lighttpd stop  
$ sudo service apache2 start
```

The Source Code for the Home Page of the Site

Next, create the directory index, `index.php`, so that the site can be accessed from the address bar of the browser simply as follows:

`https://httpsserver.eu`
or since redirection applies simply as:
`httpsserver.eu`

Create `index.php` by entering the following command at the Linux terminal:

```
$ sudo gedit /var/www/html/login/index.php
```

Enter the following code and save the file:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
body{  
background-color:brown;  
}  
.center {  
margin: auto;  
width: 80%;  
border: 3px solid black;  
padding: 10px;  
background-color:lightsalmon;  
}  
p{  
text-align:center;  
font-family: Courier New,Courier,Lucida Sans Typewriter,Lucida  
Typewriter,monospace;
```

```
font-size:32px;
color:#9F0251;
font-weight:italic;
}
span{
padding:10px;
}
a{
text-decoration:none;
}
a:link, a:visited {
    color: #9F0251;;
}
a:hover {
    color: black;
}
</style>
</head>

<body>
<br><br><br><br><br><br>
<div class="center">
<p>
<span>
<a href="account.php">Create Account</a>
</span>
|
<span>
<a href="login.php">Login &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</a>
</span>
</p>
</div>
</body>
</html>
```

As indicated in its source code, `index.php` is a simple web page that basically includes two links. Create Account leads to `account.php`, which is the PHP file that evaluates to the web page used for creating the user account, and Login leads to `login.php`, which is the PHP file that evaluates to the web page where the user logs in to the site (Figure 9-12).

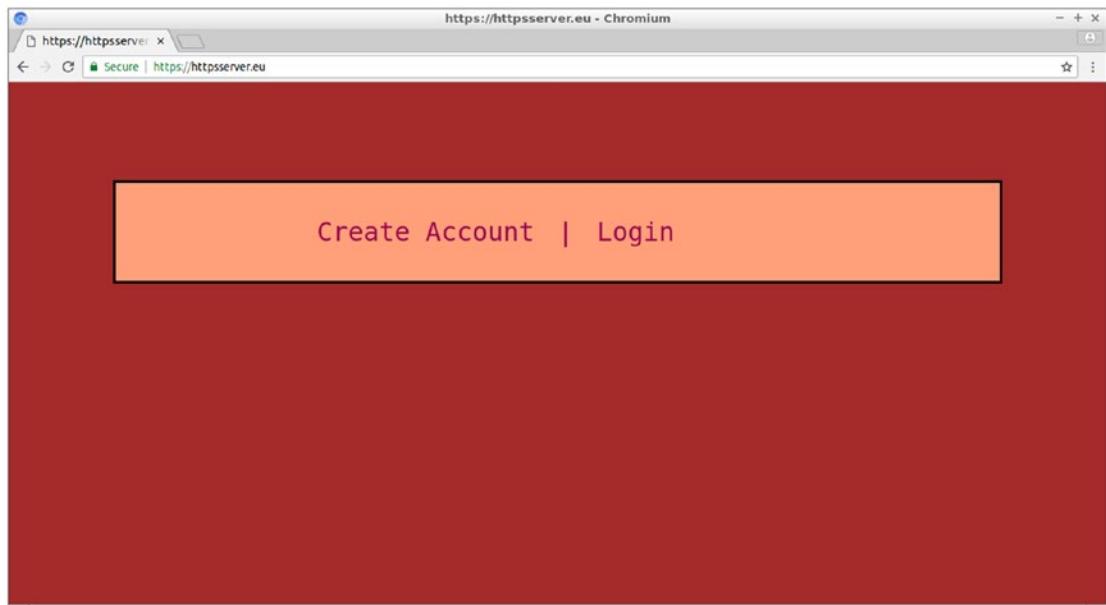


Figure 9-12. The home page includes two links for creating user accounts and for user login

Next, you will create two more web pages: `account.php` and `login.php`.

The Web Page for Creating the User Account

By following the Create Account link, the user goes from `index.php` to the `account.php` page, displayed in Figure 9-13, where the account for the specific web service is created. In this example, the service is about securely logging in to a site, where the user can view some details about the books loaned from a local library.

The user is required to enter the following details in the HTML form of `account.php` (which will be stored in the `user` table of the `login` database created in the following section with MySQL):

- First
- Last name
- E-mail address
- Username
- Password
- Password retyped

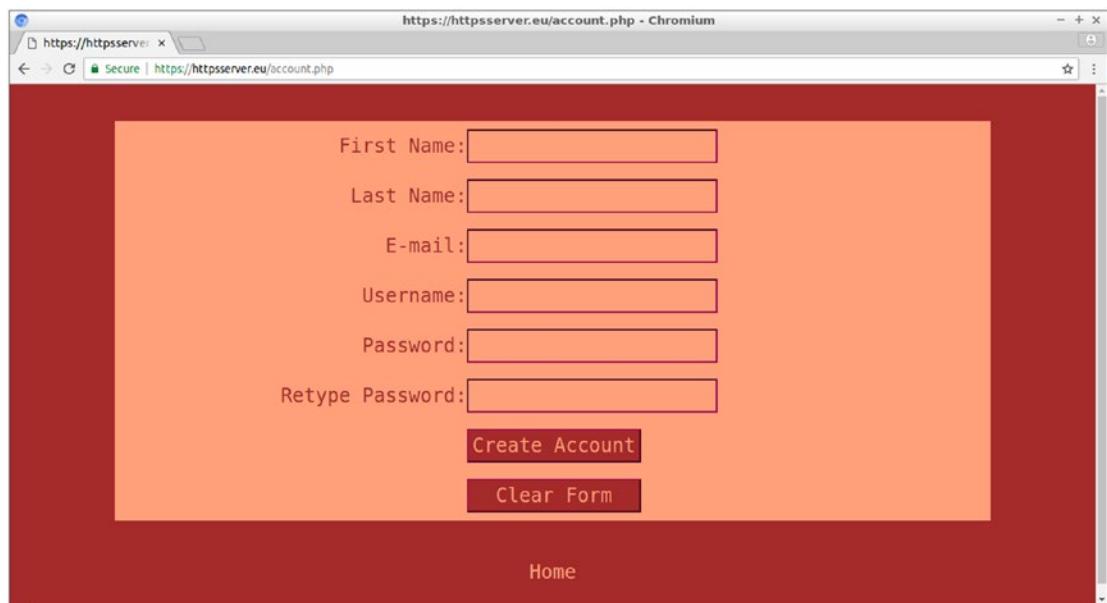


Figure 9-13. The account.php web page of the site

All form objects are of type text, except the e-mail, which is of type email, and the two password fields, of type password. The e-mail field requires an entry of the form X@Y.Z, and the password fields hide the characters by replacing them with bullets. The password has to be retyped to validate the first password entry. Two checks are done in the PHP form validation process: that all fields are nonempty and that the two password entries match. If those conditions are not met, the appropriate message appears at the top of the web page, as displayed in Figure 9-14.



Figure 9-14. Error messages are displayed when the form is incomplete or when the password entries are not identical

To create the file account.php, enter the following command at the Linux terminal:

```
$ sudo gedit /var/www/html/login/account.php
```

Enter the following source code and save the file:

```
<!DOCTYPE html>
<html>
<head>
<style>
body{
background-color:brown;
}
.center {
margin: auto;
width: 80%;
border: none;
padding: 10px;
background-color:lightsalmon;
}
```

```
.center2 {  
    margin: auto;  
    width: 80%;  
    border: none;  
    padding: 10px;  
    background-color:brown;  
}  
  
p{  
text-align:center;  
font-family: Courier New,Courier,Lucida Sans Typewriter,Lucida  
Typewriter,monospace;  
font-size:24px;  
color:white;  
font-weight:italic;  
}  
  
input{  
border-color:#9F0251;  
font-family: Courier New,Courier,Lucida Sans Typewriter,Lucida  
Typewriter,monospace;  
font-size:24px;  
color:black;  
padding:5px;  
background-color:lightsalmon;  
}  
  
input[type=submit],[type=reset]{  
background-color:brown;  
color:lightsalmon;  
padding:5px;  
}  
  
a{  
text-decoration:none;  
}  
  
a:link, a:visited {  
    color: lightsalmon;  
}
```

```
a:hover {  
    color: black;  
}  
label {  
    font-family: Courier New,Courier,Lucida Sans Typewriter,Lucida  
    Typewriter,monospace;  
    display: inline-block;  
    width: 40%;  
    text-align: right;  
    color: brown;  
    font-size: 24px;  
}  
</style>  
</head>  
<body>  
<?php  
  
$errmsg1 = "";  
$errmsg2 = "";  
$valid1 = 0;  
$valid2 = 0;  
  
if (isset($_POST['s1'])) {  
  
    $first = $_POST["first"];  
    $last = $_POST["last"];  
    $email = $_POST["email"];  
    $user = $_POST["user"];  
    $pass1 = $_POST["pass1"];  
    $pass2 = $_POST["pass2"];  
  
    if((empty($first)) || (empty($last)) || (empty($email)) || (empty($user))  
    || (empty($pass1)) || (empty($pass2))) {  
        $errmsg1 = '<p>Please complete all the fields. </p>';  
    } else {  
        $valid1 = 1;  
    }  
}
```

```
if((strcmp($pass1, $pass2))) {
$errormsg2 = '<p>Please enter the same password at the Password fields. </p>';
} else {
$valid2 = 1;
}
}

if (($valid1 == 1) && ($valid2 == 1)) {
create_entry($first, $last, $email, $user, $pass1);
}

function create_entry ($first, $last, $email, $user, $pass1) {

$servername = "localhost";
$username = "root";
$password = "";
$dbname = "login";

$mysqli = new mysqli($servername, $username, $password, $dbname);

if ($mysqli->connect_error) {
    die("Connection failed: " . $mysqli->connect_error);
}

$hashed_password = password_hash($pass1, PASSWORD_DEFAULT);
$sql = "INSERT INTO user (first, last, email, username, password) VALUES
('$first', '$last', '$email', '$user', '$hashed_password')";

if(mysqli_query($mysqli, $sql)){
    //echo "Records inserted successfully.";
}else{
    if(mysqli_errno($mysqli) == 1062) {
        echo "<p>Username already inserted (duplicate entry)</p>";
    }else{

```

```
echo "ERROR: Could not able to execute $sql. " . mysqli_
error($mysqli);
}

}

$mysqli->close();
}

?>

<?php
if(($errormsg1 != "") && isset($_POST['s1']))
echo $errormsg1;
if(($errormsg2 != "") && isset($_POST['s1']) && ($errormsg1 == ""))
echo $errormsg2;
?>

<br><br>
<form name="form1" method="post" action=<?php echo htmlspecialchars(
$_SERVER["PHP_SELF"]); ?>">
<div class="center">
<label for="first">First Name: </label><input type="text" name="first">
</div>
<div class="center">
<label for="last">Last Name: </label><input type="text" name="last">
</div>
<div class="center">
<label for="email">E-mail: </label><input type="email" name="email">
</div>
<div class="center">
<label for="user">Username: </label><input type="text" name="user">
</div>
<div class="center">
<label for="pass1">Password: </label><input type="password" name="pass1">
</div>
<div class="center">
<label for="pass2">Retype Password: </label><input type="password"
name="pass2">
</div>
```

```
<div class="center">
<label for="s1">&nbsp;</label><input type="submit" name="s1" value="Create
Account">
</div>
<div class="center">
<label for="r1">&nbsp;</label><input type="reset" name="r1"
value="&nbsp;&nbsp;Clear Form&nbsp;&nbsp;">
</div>
</form>
<div class="center2">
<p><a href="index.php">Home</a></p>
</p>
</body>
</html>
```

A large part of the code is for formatting the web page with CSS and creating the HTML form. PHP code blocks are also used in various positions in the body section of the HTML source code. In the HTML form, the `action` attribute, indicating the file that will receive the form's data, is set to `account.php`.

```
<form name="form1" method="post" action=<?php echo htmlspecialchars
($_SERVER["PHP_SELF"]); ?>">
```

When the form is completed by the user and the form data is submitted from `account.php` to itself, the PHP code included in the following `if` statement runs:

```
if (isset($_POST['s1'])) {
...
}
```

This `if` condition checks whether data is submitted by the `s1` named button of the form, with the `POST` method, and therefore whether the `$_POST['s1']` PHP global variable is set. Notice that `s1` is the name of the form's submit button. The HTTP request method `POST` is preferred over method `GET` to send SSL data, because although the `GET` data is also encrypted, it may be visible to server logs or the browser's history. Moreover, `GET` exposes the data to the URL, so the query string may be visible to anyone standing next to you. Also, `GET` limits the data sent by applying a maximum data length.

The next lines of the same PHP block assign the variables \$first, \$last, \$email, \$user, \$pass1, and \$pass2 to the corresponding values that the form's fields send. Two checks are performed with the form validation process. First you ensure that the values sent are not empty and, if all the form data is completed, that the two password fields' values match. If the checks succeed, the function `create_entry()` is called with the arguments \$first, \$last, \$email, \$user, and \$pass1.

`create_entry()` connects to the login MySQL database (which will be created in the following section) and inserts the row provided by the `create_entry()` arguments in the table `user`. These rows correspond to the table columns `first`, `last`, `email`, `username`, and `password`. Before inserting the values into the database, the PHP function `password_hash()` is called to encrypt the password value.

```
$hashed_password = password_hash($pass1, PASSWORD_DEFAULT);
$sql = "INSERT INTO user (first, last, email, username, password) VALUES
('{$first}', '{$last}', '{$email}', '{$user}', '{$hashed_password}')";
```

All passwords included in the database are therefore hashed to a value with a constant size (e.g., 60 characters when `password_hash()` is applied with the `PASSWORD_DEFAULT` argument). Hashed values are theoretically irreversible, which means you can't obtain the password from the password's hashed value. What you can do is hash the value of the user's entry and compare it with the hashed value already stored in the database.

The function `mysqli_query()` submits the query to the database. If this function returns false, the error with the number 1062 is checked, which indicates a duplicate entry. A duplicate entry for the username is not allowed by the database design, which is discussed in the following section.

In this case, an appropriate message is printed at the top of the window. Also, for other reasons of failure, the corresponding error number is printed.

Creating the Database Used for the Project

To test `account.php`, create the database used in the project. Connect with the `mysql` client to the MySQL server using the following at the command line:

```
$ sudo mysql -u root
```

At the mysql> prompt that appears, enter the following command to create the new database called login:

```
mysql> create database login;
```

The MySQL server responds with the following message:

```
Query OK, 1 row affected (0.00 sec)
```

Select the new database to work with.

```
mysql> use login;
```

The MySQL server responds with the following message:

```
Database changed
```

Create the table user with the column id as the primary key. The id field will increment each time automatically by one, starting from one, and therefore you don't have to provide it when you insert a new record into this table.

```
mysql> CREATE TABLE user(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY  
(id), username varchar(255) NOT NULL, UNIQUE KEY (username), password  
varchar(255) NOT NULL, email varchar(255) NOT NULL, first varchar(255) NOT  
NULL, last varchar(255) NOT NULL);
```

The MySQL server responds with a message similar to the following:

```
Query OK, 0 rows affected (0.43 sec)
```

At this point, you may want to view the page's structure. You can enter the following:

```
mysql> describe user;
```

The command's output is as follows:

```
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+
| id    | int(11)   | NO   | PRI | NULL    | auto_increment |
| username | varchar(255) | NO   | UNI | NULL    |                |
| password | varchar(255) | NO   |      | NULL    |                |
| email   | varchar(255) | NO   |      | NULL    |                |
| first   | varchar(255) | NO   |      | NULL    |                |
| last    | varchar(255) | NO   |      | NULL    |                |
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

In the CREATE TABLE SQL command, the UNIQUE KEY constraint was used for the username column in the user table (displayed with the UNI keyword in the table's structure) to ensure that there are no two identical usernames. Any attempts to insert a duplicate value is detected in the source code of the function `create_entry()` in `account.php`, and an appropriate error message is displayed at the top of the window.

Do not exit `mysql` yet since the MySQL client connection to the database is required for the next section.

Testing the PHP to MySQL Connection

In this section, you'll enter some records in the user table of the database from the `account.php` form. In your browser, display the directory index, `index.php`, by entering the following address in the address bar:

`https://httpsserver.eu/`

Then follow the Create Account link. You can also view the account page directly, without using the directory index first, by entering the following:

`https://httpsserver.eu/account.php`

Enter in the form's fields the details of some test users and submit them by clicking the Create Account button. For instance, you can create two users: Robert and Sophie. Figure 9-15 displays the values inserted in the form fields for user Robert.

CHAPTER 9 RUNNING YOUR SITE WITH A CERTIFICATE FROM A CERTIFICATE AUTHORITY

The screenshot shows a Chromium browser window with the URL <https://httpsserver.eu/account.php>. The page displays a form for creating a new account. The form fields and their values are:

- First Name: Robert
- Last Name: Walker
- E-mail: rob.walker@gmail.com
- Username: robert
- Password: (redacted)
- Retype Password: (redacted)

Below the form are two buttons: "Create Account" and "Clear Form". At the bottom of the page, there is a "Home" link.

Figure 9-15. The values entered in the account.php form for user Robert

Although generally not recommended, to compare the output of hashing two different-sized passwords, use a simple password for Robert, such as 123, and then a stronger one for Sophie, such as supersophie3mi8#m&&. Figure 9-16 shows the values entered in the form for user Sophie.

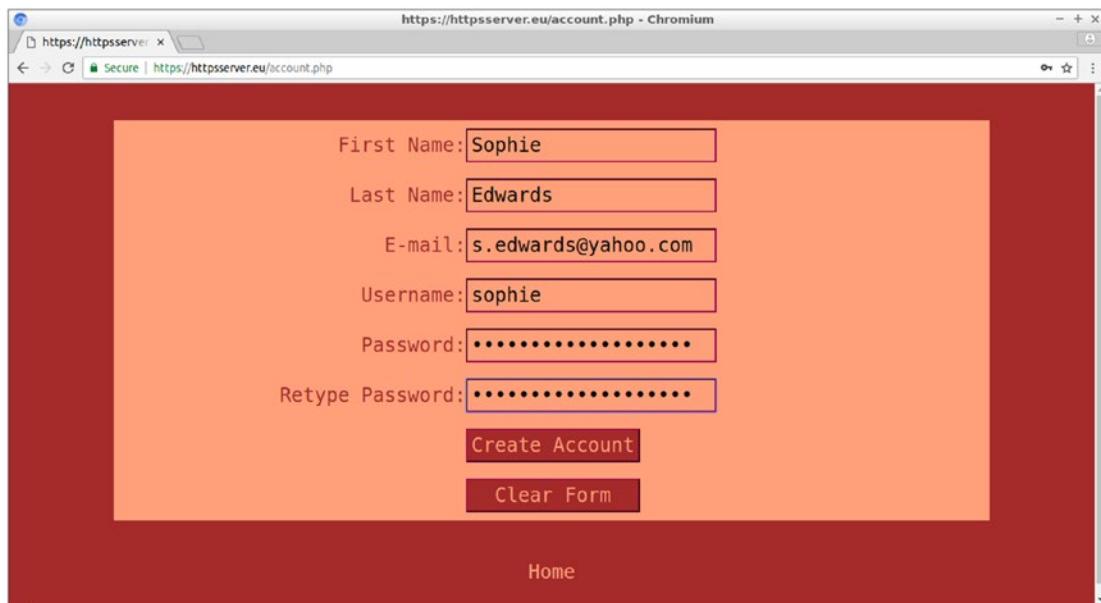


Figure 9-16. The values entered in the account.php form for user Sophie

Switch to the terminal and connect to the MySQL client. Use the following command to view the new contents of the table user:

```
mysql> select * from user;
```

The command's output is as follows:

```
+-----+-----+-----+-----+-----+
| id | username | password           | email          | first | last  |
+-----+-----+-----+-----+-----+
| 1  | robert   | $2y$10$UWc3eeIjxFlbF
                      j75muUiF.wwel2VSpk7b7
                      ytsApi5rBGQtGLLCLda | rob.walker@gmail.com | Robert | Walker |
| 2  | sophie   | $2y$10$nNTcBgLkzpSbk
                      CZTcjmdVetzk2/KYs0q2o
                      YyZpQTJI05hghzkaJwS | s.edwards@yahoo.com | Sophie | Edwards |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

As you'll notice, regardless of their initial size, both passwords after hashing have a constant length of 60 characters. For instance, for Robert, the password 123 after the hashing process has the following value:

```
$2y$10$UWc3eeIjxF1bFj75muUiF.wweL2VSpk7b7ytsApi5rBGQtGLLCLda
```

where:

- \$2y\$ is the algorithm used, which is the bcrypt algorithm for the PASSWORD_DEFAULT argument of `password_hash()`.
- 10\$ is the algorithm cost.
- UWc3eeIjxF1bFj75muUiF. is the salt.
- wweL2VSpk7b7ytsApi5rBGQtGLLCLda is the hashed password.

