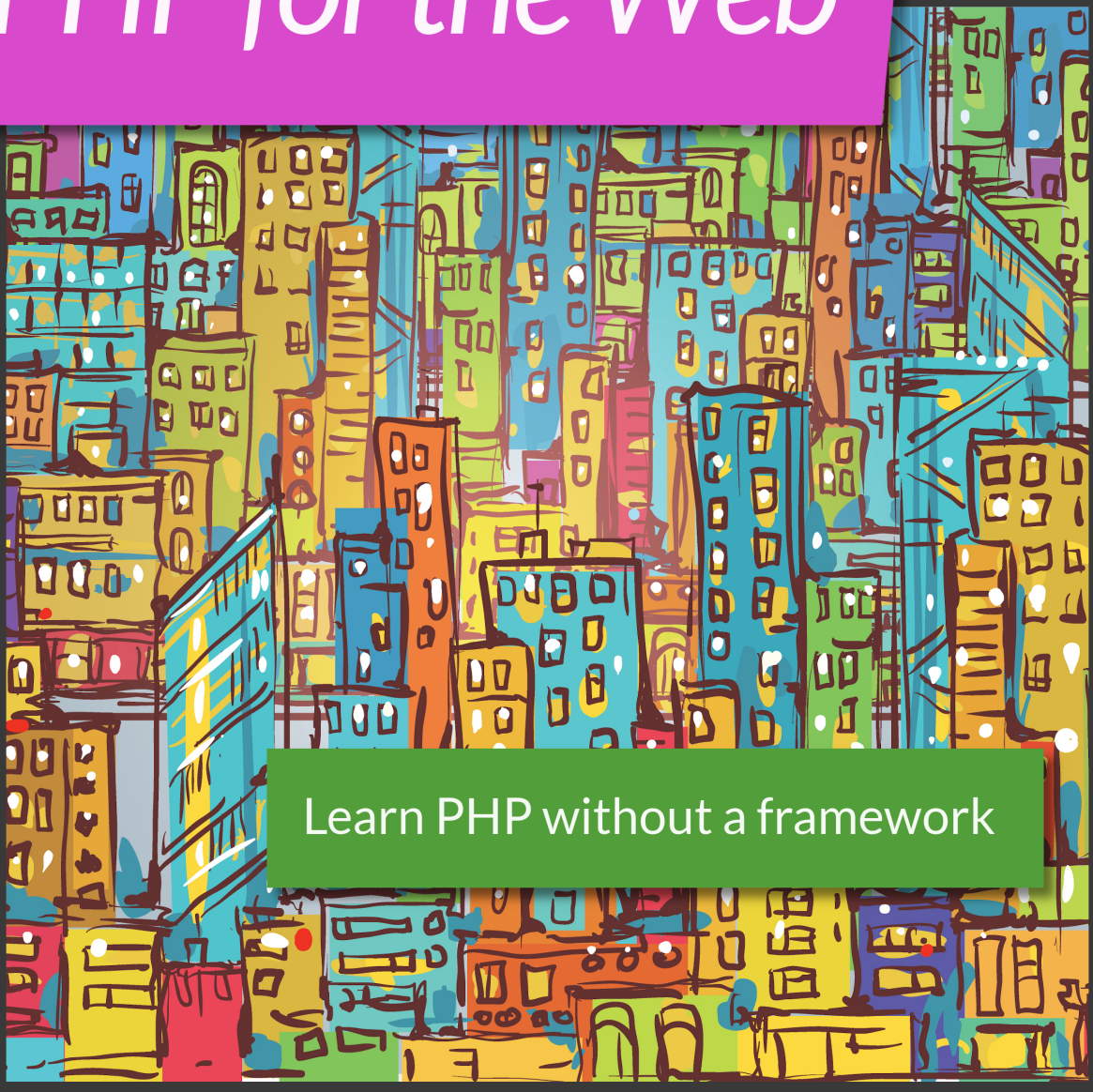
PHP for the Web



Learn PHP without a framework

Matthias Noback

PHP for the Web

Learn PHP without a framework

Matthias Noback

This book is for sale at
http://leanpub.com/learning-php-for-the-web-without-a-framework

This version was published on 2022-09-29

For Liesbeth, thanks for the inspiration!

Contents

Introduction

There are many books for people who want to learn PHP. For me the biggest issue with most of these books is that they combine too many topics, leading to many hundreds of pages of things you have to work your way through. They start with basic PHP programming, then teach you about creating websites using a relational database, how to send emails from PHP, how to create command-line applications, etc. It's all relevant and interesting, but it's way too much if you ask me. I thought it would be helpful to have a book that only covers PHP and the web.

