Effortless E-Commerce

with PHP and MySQL

SECOND EDITION

Larry Ullman

EFFORTLESS E-COMMERCE with PHP and MySQL

SECOND EDITION

LARRY ULLMAN

New Riders

Effortless E-Commerce with PHP and MySQL, Second Edition
Larry Ullman

New Riders
www.newriders.com
To report errors, please send a note to: errata@peachpit.com

New Riders is an imprint of Peachpit, a division of Pearson Education.

Project Editor: Nancy Peterson
Copyeditor: Liz Welch
Proofreader: Scout Festa
Technical Reviewer: Chris Cornutt
Production Coordinator and Compositor: David Van Ness
Cover Designer: Aren Straiger
Indexer: Karin Arrigoni

ISBN 13: 978-0-321-94936-3
ISBN 10: 0-321-94936-6

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

This book is dedicated to all the friends, family, and coworkers who have been so helpful, supportive, understanding, and generous with their time over the past few years. It's a long list, in no particular order: Roxanne, Nicole, Sarah, Meredith, Paula, Barb, Christina, Shirley, Cyndi, Sommar, Brian, Gary, Heather, Rich, Gina, Mike, Kay, Janice, David, and everyone at Peachpit Press.

A BUSHEL—THAT'S FOUR WHOLE PECKS—OF THANKS TO…

Nancy, for managing the project, for being great to work with, and for assembling such a top-notch team.

Chris, for joining in at the last minute to provide an excellent technical review.

David, for magically converting a handful of random materials into something that walks and talks like a book.

Liz and Scout, for the sharp eyes in improving my muddled words, grammar, and syntax.

Karin, the indexer who makes it easy for readers to find what they need to know without wading through all of my blather.

Aren, for the snazzy cover design.

All the readers who requested that I write this book and provided detailed thoughts as to what they would and would not want this book to be. I hope it's what you were looking for!

To all the readers who liked the first edition and made suggestions for how I could improve this one.

Gary at Kona Earth coffee (www.konaearth.com) for the ton of feedback. And for the truly excellent coffee!

Templates.com (www.templates.com) for permission to use their template in the book's Coffee example.

Jon, for permission to use his "Architecture by Hand" stencil for some of the book's figures (www.jonathanbrown.me).

Rashelle, for always entertaining the kids so that I can get some work done, even if I'd rather not.

Zoe and Sam, for being the kid epitome of awesomeness.

Jessica, for doing everything you do and everything you can.

CONTENTS

INTRODUCTION

Electronic commerce has been an important and viable part of the Internet for well over 15 years now. From the behemoths like Amazon.com to the mom-and-pop online stores to the boutiques run through Etsy, e-commerce is performed in a number of ways. Despite the dozens, or hundreds, of failures for every single commercial success, e-commerce can still be an excellent business tool when done properly. And yet, surprisingly, there are very few books dedicated to the subject.

Using two concrete examples, plus plenty of theory, this book covers the fundamentals of developing e-commerce websites using PHP and MySQL. Emphasizing security, a positive customer experience, and modular, extendable programming, this book presents tons of detailed solutions to today's real-world e-commerce demands. Whether you've been creating dynamic websites for years or just weeks, you're bound to learn something new over the course of the next 15 chapters.

WHAT IS E-COMMERCE?

In the broadest sense, the term *e-commerce* covers the gamut of possible online commercial transactions. Any website with the intention of making money for a business could fall under the "e-commerce" label. Of course, such a liberal definition encompasses the vast majority of existing websites. On the opposite end of the scale, e-commerce can be defined as strictly the online act of taking money directly from customers. And that's the kind of e-commerce this book addresses.

There are two key differences between a site hoping simply to *make* money and one intending to *take* money:

- How comfortable the customer needs to be
- How secure the site needs to be

A site can make money from selling ads, in which case all that's required of the customer is that she visits. Or a site could make money from referrals, where the hope is that the customer will use a link on the site to purchase something from another site. In both cases, what's being asked of the user is insignificant. But when a site wants a customer to provide her full name, address, and credit card information, that becomes serious business. In order for the site to succeed, the customer must be respected, her questions answered,

her concerns addressed, and her fears mitigated. And, of course, the site has to have something the customer wants to spend money on there and not somewhere else.

When it comes to e-commerce, I can't overstress the importance of security. To protect both the business and its customers, a site must be designed and programmed so as to establish and maintain an appropriate level of security. As you'll see, especially in Chapter 2, "Security Fundamentals," the overall security of a website is impacted not just by the code you write but also by some of the initial decisions that you make, such as the chosen hosting environment. With this in mind, security concerns are presented in the book from the big picture and the general theories down to the nuances of specific code. You can rest assured that the book's examples have no known security holes. Moreover, there's plenty of discussion as to how you can make specific processes even more secure, as well as warnings about what you shouldn't do, from a security perspective.

ABOUT THIS BOOK

The goal of this book is to portray the widest possible range of what e-commerce can be, in terms of PHP code, SQL and MySQL, and a site's user interface. To that end, the book is broken into four parts, cleverly named Part 1, Part 2, Part 3, and (drumroll) Part 4.

Part 1, "Fundamentals," has just two chapters, which examine

- Fundamental theories and issues surrounding an e-commerce business
- Decisions you need to make up front
- Critical aspects of online security

In Part 2, "Selling Virtual Products," you develop an entire e-commerce site. This site sells virtual products, namely access to content. With virtual products, there's no inventory management and nothing to ship. The business just needs to accept payment from customers and ensure that access is denied to nonpaying customers. For this example, PayPal is used to handle customer payments. PayPal is a wise choice for beginning e-commerce sites because it has a name that almost all customers will be familiar with (and therefore trust), and it minimizes the security risks taken by the site itself.

Part 3, "Selling Physical Products," creates an entire e-commerce site for the sake of selling physical products. This involves inventory management, an online catalog, shopping carts, order history, and more. For that example, the Authorize.net payment gateway is integrated directly into the website, creating a more seamless and professional experience.

Part 4, "Extra Touches," is entirely new in this edition of the book. Part 4 explores dozens of features, techniques, approaches, and so forth that you can apply to the two example sites or to e-commerce in general. One chapter makes specific recommendations regarding

the virtual product example site. Another chapter gives the same treatment to the second example site (which sells physical products). The third new chapter singles out JavaScript and Ajax as a great way to enhance the e-commerce experience. And the fourth new chapter explains how to use Stripe, a revolutionary way to process payments.

By using two examples with different goals and features, the book presents a smorgasbord of ideas, database designs, HTML tricks, and PHP code. The intention is that, after completing the book, you'll feel comfortable implementing any number or combination of features and approaches on your own e-commerce sites.

Technologies Used

This book, as its title implies, uses the PHP scripting language (www.php.net) and the MySQL database application (www.mysql.com) as the foundation of the websites. When writing the book, I was using version 5.5 of PHP and version 5.6 of MySQL, although you should have no problems with any of the code as long as you're using PHP 5.3 or greater and MySQL 5.0 or greater. In places where newer versions of these technologies are required, you'll see alternative ways to accomplish the same tasks.

As with any modern website, HTML is involved (of course), as is CSS. The book does not explain either in great detail, but it does show some best practices in terms of their use.

In Part 4, you'll encounter JavaScript and the jQuery framework (www.jquery.com). JavaScript, jQuery, and Ajax are used to enhance the sites and add some functionality. I explain the code in some detail, but if you're entirely unfamiliar with JavaScript, it might be daunting. JavaScript knowledge isn't necessary for either of the book's examples, however.

Part 3 also taps into some of what the Apache web server (http://httpd.apache.org) can do. As with the JavaScript, the Apache particulars aren't required knowledge, but it's worth your time to become familiar with them.

What's New in This Edition

The biggest and most obvious addition in this edition is Part 4. It consists of four chapters:

- Chapter 12, "Extending the First Site"
- Chapter 13, "Extending the Second Site"
- Chapter 14, "Adding JavaScript and Ajax"
- Chapter 15, "Using Stripe Payments"

These chapters present more ways you can implement e-commerce, from specific features you could add, to alternative coding techniques, to improving the security. And the last chapter presents a new way of taking payments online.

Besides the obvious new material, I've updated all the code in the two sites to keep them current and secure, reflecting changes in technologies or approaches since the first edition was written. For example, there are new and better ways of communicating with PayPal and Authorize.net. There's also a greatly improved and more secure technique for storing and verifying passwords in PHP. And I've changed the client-side foundation of the first e-commerce site from using a third-party template to implementing the Twitter Bootstrap framework (version 3; www.getbootstrap.com).

Finally, I've gone through all the code and fixed anything that was suboptimal, or outright wrong, in the first edition of the book. In a couple of the more complicated places, I've lengthened, clarified, or just flat-out improved the explanation of what's happening and why.

Getting Help

If you have any problems with, or questions about, what is said or done in this book, there are several resources to which you can turn, starting with the book's website, www.LarryUllman.com. There you can find all the files, code, and SQL commands used in this book.

At www.LarryUllman.com/forums/ you'll find a support forum dedicated to this book. If you post a question or comment there, you'll get a relatively prompt reply, from others or from me.

WHAT YOU'LL NEED

Just as e-commerce is a transaction between a customer and a website, a book can be viewed as a transaction between the writer and the reader (just not one that takes place in real time). I've already presented a synopsis of this book, but who do I imagine you to be and what will you need?

Some Fundamental Skills

The goal of this book is to demonstrate the application of PHP and MySQL to the task of creating an e-commerce site. Although I expect that even a seasoned web developer will learn a lot, the book does not teach the fundamentals of either PHP or MySQL. If you're not already comfortable with these two technologies, this is not the book for you. If you have no problems executing a MySQL query using PHP and then handling those query results, you'll be fine.

The same must be said for the secondary technologies involved, namely HTML and CSS. If the definition of an HTML form is foreign to you, you should learn those basics before getting immersed in this book's material.