Password hashing is a nice solution for maintaining password security even when the database is compromised. The one-way hashing applied by `password_hash()` leads to a result that cannot be reversed from the original password. Since the hashed password, and not the original one, is stored in the database, to verify the password of the user who logs in to the site, the entered password has to be hashed and then compared with the stored password.

The hashing procedure is enforced by using a salt. Applying a common hash algorithm to a password always leads to the same result for the given password. This might be proved insecure, since an attacker might try a number of passwords, hashed with the same algorithm until he succeeds. A collection of hashed passwords used for the attack is called a *rainbow table*. By attaching a salt to the password, which is a random string either generated by the hashing method or provided by the user, the hashing outcome will be different even when the same password is hashed twice. When comparing the stored password with the password entered by the visitor of the site, the latter is hashed with the salt stored in the hashed password of the database.

Before testing the user login functionality of the site, you need to create a second table for the login database with details about the books loaned by the users. Then on login, the personal information will be available for each user at the web page that the login connection leads to.

At the Linux terminal, with `mysql` open, enter the following:

```
mysql> create table book_loan (book_id char(13), primary key(book_id),  
loan_date date, user_id int, foreign key(user_id) references user(id));
```

The MySQL server responds with a similar output:

```
Query OK, 0 rows affected (0.47 sec)
```

Enter a number of insert operations on the database like the following:

```
mysql> insert into book_loan(book_id, loan_date, user_id)
values(6183443458964, NOW(), 1);
mysql> insert into book_loan(book_id, loan_date, user_id)
values(3487368817469, NOW(), 1);
```

In the previous commands, the MySQL function `NOW()` returns the current date and time as a `YYYY-MM-DD` string.

View next the contents of the table `book_loan` by using the following:

```
mysql> select * from book_loan;
```

The MySQL server outputs the following table:

book_id	loan_date	user_id
3487368817469	2018-10-04	1
6183443458964	2018-10-04	1

2 rows in set (0.00 sec)

Therefore, the user with an id value of 1 (user robert) is billed with two books so far. In the next section, this information will be available to Robert, and the information that no books are loaned will be also available to Sophie.

The Source Code of the Login Web Page

Figure 9-17 displays the `login.php` page that appears when the user clicks the Login link on `index.php`.

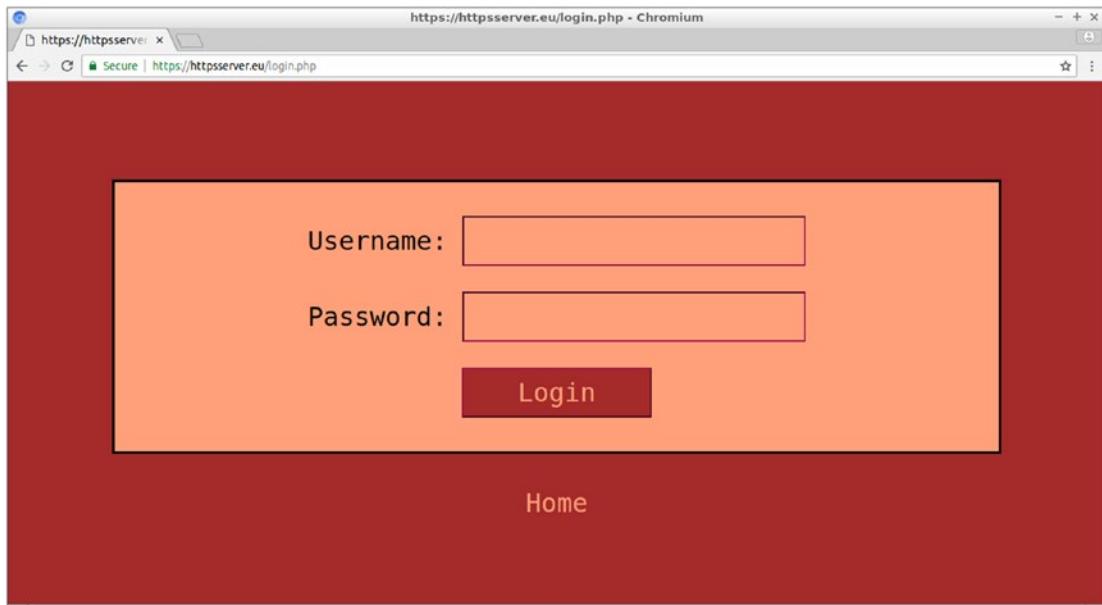


Figure 9-17. The login.php web page allows the user to log in to the site

To create the file login.php, enter the following command at the Linux terminal:

```
$ sudo gedit /var/www/html/login/login.php
```

Enter the following source code and save the file:

```
<?php  
session_start();  
?>  
<!DOCTYPE html>  
<html>  
<head>  
<style>  
body{  
background-color:brown;  
}  
.center {  
margin: auto;  
width: 80%;
```

```
border: 3px solid black;
padding: 10px;
background-color:lightsalmon;
}
.center2 {
    margin: auto;
    width: 70%;
    border: none;
    padding: 10px;
    background-color:brown;
}
p{
text-align:center;
font-family: Courier New,Courier,Lucida Sans Typewriter,Lucida
Typewriter,monospace;
font-size:32px;
font-weight:italic;
}
input{
border-color:#9F0251;
font-family: Courier New,Courier,Lucida Sans Typewriter,Lucida
Typewriter,monospace;
font-size:32px;
color:black;
padding:10px;
background-color:lightsalmon;
}
input[type=submit]{
background-color:brown;
color:lightsalmon;
}
a{
text-decoration:none;
}
```

```
a:link, a:visited {
    color: lightsalmon;
}
a:hover {
    color: black;
}
</style>
</head>

<body>
<?php
if (isset($_POST['s1'])) {
    $user = $_POST["user"];
    $pass = $_POST["pass"];

    if((empty($user)) || (empty($pass))) {
        echo '<p style="color:white">Please complete the username and the password.
</p>';
    } else {
        verify_password($user, $pass);
    }
}

function verify_password($user, $pass) {
    $dbserver = "localhost";
    $dbuser = "root";
    $dbpass = "";
    $dbname = "login";

    $mysqli = new mysqli($dbserver, $dbuser, $dbpass, $dbname);

    if ($mysqli->connect_error) {
        die("Connection failed: " . $mysqli->connect_error);
    }

    $user = mysqli_real_escape_string($mysqli, $user);
    $sql = "SELECT password FROM user WHERE username='$user'";
```

```
if($query = mysqli_query($mysqli, $sql)) {  
$rows = mysqli_num_rows($query);  
}  
  
if ($rows == 1) {  
$row = mysqli_fetch_assoc($query);  
$dbstored_pass=$row['password'];  
$isValid = password_verify($pass, $dbstored_pass);  
if ($isValid){  
$_SESSION['session_user']=$user;  
header("location: profile.php");  
}  
} else {  
echo '<p>Username or Password is invalid</p>';  
}  
  
$mysqli->close();  
  
}  
?  
  
<br><br><br><br><br><br>  
<div class="center">  
<form name="f1" method="post" action=<?php echo htmlspecialchars($_  
SERVER["PHP_SELF"]); ?>">  
<p>Username: <input type="text" name="user">  
</p>  
<p>Password: <input type="password" name="pass">  
</p>  
<p><input type="submit" name="s1" value="&ampnbsp&ampnbsp&ampnbspLogin&ampnbsp&ampnbsp&ampnbsp">  
</p>  
</form>  
</div>
```

```
<div class="center2">
<p><a href="index.php">Home</a></p>
</p>
</body>
</html>
```

Like with account.php and index.php, CSS attributes are implemented to style the login web page. Like account.php, an HTML form is used that implements the POST method to submit the values to login.php and validate the data passed. The form validation checks whether the two fields are empty, and its source code is included in the PHP block that runs when the form with the submit button s1 has sent data back to the same web page.

```
if (isset($_POST['s1'])) {
...
}
```

In the case of nonempty fields, the following function is called:

```
verify_password($user, $pass);
```

Here, \$user and \$pass are the PHP variables for the username and the password sent by the form:

```
$user = $_POST["user"];
$pass = $_POST["pass"];
```

What verify_password() does is connect to the MySQL server and specifically to the login database and run the following query, which retrieves the stored password in the database of the specific user:

```
SELECT password FROM user WHERE username='$user'
```

Before \$user is passed to the SQL query, mysqli_real_escape_string() applies to this variable to escape some characters, that is, to prepend a backslash before any special character, taking into account the character set of the current database connection, which appears as the first parameter of mysqli_real_escape_string(). The function mysqli_real_escape_string() is used for security reasons and especially for avoiding SQL injections, that is, executing commands hidden in the SQL query.

With the function `mysqli_num_rows()`, the number of rows of the query is returned, and if the row number is one (there is a password of the specific user), the row is retrieved as the `$row` array, and the password is retrieved as the array's item stored in index `password`.

```
$row = mysqli_fetch_assoc($query);
$dbstored_pass=$row['password'];
```

The following command calls the PHP function `password_verify()` to verify the password provided by the user and sent by the login form (`$pass`), with the password stored in the database (`$dbstored_pass`).

```
$isValid = password_verify($pass, $dbstored_pass);
```

As mentioned in the previous section, `$pass` is the plaintext password (e.g., 123), while `$dbstored_pass` is the hashed password. The function `password_verify()` compares the entered password with the stored one after it hashes the former using the salt stored along with the latter.

If the two passwords match, the value of `$user` is assigned to the PHP session variable `session_user`. At this point, more session variables could be used if required. Therefore, the username will be available to all other web pages of the site while the user is connected, and its unique value will be the key, which is where all information about the user will be derived from. The session variable, available for all web pages of the site that join the session, is stored on the server side, and the user cannot have any access to it and cannot substitute it with another user's.

After a successful login, the web browser redirects to `profile.php`.

```
if ($isValid){
$_SESSION['session_user']=$user;
header("location: profile.php");
```

Recall also that for the session to function as expected, the following PHP block that creates the session should be at the top of the web page:

```
<?php
session_start();
?>
```

The Source Code for the User Profile Page

Next, create `profile.php`, which is the web page the user redirects to on login. At the Linux terminal, enter the following:

```
$ sudo gedit /var/www/html/login/profile.php
```

Enter the following source code and save the file:

```
<?php
session_start();
$user=$_SESSION['session_user'];
?>
<!DOCTYPE html>
<html>
<head>
<style>
body{
background-color:brown;
}
.right {
    margin: auto;
    width: 100%;
    border: none;
    padding-right: 20px;
    text-align:right;
    background-color:lightsalmon;
    font-size:24px;
}
table{
background-color:lightsalmon;
color:black;
font-size:24px;
width:70%;
margin: 0 auto;
padding:20px;
}
```

```
td{  
text-align:center;  
}  
h1{  
text-align:center;  
}  
</style>  
</head>  
  
<body>  
  
<p class="right">  
You are currently logged in as user <?php echo $user ?>  
<a href="logout.php">Logout</a>  
</p>  
<br><br><br>  
<h1>Books Loaned</h1>  
  
<?php  
  
display_books($user);  
  
function display_books($user) {  
  
$servername = "localhost";  
$username = "root";  
$password = "";  
$dbname = "login";  
  
$mysqli = new mysqli($servername, $username, $password, $dbname);  
  
if ($mysqli->connect_error) {  
    die("Connection failed: " . $mysqli->connect_error);  
} else {  
    $sql = "SELECT book_loan.book_id, book_loan.loan_date FROM user INNER JOIN  
book_loan ON user.id = book_loan.user_id WHERE user.username='\$user'";  
    if($result=mysqli_query($mysqli, $sql)){  
        if (mysqli_num_rows($result)>=1){  
            echo "<table>";  
        }  
    }  
}
```

```

echo "<tr><td>" . "ISBN" . "</td><td>" . "Loan Date" . "</td></tr>";
while ($row=mysqli_fetch_assoc($result))
{
    echo "<tr><td>" . $row['book_id'] . "</td><td>" . $row['loan_date'] .
    "</td></tr>";
}
echo "</table>";
mysqli_free_result($result);
} else{
    echo "<h1>No results found</h1>";
}
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($mysqli);
}
$mysqli->close();
}

}

?>

</body>
</html>

```

The first block of PHP source code allows `profile.php` to join the PHP session and also retrieves from variable `$user` the value set by `login.php` for the session variable `session_user`.

```

<?php
session_start();
$user=$_SESSION['session_user'];
?>

```

In the top-right area of the web page, the message “You are currently logged in as user” appears followed by the username of the logged-in user, provided by the `$user` value. Next to the name, a Logout link appears. (I’ll discuss later in this section how this link works.)

Under the heading “Books Loaned” is a list with the book ISBNs that correspond to the `book_id` column of the `book_loan` table and also the date loaned that corresponds to the `loan_date` column of the same table. If the user has not currently loaned any books, as in the example with the user Sophie, the message “No results found” appears instead.

Like the source code in the files `account.php` and `login.php`, a connection with the `login` database is established with the function `mysqli()`.

```
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "login";

$mysqli = new mysqli($servername, $username, $password, $dbname);
```

On a successful connection, the following SQL command is executed, using the function `mysqli_query()`:

```
$sql = "SELECT book_loan.book_id, book_loan.loan_date FROM user INNER JOIN
book_loan ON user.id = book_loan.user_id WHERE user.username='\$user'";
```

This time, data from two tables is required, and an inner join between the two tables is formed. With the inner join, a new result table is created that selects only the rows from the two tables that match the join condition. In the previous SQL command, the join condition is equality among the primary key of table `user` and the foreign key of table `book_loan`. This type of join based on equality is often referred as *equijoin*, and this is often used to denormalize data. The term *normalizing* refers to the process of storing data in different tables. With the denormalizing process, data from different tables can be retrieved.

With the previous SQL command, the `book_id` and `loan_date` columns from the `book_loan` table are selected for the matching columns of `user_id` (of table `book_loan`) and `id` (of table `user`) from the rows where the `username` column (of table `user`) is the value of `$user` (e.g., `robert`). The query depends therefore on the PHP session variable `session_user`, whose value is assigned to the PHP variable `$user`.

If the return value of `mysqli_num_rows()` holds one or more rows, the results (`$row['book_id']` and `$row['loan_date']`) are included in an HTML table.

```
if (mysqli_num_rows($result)>=1){
    echo "<table>";
    echo "<tr><td>" . "ISBN" . "</td><td>" . "Loan Date" . "</td></tr>";
```

```

while ($row=mysqli_fetch_assoc($result))
{
    echo "<tr><td>". $row['book_id'] . "</td><td>" . $row['loan_date'] . 
    "</td></tr>";
}
echo "</table>";

```

The function `mysqli_fetch_assoc()` is used to provide the result rows. If no rows are returned, no table is formed, and the message “No results found” appears in a heading element.

```

} else{
    echo "<h1>No results found</h1>";
}

```

The user-specific data is thus displayed. In the following section, you’ll create the web page that logs the user out.

Allowing the User to Log Out

On `profile.php`, the PHP session is maintained, and the session variable `session_user` indicates the user who logged in and provides information specific to this user. Other web pages of the site could also be used to access the `session_user` variable and allow the user to stay connected and view user-specific data while visiting the site.

As mentioned previously, a message at the top right reminds the user of their name, and also a link called Logout allows the user to disconnect. This link leads to `logout.php`.

```

<p class="right">
You are currently logged in as user <?php echo $user ?>
<a href="logout.php">Logout</a>
</p>

```

Create the file `logout.php` using the following command at the Linux terminal:

```
$ sudo gedit /var/www/html/login/logout.php
```

Enter the following PHP code and save the file:

```
<?php  
session_start();  
setcookie(session_name(), “, 100);  
unset($_SESSION['session_user']);  
session_destroy();  
header('Location: login.php');  
exit;  
?>
```

The previous PHP source code snippet uses the function `session_start()` to join the current session and then deletes the cookie from the user's browser by setting the expiration time to a value in the past. By setting the expiration time to 100, you indicate value 100 as a Unix timestamp, which corresponds to the number of seconds after the Unix epoch, which is January 1, 1970 00:00:00 UTC. The session variable `session_user` is destroyed with the function `unset()`, and the PHP session is terminated by calling the function `session_destroy()`. With the `header()` function, the web page redirects to `login.php`.

Testing the User Connection to the Site

To test the two users, you can use the `login.php` page, which appears when you return to the home page. Just click the Home link on `account.php` and then use the Login link. Or, since you already know the site's structure, enter the following:

<https://httpsserver.eu/login.php>

Enter the username, e.g., **robert**, and the user's password, e.g., **123**, for Robert, and click the Login button, as displayed in Figure 9-18.