Narrowing down the topic of this book to just "PHP for the web" wasn't enough. There is already plenty of interesting material about building web applications with PHP. Most of this material is about how to use a framework like Symfony or Laravel to build web applications. But I think it's important to learn about all the details that a framework is hiding for you. When you know how something works, you will be better at troubleshooting issues. So in this book we're not going to use a framework.

However, without a framework your application will never be as good as it can be. You'll be fixing issues that frameworks have fixed long before you. You'll also introduce security issues that could have been prevented by using a framework. So any of the code that is in this book should be considered for educational purposes only.

In the final chapter of this book, we'll discuss what kind of code you should write in a real-world application and where to learn more about that.

Who should read this book?

This book should be interesting for beginning PHP developers who want to learn all the aspects of using PHP to create dynamic websites. I'm going to assume a small amount of PHP programming knowledge:

- Variables and assigning values to them

- Control structures like `if`, `else`, `foreach`, `for`
- Strings and string concatenation (using `.`)
- Integers
- Expressions and comparisons (`<`, `>`, etc.)
- Associative arrays (arrays with string keys and value) and indexed arrays (arrays where only the values are relevant)

I think this book will also be useful for people with experience in other programming languages who want to learn about PHP and how it deals with the web (as opposed to Java, Python, etc.).

Besides PHP, I'm relying on a basic knowledge about HTML. We'll only use a few basic elements but if you don't know any HTML yet I recommend reading MDN's Introduction to HTML[1] and Web forms - Working with user data[2].

Getting started

Before we start with the main content of the book I'm going to explain what you need to learn PHP for the web. We'll look at the software that should be installed on your computer. I'll also provide a list of the topics that I assume you know at least something about.

Bash

The first thing you need is Bash. We're going to use it to work with the command-line. On Linux and Mac you'll have it installed already. To use Bash, open the *Terminal* application.

On Windows I recommend installing Git[3], which comes with Git Bash. Once you've installed it, open the *Git Bash* application to use it.

You should see a blinking cursor. Type in the following, and press *Enter*:

[1] https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML
[2] https://developer.mozilla.org/en-US/docs/Learn/Forms
[3] https://git-scm.com/downloads

```
bash --version
```

You should now see some information about the Bash version that you have on your computer. The blinking cursor indicates that you can type in your next command.

Some tips for working with Bash

Copy and paste shortcuts (`Ctrl + C`, `Ctrl + V`) usually don't work with Bash. If you want to copy or paste values in the *Terminal*, right-click on the *Terminal* window and you should see a menu. This menu contains options for *Copy* and *Paste* that you can click on. There are usually alternative keyboard shortcuts like `Ctrl + Shift + C` and `Ctrl + Shift + V` that should work too.

If you press `Up` you'll see the previous command you ran. You can press `Up` multiple times to go further back in the history.

From now on, when I say something like "Run" or "Type at the command-line", I mean: open the terminal or Git Bash, and run a command, just like you ran `bash --version`.

PHP Runtime

You'll also need a PHP Runtime installed on your computer. **Please note: you don't need a web server like Apache, Nginx, or IIS. We'll only use PHP from the command-line.**

On Linux you may install it using `apt` or any other preferred package manager. On Mac you may use something like `brew` to install it. For Windows you may follow the instruction in Appendix A

To verify that you have installed and can use PHP from the command-line, run the following command:

```
php -v
```

It should show you some information about the current version of PHP.

An IDE that works well with PHP

I won't say you have to buy and install PhpStorm[4] but in my opinion, that will be your best choice. If you have experience with some other IDE that works well with PHP code, please go ahead. Things you should look for in an IDE:

- It should show programming mistakes
- It should have a code formatter
- It should offer suggestions while typing code ("code completion")

Firefox

One last thing we'll need is the web browser Firefox[5]. This doesn't have to become your default browser, but it will be easier to understand what I'm talking about if we both use the same browser.

An overview of the contents

A brief overview of the contents, before we dive in:

Chapter 1 shows how we can use PHP's built-in server to serve *static* resources: HTML files, pictures, CSS files, etc. In Chapter 2 we serve our first PHP script. Using PHP we can build in some dynamic aspects to our pages. Chapter 3 demonstrates the use of HTML forms, whose data can be processed by a PHP script. We talk about the difference between `GET` and `POST` requests. Chapter 4 covers cookies, and how they can be used to store some data in the browser and pass it to the next request. In Chapter 5 we use a special cookie, the *session* cookie, to store data on the server instead of in the browser. Combining techniques from all previous chapters, we build an authentication system in Chapter 6. Arriving at Chapter 7 it's time to restructure the project a bit, and make it easier to work with in the following chapters. Next we're going to build a full CRUD interface for creating, updating, and deleting tours that we're going to show on our website. We need two chapters for that, Chapter 8 and Chapter 9. To make everything look a bit nicer we're going to add the ability