As for the JavaScript, jQuery, and Apache work that you'll come across, no previous experience with them is expected, although those sections will certainly be easier to follow if you have some.

A Web Server

To develop a site using PHP and MySQL, you'll need a web server, a computer running PHP through a web server application (such as Apache, nginx, or IIS [Internet Information Services]), and the MySQL database application server. Fortunately, you can install all of these on your own computer, at absolutely no cost. The easiest way to do so is to use an all-in-one package, such as XAMPP (www.apachefriends.org) or MAMP (www.mamp.info). If you already have a website being hosted on a live server, that will work as well.

And a Bit More

A web server will let you *run* a dynamic website, but you need additional tools to *develop* one: At the very least, you'll need a decent text editor or integrated development environment (IDE). A commercial IDE like PhpStorm (www.jetbrains.com/phpstorm/) is fine, as is an open source IDE like Aptana Studio (www.aptana.com) or a plain-text editor such as SublimeText (www.sublimetext.com). Just use something with more features than Notepad!

It doesn't matter what web browser you're using, as long as you use one with great debugging tools.

And that's it! If you've already done some PHP and MySQL development (which is a requirement for following along with this book), you probably already have everything you need. So let's get started!

PART ONE
FUNDAMENTALS

1 GETTING STARTED

Just as you don't begin building a house by grabbing a hammer, creating an e-commerce site doesn't start with your computer. Well, you'll probably use your computer for research, but actual coding is a step that comes much later. In this chapter, you'll learn how to commence developing your e-commerce site. The goal of the chapter is to explain two things:

- The steps you'll need to take
- This book's perspective on e-commerce

Although the point of this book is to provide concrete answers and usable code, there will be some subjects, especially over the next few pages, for which I can't tell you what to do. In such cases, I instead try to identify what questions you'll need to answer and how you might go about doing so.

At a root level, the success of any website, regardless of whether it's intended to make money, depends on its usability, reliability, and performance: If people are attempting to use the site, can they? In this chapter, you'll encounter many of the decisions you'll need to make that impact your site's availability. The choices you make aren't permanent, but as with most things, not having to make big changes further down the road is preferable.

The success of an e-commerce site further depends on security. This chapter touches on a few security issues, but security is addressed in more detail in the next chapter, and then throughout the rest of the book.

The last thing to note is that you may be creating an e-commerce site under one of two scenarios: for yourself or for others. When creating a site for

yourself, you'll need to make most of the decisions. When creating a site for others, they'll be the ones making most of these decisions and your part in the process is, at best, advisory. Take, for example, the business's goals....

IDENTIFYING YOUR BUSINESS GOALS

Before you do anything, anything at all—mock up a web design, identify your web host, or even buy the domain name—you need to identify your business goals. For an e-commerce site, the goal is to make money, which you can do in different ways:

- Selling goods or services directly
- Advertising on the site
- Promoting goods or services that can be purchased elsewhere

In this book, I'm using the term *e-commerce* to refer to sites that directly accept money from end users. I've limited myself to that scope, because handling money directly demands a level of security well beyond other types of sites.

Say you wanted to create a site that reviews music: You might give all the content away for free but hope to make money by displaying ads on your site and/or by using affiliate links to other sites that actually sell music. In either case, the security issues you'd have are no bigger than those for most other non-e-commerce sites. As another example, my blog, www.LarryUllman.com, supports and augments the books I write, which ideally increases the sales of the books; however, the blog itself does not take money directly. The goal in this book is to create sites that sell goods or services directly to customers.

tip

A good way to get people to your site is to offer something, almost anything, for free!

Achieving a business's goals involves many components. The focus of this book is strictly on manufacturing the online experience; you'll need to follow through on your own with the other facets of running a business, such as

- Creating a legal business entity
- Properly handling business taxes
- Doing the company accounting
- Coordinating with vendors
- Marketing your business

- Managing employees and payroll
- Controlling physical inventory
- Managing shipping and returns

In short, just creating the website isn't all you'll need to do. Most importantly, know from the outset that even if you make a *fantastic* e-commerce website, that alone is no guarantee of business success.

tip

Give people a reason to visit your site even when they're not shopping, so they might buy something on impulse or think of your site first when they do want to make a purchase.

So stop reading right now and write down your business goals. What do you hope to achieve? What are your short-term goals? What are your long-term goals? Try to be realistic about them.

Next, write down (on a large piece of paper!) everything you think you'll need to do and have in order to achieve those goals. How much money can you invest up front? How much time? Who will help you? How will the helpers be compensated? From where will you get more money when you suddenly need more money? Who is going to handle the bookkeeping? How will you get people to visit your site? If you're selling physical products, where will they be stored? How will you ship the merchandise?

Clearly, you'll need to answer a lot of questions, even for the most basic of goals. But there's one key question I *can* answer for you: How do you create a good, secure e-commerce site? Answer: Read this book!

RESEARCHING LEGAL ISSUES

Whenever you're dealing with other people's money, and whenever you're creating your own business, you have to take into account legal issues. This is a big area in which I can be of little assistance: I'm not a lawyer, and I don't know in which country, state, province, territory, or city you live. But this doesn't mean I can't point you in the right direction.

National and International Laws

The legal issues involved differ when the website is for your business and when you're creating it for a client. When working for a client, you must have a sound, legal contract. In particular, the contract should limit the liability you personally have should something go wrong. As a general rule, good contracts limit your liability to the amount of money you made on the project itself, should you be at fault. Also, you should define a process for how to handle

change requests. One approach is to provide one round of requests after the initial version of the site is complete. Secondary requests, or any additions unreasonably beyond the original scope of the contract, must be renegotiated.

If you have your own business and there is no client, you still have tons of other legal issues to investigate that have nothing to do with the e-commerce site itself. For these, start by contacting every applicable governmental department to see what you must know and do. Many cities and states have small business branches dedicated to helping people like you navigate the maze of legal necessities.

In either case, you must be knowledgeable about legal issues specifically addressing online commerce. Again, your local and national governments should be able to provide you with this information. The particulars will differ greatly from one country to the next. They may even depend on where you're located, where the client is located, where the customers are, where the site is physically hosted, where the associated bank can be found, and so forth. In the United States, the Federal Trade Commission (FTC) oversees many aspects of e-commerce. On the FTC website, www.ftc.gov, you'll find excellent guidelines for e-commerce, international sales, security, and more.

As another example, in the United Kingdom, the government has exact requirements as to what information should be available on the website, as well as on order forms and in emails. This includes

- The company's physical address
- The company's registration number
- Any trade associations
- The value added tax (VAT) number

Because you'll be storing information about the customers, other laws are involved. The European Union has specific regulations as to how personal data is stored and used. The United States also has precise rules about the use of customer email addresses for advertising, promotional emails, and the handling of disclosures. All these laws apply to basic personal information; if you're storing credit card data (and you really shouldn't), even more laws apply.

You'll also need to know whether Internet sales should be taxed and, if so, at what rate. In the United States, this is still being debated and varies from state to state. And if you're shipping physical products, there are rules about when you can charge the customer based on when the order ships. If part of the order ships, you can only charge the customer part of the order total at that time.

All laws aside, treat your customers and their personal information as you would hope sites treat you and your information.

Should the worst happen—say your system is hacked and the data is breached—laws may apply as well. The state of California, for example, has strict laws as to what you must do once you find a security violation. Part of planning—a big part, really—is preparing yourself should the worst happen so that you're not scrambling to find answers in the middle of a crisis.

I understand that the number and complexity of laws that may apply can be overwhelming, but take that as an indicator of how important it is that you pursue these issues to the fullest extent. When you're building a business, and when you're trying to make money via e-commerce, instituting proper and complete compliance with all laws is the only way to go.

PCI Compliance

Another legal issue on which you should be extremely well versed is PCI DSS, short for Payment Card Industry Data Security Standard (www.pcisecuritystandards.org). This is a specific set of rules for ensuring secure, proper handling of credit cards by all commercial vendors. Any company that processes, stores, or transmits credit card information must follow these guidelines, thereby being *PCI compliant*.

Many payment gateways allow for recurring payments, meaning you can charge a customer multiple times, still without storing their payment information yourself.

By following the code in this book, you'll neither store nor process any credit cards yourself, which is for the best. *You absolutely do not want to store the user's credit card information!* There are companies that do that, yes, but that's their full-time job and they have the knowledge, resources, and money to do that properly. Still, even taking credit card information on your site and passing it off to another company means you need to be PCI compliant. The specific requirements differ based on what you do with credit cards and how many transactions per year you process. I'll get into those requirements in the next chapter.

The credit card companies, Visa in particular, have loads of documentation on their websites regarding secure handling of credit cards, what to do if your system is compromised, and more.

If your site is not PCI compliant and there's a security breach, several bad things could happen (beyond the effects of the security breach itself). First, the credit cards companies will likely escalate your security requirements to a higher level, such as requiring external security scans of your system. This means more work for you and higher expenses. Second, the credit card companies that created the PCI DSS—such as Visa, MasterCard, American Express, Discover, and JCB (Japan Credit Bureau)—could make you pay any damages they incur because of your security breach. They may even fine you as well. Third, those same companies could deny you the option of accepting their cards, which will pretty much shut down your business.

Technically, the PCI DSS isn't a law, but some parts of the specification may also be an applicable law in your country, state, province, or territory. And the

potential penalties that the credit card companies can impose can be just as scary as any legal repercussion.

CHOOSING WEB TECHNOLOGIES

Over the past 20 years, the web has changed in many ways. It has changed significantly in just the past five! But some things remain the same. For starters, there's HTML (HyperText Markup Language). Whatever else has changed—whatever image types, video options, and server-side technologies you use—the end user first interacts with HTML. This book does not, and cannot, teach HTML. If you need more information about HTML, pick up a book on that subject, such as the de facto standard, Elizabeth Castro and Bruce Hyslop's *HTML and CSS: Visual QuickStart Guide, 8th Edition* (Peachpit Press, 2013).

note

This book doesn't teach HTML, CSS, JavaScript, PHP, SQL, or MySQL; instead, it demonstrates real-world application of these technologies.