CHAPTER 9 RUNNING YOUR SITE WITH A CERTIFICATE FROM A CERTIFICATE AUTHORITY

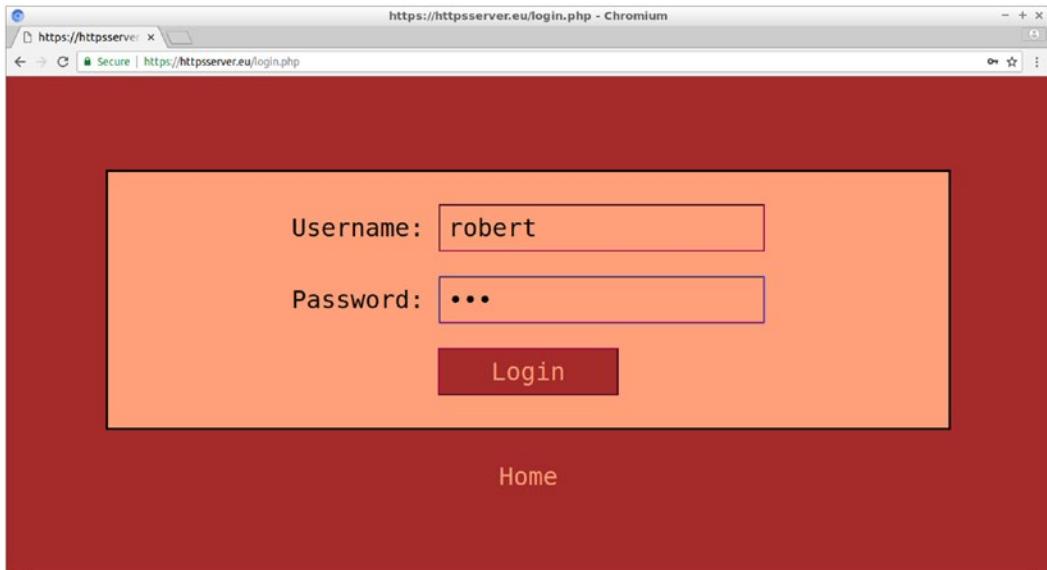
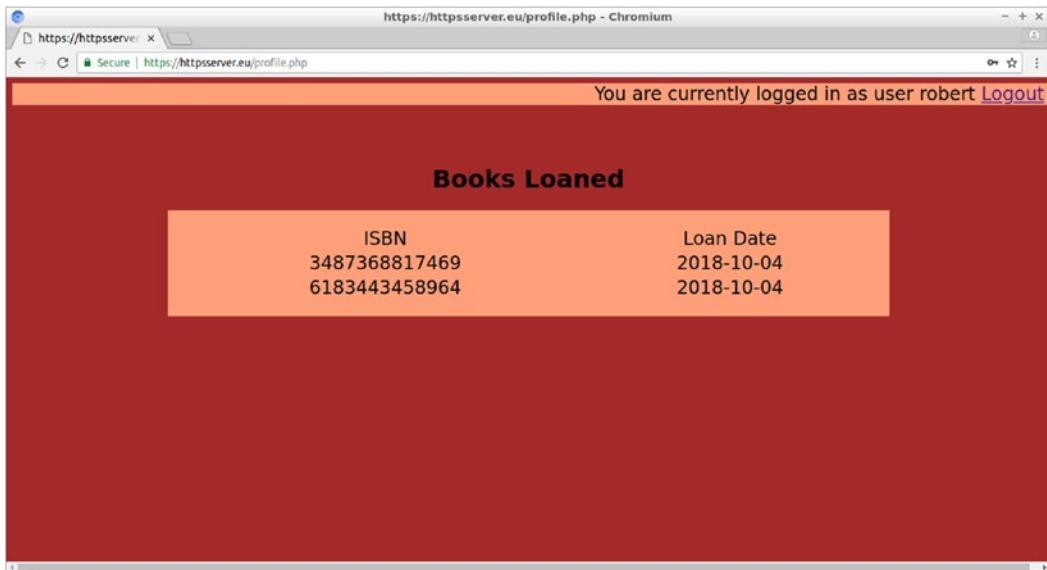


Figure 9-18. Testing the site by connecting as Robert

User Robert gets connected to his profile page with personal information about his recent activity. In the top-right area of the page, the message “You are currently logged in as user robert” appears.



ISBN	Loan Date
3487368817469	2018-10-04
6183443458964	2018-10-04

Figure 9-19. The profile web page for Robert, displaying the ISBN and loan date of loaned books

For books currently loaned to user Robert, the ISBN and loan date appear.

The Logout link leads to `logout.php`, where its evaluated PHP source code logs out the user by terminating the current session. The user then gets transferred back to the `login.php` web page.

Improving the `profile.php` Web Page

The data format for the loan date, appearing in `profile.php`, is the default one used by MySQL. To change this to a more readable format and also add a new column called Due Date, replace the `display_book()` function with the following new version:

```
function display_books($user) {

    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "login";

    $mysqli = new mysqli($servername, $username, $password, $dbname);

    if ($mysqli->connect_error) {
        die("Connection failed: " . $mysqli->connect_error);
    } else {
        $sql = "SELECT book_loan.book_id, book_loan.loan_date FROM user INNER JOIN
book_loan ON user.id = book_loan.user_id WHERE user.username='\$user'";
        if($result=mysqli_query($mysqli, $sql)){
            if (mysqli_num_rows($result)>=1){
                echo "<table>";
                echo "<tr><td>' . '<b>ISBN</b>' . "</td><td>' . '<b>Loan Date</b>' .
                "</td><td>' . '<b>Due Date</b>' . "</td></tr>";
                while ($row=mysqli_fetch_assoc($result))
                {
                    $date1 = strtotime($row['loan_date']);
                    $newformat1 = date('l, j F Y', $date1);
                    $date2 = strtotime("+ 21 days", $date1);
                    $newformat2 = date('l, j F Y', $date2);

```

```

        echo "<tr><td>". $row['book_id'] . "</td><td>" . $newformat1 . "</
        td><td>" . $newformat2 . "</td></tr>";
    }
    echo "</table>";
    mysqli_free_result($result);
} else{
    echo "<h1>No results found</h1>";
}
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($mysqli);
}
$mysqli->close();
}
}
}
```

The first call to the PHP function `strtotime()` converts the string `$row['loan_date']` to a Unix timestamp and returns this value to the variable `$date1`. The function `date()` is called to change `$date1`, passed as its second parameter, from a Unix timestamp to a readable format, indicated by the first parameter, ('l, j F Y').

```

$date1 = strtotime($row['loan_date']);
$newformat1 = date('l, j F Y', $date1);
```

In the example of the source code, the following characters are used for the format parameter:

- l, a full textual representation of the day of the week
- j, the day of the month without leading zeros
- F, a full textual representation of a month
- Y, a full numeric representation of a year

The second time `strtotime()` is called, it results in a Unix timestamp increased by 21 days, which is the specified period in the local library in this example for book loans. The `date()` call converts the Unix timestamp to the previous readable format:

```

$date2 = strtotime("+ 21 days", $date1);
$newformat2 = date('l, j F Y', $date2);
```

Finally, \$newformat1 and \$newformat2 are printed in the Loan Date and Due Date columns, respectively, as viewed in Figure 9-20.

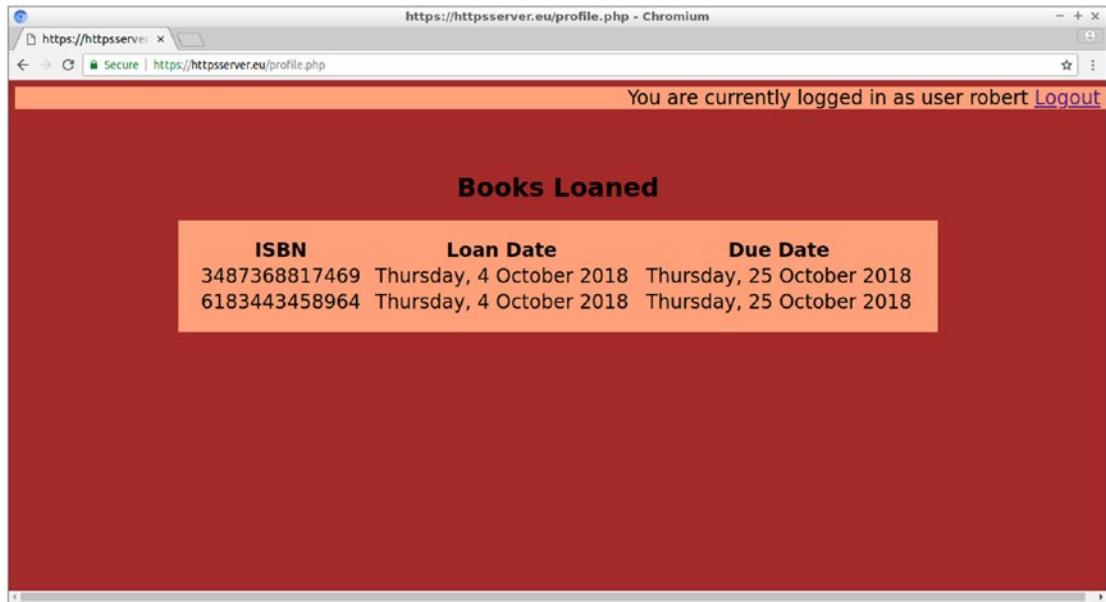


Figure 9-20. The new version of profile.php includes a Due Date column

Summary

In this chapter, you obtained an SSL certificate from a certificate authority to enable your site to implement cryptography without getting any browser warnings, just like well-known commercial sites.

You also created a project that allows the users to log in and remain connected to your site until they log out. While connected, they can view user-specific data provided by the site.

The most common ways to exchange data between the client-side JavaScript and server-side PHP language are discussed in the next chapter.

CHAPTER 10

Running Online Services Using PHP Sockets

In this final chapter of the book, you will use either web server, Apache or Lighttpd, with the SSL certificate obtained in the previous chapter. You'll start by configuring and running a DDNS client program to update your public IP address to the DDNS server provider instead of relying the IP address of your router.

The projects in this chapter require socket programming so that the PHP code executing on the server side can connect to other Internet services.

Specifically, for this chapter's projects, the web server uses PHP sockets to connect to other Internet services. As an example of applying socket programming, you'll create a PHP server first that listens to a specific port number and is tested with Telnet and then with a PHP client program. Then the web server is used to connect to this PHP server, thus implementing a web interface to the service that the PHP server offers.

In the second project, socket programming is used to create a site that simulates an open port check, similar to the online service used in Chapter 8, and a site where the web server connects to an existing quote of the day (QOTD) server.

Finally, for each site you will create in this chapter, a separate FQDN is assigned.

Updating the Domain Name IP Address with ddclient

At this point, you have a site that is enabled to run either on Apache or on Lighttpd, and both servers use the PHP language to connect to the MySQL server and provide dynamic content. For the site, you used DDNS domain names (e.g., christos.ddns.net), and you optionally purchased a second-level domain name (e.g., httpsserver.eu) so that the web servers can implement HTTPS with an SSL certificate from a CA. Also, all HTTP requests redirect to HTTPS via port 443. You can use both servers by starting and stopping them accordingly.

Hint! If you don't recall which server currently is running, use the following command to find out whether any Lighttpd or Apache process is running on your system:

```
ps xa | grep -E 'apache2 | lighttpd'
```

To toggle between Apache and Lighttpd, use `service` command to stop the server running and start the inactive one. For instance, if Lighttpd currently is running, enter the following at the Linux terminal:

```
$ sudo service lighttpd stop  
$ sudo service apache2 start
```

Both Apache and Lighttpd currently are serving the site displayed in Figure 10-1, which has the document root `/var/www/html/login` and the file `index.php` as the directory index.

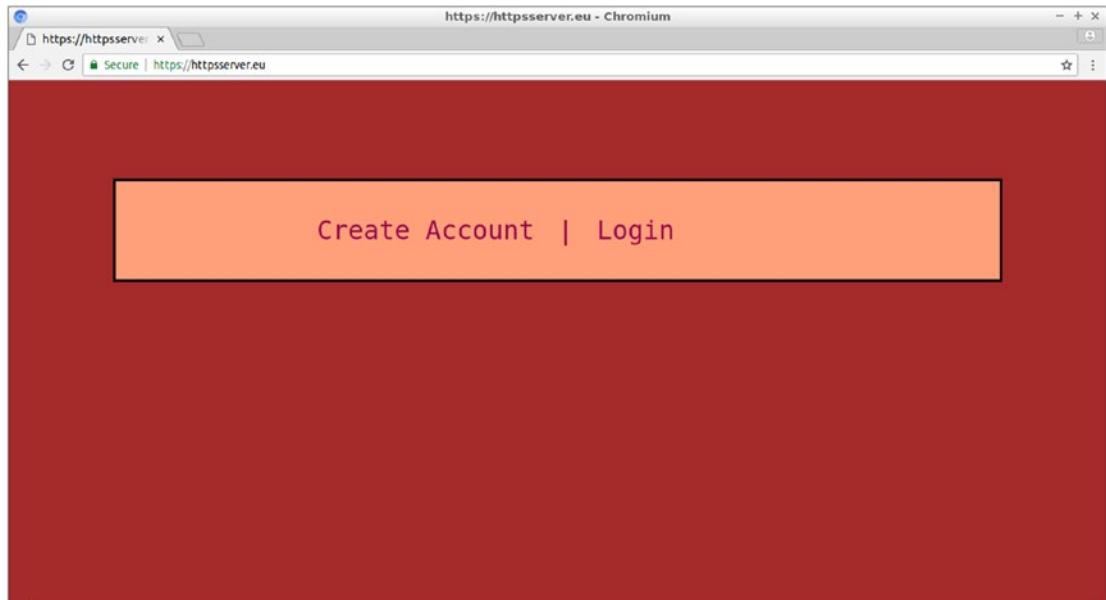


Figure 10-1. The web site currently running for both web servers

Before continuing with the projects in this chapter, let's ensure that the domain name obtained (e.g., `httpsserver.eu`) is still usable. If your ISP uses a dynamic IP address service, the domain name is expected sooner or later. For example, when the router restarts, your router's public IP address changes, and the domain name used with the DDNS service will not correspond anymore to your public IP address used when the domain was registered.

Chapter 3 discussed the router service for updating the public IP address on the domain name provider. However, if the router model does not support that specific provider, you can't rely on the router for the update. In that case, you have two options. The first is to ask your ISP to upgrade to a static IP address service. This is the best solution, especially if you really want to run a site 24/7. In that case, on your domain name's provider site, after logging into your account, you can bind the static IP address to your domain name. For instance, for the domain name provider used in the examples of this chapter, the configuration page looks like the one in Figure 10-2.

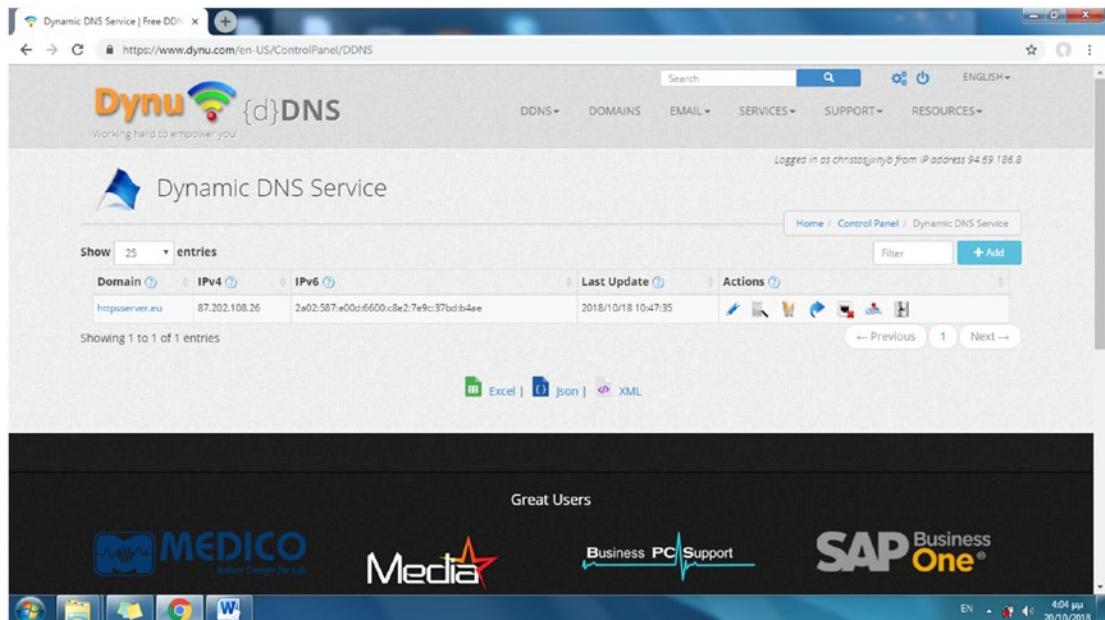


Figure 10-2. Displaying the current IP address that is bound to the domain name on the domain name provider's site

The other option is to install and run on your server a utility such as `ddclient` to do DDNS updates. You can run `ddclient` as a daemon process that periodically checks your public IP address and informs the domain name provider when this changes.

To install ddclient, enter the following at the Linux terminal:

```
$ sudo apt-get install ddclient
```

The installation process starts with the window in Figure 10-3, which asks you whether your provider appears in the included list. Use the keyboard arrows to choose Other, click the Tab key to select OK, and press Enter.

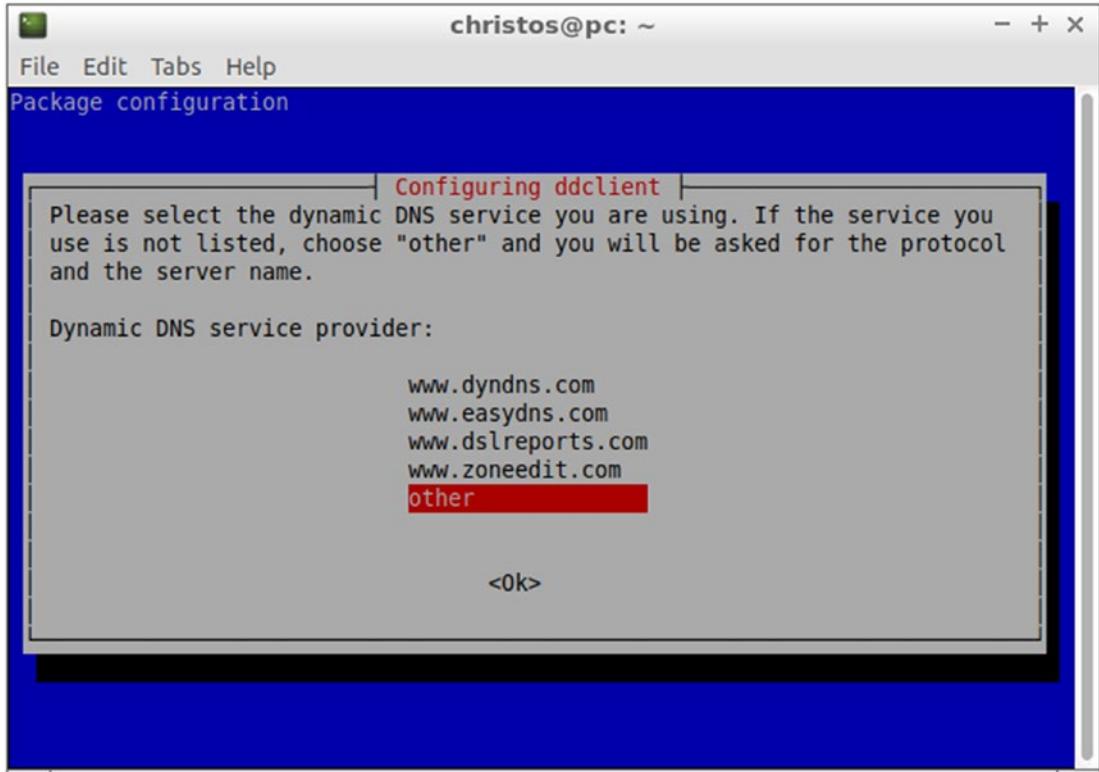


Figure 10-3. The first dialog in the ddclient installation

The installation process continues with a number of windows that retrieve information for the ddclient configuration file. At this point, you can skip each window by pressing the Esc key until the installation is complete, and then you can use one of the predefined ddclient configuration files that your domain name provider offers.

Most domain name providers provide complete guidance for setting up ddclient to use with their service, like the ones displayed in Figure 10-4.