[4] https://www.jetbrains.com/phpstorm/
[5] https://www.mozilla.org/en-US/exp/firefox/new/

to upload a picture for every tour in Chapter 10. In Chapter 11 we'll build a custom solution that catches PHP errors and exceptions and shows a proper error page for them. Chapter 12 shows how to describe and test all the features you've built in this book using an automated test runner. Finally, we'll reach the point where you've learned the basics and are ready to move on to the next book or course. I'll provide some suggestions for further learning in Chapter 13.

The source code

You can find all the code samples from this book on GitHub[6]. At the end of each chapter I've committed the current state of the project. You can look at the code in the browser or if you want you can create a copy of the project on your own computer by running:

```
git clone git@github.com:matthiasnoback/php-for-the-web.git
```

You can check out the state of the project at the end of every chapter by looking up the chapter in the list of commits[7]. Copy the corresponding commit hash (e.g. `b596da2`) from the list and run:

```
git checkout b596da21d8af0afe91f549efa2eb98da203eeeaa
```

Acknowledgements

Feedback and suggestions

Changelog

31 January 2021

[6] https://github.com/matthiasnoback/php-for-the-web
[7] https://github.com/matthiasnoback/php-for-the-web/commits/master

1. Serving resources

Serving an index.html file with the built-in web server

The simplest way of creating a web application is by "serving" `.html` files. In this chapter we're going to do that using PHP's built-in web server.

First, create a directory for your project. This could be anywhere on your computer. For example, I've created a directory called `/home/matthias/Projects/php-for-the-web`. Open this directory in your IDE (preferably PhpStorm). Now create a file in your project directory called `index.html`. In that file, paste the following HTML:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Index</title>
    </head>
    <body>
        <h1>This is the index</h1>
    </body>
</html>
```

On the command-line, navigate to your project directory using `cd` (or if you're on Windows, go to your project directory, right-click on it and select "Open Git-Bash here" in the menu that pops up).

```
cd /home/matthias/Projects/php-for-the-web
```

Now we can start PHP's built-in web server[1]:

[1] https://www.php.net/manual/en/features.commandline.webserver.php

```
php -S localhost:8000
```

In the terminal you should see something like this:

```
[...] PHP [...] Development Server (http://localhost:8000) started
```

This means that the PHP Development Server (a.k.a. the built-in server) has started, great! The server is "listening" on `localhost:8000`. That's because we told PHP to do so using the `-S` command-line option:

```
php -S localhost:8000
```

To see if the server works, open a browser and go to http://localhost:8000. You should see "This is the index":

This is the index

Adding a favicon

Security announcement: The project root should not be the document root

Communication between the browser and the server

Summary

Quiz

2. Serving PHP scripts

In Chapter 1 we've looked at serving static resources, in particular `index.html` and `favicon.ico`. These are the things a browser can deal with: it can show an HTML page, and use the `favicon.ico` as a nice visual icon inside the browser tab. However, if all we could do was return `.html` files and images to the browser, we could never build an actual web application. An `.html` file can't process data that a user provides by filling in a form. An `.html` file can't talk to a database and produce a list of available products. An `.html` file can't remember that it's me and say "Hey Matthias, welcome back!". That's because an `.html` file is a *static* resources. If the browser fetches it the second time, it will be the same file as when it fetched it the first time.

In this chapter we're going to create *dynamic* resources. When the browser fetches a dynamic resource, it may get a different response the second time it fetches the same resource.

Let's make such a dynamic resource now by creating a new PHP file called `random.php` and put it inside `public/`. Copy and paste the following code into `random.php`:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Your lucky number</title>
    </head>
    <body>
        <h1>Your lucky number is: <?php echo random_int(1, 10); ?></h1>
    </body>
</html>
```

Now let's try to run our new `random.php` script. Because we created the file inside `public/`, `random.php` is inside the document root and therefore accessible from the browser. Go to http://localhost:8000/random.php and you should see:

Your lucky number is: 7

Your lucky number is 7

Or some other number actually, because we've used the `random_int()` function[1] to generate a random number. Refresh the page and you'll see that it produces other numbers too.