With modern web browsers, most of a site's layout and design comes from CSS (Cascading Style Sheets). I'll be using CSS in this book, too, and just like with HTML, I don't explain it in much detail. Still, I won't be using CSS in any superfancy way, so you shouldn't have a problem following along.

When I first began doing web development in the late 1990s, there was this annoying little thing called JavaScript. At that time, JavaScript was largely used for petty and cutesy tricks. In short, JavaScript was almost entirely unnecessary. Today, thanks to Ajax, Web 2.0, and other marketing terms that people throw around, things are quite different. Now, JavaScript, when properly used, greatly improves the user experience. Many website features that people appreciate, such as being able to present lots of content in a limited space, being able to add something to a cart without leaving the page, and so forth, require JavaScript. Although JavaScript is valuable, it's really an "extra." With that in mind, this book will make use of some JavaScript to implement some extras. If you're not comfortable with JavaScript already, might I selfishly recommend my own book, *Modern JavaScript: Develop and Design* (Peachpit Press, 2012)?

note

After this chapter, I'll stop recommending other books to buy, I promise!

On the server side of the equation, unlike in the client, you have a vast range of web technology to consider. This book uses PHP as the programming language of choice and MySQL as the database application. These are among my personal favorite server-side technologies, and if you're reading this book, I assume you think so as well. But if you aren't already well versed in PHP and MySQL, you will have difficulty with some of this book's code. Consider reading my *PHP and MySQL for Dynamic Web Sites: Visual QuickPro Guide, Fourth Edition* (Peachpit Press, 2011) to learn more about these technologies.

EASY E-COMMERCE ALTERNATIVES

In this book, you'll learn how to write an e-commerce application from scratch, using a combination of HTML, CSS, JavaScript, PHP, SQL, and MySQL. There are, however, faster, less custom approaches you can take.

If you just want to get an e-commerce site online quickly, or if you don't know any of the listed technologies, you can use "turnkey" e-commerce sites that Yahoo, Google, and others provide. By answering some questions and using the chosen company's interface, you can create a basic e-commerce site in a day. It'll even be tied automatically into a payment system. But make no mistake: Although you'll get up and running in no time, the end result will be rather amateurish and very limited.

A middle-ground solution between using an entire third-party system and creating your own is to use an off-the-shelf e-commerce package, such as ZenCart (www.zen-cart.com), FoxyCart (www.foxycart.com), or osCommerce (www.oscommerce.com). They provide all the functionality, from creating a catalog or a shopping cart to administration, which can then be tied to one of several payment systems. These tools have been around for years; they're quite solid and well supported, but they'll still have some limitations compared to writing your own e-commerce site, especially when it's time to add features that will be uniquely yours. At the same time, these packages will also be bogged down with lots of features that you might not ever use.

SELECTING A WEB HOST

tip

You may need to put your site on a hosted server in order to test it with a payment gateway.

I strongly advocate that you *develop* your entire site using just your personal computer or other development environment that you have readily available. You can install all the necessary tools—a web server, PHP, and MySQL—on your own computer, then develop the database, write the code, test, and so on. Developing on your personal computer is faster (because you don't have to upload files), cheaper (because you're not paying for hosting during this time), and more secure (because incomplete, potentially unsecure code won't be online).

After getting the project nearly complete, you'll need to move it to your web host. Let's look at how you choose one.

Hosting Options

With regard to hosting, you can generally say that you get what you pay for, and I say that as a person who's inclined to go the cheapest route whenever possible. Over the years, I've used probably five or six hosts for my own websites and dealt with many others for clients. The old adage says that you have to spend money to make money; selecting a cheap host is a bad way to go about making money.

Hosting plans vary based on

- Price
- Features
- Performance
- Amount of control

The *price* is directly related to the quality of the other three attributes. If you spend more, you'll get more.

To be honest, the *features* don't really matter. Well, some do and many don't. Most hosting plans will offer some 56 features, of which you'll need 10. This even goes for disk space and bandwidth limitations: Hosting plans will offer you more of these than you'll ever need, thereby tempting you with trivialities. The minimally required features are PHP, MySQL, a mail server (to send and receive email), and security software, such as a firewall, a virus detector, and so forth. Additionally, beneficial features include regular backups and excellent—truly excellent—customer support. When it comes time to compare one hosting option to another, decide what really counts—like uptime, backups, security, and customer service—and ignore the rest.

The *performance* of a server will depend on the type of hosting involved, the server's specific hardware—amount of RAM, disk types, processor types, the number of processors, and the server's network connection. As I mentioned earlier, the site's performance is hugely important, but it's unfortunately something that's not easily determined in advance.

The *amount of control* you have over the server will depend on the hosting type. Different web-hosting companies offer different plans, but the basic hosting options are

- Free
- Shared
- Virtual private server (VPS)
- Dedicated or colocation (colo)

Free hosting plans are harder to come by now than they used to be, but you shouldn't even consider them for an e-commerce site. You may have a free site possibility with some account you have, or from your ISP, but you probably can't even use your domain name on them.

tip

You'll eventually come to regret using free or very cheap hosting plans for your website, so save yourself that headache!

Shared hosting plans are the most common and the cheapest (of the paid choices). Shared hosting involves putting tens of clients and possibly hundreds of websites on a single server. Shared hosting is inexpensive—decent plans range from $10 to $20 (all prices in the book will be in U.S. dollars) per month and may be a reasonable way to start. However, because there are multiple users on each server, your website will only be as secure as the weakest security link in any site on the server. The performance of the site will also suffer, as the demands are so high. Finally, you'll have little to no control over how the server runs. You won't be able to use a particular version of PHP, enable certain PHP settings or features, or tweak how MySQL runs. Shared hosts aren't likely to make any changes that might adversely impact the other clients on the same server. Still, shared hosting may be appropriate for smaller, less demanding sites without higher security concerns.

A happy medium between shared hosting and dedicated is the *virtual private server* (it's what I've personally used for several years). Instead of having tens of clients on a single server, there may be only a couple or a handful, with each client running her own virtual operating system. Although all the servers' hardware is still being shared, limitations can be placed so that you'll always get a minimum amount of RAM, thereby guaranteeing some performance no matter what happens to the other sites on the server. From a security perspective, each virtual server is a separate entity: The actions that the other clients take on their VPS instances can't impact yours. And since the VPS is yours alone, you can do whatever you want with it in terms of installing and configuring software. VPS hosting plans run from as cheap as $30 per month to around $100 per month.

tip

When using dedicated or colocated hosting, make sure that the hosting company will still provide some maintenance and security assistance.

A *dedicated* or *colocated* server is on the other end of the hosting spectrum. This kind of hosting puts an entire computer—its software and hardware—under your command, but the server is physically housed at the hosting company's location. That location should have multiple, fast connections to the Internet; redundant power supplies with battery backups; secure physical access to the server rooms; climate control; and so on. (The technical difference between dedicated and colocated hosting is that the host typically owns a *dedicated* server, whereas you typically own a *colocated* one.)

The other hosting types can't match the amount of control, the number of features, or possibly the performance of running your own entire server. But the cost of a dedicated or colocated server will be much, much higher—from a couple of hundred dollars per month to several hundred. Just as important is the fact that, depending on the particulars of the hosting plan, you may be responsible for all the maintenance and security of the server. You'll need to

decide if you think you're better suited to handle server security than someone who does that full time and has likely been doing it for years. Also, the web-hosting company will have people monitoring your server 24 hours a day, whereas you've got to sleep sometime.

CLOUD COMPUTING

Another hosting option has come up in the past few years: *cloud hosting*. Cloud computing sounds ethereal, but it's just moving some server functionality—processing of data, storing of data, handling of emails, or whatever—to a different computer (or bank of computers) not under your control and on a different network. One benefit to cloud computing is that it can automatically scale to your needs without you having to take extra steps. If, for some freak, benevolent reason, you go from processing an average of 100 sales per day to 10,000, the cloud will be able to handle the increased traffic, which might otherwise have crashed a basic hosting plan. But there are extra security concerns with cloud computing, and you'd need to be prepared to pay the price. For example, if your site gets hit with a denial-of-service (DoS) attack (discussed in Chapter 2, "Security Fundamentals"), you'll have to foot the bill for the extra cloud computing, but the attack itself will have generated no extra revenue.

A cloud hosting option, such as Amazon's Web Services, is fantastic in many ways. You can expand easily and still only pay for what you use. But cloud hosting is implemented differently than any other type of hosting, and those differences present another hurdle to overcome when you're just starting out. On the other hand, you can start with a traditional hosting scenario and later add extended networking (for example, a content delivery network) to gain some of the benefits of cloud hosting.

This book doesn't discuss cloud computing beyond what I've just said. But be aware of this potential avenue, and you may want to look into vendors and pricing if you suspect that cloud computing could be a good fit for your site and situation.

My Hosting Recommendation

As a reader, you're probably looking for as many definitive answers as possible, so my recommendation is to select a quality shared or VPS hosting plan to begin, depending on the project itself and your budget. You absolutely don't want to host the site on your personal computer; you absolutely don't want to use free hosting; and you most likely shouldn't go with dedicated hosting to start, unless you have money to waste.

One important thing to know is that you're not permanently locked into a given hosting plan or even a web host. A good web host should be able to upgrade or expand your hosting plan with little or no downtime. Start with a plan that's reasonably basic, and should you have the good fortune of profound success, you can scale up your plan to meet the increased demands over time.