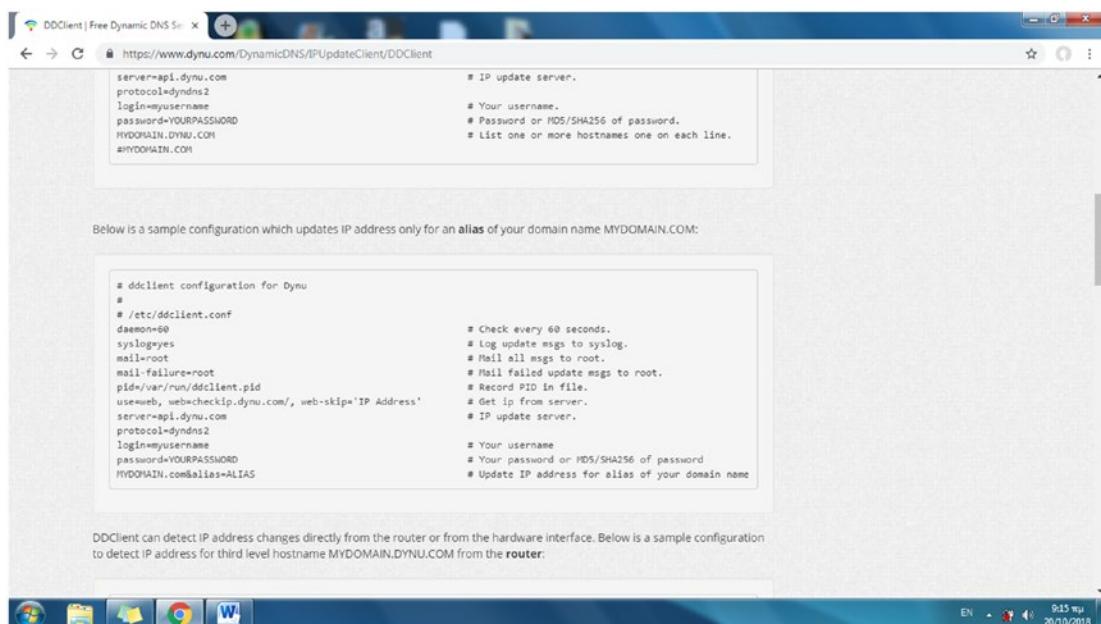


Figure 10-4. Viewing instructions for ddclient configuration from the DDNS provider's site

To edit the ddclient configuration file, enter the following at the Linux terminal:

```
$ sudo gedit /etc/ddclient.conf
```

Enter the settings recommended by your domain name provider; replace the username, password, and domain name with yours; and save the file. In the following sample file, used for [Dynu.com](#), the three last lines were changed. You can also modify the time interval that ddclient uses to check for changed IP addresses. In the following configuration file, the time interval is set to 60 seconds.

```
daemon=60
syslog=yes
mail=root
mail-failure=root
pid=/var/run/ddclient.pid
use=web, web=checkip.dynu.com/, web-skip='IP Address'
server=api.dynu.com
protocol=dyndns2
```

```
login=christosffxdyv  
password=somePsw  
httpsserver.eu
```

Configure ddclient to run as a daemon process. Run the following to edit the /etc/default/ddclient configuration file:

```
$ sudo gedit /etc/default/ddclient
```

Change false to true in the following line:

```
run_daemon="false"
```

Save the file and run the following commands at the Linux terminal:

```
$ sudo update-rc.d ddclient enable  
$ sudo systemctl start ddclient.service
```

Test next the ddclient daemon. Switch off your router so that the public IP address of your router renews and then switch it back on. Run the following command that filters the lines with the word SUCCESS on it, created by ddclient:

```
$ sudo cat /var/log/syslog | grep SUCCESS
```

The output indicates the successful update of the IP address.

```
Oct 20 16:21:44 pc ddclient[4014]: SUCCESS: updating httpsserver.eu: good:  
IP address set to 94.69.186.8
```

View also the DNS configuration web page at your DNS provider, as displayed in Figure 10-5.

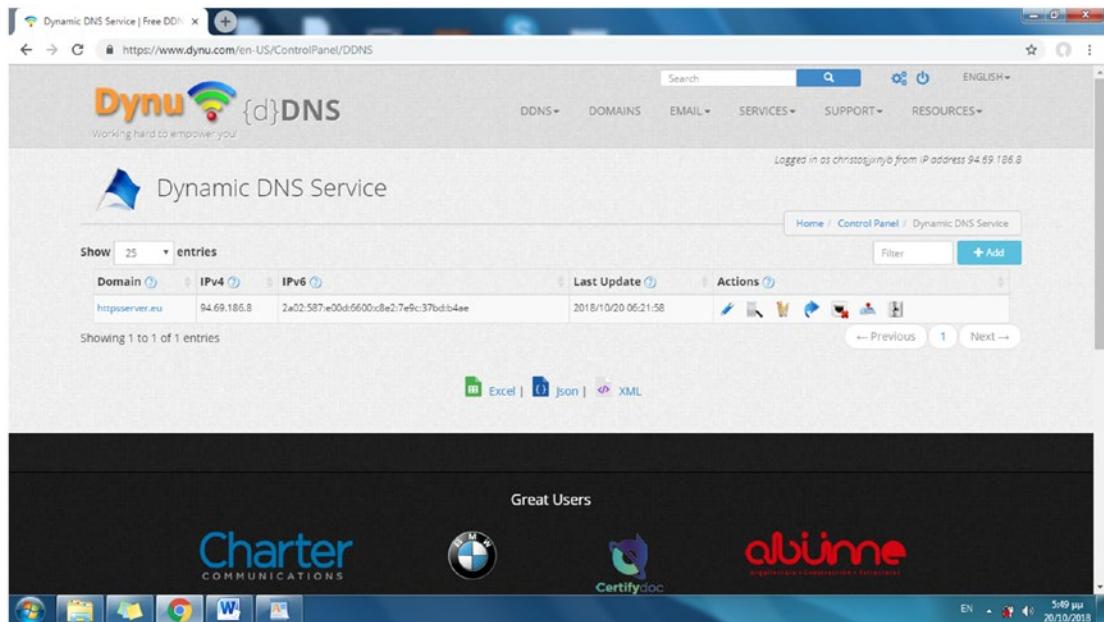


Figure 10-5. Displaying the new public IP address of your router, corresponding to your domain name at the DDNS service provider's site

As viewed in the web page, the IP address corresponding to the domain name `httpsserver.eu` is indeed successfully updated to 94.69.186.8.

With your domain name always corresponding to the public IP address of your router, even when the IP address changes, you can use this domain name for the sites that run the chapter's projects. Because the projects are based on TCP/IP sockets, the basic notions of the socket programming are discussed next.

Utilizing PHP Sockets

A *socket* as a networking term is a pair of an IP address and a port number that belongs to each one of the two ends of a network connection, e.g., a client and a server socket for a client-server connection. The socket in an application programming interface (API) provides the routines for interprocess communication between applications at the TCP/IP or at the UNIX local level.

The PHP language supports a full-fledged sockets library similar to the one introduced with Berkeley Sockets (POSIX Sockets). Berkeley Sockets were initially released in 1983 as a library of the C language. Since then, many other programming

languages include the Sockets API. Sockets are usually implemented as system calls, routines that allow a programming language to interact with the operating system (OS), and request services, in the case of socket networking services. With PHP sockets, you may connect to a port number at a given IP address and connect therefore to any service bound on the specific port number on that system. Your browser uses sockets to connect to a given web server. With sockets you can implement a client or a server for any Internet service: email, DNS, a network online game, and so on.

The ability for the PHP code to connect to Internet services with sockets is very important to web development because the web server may interact with other services. You have already created a few sites where the web server exchanges information with the MySQL server. Consider also another example like the project of Chapter 8, where a user account is created by submitting personal details such as the first name, the last name, and the e-mail address. Most sites verify the e-mail address of a new member by submitting an e-mail to the client that includes a link, which activates the account. As with the MySQL interaction the PHP code of the web server to provide e-mail handling it must interact with the e-mail services. The final project included in the current site enables the web server PHP code to connect as a client with an Internet service using a more simplified protocol like the quote of the day (QOTD).

In the following section, you will create a PHP command-line program that implements a server listening to a specific port number, used as a first example for connections to non-HTTP services.

The Code for the Command-Line PHP Socket Server

You will create a server listening to a specific port number with a PHP terminal program. In the following example, the server listens to port 33000 and does not implement any well-known service but simply returns to the client the message previously sent in uppercase characters. At the Linux command line, enter the following:

```
$ gedit server.php
```

Enter the following PHP source code and save the file:

```
<?php
```

```
// Set the IP address and port number the server will listen on  
$address = '192.168.1.100';
```

```
$port = 33000;

// Create the server socket, a TCP Stream socket
$server = socket_create(AF_INET, SOCK_STREAM, 0);

// Bind the socket to an IP address / port number pair
socket_bind($server, $address, $port) or die('Could not bind to address');

// Start listening for connections
socket_listen($server);

//loop for and listen for connections
while (true) {

// Accept incoming requests and handle them as child processes
    $client = socket_accept($server) or die("Could not accept incoming
connection\n");

    if($client){
        // Read client input of 1024 bytes
        $input = socket_read($client, 1024) or die("Could not read input\n");

        // Strip all white spaces from input
        $output = strtoupper($input);

        // Send $output back to client
        socket_write($client, "you wrote " . $output . "\n") or die("Could not
write output\n");

        // Close the client socket
        socket_close($client);
    }
}

?>
```

The PHP program creates a socket descriptor (a notion similar to a file descriptor for accessing a connection) `$server` with the function `socket_create()`. The first parameter of `socket_create()` is `AF_INET`, which indicates that the IPv4 protocol will be used. Other options are `AF_INET6` for IPv6 Internet-based protocols and `AF_UNIX` for process communication in the same computer. The second parameter is set to

`SOCK_STREAM`, used for TCP connections. Other options are the `SOCK_DGRAM` for UDP connections, `SOCK_SEQPACKET`, `SOCK_RAW`, and `SOCK_RDM`.

The socket is then bound, with `socket_bind()`, to the private IP address of the computer, 192.168.1.100 in this example, and to port number 33000, which is randomly selected out of the range of the system ports numbers (0–1023).

With the function `socket_listen()`, the server socket listens for connections and enters an infinite loop where if a client connects, the function `socket_accept()` returns `$client` as its socket descriptor.

Next, with the function `socket_read()`, the socket reads user data up to 1,024 bytes in this example, and with the PHP function `strtoupper()` converts the lowercase characters entered by the client to uppercase. The function `socket_write()` sends a message back to the client, which includes the client's text in uppercase characters. Finally, the function `socket_close()` closes the client socket descriptor.

Testing the PHP Command-Line Socket Server

To test the command-line PHP server, you can create a PHP command-line client. However, instead of this, you will simulate a client by using Telnet, the Swiss Army knife for networking. At the Linux terminal, enter the following command to start `server.php`:

```
$ php server.php
```

Next, switch to another terminal by using Ctrl+Alt+T and run Telnet as follows:

```
$ telnet 192.168.1.100 33000
```

Telnet responds with the following message, which indicates that a connection is issued to IP address 192.168.1.100, the host of the server program in this example, at the selected port number (33000):

```
Trying 192.168.1.100...
Connected to 192.168.1.100.
Escape character is '^]'.
```

Enter some text and press Enter.

```
hello world
```

The server responds with the following message:

```
you wrote HELLO WORLD  
Connection closed by foreign host.
```

In the following section, you'll connect to the PHP server by implementing a PHP client.

Implementing a Command-Line PHP Client

Next, you'll create a command-line sockets client with PHP code. Similar source code will be used in the PHP source code of the web server to make a connection to the command-line PHP server.

To create a PHP client program, use the following command at the Linux terminal:

```
$ gedit client.php
```

Enter the following PHP source code and save the file:

```
<?php  
  
$host = "192.168.1.100";  
$port = 33000;  
  
$server = socket_create(AF_INET, SOCK_STREAM, 0) or die("Could not create  
socket\n");  
  
$result = socket_connect($server, $host, $port) or die("Could not connect  
to server\n");  
  
$prompt = "Write your message here: ";  
  
$message = readline($prompt);  
  
socket_write($server, $message, strlen($message)) or die("Could not send  
data to server\n");  
  
$result = socket_read($server, 1024) or die("Could not read server response\n");  
  
echo "Reply: " . $result;  
  
socket_close($server);  
  
?>
```

The `socket_create()` function returns `$server`, which is a socket descriptor and is used by the function `socket_connect()` to connect to IP address 192.168.1.100 on port 33000, where the PHP server listens. A message is then entered by the user with the PHP function `readline()`, and this message is sent to the server with the function `socket_write()`. Then `socket_read()` function returns the server's response, up to 1,024 bytes in this example, to the client. The reply is printed to the terminal with the `echo` command, and the `$server` socket closes with `socket_close()`.

Now test the command-line server with the command-line client. Start the PHP server using the following command at the Linux terminal:

```
$ php server.php
```

Then switch to a different terminal and start the PHP client using the following command:

```
$ php client.php
```

The client responds with the following message:

Write your message here:

Enter your message. For instance, enter `hello world` at the client's prompt and press Enter.

Write your message here: `hello world`

The response from the PHP command-line server client with PHP code. Similar source code will be used in the to the client terminal is as follows:

Reply: you wrote HELLO WORLD

Next, you will connect to the terminal PHP server from the PHP source code run by the PHP engine of the web server. You have to create a new site to allow the web interface into the terminal PHP server, but first you have to set up the web servers.

Configuring the Web Servers for the New Site

Now you'll create another site to use for the chapter's projects in which I'll show the directory index and document roots for the next three projects. The site will be served from both the Apache and Lighttpd servers. Create a subdirectory called `sockets` of the

default document root used for those servers, `/var/www/html/`, and set this directory as the document root. At the Linux terminal, enter the following:

```
$ sudo mkdir /var/www/html/sockets
```

Edit the configuration files of the two web servers so that both are enabled to run the site, although only one of the two will be active. For Apache, edit `default-ssl.conf`, which is the configuration file for HTTPS, using the following:

```
$ sudo gedit /etc/apache2/sites-available/default-ssl.conf
```

Set the new document root with the `DocumentRoot` directive and save the file.

```
DocumentRoot /var/www/html/sockets
```

Reload the Apache server so that the new configuration is enabled.

```
$ sudo service apache2 force-reload
```

Edit next the Lighttpd configuration file. Use the following:

```
$ sudo gedit /etc/lighttpd/lighttpd.conf
```

Set the value of the `server.document-root` directive to the new document root and save the file.

```
server.document-root      = "/var/www/html/sockets"
```

Reload the Lighttpd server to enable the changes in the configuration file.

```
$ sudo service lighttpd force-reload
```

Create the Site That Interfaces with the Command-Line Server

One option for connecting a remote client to the PHP command-line server is to use a client program that connects to the public IP address of the router and uses port forwarding for port 33000 to access the computer that hosts `server.php`. The other option used in this section is to create a web-based interface that connects to the command-line server via port 33000 with no specialized client program required.

Create the web page `message.php`, which will include an HTML form with a textbox for the user to type a message. The message is then submitted to `client.php`, the action

CHAPTER 10 RUNNING ONLINE SERVICES USING PHP SOCKETS

file of the form on the web server. This file uses PHP source code to interface as a client with the Linux terminal program `server.php`. The user's message from `client.php` is relayed to the terminal server, and the resulting uppercase message is returned to `client.php` and displayed on the evaluated HTML web page.

In the document root, create two PHP files. Use the following commands on the Linux terminal to create `message.php`:

```
$ cd /var/www/html/sockets  
$ sudo gedit message.php
```

Enter the following source code and save the file:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
body{  
background-color:lightsteelblue;  
}  
p{  
text-align:center;  
font-size:32px;  
}  
</style>  
</head>  
<body>  
<?php  
  
if(isset($_POST["t1"])) {  
$text = $_POST["t1"];  
}  
  
$host = "192.168.1.100";  
$port = 33000;  
  
$socket = socket_create(AF_INET , SOCK_STREAM , 0);  
$connection = socket_connect($socket , $host , $port);
```

```
socket_write($socket, $text, strlen($text));
$result = socket_read ($socket, 1024);
socket_close($socket);

echo "<p>" . $result . "</p>";

?>

</body>
</html>
```

Test the client-server system. First open `server.php` at the Linux terminal as follows:

```
$ php server.php
```

Switch next to the browser and enter the following URL:

<https://httpsserver.eu/sockets/message.php>

Figure 10-6 shows the web page `message.php`.

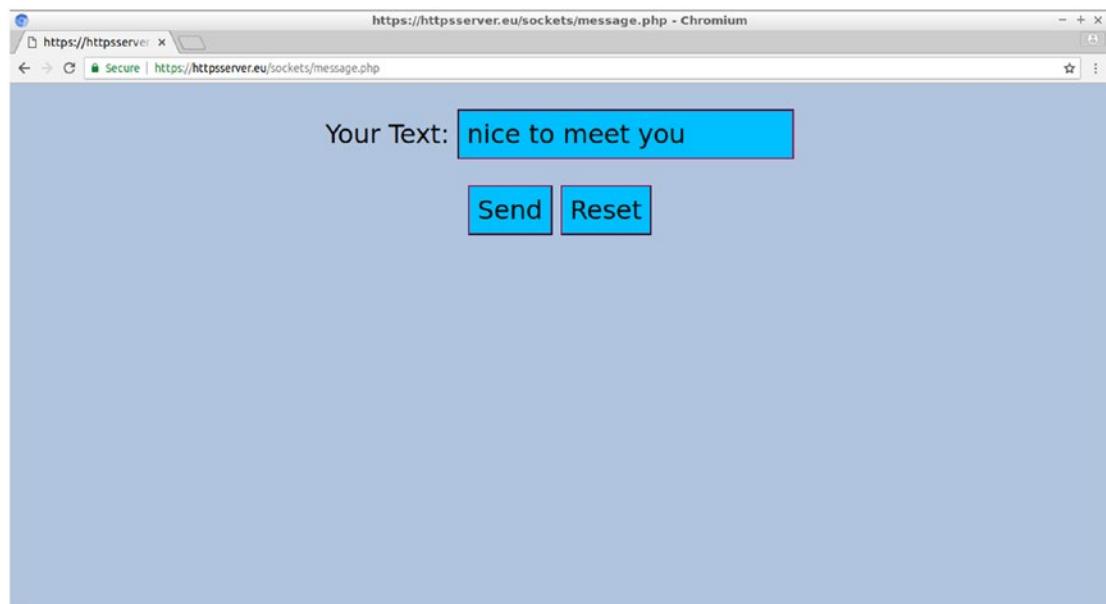


Figure 10-6. The HTML form that submits the user's text to `client.php`, which connects with sockets to the terminal server

In the textbox, enter some text, such as **nice to meet you**. Click the Send button. The server's response, which is the message in capital letters, is shown in Figure 10-7.

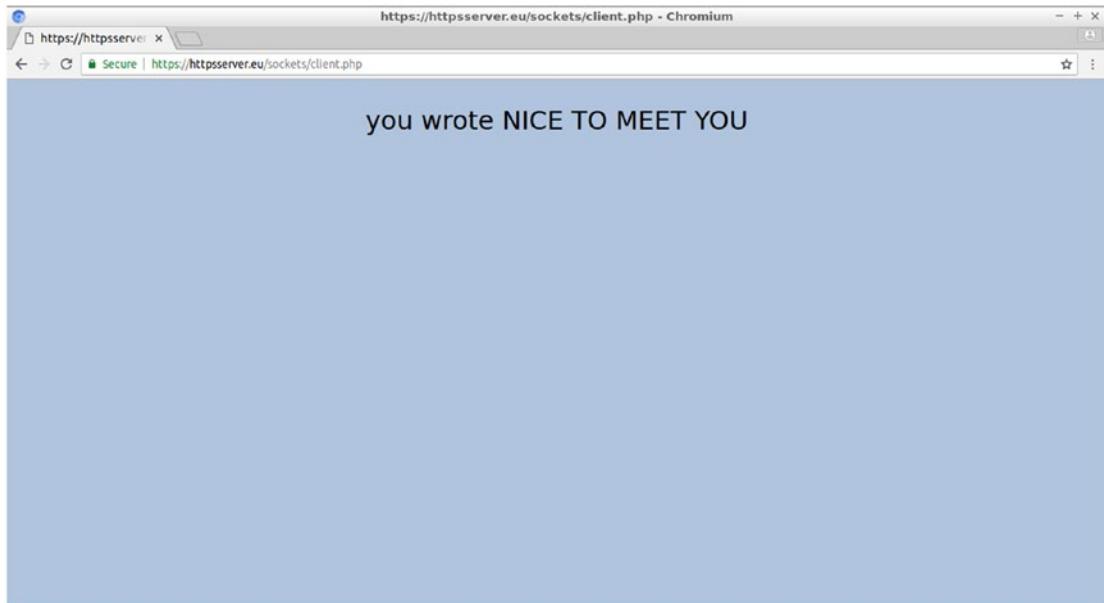


Figure 10-7. The terminal server's output as it appears to the evaluated client.php

In the following section, you'll create the first of the two main projects for this chapter, where the server-side programs use sockets to connect to remote computers.

A TCP Port Check Site

Create a project that uses TCP/IP sockets to simulate an online service and checks whether a port on the client system is open. This is similar to the service used in Chapter 8 (e.g., www.yougetsignal.com or www.canyouseeme.org), as displayed in Figure 10-8.

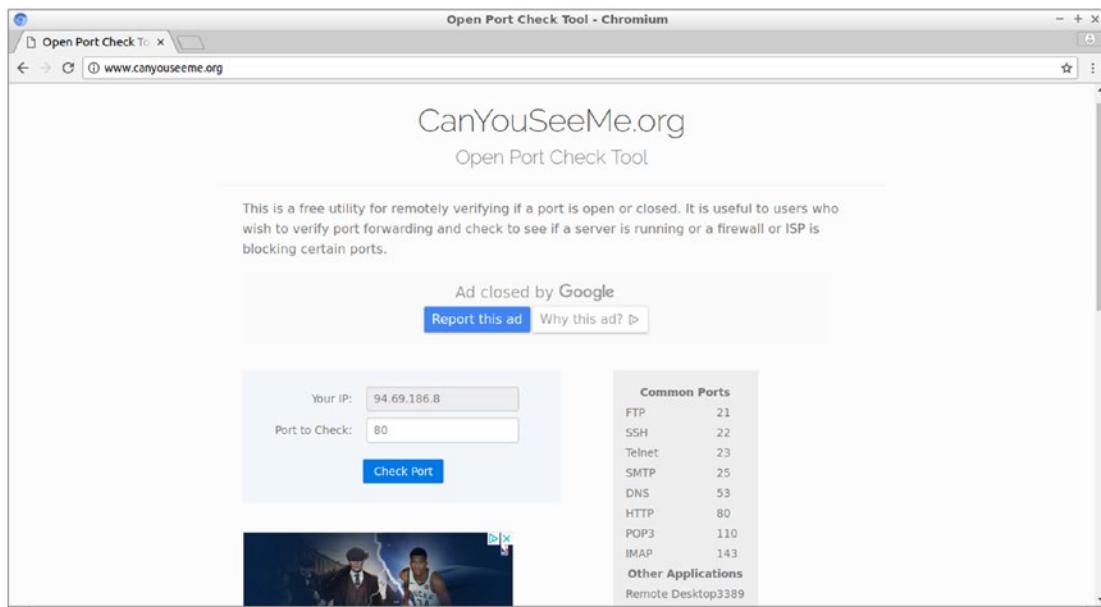


Figure 10-8. An online open port check tool

Two PHP files are required for this file. The file `index.php`, displayed in Figure 10-9, is the directory index that the user views when it uses the domain name of the site as the URL, as shown here:

<https://httpsserver.eu>



Figure 10-9. The home page of the open port check tool site

On `index.php`, the user enters the port number to be checked and clicks the Go button. The form data is submitted to `ports.php`, the second PHP file used in this project, and the PHP code connects to the client computer via the specific port number. According to the connection status, the port is considered open or closed. The server's response message is displayed in a dialog that is created by the JavaScript source code injected in PHP.

The Source Code for `index.php` and `ports.php`

Create the first PHP file using the following command in the Linux terminal:

```
$ sudo gedit /var/www/html/sockets/index.php
```

Enter the following source code and save the file:

```
<!DOCTYPE html>
<html>
<head>
<style>
```

```
body{
background-color:orangered;
}
input[type=number]{
width: 300px;
}
p{
text-align:center;
font-size:32px;
color:lightsalmon;
}
input{
border-color:#9F0251;
font-size:32px;
color:black;
padding:10px;
background-color:lightsalmon;
}
</style>
</head>
<body>
<form method="post" action="ports.php">
<p>
<label for="port">Enter TCP Port Number:</label>
<input name="port" type="number" placeholder="Min: 0, Max: 1023" min="0"
max="1023" maxlength="25">
</p>
<p>
<input type="submit" name="s1" value="Go">
<input type="reset" value="Reset">
</p>
</form>
</div>
</body>
</html>
```

The web page created with the `index.php` source code consists of a form that includes three objects: one field of type `number`, and two buttons, one of type `submit` and one of type `reset`. By using the `input` element of type `number`, you apply the restriction that the value entered by the user should be an integer number. In addition, the `max` and `min` attributes of the element are set to the upper and lower limits of the number entered. The range is between TCP port numbers 0 and 1023, the *well-known ports*, also known as *system ports*. The well-known ports are assigned to commonly used types of Internet services. For instance, the web server utilized two system ports, ports 80 and 443, in the previous chapters.

When the number entered by the user in the number field exceeds the well-known port number range, a warning message is displayed, as viewed in Figure 10-10.

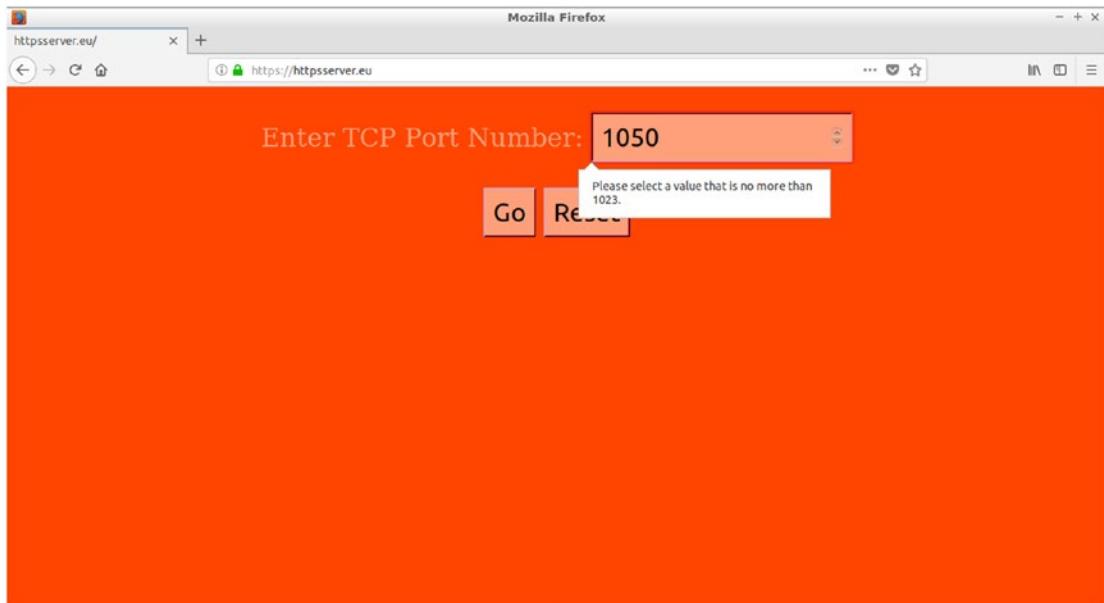


Figure 10-10. A warning message is displayed when the user exceeds the system port number limits

A similar message appears when a non-numeric value is entered. Also, the arrows that appear in the Firefox browser on the right of the number field increase/decrease the number entered by one.

The `method` attribute of the form is set to `POST`, and the `action` attribute is set to `ports.php`. Create this file using the following:

```
$ sudo gedit /var/www/html/sockets/ports.php
```

Enter the following source code and save the file:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Port Scanning</title>
</head>
<body>

<?php

$host = $_SERVER['REMOTE_ADDR'];

if (isset($_POST['s1'])){

    $port = intval($_POST["port"]);
    if (isset($port)) {

        $socket = socket_create(AF_INET , SOCK_STREAM , SOL_TCP);
        $connection = socket_connect($socket , $host , $port);

        echo '
<script>
alert("Port number ';
        echo $port;
        echo ' is';

        if (!$connection){
            echo ' not';
        }

        echo ' open.");
        window.location = "index.php";
        </script>';

    }

}

?>

</body>
</html>
```

The PHP source code of `ports.php` creates a TCP socket that binds to the IP address of the visitor's computer and also to the port number to be tested. The IP address is assigned to the PHP variable `$host` by the PHP global variable `$_SERVER['REMOTE_ADDR']`, and the TCP port number is assigned to the PHP variable `$port` from the data submitted by the HTML, `$_POST["port"]`. The value is typecast to an integer with the PHP function `intval()`. A socket connection is created as follows:

```
$connection = socket_connect($socket , $host , $port);
```

If the value returned to the `$connection` variable is `true`, a connection to the specified IP address on the tested port succeeds, and the port is considered open to the client computer. Otherwise, it is considered closed. The PHP code prints with the `echo` command some JavaScript source code that will be executed on the client side by the client's browser. The script creates a dialog that reports the state of the specific port. Also, the following JavaScript command redirects the web page to `index.php`, where another port can be tested:

```
window.location = "index.php";
```

Testing the Online Open Port Check Site Locally

Enter the URL of your site in the address bar of your browser.

<https://httpsserver.eu>

Test first a port number that is open. If your router, for instance, port forwards to ports 443 and 80, those are the open ports, with all the others closed. Enter, for instance, **443** in the form's number field, as displayed in Figure 10-11.

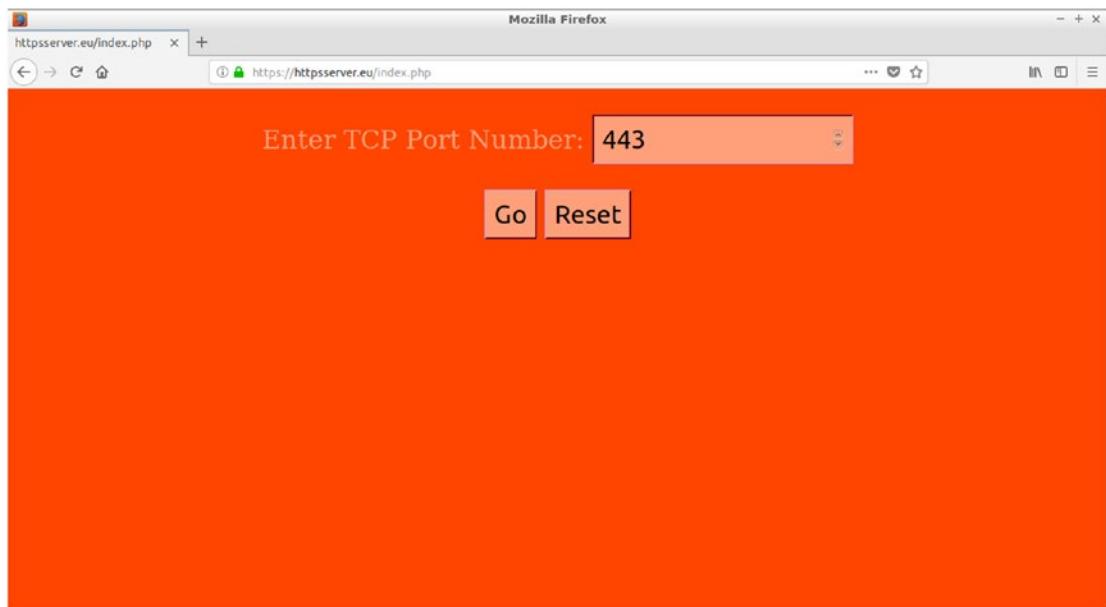


Figure 10-11. Testing the port number check tool for port 443

Click the Go button. The web page shown in Figure 10-12 appears.

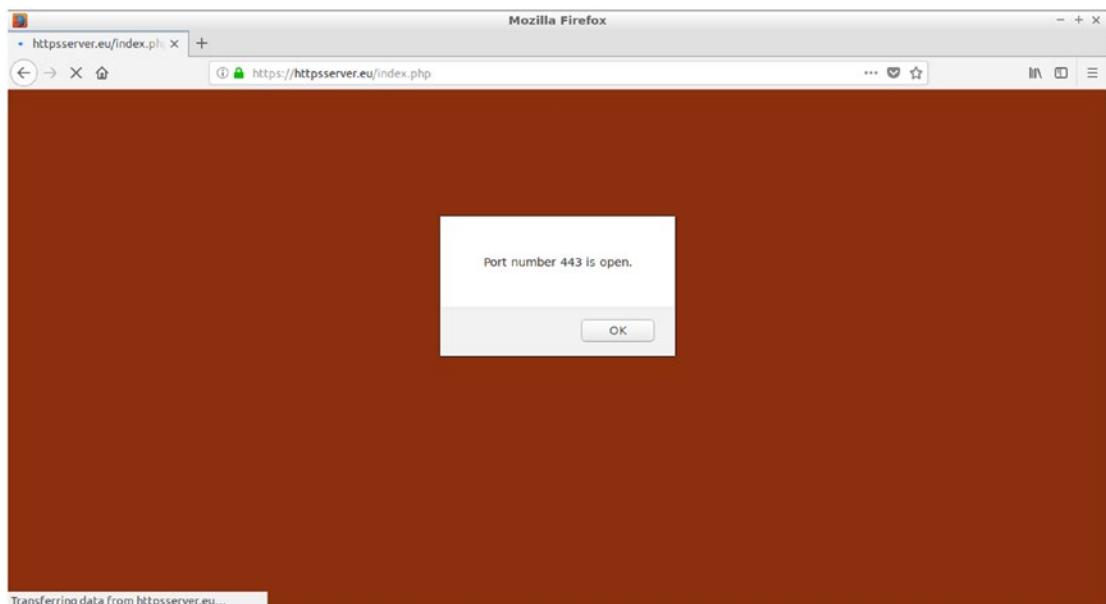


Figure 10-12. The port check output indicates an open port for port 443

CHAPTER 10 RUNNING ONLINE SERVICES USING PHP SOCKETS

As indicated in the message, “Port number 443 is open.” Click the OK button to return to `index.php` and test another port. Enter **25** this time as the port number, as displayed in Figure 10-13.



Figure 10-13. Testing the port number check tool for port 25

Click the Go button. The web page shown in Figure 10-14 appears.

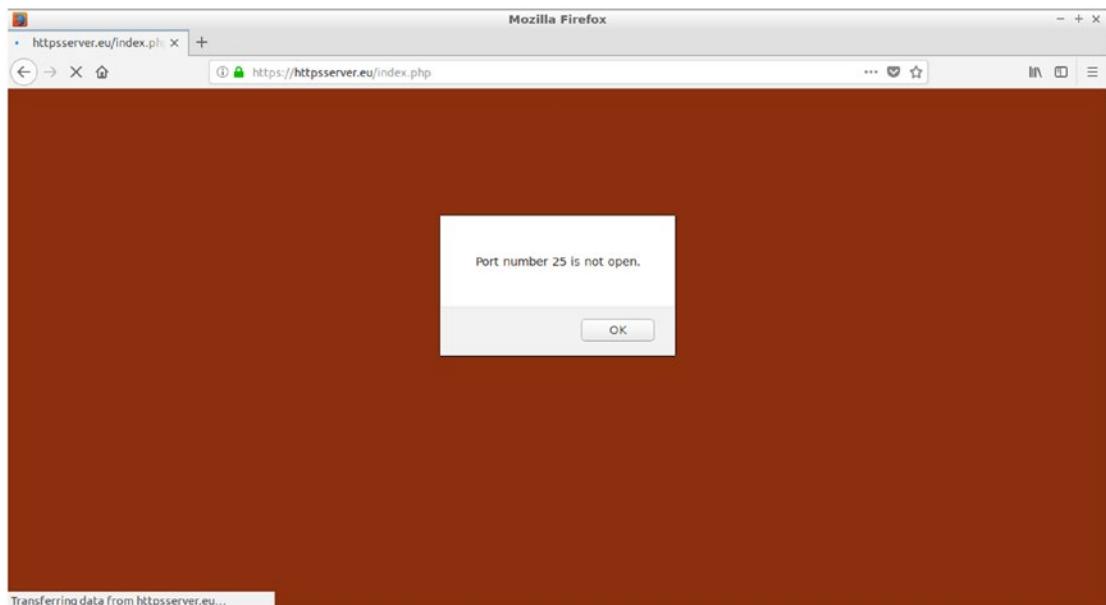


Figure 10-14. The port check output indicates a closed port for port 25

As indicated in the message, “Port number 25 is not open.”

Next, you’ll test the online server from a computer external to your LAN to ensure it functions as expected.

Testing the Online Port Test Site Remotely

To test the port check tool remotely from a remote workstation, you cannot use online web page test services like webpagetest.org like in the previous chapters because this time the remote system needs to provide some form data. Instead, use an online service that allows you to interact with the remote browser such as browserling.com. In your browser, enter the following URL:

<https://www.browserling.com>

In the browser’s drop-down list, select one of your favorite browsers, for instance Chrome, as viewed in Figure 10-15.