By the way, if your browser tells you that it's "Unable to connect", your PHP server isn't running yet. In that case, open the *Terminal*, go to your project directory and run:

```
php -S localhost:8000 -t public/
```

[1] https://www.php.net/random_int

The response: status, headers and body

Building up a response

Linking to other pages

Passing values between requests

Security announcement: user input can’t be trusted

Summary

Quiz

3. Forms

In Chapter 2 we've seen how you can let the PHP server serve PHP scripts. We also saw how you can provide input to these scripts using query parameters, which are part of the URL. This works great if the input is something the application produces, like the random number in `/random.php`. But if we want the user to provide the input themselves, we'd better use an *HTML form*. A form is much more user-friendly since the user doesn't have to modify the URL manually. They can just type something in a form field, select an item from a combobox, put a checkmark somewhere, and submit the data to the server.

Submitting form data as query parameters

We're going to add a simple form to the `/kittens.php` page. The form has a field for providing a number. Once you submit the form, the form data will be sent as query parameters.

First, add the form to `kittens.php`, just before the closing `</body>` tag:

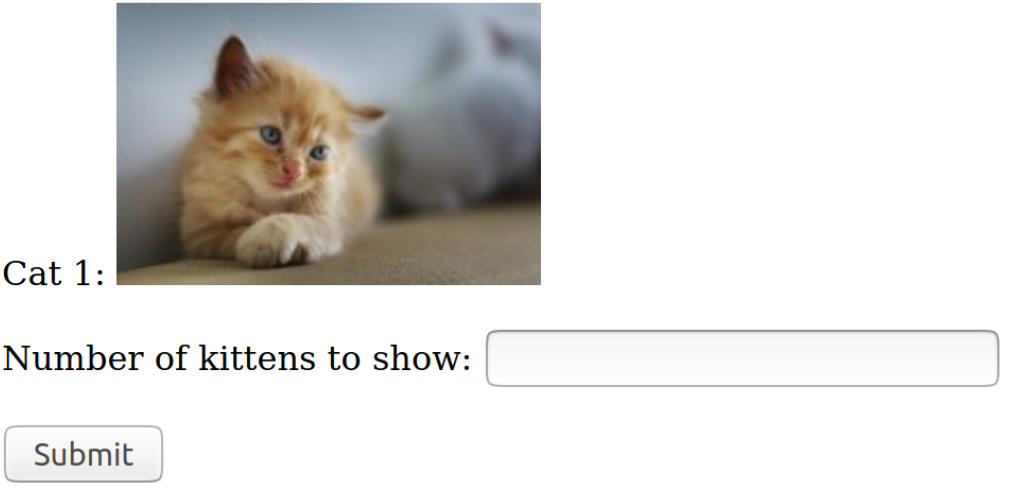
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Kittens</title>
</head>
<body>
<?php

$numberOfKittens = isset($_GET['number']) ? (int) $_GET['number'] : 1;

// ...
?>
    <form>
```

```
        <div>
            <label for="number">
                Number of kittens to show:
            </label>
            <input name="number" id="number">
        </div>
        <div>
            <button type="submit">Submit</button>
        </div>
    </form>
</body>
</html>
```

Now go to http://localhost:8000/kittens.php and you should see the form at the bottom of the page:



The new form on `/kittens.php`

Fill in a number and submit the form. You should now see the same number of pictures on your screen. Nice!

But why is that? Take a look at the URL: if you filled in 4, the URL is now http://localhost:8000/kittens.php?number=4 Exactly what we wanted. There's only

one usability issue: after submitting the form the number field is empty. That's inconvenient, because to correct a mistake you have to type in the whole number again instead of just making the change to the number you already provided.

Security announcement: Always use output escaping

Adding a select element to the form

Submitting data via the request body

Summary

Quiz

4. Cookies

HTTP is what they call a *stateless* protocol. The server processes a request, and when the response has been delivered, the server forgets all about that request and starts processing the next request. There is no link between these requests, not even a way to know if a request comes from the same person as the previous request. Unless you find a way to pass data between requests. This is what *cookies* are for.

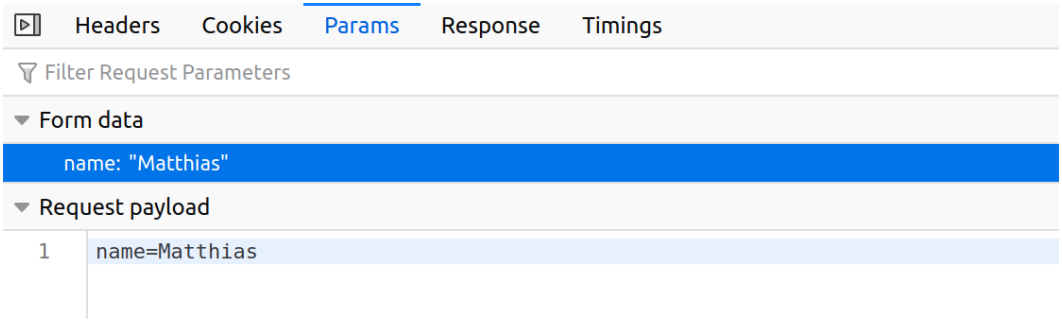
Say we want to know the user's name, and show it on every page they visit. We start by providing a page where the user can enter their name. Create a script called `name.php` in `public/` and copy-paste the following code into this file:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Name</title>
</head>
<body>
<form method="post">
    <p>
        <label for="name">
            Your name:
        </label>
        <input type="text" name="name" id="name">
    </p>
    <p>
        <button type="submit">Submit</button>
    </p>
</form>
</body>
</html>
```