It's possible to change web hosts as well, just not as easily. It's best to start with a great host that you'll be able to stick with for years and years. This

means not only someone reliable, but also a host that's established in such a way to allow for your site's expansion. For example, a really cheap host probably does only shared hosting. You'd never be able to move to a dedicated server with them, and you probably wouldn't want to. Conversely, the hosting company I use provides only VPS and dedicated hosting plans. The VPS works for me for now, and I can move to one or more dedicated servers with this same company when I have that need.

tip

You can save yourself some money by developing the entire site on your own computer before you purchase a hosting plan.

My final piece of advice is not to spend dramatically more than you need to earlier than you need to. By that I mean, you many think you've got a site that will someday have millions of users, and therefore you'll need dozens of servers, but today you've got no site and no users, so a single server (or hosting plan) will be more than sufficient.

Finding a Good Host

The final question, then, is how do you know if a web host is good? First, go online and search using terms like *web host review* or *best web host*. In the search results, *ignore* every site whose sole purpose is to rate and review web hosts. Yes, that's right: ignore those. They're unreliable and built on advertising, and you'll never know what kind of relationship they may have with the companies they're "ranking." Plus, in my experience, such sites are ranking web hosts for the masses, for those who don't know any better. If you want to find a couple of recommendations this way, mostly as a basis of comparison, that's fine, but these rankings should not be used to make a decision.

note

All of the lousy web hosts I've used over the years were found by listening to "official" rankings of the best web hosts!

The best way to find a good host is to get real-world feedback and comments from real people. One way to do so is by finding forums where people talk about their hosting experiences. In the past, I've also emailed people to ask them if they're happy with their host, prior to making a decision. You can also get recommendations through mailing lists and the like.

If you're curious, I personally use ServInt (www.servint.net) and have for years. I've also heard good things about DreamHost (www.dreamhost.com) but have no personal experience with them.

Once you've got a few potential candidates, start by excluding those that are really cheap. You don't want to try to save money by skimping on web hosting. It's not a good long-term plan. Cheaper hosting options than the one I use are out there, but my site is always available. I've got peace of mind, and you can't put a price on that. Interestingly, my current host doesn't even offer a free month of hosting, as many companies do. Their argument, which I buy into, is that providing a free month invites malicious people to temporarily get a

server just to send spam or do other harmful or annoying activities. You don't want to be part of a network where that's happening.

You should also rule out those companies that try to do too much: better to have a host that excels at one or two things than one that is average at several. One of the worst hosting experiences I ever had, if not the worst, was with a company whose primary function was as a domain registrar. They were fine as registrars but *terrible* as hosts.

note

Don't let your registrar host your site, and don't let your host register your domain name!

As I already said, all web hosts will offer tons of features and more disk space, bandwidth, and add-ons than you'll ever need. And it's almost impossible to compare performance from one host to the next. For me, then, I focus on a host's security approach and customer service. If your hosting company strives for top-level security, that minimizes the chances of a problem occurring in the first place. If your hosting company also offers great customer service, they'll provide a quick fix should a problem arise.

USING A PAYMENT SYSTEM

As with your choice of a web-hosting company, the payment system you use for your e-commerce site will have a significant impact on the end result. This is not to say that the site will be married to a single payment system for eternity, but as with any divorce, ending a relationship with a payment system can be tedious and costly for your business. Furthermore, it's possible, if not common, to use more than one payment system at the same time (for example, PayPal and another).

The payment system is the differentiating element between a standard website and an e-commerce one. The whole point of a payment system is to transfer money between the customers and the business.

When I wrote the first edition of this book in 2010, there were only two broad types of payment systems. These are frequently known by a variety of names but can be described as either a *payment processor* or a *payment gateway*.

In the years since I wrote this book, a third approach has arisen. For lack of an existing label, I'm going to call this option "the middle way," for reasons to be explained shortly.

note

Some companies, like PayPal, offer more than one type of payment system.

In this book, I'll demonstrate an example of each of the three types, but here, I'll outline their pros and cons.

Payment Processors

A *payment processor* is a delayed payment system that normally goes through a third-party site (**Figure 1.1**). The best example is the *Payments Standard* option at PayPal (www.paypal.com). If you want to accept payment through PayPal using their basic service, you'll send the customer to PayPal's site along with your PayPal identification and some other information. The customer then uses PayPal to authorize the transfer of that amount of money. PayPal (hopefully) returns the customer to your site, and at some later point in time PayPal will make the funds available to you, minus their fees.

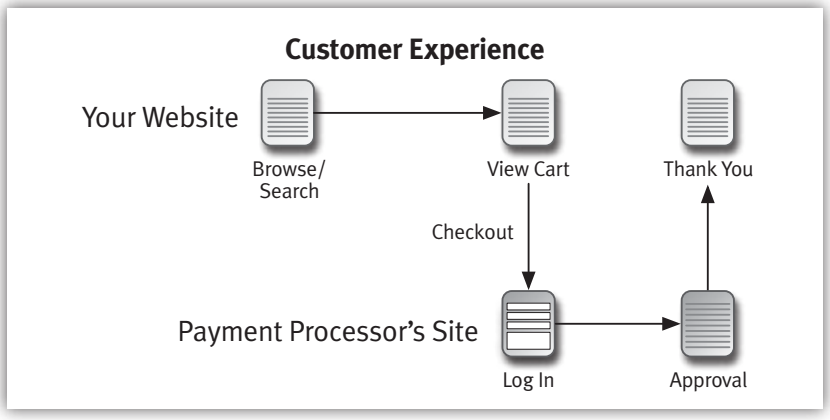

Figure 1.1

Using a payment processor like PayPal Payments Standard or Google's Checkout may be easy to establish, has little-to-no up-front costs, and taps into a service that many customers will be familiar with (customers are especially comfortable with PayPal). On the other hand, these systems aren't as integrated into your site as the alternatives, and sending customers away from your site is a risky e-commerce move, increasing the odds of losing the sale. Also, the per-transaction costs tend to be a bit higher, and deposits most likely won't be automatically made into your business's bank account (that is, you may need to go into the payment processor's system in order to accept and then transfer your credits).

In this book, I'll demonstrate how you can integrate PayPal, which is perhaps the most common payment processor, into your site. However, it can be a real chore to work with, even for experienced developers. But I'll help you navigate that morass in the first e-commerce example in this book, which I'm calling "Knowledge Is Power."

Payment Gateways

A *payment gateway* is a real-time payment system that can be directly integrated into your own site, resulting in a process that's more professional and seamless. Instead of sending users away in the hopes they'll come back, transaction data is transmitted behind the scenes, and the customer won't leave your site at any point in the entire process (**Figure 1.2**). Also, a gateway offers much better fraud prevention, among other extra features (more on fraud protection in the next section). The gateway will deposit your monies into a *merchant account* automatically, normally charging less per transaction than payment processors do.

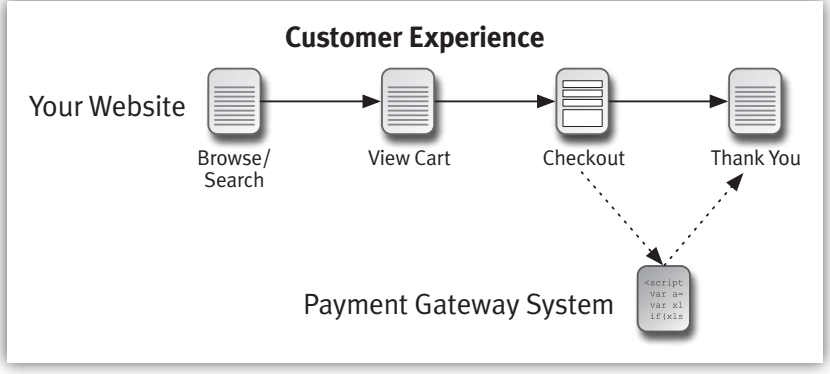

Figure 1.2

On the other side of the equation, a payment gateway may have higher setup costs and will require more programming to integrate the system into your site. Payment gateways also require a *merchant account*, which is an account into which credit card charges can be deposited and refunded (for customer returns). You may or may not be able to use your business bank as your merchant account, depending on your bank.

Further, because your site will be temporarily handling the customer's payment information, you'll need to maintain PCI compliance. This means more work and more money. PCI compliance will also dictate what hosting options are viable for you: shared hosting, for example, would be unacceptable.

Tons of payment gateways are available. Moreover, some gateway systems are resold through other vendors, giving you the ability to shop around for the best deal. Authorize.net (www.authorize.net) is perhaps the best-known payment gateway, and it will be used in the book's second example, which I'm simply calling "Coffee."

The Middle Way

In the three years since I wrote the first edition of this book, a new type of payment option has arisen. I have yet to see a label applied to this approach, so I'll refer to it as "the middle way," because it offers the best aspects of both traditional models (payment processors and payment gateways).

With the middle way approach, customers enter their payment information on your site, as when using a payment gateway. But the payment information will be directly sent to the payment company (using JavaScript) and never touch your server. This simple difference greatly reduces the steps required for you to be PCI compliant, while simultaneously offering an optimal customer experience. For the developer, the middle way approach is surprisingly simple to set up and integrate. For the business, the fees are competitive, generally comparable to PayPal's fees, without any monthly or setup fees. And the middle way is a "full-stack" solution, which means you don't need to find and use a merchant account.

note

Both PayPal and Authorize.net offer "middle way" solutions as well.

The most popular middle way payment options available at the time of this writing are as follows:

- Stripe (https://stripe.com)
- Braintree (www.braintreepayments.com)
- Paymill (www.paymill.com)

These aren't perfect solutions, of course. The fees may be higher than other payment options, and there may be a longer delay in getting the money transferred to your bank account than you'd prefer. From a technological perspective, the middle way route tends to require JavaScript. Although only a very small percentage of web users have JavaScript disabled (between 1 and 3 percent), this requirement could be a factor.

In my experience, the biggest negative to these options is that they may not be available yet in your country. These are new approaches, by new companies, and although each company is expanding quickly, they may not be able to service your country quite yet.