CHAPTER 10 RUNNING ONLINE SERVICES USING PHP SOCKETS

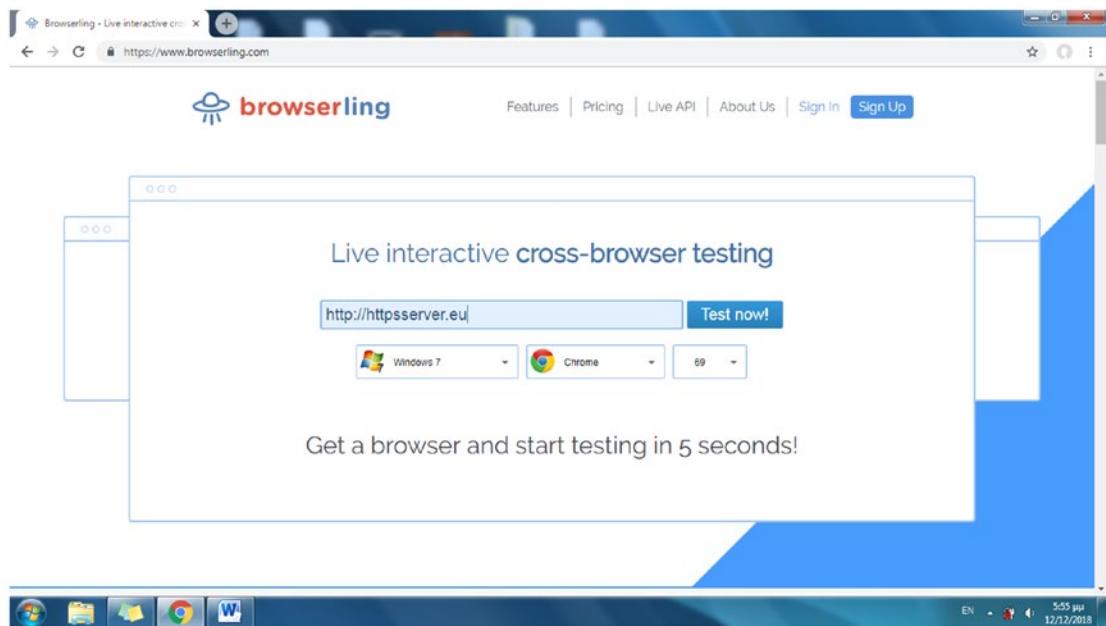


Figure 10-15. The [browserling.com](https://www.browserling.com) online service home page

In the textbox, enter the URL of your site, e.g., `httpsserver.eu`, and click the “Test now!” button. In a few seconds, the remote browser loads the web page of your site, as indicated in Figure 10-16.

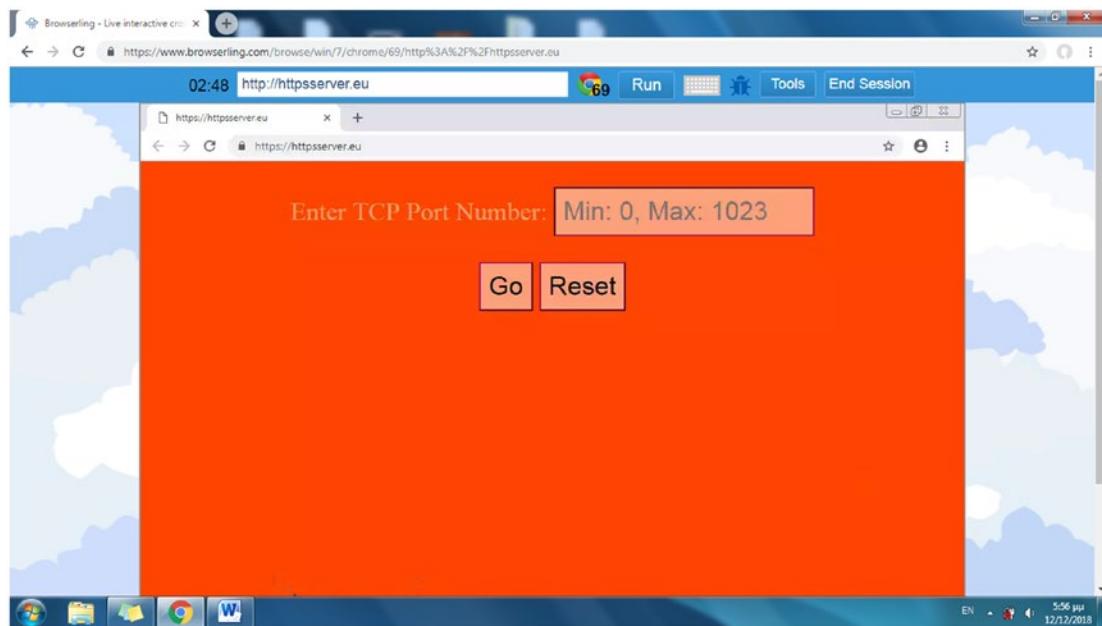


Figure 10-16. The open port check tool site as viewed from the remote browser

Test port 443 remotely first by using the online service's browser. Enter **443** in the number field of the form and click the Go button. Your web server will attempt to connect to the remote client on the specified port number, and accordingly a message will appear in the dialog of the evaluated PHP source code. As indicated in Figure 10-17, the connection succeeded, and a message appears to indicate that port 443 is open.

CHAPTER 10 RUNNING ONLINE SERVICES USING PHP SOCKETS

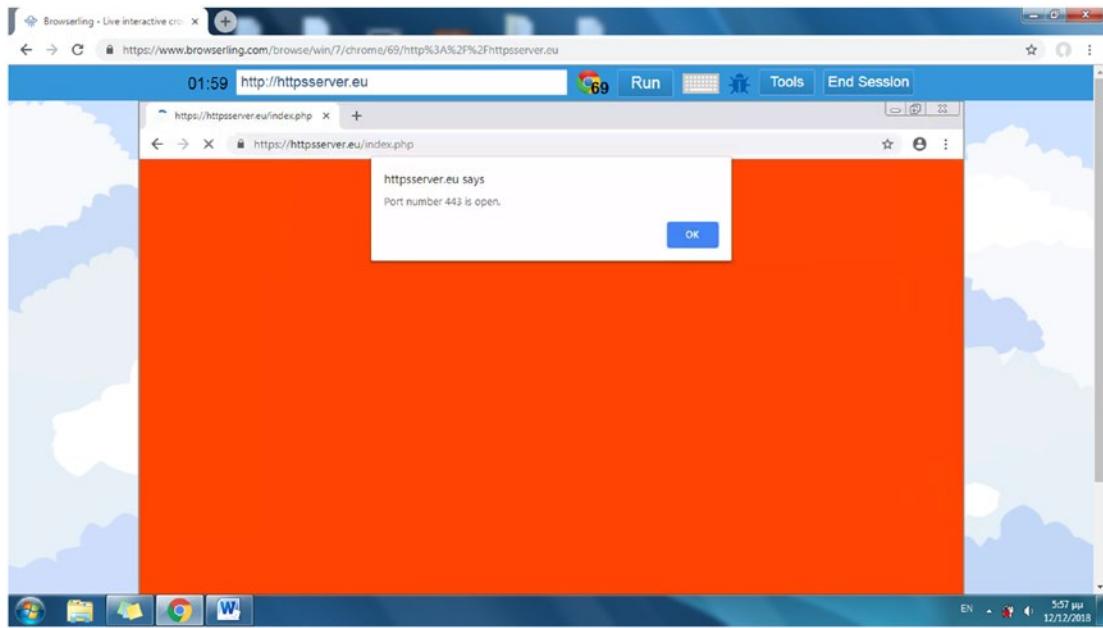


Figure 10-17. Verifying that port 443 is open on the remote system

Test for another port, e.g., port 80. This time, as viewed in Figure 10-18, the port is closed.

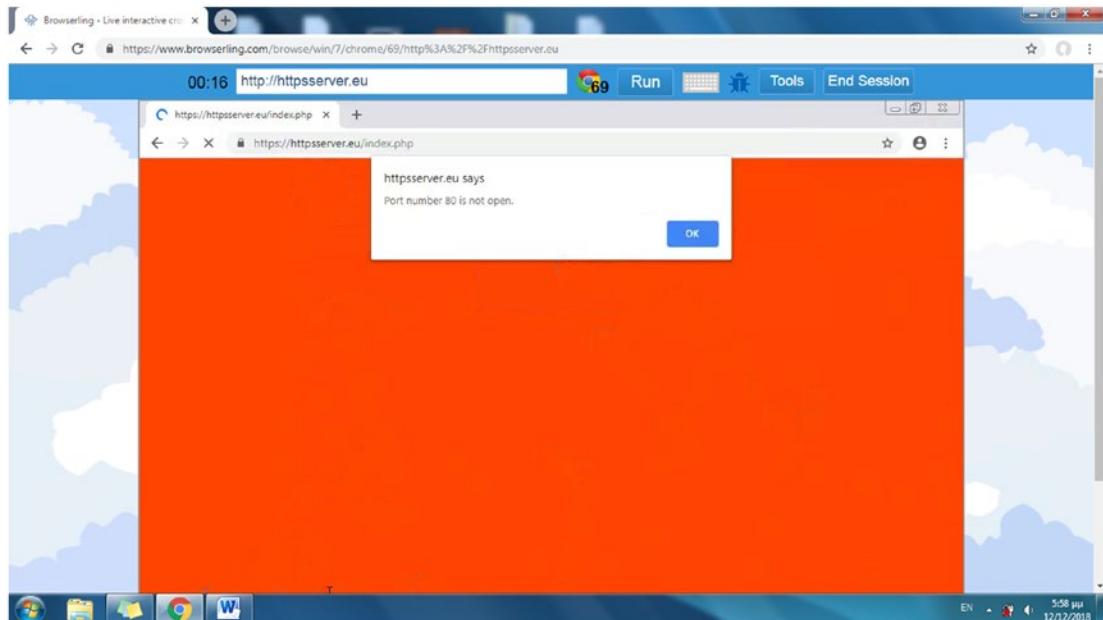


Figure 10-18. Verifying that port 80 is closed on the remote system

A Second Version of the Open Port Check Tool Source Code

Now try the following version of `index.php`. Because it submits data to itself, it does not require a second PHP file (`ports.php`) for processing the data sent.

```
<!DOCTYPE html>
<html>
<head>
<style>
body{
background-color:orangered;
}
input[type=number]{
    width: 300px;
}
p{
text-align:center;
font-size:32px;
color:lightsalmon;
}
input{
border-color:#9F0251;
font-size:32px;
color:black;
padding:10px;
background-color:lightsalmon;
}
</style>
</head>
```

CHAPTER 10 RUNNING ONLINE SERVICES USING PHP SOCKETS

```
<body>

<?php
if (isset($_POST['s1'])) {
$host = $_SERVER['REMOTE_ADDR'];

if (isset($_POST['s1'])){
    $port = intval($_POST["port"]);
    if (isset($port)) {

        $socket = socket_create(AF_INET , SOCK_STREAM , SOL_TCP);
        $connection = socket_connect($socket , $host , $port);

        echo '
<script>
alert("Port number ';
        echo $port;
        echo ' is';

        if (!$connection){
            echo ' not';
        }

        echo ' open.');
    }

    echo '</script>';
}

}
}

?>

<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>">
<p>
<label for="port">Enter TCP Port Number:</label>
<input name="port" type="number" placeholder="Min: 0, Max: 1023" min="0"
max="1023" maxlength="25">
</p>
```

```

<p>
<input type="submit" name="s1" value="Go">
<input type="reset" value="Reset">
</p>
</form>
</div>

</body>
</html>

```

In this version, the form submits data to the current PHP file, indicated by the global PHP variable `$_SERVER["PHP_SELF"]`.

```
<form method="post" action=<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>>
```

The source code that accepts the form values is the first PHP block. A socket connection is then issued with the user's IP address on the port to be checked. Also, a JavaScript block is created by the PHP code and is executed by the user's browser, creating a windowed message with the `alert()` method about the state of the port on the user's computer.

In this project, the web server connected to a client system with sockets. In the project in the following section, the server will connect with sockets to a server system. This time it's a server that implements another Internet service, namely, a QOTD service.

Creating an Online Service Displaying QOTD Messages

With PHP socket programming, the web server can connect to any open port number on a given computer and therefore connect to other Internet services this time as a client. Earlier in the chapter, you learned how to connect the web server through PHP sockets to the local terminal PHP server. In this project, you will connect the web server to a remote Internet service. Other than the Web, there are a large number of Internet services such as e-mail, File Transfer Protocol (FTP), Telnet, and DNS that follow the client-server model of a server that accepts a connection on a specific port number using the message formats defined by the corresponding protocol. To interface with those services from the Web, you can't rely on a client-side language like JavaScript, because

a nonsystem language cannot support sockets and make a direct connection to one of these services. This can be performed, however, by the web server using PHP on behalf of the web client.

The service the web server connects to in this project will be a quote of the day (QOTD) service. This service implements a simple protocol, defined in RFC 865, to deliver daily quotes. RFC 865 recommends that the quotes should be limited to the ASCII printable characters, spaces, and newlines; their length should be less than 512 characters; and the port used for both TCP and UDP should be port 17.

Hint! You can read RFC 865 at <https://tools.ietf.org/html/rfc865>.

The user can choose one of the two available QOTD servers to receive its quote of the day. This happens through an Ajax request, which sets the values of the PHP code on the server side.

- cygnus-x.net
- dixmmx.net

To test the QOTD servers before implementing the PHP server-side sockets program, use the following Telnet command at the Linux terminal:

```
$ telnet cygnus-x.net 17
```

Telnet connects to the cygnus-x.net QOTD server on port 17 and prints a quote of the day. The command's output is as follows:

```
Trying 66.229.218.72...
Connected to cygnus-x.net.
Escape character is '^]'.
```

"Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time."

- Thomas Alva Edison

Connection closed by foreign host.

Next, you will create the source code that utilizes the Ajax technique to request the QOTD message without reloading the initial web page with the form.

The Source Code for the QOTD Project

Create a file called `index2.php` in the document root with a text editor.

```
$ sudo gedit /var/www/html/sockets/index2.php
```

Enter the following source code and save the file:

```
<!DOCTYPE html>
<html>
<head>
<title>Quote of the Day</title>
<style>
body {
background-color:lime;
}
select{
padding:20px;
font-size:32px;
color:navy;
background-color:yellow;
}
#demo{
color:deepskyblue;
padding:20px;
font-size:24px;
}
p{
text-align:center;
}
</style>
</head>
```

CHAPTER 10 RUNNING ONLINE SERVICES USING PHP SOCKETS

```
<body>
<p>
<select name="sel1" id="sel1" onchange="sendRequest()">
    <option value="-1">Select a QOTD server</option>
    <option value="1">QOTD server 1</option>
    <option value="2">QOTD server 2</option>
</select>
</p>
<script>
function sendRequest() {
if ((document.getElementById("sel1").value == 1) || (document.
getElementById("sel1").value == 2)){
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            document.getElementById("demo").innerHTML = xhr.responseText;
        }
    };
    xhr.open("POST", "qotd.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    var data = document.getElementById("sel1").value;
    xhr.send(`var1=${data}`);
    document.getElementById("demo").style.backgroundColor = "yellow";
} else{
    document.getElementById("demo").style.backgroundColor = "lime";
    document.getElementById("demo").innerHTML = "";
}
}
</script>
<div>
<p id="demo"></p>
</div>
</body>
</html>
```

The source code creates the web page displayed in Figure 10-19.

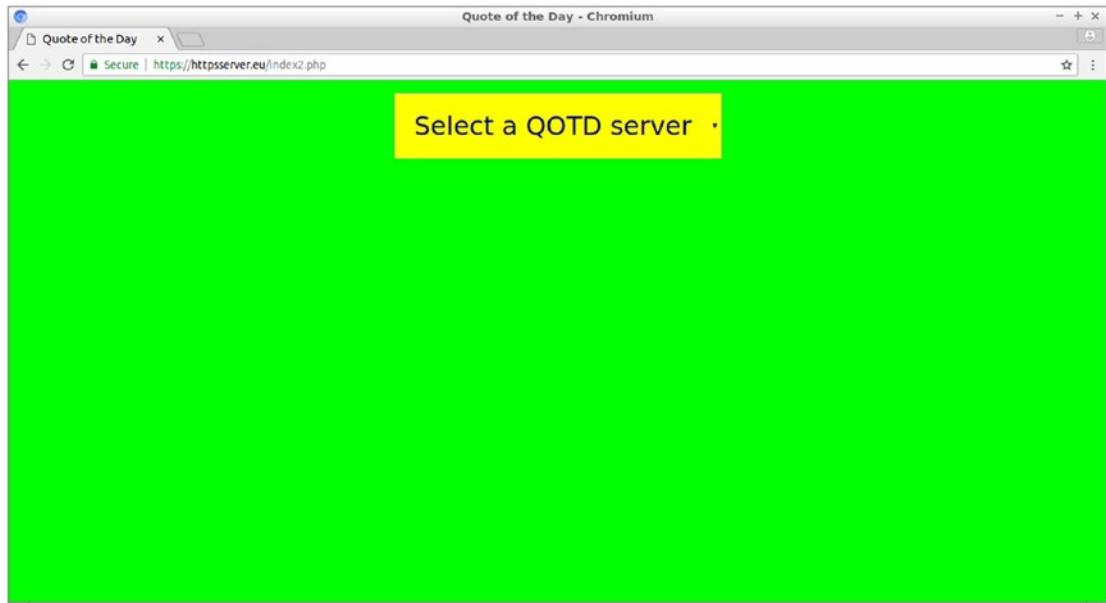


Figure 10-19. The home page of the QOTD online service

To view the page, use the following URL:

<https://httpsserver.eu/index2.php>

The source code creates a drop-down list with two active elements: one submitting the value 1, which for the PHP code on the web server corresponds to the domain name of the first QOTD server, cygnus-x.net, and the other submitting the value 2, which corresponds to the domain name of the second QOTD server, djkxmmx.net. The list responds to the onchange JavaScript event, which is activated each time a new selection is made, handled by the function sendRequest().

```
<select name="sel1" id="sel1" onchange="sendRequest()">
```

The function sendRequest() creates xhr, an XMLHttpRequest instance, that submits a POST request to the web server by implementing the Ajax mechanism. The data is sent behind the scenes without reloading the web page, and only the specified HTML element is updated, which for this example is a paragraph with an id value of demo.

```
if (xhr.readyState == 4 && xhr.status == 200) {  
    document.getElementById("demo").innerHTML = xhr.responseText;
```

In the second parameter of the `open()` method, belonging to the `XMLHttpRequest` instance, the `qotd.php` page was set as the destination program to receive the submitted data.

```
xhr.open("POST", "qotd.php", true);
```

The drop-down list selection is then sent as the value of variable `var1`.

```
var data = document.getElementById("sel1").value;
xhr.send(`var1=${data}`);
```

Notice that the function `sendRequest()` submits the data only if it is the currently selected element with value 1 or the element with value 2. There is also a third, dummy element to indicate the list functionality. It prints “Select a QOTD server” and corresponds to value -1. If this element is selected after a valid selection that prints a quote, `sendRequest()` is also called, and the `else` statement executes to clear the quote and reset its background color from yellow to lime.

Create a file called `qotd.php` using the following command at the Linux terminal:

```
$ sudo gedit /var/www/html/sockets/qotd.php
```

Enter the following source code and save the file:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<?php
if($_POST['var1'] == "1") {
    $host = gethostbyname("cygnus-x.net");
} elseif($_POST['var1'] == "2") {
    $host = gethostbyname("djxmmx.net");
}
$port = 17;
$socket = socket_create(AF_INET , SOCK_STREAM , SOL_TCP);
$connection = socket_connect($socket , $host , $port);
$result = socket_read ($socket, 512);
```

```

socket_close($socket);
echo $result;
?>
</body>
</html>

```

The file `qotd.php` is the one assigned in the `index2.php` source code as the server-side program to receive the data sent with the Ajax mechanism. The PHP source code block checks the value submitted with Ajax. If the value sent is 1, a connection with the QOTD server with the domain name `cygnus-x.net` will be created. Also, for a value of 2, a connection to the QOTD server `djxmmx.net` is issued. The function `gethostbyname()` returns the IP address corresponding to the given QOTD server FQDN, passed as the function's argument. With this IP address and port 17, the port used for the QOTD service, a TCP socket is created.

```

$socket = socket_create(AF_INET , SOCK_STREAM , SOL_TCP);
$connection = socket_connect($socket , $host , $port);

```

For the QOTD service, the client just reads the data (the quote) sent by the QOTD server, which according to RFC 865 has a maximum length of 512 bytes.