Go to http://localhost:8000/name.php and you should see this simple form:

The new form asking for the user's name

Open the *Network* panel, fill out the form, and submit it. In the list of requests you should see a new `POST` request for `name.php`. When you select it and open the *Request* tab, it shows that the name is submitted correctly:



The submitted form data

Setting a cookie

Using a cookie

Set-Cookie is a response header, Cookie a request header

Redirecting after processing a `POST` request

Security announcement: cookies can be manipulated without you knowing

Summary

Quiz

Challenge

5. Sessions

In Chapter 4 we saw how you can use cookies to pass data between requests. We ended with a warning that cookies can be manipulated, because they are headers, so they can't really be trusted. Another security aspect is that the contents of cookies are visible. Both of these aspects of cookies make them unsuitable for sensitive information, or information that only you as the programmer should be able to manipulate.

For situations where using a cookie isn't a good idea, but you still want to keep some kind of information between requests, you can use a *session*. Cookie data is managed by the browser, session data is managed by the server. This gives you full control over the data, and prevents it from being visible elsewhere, or being manipulated without you knowing. Let's see how this works.

We're going to rewrite the `name.php` script to store the user's name in the session, not in a cookie. The first thing we should do in the script is start the session using the `session_start()`[1] function. Session data should be saved in a superglobal variable called `$_SESSION`, which is an associative `array`. In this case we assign the value of `$_POST['name']` to `$_SESSION['name']`:

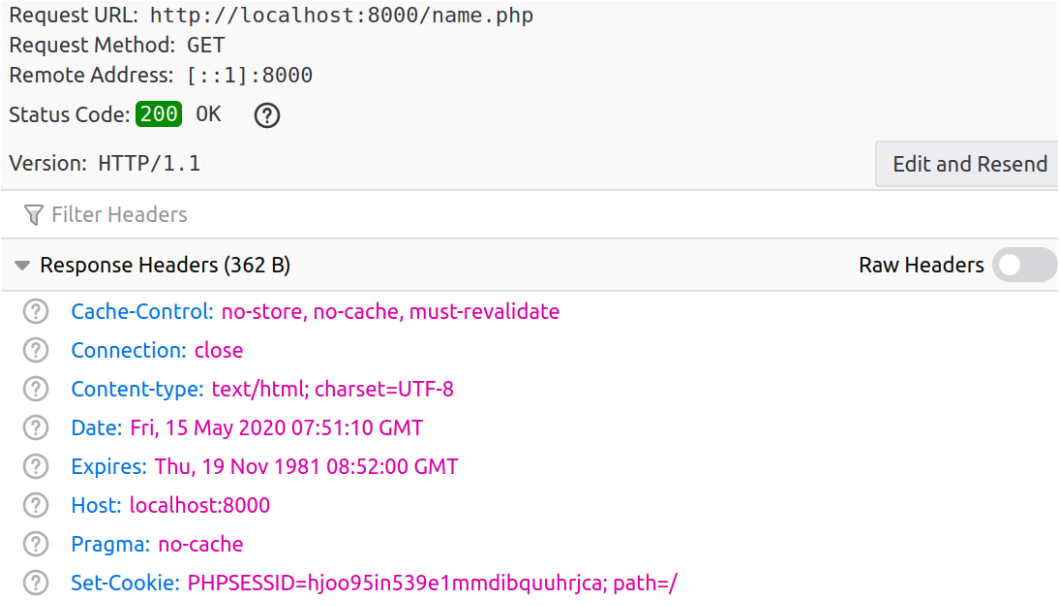
```
<?php
session_start();

if (isset($_POST['name'])) {
    $_SESSION['name'] = $_POST['name'];
    header('Location: /random.php');
    exit;
}
?>
```

Let's see what happens. First, open the *Storage* panel and remove all existing cookies, just so we don't get confused. Now open the *Network* panel and go to http://localhost:

[1] https://www.php.net/session_start

8000/name.php. In the list of requests you should see a `GET` request for `name.php`. Select this request and take a look at the *Response Headers* section of the *Headers* tab:



Response headers of the `GET` request for `name.php`

Session files and serialized data

Flash messages

Using flash messages everywhere

Summary

Quiz

6. Authentication

A web application usually has:

1. A public part
2. A part for known users who are logged in
3. A part that's only accessible for administrators

Public pages will be accessible by so-called *anonymous* users. For other pages, like a list of previously created orders, or a page where the user can edit their profile, they need to authenticate first. Authentication means we establish the identity of the user. After providing their username and a password, we know that it's them, not someone else. After all, nobody else is supposed to know someone else's password; it's a secret. As you know, this assumption is where a lot of security risks are: the user could have their password written on a post-it next to their monitor. Or they could have shared their credentials with their grandson. Or they could have a password that is easy to guess. There are many things to consider, and I definitely recommend you doing that, but it's beyond the scope of this book. I can highly recommend OWASP as a source of security-related instructions for programmers. For the topic of authentication, check out their Authentication Cheat Sheet[1].

[1] https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

A secret page

Setting up a login form

Validating the username and the password

Logging out

Summary

Quiz

7. Project structure

Header and footer snippets

Along the way, you may have noticed some duplication in the code we copied and pasted in several places. For starters, we've been duplicating most of the basic HTML structure (`<!DOCTYPE html><html...`) every time we created a new page. What's inside `<body>...</body>` will be different for every page, but the rest of the HTML can be included from a shared snippet. Let's start by moving the common HTML code out of `login.php` into reusable snippets. First, create a file in the root directory of the project (not in `public/` since this is a snippet, not a page) and call it `_header.php`. In this file, we're going to put the shared code for the top part of the HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Login</title>
</head>
<body>
```

We're going to do the same for the bottom part of the HTML. Create a file called `_footer.php` and put the following code in it:

```
</body>
</html>
```

Now go back to `login.php` and include both of these files in the right place:

```php
<?php
include(__DIR__ . '/../_header.php');
?>
<form method="post">
    <div>
        <label for="username">
            Username:
        </label>
        <input type="text" name="username" id="username">
    </div>
    <div>
        <label for="password">
            Password:
        </label>
        <input type="password" name="password" id="password">
    </div>
    <div>
        <button type="submit">Submit</button>
    </div>
</form>
<?php
include(__DIR__ . '/../_footer.php');
```

Passing variables to snippets

Flash messages revisited

Bootstrapping

From .html to .php

Adding navigation

Adding a stylesheet

Routing

Summary

Quiz

Challenge

8. CRUD part 1: Create

In this chapter we'll combine all of the previously discussed techniques by creating something many applications have: a CRUD interface.

CRUD stands for "Create - Read - Updated - Delete". And although at some point you'll talk to people who don't like it at all, it can be a powerful way of looking at your application's abilities. In the following sections we're going to expand our online business. We're going to become a serious travel agency which offers tours from Amsterdam to all kinds of interesting cities nearby. The first thing our business needs is to show a list of all the available tours. Also, since we think traveling should be for everyone, we're going to take special care to make most of our tours accessible.

Before we can show a list of tours on the website, we first have to define them somewhere. We could hard-code them as an array:

```
$tours = [
    [
        'destination' => 'Berlin',
        'number_of_tickets_available' => 10,
        'is_accessible' => true
    ],
    // and so on...
];
```

But if we do this, we'd have to redeploy the website every time we want to add a new tour, edit an existing one, or remove one. This is where CRUD comes in: we'd want to offer the user to make all of these changes on the website itself. A user who is logged in and is known to be an administrator should have the ability to edit a tour, add a new one, or delete one. Normal users should be able to list tours and select a tour to take a closer look at its details. Most web applications would use a database to store this kind of data, but since this book doesn't cover databases, nor assumes any prior experience with them, we're going to store the data in a file for now.

Saving JSON-encoded data in a file

Adding a tour

Form validation

Showing the submitted data in the form

Listing tours

Summary

Quiz

Challenge

9. CRUD part 2: The rest

We've successfully built a form that enables the user to add a tour to the list of tours. The next step will be to allow the user to edit the tour, in case they made a typo, or they just want to change part of its data. Before we can do that, we need a way to identify the tour and say: I want to edit this tour, not that other tour. So whenever we create a tour and add it to the `$toursData` array, we should also add an `id` property to give it a unique identifier (ID). A unique ID could be a number that hasn't been used before. My suggestion would be to count the number of tours we have in `$toursData` and add one to it. If we use the resulting number as an ID, the first tour would have ID 1, the second 2, and so on. As long as we don't delete elements from the array this ID will be unique. Let's modify `create-tour.php` so it will set an ID for every tour:

```
if (count($formErrors) === 0) {
    // Provide a unique ID for this new tour:
    $normalizedData['id'] = count($toursData) + 1;

    $toursData[] = $normalizedData;
```

If you didn't delete your `tours.json` at the end of the previous chapter, now is a good time to do it. Existing data in `tours.json` doesn't have an ID, so to ensure that all tours will have an ID we need to get rid of the existing ones. Or, if you like, you can also modify the JSON data and manually add IDs to the existing tours.

Now let's see if the code actually works. Go to http://localhost:8000/create-tour, fill out the form and submit it. Take a look at `tours.json` and you should see that the first tour has `"id": 1`:

```
[
    {
        "destination": "Berlin",
        "number_of_tickets_available": 10,
        "is_accessible": true,
        "id": 1
    }
]
```

Create some other tours, and you should see that they each get their own unique ID.

Introducing some reusable elements

Editing tour data

Deleting tours

Summary

Quiz

Challenge

10. File uploads

What's missing in the catalog of tours is a nice picture of the destination. We'd like to allow administrators to upload a picture and show that picture on the detail page of the tour (which, by the way, we don't have yet). Allowing users to upload files to the server is even more of a potential security risk than any other thing we did before. People could upload all kinds of bad files: files that look like pictures but are programs, pictures that show inappropriate things, crash the browser, etc. However, since only an administrator should be able to upload pictures we don't have to assume the worst.

We need several ingredients that we'll add one by one:

1. The create and edit form should be prepared for uploading a file.
2. After the form has been validated we should take the uploaded file and move it to a place inside the document root.
3. We should save the filename in `tours.json` as well.
4. We should then show the image on the details page of a tour.

Adding a details page

First, let's prepare the details page for the tour. This is where we show all the information we have about a tour, including the picture. For now, we only have the destination, the number of available tickets, and whether or not the tour is accessible. So let's start with that information. We assume that the tour ID is provided as a query parameter. Then we can use the existing `load_tour_data()` function to load the data for this specific tour. In `pages/` create a new file called `tour.php` and copy/paste the following code into it:

```
<?php

include(__DIR__ . '/functions/tour-crud.php');

include(__DIR__ . '/../bootstrap.php');

if (!isset($_GET['id'])) {
    header('Location: /list-tours');
    exit;
}

$tourId = (int)$_GET['id'];
$tourData = load_tour_data($tourId);

include(__DIR__ . '/../_header.php');

?>
<h1>Tour to <?php
    echo htmlspecialchars($tourData['destination'], ENT_QUOTES);
?></h1>
<p>This tour is <?php echo $tourData['is_accessible']
    ? 'accessible'
    : 'not accessible'; ?>.</p>
<p>There are <?php
    echo htmlspecialchars(
        $tourData['number_of_tickets_available'], ENT_QUOTES
    );
?> tickets available.</p>
<?php

include(__DIR__ . '/../_footer.php');
```

To make this new page accessible we should add it to the `$urlMap` in `index.php` too:

```
$urlMap = [
    '/create-tour' => 'create-tour.php',
    '/list-tours' => 'list-tours.php',
    '/edit-tour' => 'edit-tour.php',
    '/delete-tour' => 'delete-tour.php',
    '/tour' => 'tour.php',
    // ...
];
```

Uploading a file

Processing the file upload

Showing the uploaded picture

Replacing the existing image

Form validation for file uploads

Summary

Quiz

11. Error handling

As soon as we started using a PHP server to serve `.php` scripts in Chapter 2 we had to worry about errors and showing them in the browser. I mentioned back then that you need to make a distinction between the website as it is still running on your own computer and the website as it is running on a publicly accessible server. You may find that people talk about this distinction in different ways. When you're working on your website on your own computer you're running it "locally" or on your "development server". When it runs on a publicly accessible server it has been "deployed" to the "production server". We use different words here because these are different contexts or environments and there will be some differences in server configuration and behavior of the website depending on whether it runs locally or on the production server. In this chapter we'll improve the way our website handles errors and we'll make this dependent on the environment in which the website runs.

Producing an error

Before we can improve error handling, let's create a file that produces an error, so we can see how our website handles it. Create a new script in `pages/` called `oops.php`. Also add it to the `$urlMap` in `index.php` so we can open the page in the browser:

```
$urlMap = [
    '/oops' => 'oops.php',
    // ...
];
```

This isn't going to be a real page, and we should remove it later, but we just need a place where we can freely produce errors. The first type of error we have to deal with is an *exception*[1]. You can use an exception to indicate that the script can't do what it was asked to do. We already saw one back in Chapter 9 where the function

[1] https://www.php.net/manual/en/language.exceptions.php

`load_tour_data()`. The function "throws" an exception when it is asked to load data for a tour that doesn't exist:

```
function load_tour_data(int $id): array
{
    $toursData = load_all_tours_data();

    foreach ($toursData as $tourData) {
        if ($tourData['id'] === $id) {
            return $tourData;
        }
    }

    throw new RuntimeException('Could not find tour with ID ' . $id);
}
```

In `oops.php` we'll also throw an exception to see what that looks like for a user:

```
<?php

throw new RuntimeException('Something went wrong');
```

Start the PHP server if it isn't already running:

```
php -S 0.0.0.0:8000 -t public/ -c php.ini
```

Then go to http://localhost:8000/oops. You should see the following:

Fatal error: Uncaught RuntimeException: Something went wrong in /app/pages/oops.php:3 Stack trace: #0 /app/public/index.php(21): include() #1 {main} thrown in **/app/pages/oops.php** on line **3**

Fatal error: Uncaught RuntimeException

The reason the error shows up on the page is because we have set the PHP setting `display_errors` to `On` in our custom `php.ini` file. We loaded this file using the `-c` command-line option.

Seeing error messages on the screen is very useful for a developer like yourself: it will help you fix issues quickly. But it would be quite embarrassing if this showed up in the browser of an actual visitor of the website.

Using different configuration settings in production

PHP errors

Summary

Quiz

12. Automated testing

I'm always disappointed when book authors mention test automation only in the last chapter, but here we are. Let's talk about automated testing.

Testing is a very important part of a developer's life. So far we've been making changes in the code and "testing" it by going to the browser. This is called *exploratory* testing, and you need a person for that. In this final chapter I'd like to show you a technique that let's the computer do the testing. By doing so you can save yourself a lot of time clicking around the website.

What I think is even more useful about automated testing is that you can save the tests and run them whenever you like. When you're programming, a change in one file might cause a problem in another file. Running the tests after every change will expose such problems. This way tests become a safety net. That's why they are sometimes called *regression tests*: they will help you keep going forward.

Using Composer to install testing tools

Before we can write tests and run them we should first install some tools. There are some excellent libraries available that you can use for the automated testing of websites. Most frameworks offer their own tools for this too. In this case, I'd like to use PHPUnit[1] in combination with Panther[2]. You can only install it using Composer which is a tool for installing PHP packages (including libraries and frameworks) in your project. In order to use Composer you first have to install it on your computer. In the *Terminal* make sure that you are in the root directory of your project. Then follow the setup instructions on Composer's website[3]. When you're done you should have a `composer.phar` file in the root of your project. Now we can install PHPUnit and Panther:

[1] https://phpunit.de/
[2] https://github.com/symfony/panther
[3] https://getcomposer.org/download/

```
php composer.phar require --dev \
   phpunit/phpunit symfony/panther symfony/css-selector symfony/mime
```

It may take some time, but the output should look something like this:

```
Using version ^0.7.1 for symfony/panther
...
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 15 installs, 0 updates, 0 removals
  - ...
  - Installing symfony/panther (v0.7.1): Downloading (100%)
...
Writing lock file
Generating autoload files
```

In your project you'll now have a `vendor/` directory (if you use Git for version control, make sure that `/vendor/` is in your `.gitignore` file). The `vendor/` directory is where all the third-party code can be found (so far Panther, PHPUnit and its dependencies). You'll also see a `composer.json` file in the root directory of your project. This file contains a list of all the packages that you have declared as dependencies of your project. Yes, we'll need a lot of packages just to start creating automated tests. But that's because all these packages make a lot of complicated things really easy for us.

A first test

Creating our first browser test

A test for the pictures page

Starting with a clean slate

Troubleshooting and suggestions

Summary

Quiz

Challenge

13. Conclusion

Object-oriented programming

Frameworks

Testing

Parting words

14. Appendix A: Installing PHP on Windows

15. Appendix B: Answers to the quiz questions

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 9

Chapter 10

Chapter 11

Chapter 12

16. End of the sample file

Thanks for downloading the sample file! I hope you liked it and of course I hope you'll buy the full version of the book. Use this link to get it for only 9 dollars: http://leanpub.com/learning-php-for-the-web-without-a-framework/c/SAMPLE

You can reach me on Twitter (@matthiasnoback[1] or send an email to info@matthiasnoback.nl[2].

[1] https://twitter.com/matthiasnoback
[2] mailto:info@matthiasnoback.nl