New in this edition of the book, I'll demonstrate how to use Stripe as a payment option. As a disclaimer, as of this writing, I'm employed by Stripe (as a Support Engineer). Although I'm clearly biased toward Stripe, know that I'm working for Stripe because I love what they're doing, not the other way around. I started using Stripe, and writing about Stripe, long before they hired me.

Which Should You Use?

If I were to start an e-commerce site today, I'd use one of these middle way companies, if available. Obviously I'm biased here, as I work for one of them, but whether or not you use Stripe, I think that the middle way approach offers the best of all the possible pros and cons.

That being said, many customers prefer to use PayPal, and having more payment options is always better. You might consider using a middle way approach *and* PayPal.

If you can't or don't want to use a middle way approach, you'll need to select the best traditional payment gateway. When selecting among payment providers, you should first determine if your business bank or web-hosting company has an arrangement with any payment companies. By choosing a preapproved vendor for this important service, you'll minimize some of the potential headaches and hopefully have an expert to turn to when you need technical support.

Another factor is geography: Different providers will work in your part of the world and will be limited as to what other regions they support. Also, you'll want to check that the currency the provider uses matches your business's currency, and that of your customers.

There are many features to weigh when making your selection:

- Tools for fraud prevention
- Ability to perform recurring billing
- Acceptance of eChecks
- Automatic tax calculation
- Automatic shipping calculation and processing
- Digital content handling
- Integrated shopping cart

Clearly, many of these features can greatly simplify the development of your e-commerce site and result in a more professional web application, but I'd like to highlight fraud prevention. You may not have given much thought to the subject, but excellent fraud prevention is in the best long-term interest of your site. If someone can use a credit card at your site that isn't valid or isn't theirs, you'll have a false sale and later have some cleanup to perform to undo the transaction. Further, the person whose credit card was fraudulently used will think poorly of your business for allowing the fraud in the first place. For these

Payment systems will provide test accounts, dummy credit card numbers, and false processing systems through which you can test your site before going live.

Make sure your payment solution provider is in full PCI compliance and can assist in guiding your site's compliance, too.

Some gateways offer virtual terminals where the merchant can process credit card payments manually. These can be used to issue returns, for example.

reasons, using a gateway with sophisticated fraud-prevention tools is a must. The two most common techniques are to verify the billing address and the card verification value (CVV; those numbers on the back of the card).

A final, obvious factor that I didn't list earlier is cost. You'll need to consider the initial setup costs, the monthly fees, and the individual transaction expenses. If you require features that come at an extra cost, factor those in as well.

tip

If the price of your transactions will be small—less than $10 on average—find a payment provider that supports *micropayments*, which have smaller transaction fees.

THE DEVELOPMENT PROCESS

After you've finalized your business plan, researched the laws, decided on a hosting company, and selected a payment system, it's time to start putting down HTML tags, SQL commands, and `if`-`then` statements. The development process itself is the point of this book, so let's take a look at it in detail (**Figure 1.3**).

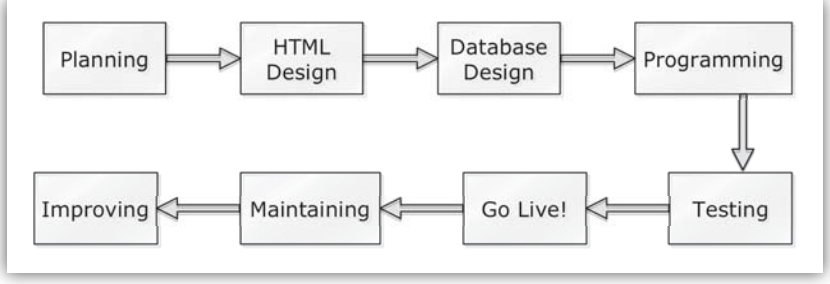

Figure 1.3

The development process occurs in phases. If each phase is approached deliberately and the end results are properly generated, you'll develop a great e-commerce site as efficiently as possible. If, on the other hand, you jump around, rush the process, skip steps, and make omissions, the whole procedure will take much longer, and the end result will be buggier.

At the end of the development process, you'll hopefully have created the best possible e-commerce site, but that site will undoubtedly need to be changed next week (as clients always want), next month, or next year. If the first goal is a smooth, optimal process, then the second is output—specifically PHP code and a MySQL database—that's flexible and scalable. When those inevitable changes need to be made, you should be able to do so without breaking or rewriting the entire system.

Site Planning

The first step in the development process is planning a generic site. This is much like establishing your business goals, but specifically with the site itself. What should the site do? What should it look like? Who are the target users? What browsers and/or devices should the site support? Use pen and paper, or any application in which you can make notes, and be as inclusive as you possibly can. It'll be much better, further down the road, if you considered an idea and ruled it out than if you never thought of it in the first place.

As a model for how to do e-commerce well, you can't do much better than examining Amazon.com in detail.

The best thing you can do at this point is look online. The web is a rich tapestry of both the good and the bad, so look at the sites you like and use. What do they do well? What would you do differently? What fonts, colors, and designs appeal to you? There's an old adage about writing: *good writers plagiarize; great ones steal*. That's kind of true for websites, too.

HTML Design

The next thing you should do in the development process is mock up the HTML designs for the site. I, for one, have absolutely no design skills whatsoever. If you can say the same, here are two simple solutions:

- Hire a qualified designer to create the HTML templates.
- Use an off-the-shelf design that you tweak a bit.

The HTML design process will, rightfully, include a few iterations of feedback, followed by updated designs.

I've taken both approaches several times; which you use depends on the site and your budget. If you're hiring someone, at a minimum, you'll want her to create a few templates:

- The home page
- An inner, basic content page
- A styled form

From these you can easily generate the appearance of most of your site. If you're developing an e-commerce site that sells products, you'll also want *representative browsing* (that is, showing multiple products at once) and detailed listing pages.

As I'm not a web designer, I've relied on a design framework and an available third-party template for the two e-commerce sites in this book.

If you don't have the budget or time to purchase a custom design, you can take an existing one and modify it to your needs. Both free and commercial designs are available, although you'll need to abide by the licensing where applicable. For example, some designs are free to use as long as you give credit to the designer in the footer. Other designs are free for noncommercial use but

require licenses for commercial endeavors. In any case, you can take the existing template and then adjust the HTML and CSS to personalize the design for your or your client's tastes.

tip

Your site's design should include obvious links for contacting the administrator, finding the site's return policy, and viewing the privacy policy.

The goal at this point is to get the client (or you) to sign off on the look of the site. Moreover, the design also implies much of the functionality; getting approval of that is even more critical to the process. Think about: How will the look and function of the site be different if the user is logged in? How will navigation be handled? How are items added to the cart? How will the cart contents be shown? Also pay attention to the fundamentals of the user interface: simplicity, ease of use, proper navigation, breadcrumbs, obvious access to the cart, and so on.

Database Design

Designing the database is a key step, largely because changes to the database at a later date have far larger implications and potential complications than changing any other aspect of the site. Adding functionality through database changes is a steep challenge, and fixing database flaws is excruciating. Make every effort you can to get the database design right the first time.

note

This is absolutely the last reference I make to another book, I promise. Unless I think of another....

Good database design begins, naturally, with *normalizing* the database. If you aren't familiar with normalization, see any good resource on the subject, including my *MySQL: Visual QuickStart Guide, 2nd Edition* (Peachpit Press, 2006). Normalization and performance mean that you also

- Use the smallest possible column types.
- Avoid storing **NULL** values as much as possible.
- Use numeric columns as much as possible.
- Use fixed-length columns when you can.
- Provide default values for columns, if applicable.

tip

If MySQL is running with the `--log-long-format` feature enabled, the database will write to the log any queries that aren't using indexes.

Performance is also greatly affected by using indexes properly. Declaring indexes is somewhat of an art, but here are some general rules:

- Index columns that will be involved in **WHERE** and **ORDER BY** clauses.
- Avoid indexing columns that allow **NULL** values.
- Apply length restrictions to indexes on variable-length columns, such as indexing only the first 10 characters of a person's last name.

- Use **EXPLAIN** queries to confirm that indexes are being used.
- Revisit your indexes after some period of site activity to ensure they're still appropriate to the real-world data.

A final consideration in your database design, which gets less attention, is the storage engine (or table type) in use. One of MySQL's strengths is its support for multiple storage engines, meaning you can select the one whose features best match your needs. For example, you can create MySQL tables in memory, which will perform exceptionally well but provide no data permanence. The two most common MySQL storage engines are *InnoDB* and *MyISAM*. The InnoDB engine is now the default for MySQL, and it's an excellent fail-safe in sensitive situations.

The **--log-slow-queries** option in MySQL can be used to help you catch detrimental queries.

If you have administrative-level control over your database, you should know about a number of configurations that impact MySQL's performance. To start, there's **back_log**, **key_buffer_size**, **max_connections**, and **thread_cache_size**. You can use a configuration file to change these settings from their defaults to values more appropriate to your server and site. See the MySQL manual for more information for the version of MySQL that your server is running—assuming that you have that kind of control over your server, of course.

Should you get to a point where your site is so active that multiple servers are appropriate, you can consider *replicating* the database. Database replication stores the same data on more than one server, with that data automatically synced among them. By adopting replication, you'll get improved security, reliability (if one server fails, the data still lives on elsewhere), and performance.

Programming

The primary focus of this book is the PHP programming, where PHP acts as the glue between the user/browser and, well, pretty much everything else: the database, email, payment systems, and more. From a programming perspective, you'll want to create code that's not only functional but also reusable, extendable, and secure.

To make reusable, extendable code, it must be well organized and thoroughly documented. I can't stress this enough: Document your code to the point of overkill. As you program, begin with your comments and revisit them frequently. When you make any changes to your code, double-check that the comments remain accurate. You should also use flowcharts, UML (Unified Modeling Language) diagrams, and other tools to outline and represent your site in graphical and noncode ways.