```
$result = socket_read ($socket, 512);
```

The text read is assigned to the `$result` variable, whose value `qotd.php` is sent to `index2.php` with the `echo` command.

```
echo $result;
```

In the `index2.php` source code, the `$result` value (the quote) is received by the `xhr.responseText` attribute via the Ajax mechanism and is printed as the content of the paragraph `demo`, thus updating only a small portion of the web page.

Testing the QOTD Site

Next, you'll test the QOTD site by using the URL of the site. For example, use the following in the address bar of your browser:

`https://httpsserver.eu/index2.php`

CHAPTER 10 RUNNING ONLINE SERVICES USING PHP SOCKETS

In Figure 10-20, the user has selected QOTD server 1, and the quote of the cygnus-x.net QOTD server is displayed in the web page.



Figure 10-20. Displaying the quote of the first QOTD server

In Figure 10-21, the user has selected QOTD server 2, and the quote of the dixmmx.net QOTD server is displayed on the web page.



Figure 10-21. Displaying the quote of the second QOTD server

Select next the dummy option “Select a QOTD server.” As displayed in Figure 10-22, the quote is deleted, and its background color is reset from yellow to lime.



Figure 10-22. By selecting the dummy option, the web page is reset to its initial state

In the final section, you'll separate the last two sites you created by using a different fully qualified domain name for each one.

Using Different FQDNs for Your Sites

In this section, you'll implement two different fully qualified domain names for your domain, e.g., `httpsserver.eu`, with each one serving a different site. For instance, you can use `sockets.httpsserver.eu` for the site that runs the open port check tool and `qotd.httpsserver.eu` for the site that interfaces with a QOTD server.

You may at this point want to separate the files of the two sites. Use the following:

```
$ cd /var/www/html/sockets
$ sudo mkdir ports qotd
$ sudo cp index.php ports
$ sudo cp index2.php qotd.php qotd
```

In the `default-ssl.conf` Apache configuration file, add two new name-based virtual hosts, with each one serving a FQDN.

```
$ sudo gedit /etc/apache2/sites-available/default-ssl.conf
```

Add the following lines at the end of the configuration file to create the two vhosts and save the file:

```
<VirtualHost *:443>
    ServerName ports.httpsserver.eu
    DocumentRoot "/var/www/html/sockets/ports"
</VirtualHost>

<VirtualHost *:443>
    ServerName qotd.httpsserver.eu
    DocumentRoot "/var/www/html/sockets/qotd"
    DirectoryIndex index2.php
</VirtualHost>
```

Reload the configuration file to enable the new settings.

```
$ sudo service apache2 force-reload
```

Test the first vhost using the following in the address bar of your browser:

<https://ports.httpsserver.eu>

The web page loading is displayed, as shown in Figure 10-23.

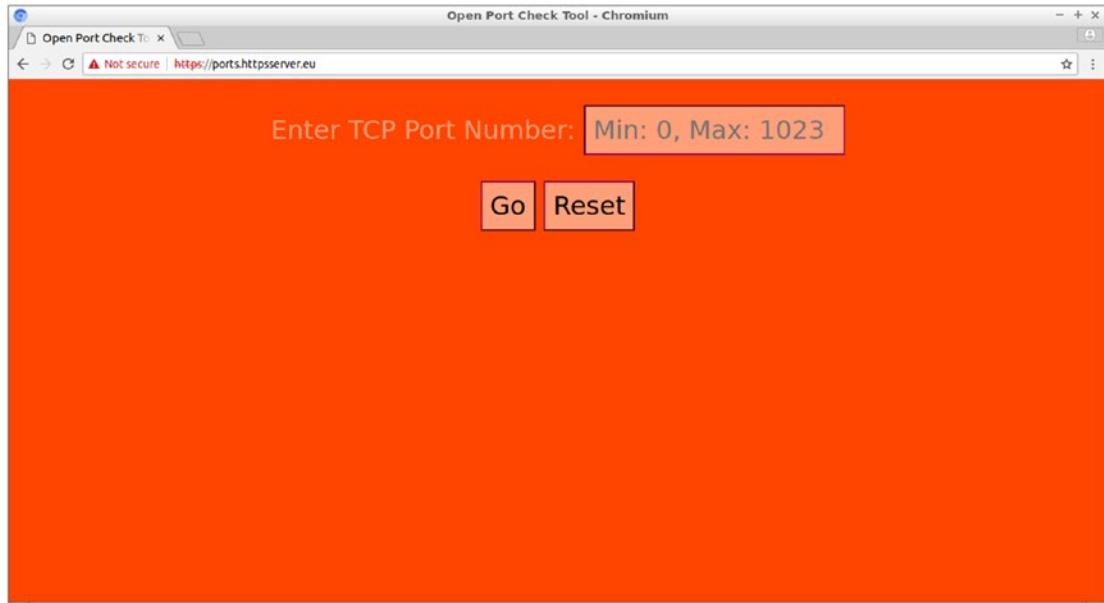


Figure 10-23. The FQDN `ports.httpsserver.eu` is used for the open port check site

Test the second vhost using the following in the address bar of your browser:

<https://qotd.httpsserver.eu>

The web page loads, as shown in Figure 10-24.

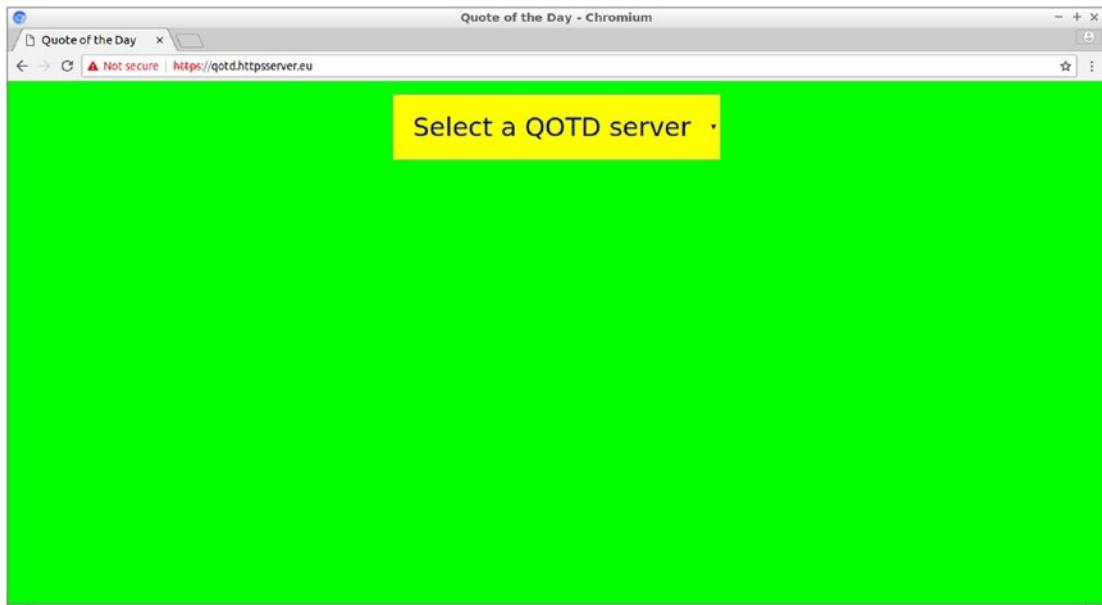


Figure 10-24. The FQDN `qotd.httpsserver.eu` is used for the QOTD site

For both sites, the browsers display the insecure site warning, while the SSL certificate verifies the domain names.

`https://httpsserver.eu`
and also the same for the hostname `www`:
`https://www.httpsserver.eu`

The other two hostnames for the `httpsserver` domain, `ports` and `qotd`, are not supported with the SSL certificate. To use one SSL certificate for any FQDN for your domain name, you have to purchase a wildcard SSL certificate, which usually costs more than a simple certificate.

Summary

In this chapter, you installed `ddclient` to update your router's public IP address to the DDNS service provider. You then used sockets programming to allow the web server to connect to a server listening to a non-HTTP port and interface with other services. Finally, you created an online open port check tool and a site that interfaces with a QOTD site.

APPENDIX

Exchanging Variables Between JavaScript and PHP

In this book, the JavaScript language is used for the client side of the HTTP client-server model, and PHP is used for the server side. In some cases, the two languages need to exchange variables, and there are a number of ways to do that. The examples in this appendix display the main ways to exchange variable values between JavaScript and PHP.

Example 1: Passing Variables from PHP to JavaScript Using the echo Command

To allow PHP to set JavaScript variables, you can simply inject PHP echo commands into the script in the PHP file. These commands are then evaluated on the server side and create the JavaScript code that is executed locally at the browser. The JavaScript code that appears within the PHP code has to be included in an echo message; otherwise, it is omitted when the PHP code is evaluated. In the following code, which shows the test1.php PHP file, the PHP variable \$p holds the current IP address of the client, and it is assigned to the JavaScript variable j. The value of variable j is then printed on the web page.

```
<?php  
$p = $_SERVER['REMOTE_ADDR'];  
echo '  
<script>
```

```

var j = "'";
echo $p;
echo '';
document.write(j);
</script>';
?>
```

The previous PHP code evaluates, for the IP address 94.69.186.8, to the following code:

```

<script>

var j = "94.69.186.8";
document.write(j);
</script>
```

Example 2: Passing Variables from JavaScript to PHP Using the location Object

PHP works on the server side, in other words, at the web server. Therefore, for JavaScript to assign values to PHP variables, the values must be submitted to the web server.

The following HTML file, called `index.html`, includes the JavaScript source code. The `window.location` object (or simply `location`) is used to get information about the URL of the current web page (`document`); it also redirects the browser to another web page. In this example, the `location.href` property sets the URL that the current page redirects to. The new page is `test2.php`, and the `p1` and `p2` values are included in the query string.

```

<!DOCTYPE html>
<html>
<head></head>
<body>
<script>
var j1 = "hello";
var j2 = "world";
location.href = "test2.php?p1=" + j1 + "&p2=" + j2;
</script>
```

```
</body>
</html>
```

The source code of `test2.php` is as follows:

```
<?php
if (isset($_GET["p1"]) && isset($_GET["p2"])) {
    $p3 = $_GET["p1"] . " " . $_GET["p2"];
    echo $p3;
}
?>
```

The request issued by JavaScript passes the values of `p1` and `p2`, which can be retrieved by PHP as `$_GET["p1"]` and `$_GET["p2"]`, respectively. The echo message printed to the web page when PHP resolves is `hello world`.

Example 3: Passing Variables from JavaScript to PHP with HTML Form Submission

In the previous example, the HTTP method GET is only for submitting JavaScript data to PHP. With HTML forms, you can utilize either GET or POST. Consider the following example of `index.php`, where JavaScript fills the value of a hidden element and submits it with the form.

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<form id="f1" method="POST" action="test3.php">
<input type="hidden" id="p1" name="p1" value="">
</form>
<script>
var j1= "hello world";
document.getElementById('p1').value = j1;
document.getElementById("f1").submit();
</script>
</body>
</html>
```

The form is submitted from the JavaScript source code with the `submit()` method of the form object.

```
document.getElementById("f1").submit();
```

The source code of `test3.php`, the program that receives the form data, is as follows:

```
<?php
if (isset($_POST["p1"])) {
    $p2 = $_POST["p1"];
    echo $p2;
}
?>
```

The value of the JavaScript variable `j1` is submitted using the form and is assigned then from the PHP source code to the PHP variable `$p2`. The value of `$p2` is printed to the user's web page with the `echo` command.

Example 4: Passing Variables from JavaScript to PHP and Back with Ajax

Another way to pass variables from JavaScript to PHP is to use the Ajax mechanism.

Ajax is not a programming language; rather, it is a technique that involves the JavaScript XMLHttpRequest object, which creates an asynchronous connection to the web server in the background. *Asynchronous* means that the script sends a request to the web server and continues its execution without waiting for a reply. For the asynchronous request, a callback function is set that handles the returned data from the server when it arrives. Initially Ajax was implemented with the XML language, which handled all the requests to the server.

Another way to use the Ajax method is with the DOM model. JavaScript uses the DOM to provide positions to the web page for where to place the server's reply, such as at a paragraph with a specific ID. These positions are the only parts of the web page that are updated, instead of reloading the whole page. DOM objects provide the positions of the data that will be submitted to the web server.

CSS also plays an important part in the Ajax method because the returned data can be formatted with CSS to retain the styling of the web page that submits the Ajax request.

`XMLHttpRequest()` is the constructor method that is used to create `XMLHttpRequest` instances (that is, objects) for the `XMLHttpRequest` class. The objects are used to connect to a web server, dispatching an HTTP request and receiving the reply. The `XMLHttpRequest` class defines methods (routines) and properties (variables) for the request handling. The following are the `XMLHttpRequest` class methods:

- `XMLHttpRequest()` is the constructor of the `XMLHttpRequest` object.
- `abort()` cancels the current request.
- `getAllResponseHeaders()` returns the headers of the response message.
- `getResponseHeader()` returns a specific response header.
- `open()` initializes the request by setting its attributes.
- `send()` sends the request to the web server.
- `setRequestHeader()` sets the headers of the request.

The following are the `XMLHttpRequest` class properties:

- `onreadystatechange` defines the function that will be called when the `readyState` property changes.
 - 0: Request not initialized
 - 1: Server connection established
 - 2: Request received
 - 3: Processing request
 - 4: Request finished and response ready
- `responseText` holds the response data as a string.
- `responseXML` holds the response data as XML data.
- `status` holds the status of the request reply as an HTTP status code. Examples of HTTP status codes are the following:
 - 200, 301, 400, 403, 404, and 501
- `statusText` holds the corresponding text of the HTTP status code. Here are some examples:

APPENDIX EXCHANGING VARIABLES BETWEEN JAVASCRIPT AND PHP

OK for status code 200
Moved Permanently for status code 301
Bad Request for status code 400
Forbidden 403
Not Found for status code 404
Not Implemented for status code 501

The following file, test4.html, uses the POST method to submit data to the program test4.php with the Ajax mechanism. The JavaScript function sendRequest() runs when the web page is loaded.

```
window.onload = sendRequest;
```

In this example, the JavaScript variable j1 is sent via Ajax as value1 to the PHP source code and is received in the global variable `$_POST["value1"]`. This is echoed to the web page returned to the browser, and the message hello world, the initial value of j1, is printed to the predefined paragraph element, p1.

```
<!DOCTYPE html>
<html>
<head>
<script>
function sendRequest() {
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            document.getElementById("p1").innerHTML = xhr.responseText;
        }
    }
    xhr.open("POST", "test4.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    var j1 = "hello world";
    xhr.send(`value1=${j1}`);
}
window.onload = sendRequest;
</script>
</head>
```

```
<body>
<p id="p1"></p>
</body>
</html>
```

The source code for test4.php is as follows:

```
<?php
if (isset($_POST["value1"])) {
    $p2 = $_POST["value1"];
    echo $p2;
}
?>
```

The equivalent version for the HTTP method GET would be as follows:

```
<!DOCTYPE html>
<html>
<head>
<script>
function sendRequest() {
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            document.getElementById("p1").innerHTML = xhr.responseText;
        }
    }
    var j1 = "hello world";
    xhr.open("GET", `test4b.php?value1=${j1}`, true);
    xhr.send();
}
window.onload = sendRequest;
</script>
</head>
<body>
<p id="p1"></p>
</body>
</html>
```

The source code for the action file, test4b.php, is as follows:

```
<?php
if (isset($_GET["value1"])) {
    $p2 = $_GET["value1"];
    echo $p2;
}
?>
```

Example 5: Passing Variables from JavaScript to PHP with Cookies

Another option for exchanging variables between JavaScript and PHP is through cookies. Cookies can be set and retrieved from JavaScript and also from PHP. In addition, a cookie set with JavaScript can be retrieved by PHP and vice versa.

The following source code, used in a PHP file such as test5.php, uses JavaScript to set the value of the cookie message in the head section of the HTML source code. Then in the body section of the HTML source code, the PHP code retrieves the cookie's value and prints this message: Cookie 'message' has value hello.

```
<!DOCTYPE html>
<html>
<head>
<script>

function setCookie(name, value, expires, path, domain, secure){
    cookieString = name + "=" + encodeURIComponent(value) + "; ";
    if(expires){
        expires = setExpiration(expires);
        cookieString += "expires=" + expires + "; ";
    }
    if(path){
        cookieString += "path=" + path + "; ";
    }
    if(domain){
        cookieString += "domain=" + domain + "; ";
    }
}
```

```

if(secure){
    cookieString += "secure; ";
}
document.cookie = cookieString;
}

function setExpiration(days){
    var today = new Date();
    var expires = new Date(today.getTime() + days * 24 * 60 * 60 * 1000);
    return expires. toUTCString();
}

window.onload = setCookie("message", "hello", 1);
</script>
</head>

<body>
<?php
$c_name = "message";
if(!isset($_COOKIE[$c_name])) {
    echo "Cookie named '" . $c_name . "' is not set";
} else {
    echo "Cookie '" . $c_name . "' has value " . $_COOKIE[$c_name];
}
?>
</body>
</html>

```

In the JavaScript source code, the cookie string is formed with the required information from these properties: name, value, expires, path, domain, and secure. The function `encodeURIComponent()` is used to encode the following special characters with their UTF encoded versions:

, / ? : @ & = + \$ #

For the expiration date, the current date is calculated in milliseconds via `today.getTime()`, and the days the cookie is valid, also in milliseconds, is calculated and added to the current date to form the Unix timestamp. The `toUTCString()` method converts this date to a string according to Coordinated Universal Time (UTC) standard. Here's an example:

Mon, 29 Oct 2018 06:35:38 GMT

The cookie is set when the web page is loaded with the following JavaScript command:

```
window.onload = setCookie("message", "hello", 1);
```

The PHP code retrieves the cookie in the global variable `$_COOKIE[$c_name]`, where `$c_name` is the message in the current example.