Formal PHP documentation can be achieved using phpDocumentor (www.phpdoc.org).

note

Because this book is one giant comment on entire sites of PHP code, the scripts displayed in the book won't be as documented as yours should be.

The security of your code is based on so many factors that the next chapter will start discussing just this one subject. Secure programming is even more critical in e-commerce sites, however, so the topic will be reinforced time and again throughout the entire book.

Depending on the circumstances, you may also want to look into version-control software such as Subversion (http://subversion.tigris.org) or Git (http://git-scm.com). Version-control software makes site updates a smoother process, allowing you to accurately implement all site changes or roll back problems to previously sound states. If you're developing a site with a team of people, version control will almost certainly be mandatory.

With PHP, unlike with many other languages, you have a choice of using an *object-oriented* or *procedural* approach. I'm perfectly comfortable doing either, and I don't believe one approach is clearly better than the other. I advise against buying into the myth that object-oriented programming (OOP) is more extendable or secure than procedural code. Poorly written OOP will cause you endless headaches, whereas well-written procedural code won't hamper your site's long-term development in any way.

When asking for reader input on this book, there was a moderately heated discussion as to which approach I should use and to what extent. Some feel that OOP is the hallmark of professional programming; others don't know or care for it and wouldn't get much value out of an OOP-based book. In the end, I decided to use a mostly procedural approach, as it's the common denominator of all PHP programmers, and procedural code can more easily be turned into OOP than vice versa. New in this edition of the book, however, is Chapter 16, "An OOP Example." It demonstrates how key components of an e-commerce site would be programmed using OOP.

Similarly, there was some discussion as to whether I should incorporate a framework. Again, my heart is not set one way or the other on frameworks. Sometimes I use them; sometimes I don't. In the end, I decided against using any framework in this book, because those chapters would inherently be more about the framework than the underlying example—an e-commerce site, the real focus of the book.

All that being said, when it comes to your own projects, you'll need to make the decision on procedural versus object-oriented, frameworks or not, and if using a framework, which one. Know up-front that these decisions will neither adversely affect nor guarantee the success of your e-commerce site. The only thing you want to avoid doing is starting off on one path only to later change course. That's a recipe for frustration and a likely guarantee of disaster.

Testing

Testing your website isn't a onetime, standalone step, but rather something you'll need to do often. You can't test your site too much! Unfortunately, it's hard for the site developer to perform a truly good test of the site: He created it, so he knows how it should work and uses it accordingly. A better test is what happens when your family, coworkers, and annoying friends give the site a whirl. And I specify the *annoying* friends, because they're the ones who will attempt to do things you never would have imagined. When these people, who aren't web developers themselves, purposefully or accidentally misuse the site, what happens? From these experiences you can improve the user interface and security of the whole application. Improving those two things will go a long way toward a successful e-commerce venture. Still, there are steps you can take to effectively test your site yourself.

Relatively new to PHP is the concept of *test-driven development* and *unit testing* (although unit testing generally requires that you're using OOP). You define concrete and atomic tests of your code, and then run the tests to confirm the results. Each test should be concise and clear. As you write more code, you define more tests and continue to check each test to ensure that what you just did didn't break any other functionality. Test-driven development and unit testing are big enough subjects that I recommend you research both further on your own, when you're ready.

A different type of site testing you could address is *performance*. If you want to start with the big picture—how well the server copes with demand—software like ApacheBench (https://httpd.apache.org/docs/2.2/programs/ab.html) and Siege (www.joedog.org/index/siege-home) will run benchmarks on your web server, reporting on how many requests can be handled per second. Requests per second (RPS) is the standard measuring tool for a site's performance. Once you start checking your site's performance, you'll find that big, systemwide changes you make will have the greatest impact. These include

- Changing the server hardware: increasing memory, installing faster hard drives, and using faster processors
- Changing the demands on the server: disabling unnecessary features, putting fewer users or sites on a single server, and balancing loads across multiple servers
- Caching the PHP output
- Caching the PHP execution
- Caching the database results

tip

Look online for specifics on implementing any of these caching techniques.

If you think about the process involved for handling the request of a PHP-MySQL based page, you'll see three areas where caching can be applied (**Figure 1.4**). First, if the database or PHP is caching the results of a database query, then that query won't need to be executed with each request. Second, by default, each request of a PHP script requires that the PHP code be executed as if it had never been run before. By applying an *opcode* cache such as the Alternative PHP Cache (APC; www.php.net/apc), the PHP code itself is cached by the system, making that execution faster. Finally, the end result is that HTML is sent to the web browser. If you can cache the dynamically generated HTML, then no PHP code will be executed at all, no database queries are required, and the request itself becomes as fast as a request for a static HTML page.

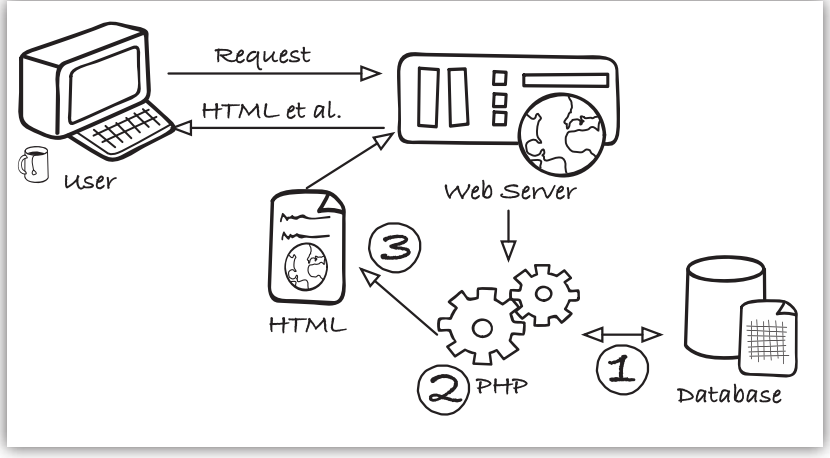

Figure 1.4

You can also spend some time profiling your code, using tools like Xdebug (www.xdebug.org) or the Advanced PHP Debugger (APD; www.php.net/apd) to see where potential bottlenecks are in the PHP itself. Bottlenecks usually occur when PHP interacts with the filesystem, whether that means literal files on the server (like reading and writing text files or using sessions) or through the database application. I caution you against spending too much time worrying about profiling individual sections of code, because the improvements you can make that way will be relatively minor and possibly not worth your time. Better to learn good programming habits so that you don't have to worry about profiling after the fact.

Going Live

Once a site has been completely developed and tested, then updated to include the latest bug fixes and customer requests, it's time to go live. Before doing so, you should revisit all the legal and security issues to make sure the site is in full compliance. Second, have a plan in place for what should be done when something goes wrong (notice I said *when*, not *if*). Third, if any assumptions were made in the code, or any dummy processes installed, remove those. By "assumptions," I mean things such as using the test version of the payment system, not requiring real authentication to the administration pages, and so forth.

Brick-and-mortar stores normally have what's called a *soft opening*. During this period, the business is open and fully functioning but not promoting itself actively. The hope is that the arrival of some traffic will catch issues and allow for improvements, without attempting to do so under the burden of a full user base. This is something you may want to consider as well, although in truth, pretty much every website that doesn't have millions of dollars of advertising behind it has a soft opening.

Maintaining

Depending on the situation, going live may or may not be the end of your involvement with the project. If it's not, such as when it's your site, you'll need to have a plan in place for maintaining the project. Site maintenance begins and ends with creating good, frequent backups of your site's data. This is something the hosting company should be doing for you (check when you are researching hosts) and something you should be doing as well. Make sure that backups are kept in multiple locations, too, so that a natural or man-made disaster doesn't wipe out both your server and your backups. Keep in mind that laws will likely apply if you're storing customer information in your backups (which, presumably, you will be).

The maintenance of a site also requires that you keep an eye on the data itself. Check and optimize your database tables to improve their performance. Watch your database logs for slow and underperforming queries. Review your web server logs for *file not found* errors, high loads, and potential security problems. Analyze your data to find sales trends and places where you can make improvements. In short, collect and examine as much information as you can. And keep making backups!

tip

For security purposes, safely store your backups, such as in a locked safe or a bank deposit box.

note

The running of the site—dealing with customers, handling inventory, processing orders, reviewing customer feedback, and so on—is a whole separate way that a site has to be maintained.

Improving

The final step in the development process is improving what you've created. Improvements may stem from the client, from customer feedback, or from changes in available technologies. Improving a site is a subroutine of this entire development process: Think about what you want to change, plan its implementation, mock up the design, retool the database, write the code, test the end result, go live, and maintain the updated version of the site.

Although it's best to treat the development process as a linear progression of discrete steps, when you factor in repeated places for feedback, and the high potential for the process to be revisited for improvements, the real design process is best represented by **Figure 1.5**.

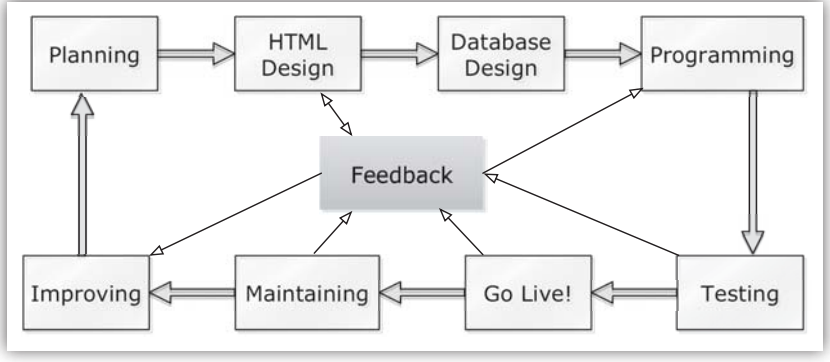

Figure 1.5

This page intentionally left blank

2 SECURITY FUNDAMENTALS

Although every chapter in the rest of the book will include recommendations for improving the security of your website, security is such an important subject that this chapter focuses on it alone. There are three broad topics:

- Exploring general theory and background information
- Creating a secure environment
- Recognizing and combating common vulnerabilities

Some of the topics discussed here will be implemented in real-world code in subsequent chapters. A few of the other recommendations are steps to implement a single time. And a handful of tips will apply only if you have administrative-level influence over the server. Still, it's only by grasping the whole picture that you can implement security on a high level.

SECURITY THEORY

Before getting into security specifics, let's think about what it means to be secure. I want to start with two simple, but perhaps heretical, ideas:

- No website is completely secure.
- Maximum security isn't the goal.

These two statements probably sound so absurd that I've lost all credibility, but in no way am I saying that security isn't important. In fact, when it comes to e-commerce sites, security is the most vital criterion. I'm just saying

that you may need to think about security differently than you currently do. I'll explain....

No Website Is Secure

The first fact you have to accept about any type of security is that security isn't a binary thing, where a site, application, or computer is either secure or not. Security is measured on a spectrum (**Figure 2.1**). The code, software, environment, people involved, and other factors move the security rating up and down that scale. No matter what you know or do, you'll never create a website that's *absolutely secure*; the only thing you can do is attempt to make it *more secure*.

Figure 2.1

I've had people—well, one person—say this approach is wrong and dangerous, but I think quite the contrary is true. When you begin to believe that your site is absolutely secure, that's when it's the most vulnerable, because you've let your guard down. What you should be doing is taking steps so that your site is *secure enough*.

note

Accepting that you can't create absolute security doesn't mean you shouldn't try but rather that you should never stop trying.

As an analogy, think about a car. If you drive somewhere and get out but don't lock the car, it still may be relatively secure, depending on the time of day, the type of car, the area in which it's parked, and the length of time you'll leave it there. Just leaving a car unlocked doesn't mean it's guaranteed to be broken into, just as locking it doesn't mean it won't be. It's certainly harder to break into a locked car, but it's not impossible. If you leave the car in your garage, it's much, much less likely to get broken into, until you leave the garage door open or someone breaks into the garage. And, of course, never taking a car out of the garage defeats the whole point of having a car.

The same is true for a website. Doing X, Y, and Z will make it harder—but not impossible—to break into, because there's always a potential flaw just around the corner. Even if the server isn't turned on and is sitting in a locked hosting cage somewhere, people who work for that hosting company can still access the machine. Simply put, there's nothing you can do to guarantee absolute security.

I say that no website is completely secure for two reasons. First, I want to promote eternal vigilance when it comes to your site's security. *Complacency is dangerous*. The second reason is...

Maximum Security Isn't the Goal

Again, this may sound blasphemous, but it's really not. You have to first accept that security comes at a cost. Making something—pretty much anything—more secure requires more time and money. Also, anything that's more secure is inherently less usable and, in terms of computers, slower (**Figure 2.2**).

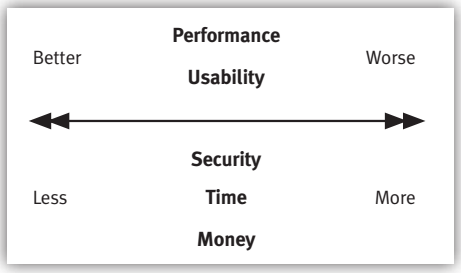

Figure 2.2

tip

The success of a site will increase its risks, because the extra attention will make it a bigger target for hackers.

Returning to my car analogy, if you live in a city, you likely lock the car when you drive it somewhere and park. But when you park it in your garage, you probably don't lock it. The same might be true if you live in a small town or if your car is a total beater. In some situations, maybe you use a secondary antitheft device, like a steering wheel lock. What you're doing with your car, consciously or not, is adjusting the security measures in place based on the perceived level of risk and the potential loss (for example, an expensive car versus a cheap one).

The same is true of websites: Different types of sites require different levels of security. A site on which I list my favorite books is at a different point on the security spectrum than one that stores user information. Even that is on a less critical plateau than a site that handles credit cards. Even beyond that high-security level are sites for online banking, sensitive government and/or military data, and so forth (**Figure 2.3**).

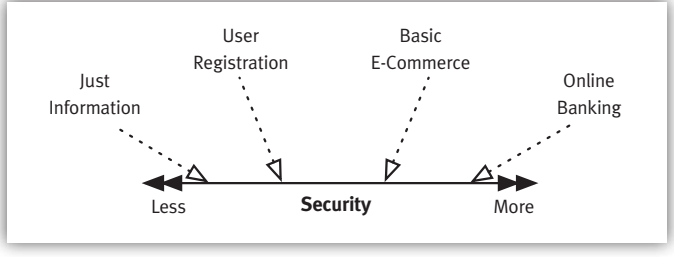

Figure 2.3

The goal, then, isn't to implement the highest level of security but rather the *highest level of security that's appropriate for the site*. Let's look at two examples of what I mean.

You may already know that Secure Sockets Layer (SSL) is an essential part of the e-commerce system. SSL provides the first line of defense for protecting user-submitted information; when the time comes to take the customer's credit card, SSL must be used. But this doesn't mean that SSL must be used for every page on your site. SSL puts a strain on the server, and only a fraction of SSL requests can be handled simultaneously compared to non-SSL requests. As a compromise between security and performance, you may choose to use SSL only for the checkout process and use a non-SSL connection for the bulk of your site.

note

As server capabilities have improved, the performance hit for using SSL has decreased.

As another example, a shared host is going to be less secure than a dedicated host simply because more people have access to, and more software is running on, the server. On the other hand, a shared host will cost a tenth or less of what a dedicated host costs. You can increase the security by purchasing a more expensive hosting plan, but that may not be necessary, let alone prudent.

All this being said, I don't want you thinking that I'm cavalier about security or that you should be. In this chapter, you'll learn the fundamentals for creating a secure website, but it's not reasonable to think that your site has to be secure to the *n*th degree. There are many baseline recommendations—in terms of code and server environments—that you should ideally implement for any site. But you'll be presented with plenty of choices for which you must weigh all the pros and cons before coming to a decision. The goal is to hit the appropriate mark on the security spectrum for the given situation (as in Figure 2.3). Then, give your site a nudge just a wee bit to the right on that spectrum, just to be safe.

tip

When making security decisions, always err on the side of being overprotective.

Security for Customers

E-commerce sites and websites in general have a client-server relationship: two parties equally participating in an event. There are two sides to security as well: one you implement as the site developer and/or server administrator and one the customer is aware of. Now let's take a couple of pages to talk about security from the customer's perspective.

tip

The success of an e-commerce site partly depends on a customer's comfort in spending money there.

There's an old expression that says cleanliness is next to godliness. My house wouldn't suggest that I live by that expression, but it leads me to an analogy I have for website security:

Security Is Next to Godliness.

Think of security the way you might think about cleanliness. Say you go to eat at a restaurant. The restaurant may or may not *look* clean, and it may or may not *be* clean. But if the restaurant doesn't look clean, then it probably isn't

actually clean, and you don't want to eat there. The same goes for a website's security: If it doesn't give the appearance of being secure, it probably isn't secure, and potential customers won't want to use the site (and shouldn't). So how does a site look secure to the lay user?

- It's professional in appearance.
- It's honest and transparent with respect to what the business is, what its policies are, how customer information will be used, and so on.
- It uses SSL.
- It doesn't do anything that may make the customer feel the site isn't secure.

tip

Having friends and family test your site is a good way to get feedback on potentially confusing or problematic parts.

This last quality is the most important, as the common person may not know the difference between a secure-looking and unsecure-looking site. A successful e-commerce site gives customers every reason to complete their sale and absolutely no reason not to. If a customer goes to a site and sees technical error messages, alerts from their browser (for example, because of poor JavaScript or improper use of SSL), and so forth, she'll likely (or hopefully) take her business elsewhere. If the website does something that makes the customer say "Huh?," think of it like seeing a rodent scurry across the restaurant floor: It's time for the customer to go.

The second part of this analogy is that though it's important for a restaurant to *look* clean (so people will eat there), it's more important that it's *actually* clean (so that patrons don't get sick, so that the inspector doesn't shut it down, and so on). Your website must be secure so that nothing bad can happen to the customers or your client.

tip

If you're creating a site for a client, put a plan in place so that someone continues to maintain the site's security after you're done with the project.

The final reason I believe this analogy works is that it also supports the two maxims I already put forth. Security, like cleanliness, isn't an absolute, and the amount of effort you put into it should depend on the situation. The place where a restaurant keeps its garbage doesn't need to be that clean, but the kitchen sure does. Maybe you're the kind of person who would thoroughly clean monthly, weekly, or daily. Maybe you're the kind who will take cleaning to the disinfecting level. There's no right answer in these situations: There's better and there's worse, and there's what's right for you and your situation. The same goes for security. Most importantly, just because you cleaned today doesn't mean it will stay clean forever. And the website that went live today without any issues could become vulnerable tomorrow, even if that's through no fault of your own.

PCI REQUIREMENTS

In Chapter 1, "Getting Started," I mention that you'll need to be aware of *PCI compliance*. Compliance means abiding by all 12 requirements outlined in the PCI DSS. Depending on your level of involvement in the e-commerce project, some of these may not be applicable to you personally, but you should still be aware of them and pass them along to those who are responsible.

Taken verbatim from www.pcisecuritystandards.org, the requirements are as follows:

Build and Maintain a Secure Network

Requirement 1: Install and maintain a firewall configuration to protect cardholder data.

Requirement 2: Do not use vendor-supplied defaults for system passwords and other security parameters.

Protect Cardholder Data

Requirement 3: Protect stored cardholder data.

Requirement 4: Encrypt transmission of cardholder data across open, public networks.

Maintain a Vulnerability Management Program

Requirement 5: Use and regularly update anti-virus software or programs.

Requirement 6: Develop and maintain secure systems and applications.