Example 6: Passing Variables from PHP to JavaScript with Cookies

The following PHP code creates a cookie named `message`, assigns the value `hello` to it, and specifies that the cookie expires in one hour (60 minutes * 60 seconds) from now. The function `time()` returns an integer containing the current time as a Unix timestamp.

```
<?php
$c_name = "message";
$c_value = "hello";
setcookie($c_name, $c_value, time() + 3600);
?>
```

In the following example, which is included in a PHP file called `test6.php`, the PHP code sets the value of the cookie named `message` to `hello`, and then JavaScript retrieves this value and prints it on the web page:

```
<?php
$c_name = "message";
$c_value = "hello";
setcookie($c_name, $c_value, time() + 3600);
?>
```

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<script>
function retrieve_cookie(w){
    var c_value = "";
    var dUC = decodeURIComponent(document.cookie);
    cArray = new Array();
    cArray = dUC.split(';');
    for(i = 0; i < cArray.length; i++){
        cValues = new Array();
        cValues = cArray[i].split('=');
        if(cValues[0] == w){
            c_value = decodeURIComponent(cValues[1]);
        }
    }
    return c_value;
}
var cookie_name = "message";
var val = retrieve_cookie(cookie_name);
if(val) {
    document.write('Cookie with name "' + cookie_name + '" has value ' + '''+ val + ''');
} else {
    document.write('Cookie with name "' + cookie_name + '" does not exist');
}
</script>
</body>
</html>
```

The function `retrieve_cookie()` returns the value for the cookie name passed as a parameter. This function splits `document.cookie`, a string containing a semicolon-separated list of all cookies of the site, and creates an array where each element includes

APPENDIX EXCHANGING VARIABLES BETWEEN JAVASCRIPT AND PHP

information for a single site cookie. Consider, for instance, the case where two cookies were set with the following PHP code:

```
<?php  
$c_name = "message";  
$c_value = "hello";  
setcookie($c_name, $c_value, time() + 3600);  
$c_name = "message2";  
$c_value = "hello2";  
setcookie($c_name, $c_value, time() + 3600);  
?>
```

The document.cookie string in this case would be as follows:

```
message=hello; message2=hello2
```

To print the value of document.cookie, use the following command in retrieve_cookie():

```
document.write(document.cookie);
```

In a for loop in retrieve_cookie(), each name-value pair is split across an equal sign, and a new array is created with the first element as the name and the second element as the value. If the given name that was passed as a parameter to the function matches the first element, the second element (the value) is returned. Notice that encodeURIComponent() is used to ensure that document.cookie does not contain any commas, semicolons, or whitespace.

Index

A, B

a2enmod (apache2 enable module)
 command, 102
Ajax, 474, 488
alert() method, 52, 473
alter table statement, 230
Amazon Standard Identification
 Number (ASIN), 241
Animated PNG image
 ezgif.com web page
 color transparency, 213
 display, 212
 effects tool, 212
 process, 213
 frames, 209
 getElementById(), 216
 pattern, 209
 0.png file, 209
 runs, 216
 six PNG files, 209
 source code, 214–215
 sub() runs, 216
 uploaded
 images, 211
 web page, 211
 zipped file, 210
Apache configuration,
 139–141, 336–338
apache_note() function, 103
Apache Vhosts, 94–96

Apache web server
 administrative features, 1
 Apache process, 16–18
 apache2 process, 2
 CA certificate, 392
 command ps output lists, 2
 directory index, Ubuntu, 3
 download and install, 2
 ifconfig command, 7, 8
 LAN, 7–9
 Linux firewall, 14, 16
 logging, 36
 new directories and web pages, 4–7
 PHP engine, 37
 static private IP address, 9–14
 testing, IP address, 8
 vhosts (*see* Virtual hosts (vhosts))
appDataLayer struct, 251, 252
Application programming
 interface (API), 449
Asymmetric cryptography, 328
ATCP port check site
 home page, 460
 index.php, source code, 460–462
 system ports, 462
 warning message, 462
 online open port check tool, 459
 online port test site, 467
 home page, 468
 port verification, 470
 remote browser, 469

INDEX

ATCP port check site (*cont.*)

- port 443 testing, 465
- port.php, source code, 463–464
 - port 25 testing, 466–467

auto_increment keyword, 226

awk tool, 257

C

cat command, 257

ceil() function, 316

Certificate authorities (CAs), 328

Certificate-signing request (CSR), 329, 388

chmod (change mode) command, 256

Client-server communication, 330

Client-side *vs.* server-side programs

- JavaScript/PHP addition web page
 - action attribute, 65
 - addition2.html, 69–71
 - addition.php, 66
 - addition2.php, 72, 74
 - HTML file, 62–64
 - JavaScript calculator, 66
 - Number() function, 64
 - onclick event, 64
 - program.html, 62
 - program2.html, 67–69, 74
 - result, 66

\$cnt variable, 291

Command-Line PHP client, 453–454

Command-Line PHP socket server code, 450–452

Common Gateway Interface (CGI), 39

\$connection variable, 464

Cookies, 353–354, 492

create_entry() arguments, 419

CREATE TABLE SQL command, 421

cron, database update

- Bash script, 283
- crontab, 280–281
 - gedit, 283
 - HTML files, 284
 - MySQL server, 282
 - web pages, 284–285

crontab command, 280

Cross-site scripting (XSS), 81

D

Database, search-enabled site

- Bash shell script, 275
 - chmod command, 278
 - do...done commands, 277
 - info database, 279
 - MySQL server, 277–279

creation

- info database, 273
- MySQL client, 275
- MySQL server, 274

Data layer, 252

ddclient daemon, 448

delete SQL statement, 230

describe SQL statement, 226

DocumentRoot directive, 397

Domain name IP address, ddclient

- Apache/Lighttpd, 443
 - current IP address, 445
 - ddclient installation, 446
 - DDNS provider's site, 447–448
 - Linux terminal, 444
 - new public IP address, 449

Domain name registration

- CA SSL certificate, 387
 - CSR, 388
 - directory, 389
 - OpenSSL, 389

- newly registered domain name, 387
- search availability, 386
- TLD, 386
- Domain Name System (DNS) service, 125
- drop table statement, 226
- Dynamic content web site
 - pagination (*see* Pagination)
 - project design, 271
 - search-enabled site, 269–271
 - submit buttons, 318, 320
 - testing
 - directory index, 294
 - echo \$items command, 298
 - explode() function, 300
 - \$items value, 299
 - multiple keywords, 296–297
 - preg_split() function, exclude, 301
 - query, 295–296, 299
 - trim() function, exclude, 300
 - web content samples, 272–273
- Dynamic DNS (DDNS) service
 - account creation, 130
 - no-IP, register, 126
 - router configuration, 131–132
 - updation, 125
- Dynamic Host Configuration Protocol (DHCP), 9, 10
- Dynamic IP address service, 445

- E**
- echo command, 52, 61, 73, 115, 265, 454
- e-mail account confirmation, 128
- equijoin, 435

- F**
- Favorite icon (favicon), 156–163
- fgets() function, 152

- G**
- GeoIP Apache module
 - a2enmod command, 102
 - apache2ctl-M command, 101
 - apache_note() function, 103
 - GeoIP.dat.gz, 103
 - geo2.php, 103
 - geoip_module, 101–102
 - mods-available, 102
 - test location, 105–106
 - webpagetest.org, 104–105
- GeoIPCity, 113
- GeoIP.dat.gz, 103
- geoip_module, 101–102
- getElementById() function, 216
- GET method, 54
- GET method, site implementation
 - amazon.co.uk site, 324–325
 - offset value, 323–324
 - POST references, 321

INDEX

GET method, site implementation (*cont.*)
 query string, 322–323
 test, 321
grep command, 257

H

header() function, 82, 437
Home page, dynamic content web site
 directory index, 286–288
 HTML files, 285
PHP
 echo command, 293–294
 like operator, 291
 MySQL server, 292–293
 search.php, 288–290
 PHP-MySQL interface, 286
Hostnames, 133, 135
htmlspecialchars() function, 81
HTTP alternate port, 89

HTTP cookies
 browser tools, 365–366
 display details, 361
 e-commerce site, 352
 Lighttpd configuration file, 362–363
 Linux, 363
 and PHP sessions (*see* PHP sessions)
 remove, PHP, 355
 retrieve, PHP, 355
 settings, 353–354
 site creation, PHP, 355–357, 359–360
 web server programs, 353
Wireshark packet analyzer, 361, 364

I
info.php web page, 42
INSERT INTO command, 263–264

insert into SQL statement, 228
International Standard Book Number
 (ISBN), 243
Internet protocol, 343
intval() function, 464
IP-based virtual hosts
 address bar, 24
 a2ensite (apache2 enable site)
 command, 22
 configuration rules, 21
 DirectoryIndex directive, 21
 directory indexes, 22
 index1.html, 21, 24
 index2.html, 23
 Linux terminal, 22
 testing, 24–25
 start/end VirtualHost
 containers, 21
isset() function, 73, 82

J, K

JavaScript to PHP
 Ajax, 488–491
 cookies, 492–493
 HTTP method, 487–488
 location object, 486–487

L

Lighttpd
 animated PNG image (*see* Animated
 PNG image)
 basic configuration, 170–171
 change, document root, 173
 configuration, 334–336
 created online services (*see* Online
 services with Lighttpd)

enabling and disabling the directory, [174–175](#)
 error message, [176, 178](#)
 GUI, [165](#)
 installation, [165–166](#)
 log files access, [178–180](#)
 multiple client requests, [203–208](#)
 object-oriented programming
 principles, [170](#)
 PHP 7.0, [189](#)
 phpinfo() function, [189](#)
 server.modules directive, [170](#)
 specific IP address/hostname,
 [171–172](#)
 testing
 basic configuration, [166–167](#)
 directory index, [167–168](#)
 document root, [166](#)
 HTML source code, [167](#)
 index.lighttpd.html, [166](#)
 virtual hosts, [181–189](#)
 Lighttpd web server,
 CA certificate, [399](#)
 Linux firewall, [14, 16](#)
 Local area network (LAN)
 FQDN, [350](#)
 HTTPS connection testing,
 [347–349, 352](#)
 port 443, [346, 350–351](#)
 port locating, [346](#)
 router web interface, connection, [345](#)

M

match string() method, [145](#)
 method attribute, [462](#)
 mods-available, [102](#)
 mysql-client package, [219](#)

MySQL database
 alter table statement, [230](#)
 describe SQL statement, [232–233](#)
 download, [219](#)
 help command, [221, 222](#)
 insert SQL statement, [231](#)
 installation, [219–220](#)
 mysql client, [221](#)
 MySQL server, [223–224](#)
 select statement, [232](#)
 status, [220](#)
 table
 connection, [233–235](#)
 CREATE TABLE command, [227](#)
 describe SQL statement, [226–227](#)
 drop table statement, [226](#)
 foreign key, [226](#)
 primary key, [226](#)
 mysqli() function, [292](#)
 mysqli_fetch_assoc() function, [293](#)
 mysqli_free_result() function, [294](#)
 mysqli_query() function, [293, 419](#)
 mysqli PHP command, [288](#)
 MySQL-PHP interface, [269](#)
 mysql-server package, [219](#)

N

Name-based vhosts
 create, [31](#)
 000-default.conf, [30](#)
 /etc/hosts file, [30](#)
 HTML code, [32](#)
 index.html, [32](#)
 Linux terminal, [32](#)
 localhost, [30](#)
 reload, web server, [33](#)
 ServerName directive, [30](#)

INDEX

Name-based vhosts (*cont.*)

testing

 first name-based, 34

 second name-based, 34

 web page downloaded, 30

Network Address Translation (NAT)

 protocol, 89–90

nl2br() function, 84

no-IP

 account activation, 129

 Confirm Account button, 129

 e-mail account confirmation, 128

 home page, 127

 hostname, 128

Normalizing, 435

O

onclick event, 64

Online services with Lighttpd

 action file creation

 directory screenshot, 201

 exec() PHP function, 202

 PHP code, 198

 php.png image, 203

 save file, 197

 screenshot.php, 197

 shell_exec(), 202

 wkhtmltoimage, 199–200

 Xvfb server, 201

basic web pages, 190

create directory index, 195–196

home page, 191

index.php, 191

invalid URL, 194

screenshot.php, 193

URL inserts, 191

web server responses, 193

Online web service

 site creation, 138–139

 WHIOS testing, 154–155

open() method, 478

Open port check tool, 471–473

openssl req command, 388

P

Pagination

 forward or backward, 308

 HTML form, 307

 index.php version, 310–311

 limit clause, 305

 limit MySQL, 304–305

 PHP engine, 306

 search.php, 307, 311–315

 \$count, 315

 echo command, 316–317

 \$offset, 315

PCRE Regular Expression (PREG), 291

PHP

 CLI results, 45

 echo command, 51

 engine, 39

 epigrammatic uses, 40

 form validation (*see* Form validation)

 installation, 41–42

 interpreters, 43

 testing, 41–45

 user-defined variables, 57

 variables setting

 GET method, 54

 POST method, 58

 web page source code, 46

PHP Command-Line socket server, 452–453

PHP data objects (PDO), 288

- PHP-enabled web server, 39
 - `phpinfo()` function, 380
 - PHPSESSID cookie, 383
 - PHP sessions
 - cookie modification, 366
 - CSS section, 371
 - directory index, 368
 - `index.php`, 371–376
 - `page1.html`, 376–380
 - `page2.html`, 378–380
 - PHPSESSID cookie, 367
 - project experimentation, 380–383
 - session ID, 367
 - web page, 369, 371
 - PHP sockets, 449–450
 - command-line server, 455–456, 458
 - web server, configuration, 454–455
 - PHP to JavaScript
 - cookies, 494–495
 - `echo` command, 485–486
 - ping.eu WHOIS service, 138
 - Port-based virtual hosts
 - `a2ensite` command, 26
 - configuration rules, 27
 - create, 26
 - directory indexes, 27
 - document root directory, 28
 - HTML source code, 27
 - Linux command line, 27
 - listening ports, 26
 - testing, 29
 - Port forwarding
 - Apache vhosts, 94, 96
 - configuration web page, 92
 - form details, 92
 - LevelOne FBR-1161 ADSL router, web page, 93
 - routers, 90
 - settings, 93
 - web-based configuration interface, test router, 91
 - Port number, 94
 - POST method, 58, 321, 357, 359
 - `preg_split()` function, 291
 - Privacy Enhanced Mail (PEM), 333
 - Private IP address, 94
 - Public and private addresses, 89
 - Public IP address, 96
 - Public key infrastructure (PKI), 327, 328
- Q**
- Query results, two-colored table, 302–303
 - Quote of the day (QOTD) protocol, 450
 - online service creation, 473
 - ajax technique, 474
 - QOTD servers, 474
 - project, source code, 475–476
 - site testing, 479
 - source code
 - `cygnus-x.net`, 477
 - `djxmmx.net`, 477
 - online service homepage, 477
 - `qotd.php`, 478
 - TCP socket, 479
- R**
- Rainbow table, 424
 - Regular expressions, 145–146
 - Requests for comments (RFCs), 269
 - Root certificates, 328
 - route command, 10
 - Router configuration options, 91

S

Search-enabled site

- home page, 270

- RFC editor, 270

- rfc-editor.org site, 269

search.php, 288

Second-level domain (SLD), 385

Secure logging, project

- database creation, 419, 421

- design, 408–409

- home page, 409

- login web page, 425

- PHP to MySQL connection, testing, 421

- profile.php web page, 439

- user account, web page, 411

- user connection, testing, 437, 439

- user log out, 436–437

- user profile page, 432

Secure Sockets Layer (SSL), 328

- CA certificate, testing

 - certificate viewer, chromium, 404

 - chain of trust, 406

 - home web page, 402–403

 - online tool, 405

 - secure connection, 404

- configuration, Lighttpd

 - Apache, 336–338

 - Lighttpd, 334–336

select SQL statement, 229–230

Self-signed certificates

- openSSL

 - home directory, 330–333

 - HTTPS protocol, 334

- testing

 - chromium browser warning, 338–339

 - directory index, 340, 343

 - HTTPS protocol, 343–344

 - insecure connection, 341

IP address, 344

local host link, 339–340

security exception, 342

untrusted site, 341–342

sendRequest() function, 477

Server-side programming with PHP

- echo command, 52

- GET method, 54

- POST method, 58

variables and strings

 - escaping double/single quotes, 49

 - evaluated test.php web page, 47

 - HTML tags, 50

 - JavaScript source code, 51

 - mixing PHP and HTML tags, 53

 - \$name variable, 52–53

 - .php file extension, 46

 - test.php file, 46, 48

 - user preferences, 53

session_start() function, 367, 380

session_user variable, 406, 436

setcookie() function, 354

shell_exec() function, 44

Shell script, 245

show databases statement, 224

show tables statement, 253

socket_bind(), 452

socket_close(), 452

socket_listen(), 452

Speedport Entry 2i VDSL router, 91

Spyware programs, 353

SQL queries

- group by clause, 237

- inner join clause, 236

- like operator, 239–240

- select statement, 236

- update command, 238

sqlstring, 263

- SSLEngine directive, 397
 ssl module, 337, 397
SSL/TLS
 certificates, 328–329
 public key cryptography, 328
 TCP/IP protocol, 328
 Static IP address service, 445
 Static Map Maker tool, 116
 Static map web page
 eyedropper tool of mtPaint, 124
 geo6.php, 120–122
 new version, 121
 shutter window, 123
 Static private IP address
 Add button, 13
 DHCP, 9
 DNS server, 13
 Editing Wired connection 1
 window, 12
 ifconfig, 10
 IPv4 and IPv6 settings, 11
 LAN, 11, 13
 manual method, IPv4 settings, 13
 Network Connections window, 11
 options, 10
 route command returns, 10
 Save button, 14
 testing, 14
 Stream Editor (sed), 257, 259
 String delimiters, 50
 strpos() function, 153
 strtotime() function, 440
 strtoupper() function, 452
 Structured Query Language (SQL), 219
 Submit buttons, images
 CSS properties, 318
 CSS disabled selector, 320
 opacity property, 320
 search.php, 318–319
 submit property, 319
- ## T
- TCP/IP sockets, 449
 Telnet protocol, 91
 Top-level domains (TLDs), 126, 386
 toUTCString() method, 494
 Transport Layer Security (TLS) protocol, 328
 Transport protocol, 94
 trim() function, 152
- ## U
- Unix pipe operator (|), 258, 277
 user data, file edition, 149, 151–153
 URL mask option, 137
- ## V
- validate() function, 144
 Virtual hosts (vhosts)
 categories, 19
 download, directory index, 21
 enabled sites and details, 35
 IP-based, 21–22, 24
 Lighttpd
 create directory index, 184
 Dashboard web page, 181
 DDNS configuration, 181
 directory index, 187–188
 document root directory, 184
 Host header value, 186
 Hostnames web page, 183
 HTML source code and
 save file, 184–186
 multiline options, lighttpd.conf, 188
 loaded web page, 183

INDEX

Virtual hosts (vhosts) (*cont.*)
 reload, 189
 secureserver, 182
 settings, 182
 URL, 187
ls command lists, 35
name-based vhosts (*see* Name-based vhosts)
port-based vhosts, 25–29
 VirtualHost pair, 20
Virtual host, testing
 public IP address, router, 96–98
 result web page, 100
 router’s port forwarding service, 96
 webpagetest.org home page, 98–99
Visitor’s location, map
 Apache module, 114
 echo commands, 115, 117
 GeoIPCity.dat database, 115
 geoip.conf, 114
 geo4.php, 115, 118
 geo5.php, 118, 119
 PHP global variables, 119
 src attribute, 116
 Static Map Maker tool, 116
 sudo su, 113
Visitor’s native language
 GEOIP_COUNTRY_CODE
 environmental variable, 108
geo3.php, 107
geoscreenshot.com, 113
test location list
 California, 109–110
 France, 111
 Korea, 112
 Seoul, 112
 Strasburg, 110
webpagetest.org home page, 108

W

Web page, editing, 141–145
Web page testing, IP address, 147
webpagetest.org home page, 98–99
Web Redirect option, 134
Web scraping, 219, 241
 args array, 258
 awk, 259
 categories menu, 246–247
 echo commands, 260
 file.txt, 254, 260
 for loop, 260
 grep, 259
 HTML source code, 249–250
 ISBN, 249, 254, 260
 Linux, 248
 MySQL database, 253–254
 product link, 249
 shell program testing, 264–268
 shell.sh, 254–257, 262
 URL, category, 247
 wget, 259, 261
Web scraping, MySQL and Linux Shell
 URL
 address bar, 244–245
 Apress home page, 243–244
 ASIN, 241, 243
 ISBN, 243
Web Server, 93
wget (web get) command, 257
where clause, 235
while loop, 152
Whois installation, command line, 148–149
Wireshark packet analyzer, 361

X, Y, Z

XMLHttpRequest(), 489