Implement Strong Access Control Measures

Requirement 7: Restrict access to cardholder data by business need to know.

Requirement 8: Assign a unique ID to each person with computer access.

Requirement 9: Restrict physical access to cardholder data.

Regularly Monitor and Test Networks

Requirement 10: Track and monitor all access to network resources and cardholder data.

Requirement 11: Regularly test security systems and processes.

Maintain an Information Security Policy

Requirement 12: Maintain a policy that addresses information security for all personnel.

If you go to the PCI website, you can download a 70-plus-page PDF that explains each of these regulations in more detail. The document also

note

Extra precautions apply if wireless technology will be used on your business's internal network.

note

Depending on the level of PCI compliance that applies to your business, you may be required to perform annual validation tests.

discusses how to test each condition and provides a worksheet to annotate your results. You should read this PDF at some point, but I want to add a few notes of my own here.

tip

The PCI DSS has lots of useful security recommendations, applicable to even non-e-commerce sites.

First, some of these rules, such as using a firewall and antivirus software, may be beyond your role and server authority, but they still need to be done. In fact, you should use a firewall and antivirus software on any server. Changing the default passwords is also a must, but the second requirement goes well beyond just changing passwords, into areas such as disabling unnecessary software.

As for requirements 3 and 4, the best advice I can give is *not to store credit card information at all,* but if you do, ratchet your security up many, many levels. Also know that there are key pieces of data that you're not allowed to store, such as the card verification value (CVV) or its PIN. Storing credit card information is not for the beginning developer or the small business, so please design your site and use payment gateways in such a way to relieve you of that burden.

Requirement 6—develop and maintain secure systems and applications—is what this book is all about. That's a big topic whose bottom line is to program securely.

Requirements 7–9 are impacted by both the business and the hosting company. But one rule you can use in just the programming facet is requirement 8: providing unique identifiers to each administrator. Unless you'll have only one person administering your site, create a system and get in the habit of defining multiple users with appropriate permissions. In this book's first e-commerce example (paid access to content), one administrator type might only be able to manage the site's content. Another administrator might be able to do that and access the noncommercial customer data. The highest level of access might also allow for viewing payment data (in this case, a record of payments made, not the actual customer information charged for payments). Extra security can be achieved by forcing regular password changes, adding password requirements (length, use of capital and nonletters), and disabling inactive accounts.

The remaining three requirements are ongoing tasks, necessary even if none of the site's code changes. I talk about this some in the first chapter—keeping a close eye on the server to catch something bad, and having a plan in place when it does.

These 12 requirements are excellent and appropriately encompassing. When you read the full PCI DSS document, you'll get tons of specific recommendations, each of which will improve your site's security that much more.

Some people feel that the PCI DSS is too demanding, although I think it's better to overdo security than to take risks. An opposite complaint about the PCI DSS

is that it can fool people into thinking that their site or system is secure just because they've abided by these requirements. Remember that the PCI DSS establishes a *baseline* for the *minimum* you must do toward improving security. If your situation warrants, there are always more steps you can take.

SERVER SECURITY

Most of the rest of the book will address security as affected by the PHP code you write, but let's first look at many of the server-based factors that play into the overall security of your e-commerce site. The approach to server security is simple:

1. Deny
2. Authorize
3. Record

You should first deny everybody and everything you can. Then allow limited capability only after proper authorization and authentication. Finally, record pretty much everything so that you know what people might be trying to do (but failing) and what they did do.

tip

A public website page has an opposite security model: anyone and everyone is allowed to view it.

Hosting Implications

The biggest question with respect to server security will be the hosting of the site. A shared host will be less secure than a VPS or dedicated hosting plan just by the virtue of having more people with access to the server itself. Further, any hosting that gives you some administrative-level control over the server *can* be more secure, as you'll be able to customize how the server runs and better lock it down. I would argue, however, that unless you're an expert in server administration, using a managed server is better than trying to do it all by yourself. Whatever your situation, do your best to limit the number of people who have physical and network access to the server.

note

An organization's own employees can be a weak security link. Be aware of the potential for "inside jobs."

Per the PCI DSS, the server should also be running a firewall and an antivirus program. The antivirus program has to be kept up to date if it's to be any good at all. But the same can be said for all the software on the server. Quite frequently, security holes are introduced when a design flaw (a bug) is found in common applications, like the Apache web server, DNS, or an email system. Upgrading and patching these tools when new versions are released is a key component to your server's security.

tip

Subscribe to software mailing lists so that you're contacted when new versions are released.

And, of course, you should create very secure passwords for accessing the server and change them regularly.

Finally, use the server's logs to track who accesses a site and when. This way you have a record of who could've done something bad. You may also want to be notified (via email or text message) when anyone logs into the server at all.

PHP and Web Security

A secondary level of security is controlled by how the web server (for example, Apache, nginx, IIS, and so on) and PHP are configured. First, keep both up to date, along with any related software. Specific security issues will depend on the web server in use, so research and stay abreast of security factors surrounding your particular web server. By learning more about Apache, for example, you'll find that the `DocumentRoot` directive, which limits the web server to working only with files found within that directory, can be a great security asset.

tip

The PHP manual lists other security recommendations depending on PHP's relationship to the web server (CGI binary versus Apache module).

You can make a number of adjustments to affect how PHP runs. View the current settings by invoking the `phpinfo()` function (**Figure 2.4**). You'll find each setting listed with two columns: Local Value and Master Value. The Local Value column indicates settings that are being overridden within the current directory.

Configuration

PHP Core

Directive	Local Value	Master Value
allow_call_time_pass_reference	On	On
allow_url_fopen	On	On
allow_url_include	Off	Off
always_populate_raw_post_data	Off	Off
arg_separator.input	&	&
arg_separator.output	&	&
asp_tags	Off	Off
auto_append_file	*no value*	*no value*
auto_globals_jit	On	On
auto_prepend_file	*no value*	*no value*
browscap	*no value*	*no value*
default_charset	*no value*	*no value*
default_mimetype	text/html	text/html
define_syslog_variables	Off	Off

Figure 2.4

tip

Don't leave a `phpinfo()` script publicly available on your server. It displays too much information about your server!

To modify how PHP runs, edit the `php.ini` configuration file. The `phpinfo()` function also reveals its location. You can modify some PHP settings within a PHP script, although in terms of security, it's best to make such changes on a global basis (otherwise, if you forget to make the change in any given script, you'd create a security hole). After making changes to the web server or PHP configuration, restart the web server to enact those changes.

In terms of specifics, you should start by using the `open_basedir` setting, which limits the directories from which PHP can open files. If you set this

value to your web directory, or the parent of the web directory, malicious PHP code can't be used to read important system files located in other places.

On a similar note, you should, if you can, take advantage of non-web directories as a place to store sensitive information. For example, your URL, www.example.com, might point to the actual server directory **/var/www/username/htdocs**, so that loading http://www.example.com/home.php executes (through the web browser) **/var/www/username/htdocs/home.php** (**Figure 2.5**). In this case, the **htdocs** folder is called the *web root directory*. Anything placed within that directory is theoretically accessible via the HTTP protocol. For example, the **image.png** file stored in the **images** subdirectory is available via http://www.example.com/images/image.png. Anything stored above the **htdocs** directory—**/var/www/username**, the parent of the web directory—isn't available via HTTP (and, therefore, not available over a network using a browser). Files and folders placed there can still be accessed by PHP running on the server, but they cannot be directly accessed remotely via HTTP.

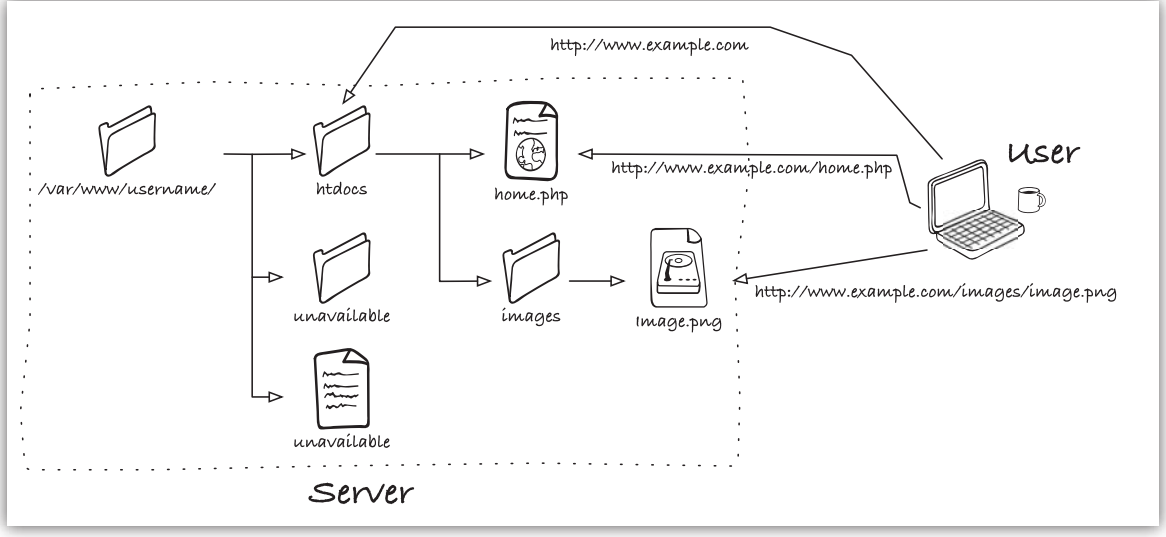

Figure 2.5

How errors are handled can often undermine the security, and the professionalism, of a website. A common attack is for hackers to supply problematic data in the hopes that generated errors will be revealing. To safeguard against this type of attack, start by developing your site under the error level of **E_ALL | E_STRICT**. By doing so, any potential problem will be reported to you via email or a log. Then, before the site goes live, disable **display_errors** so that no PHP problem will be shown to the user. Instead, use a custom error