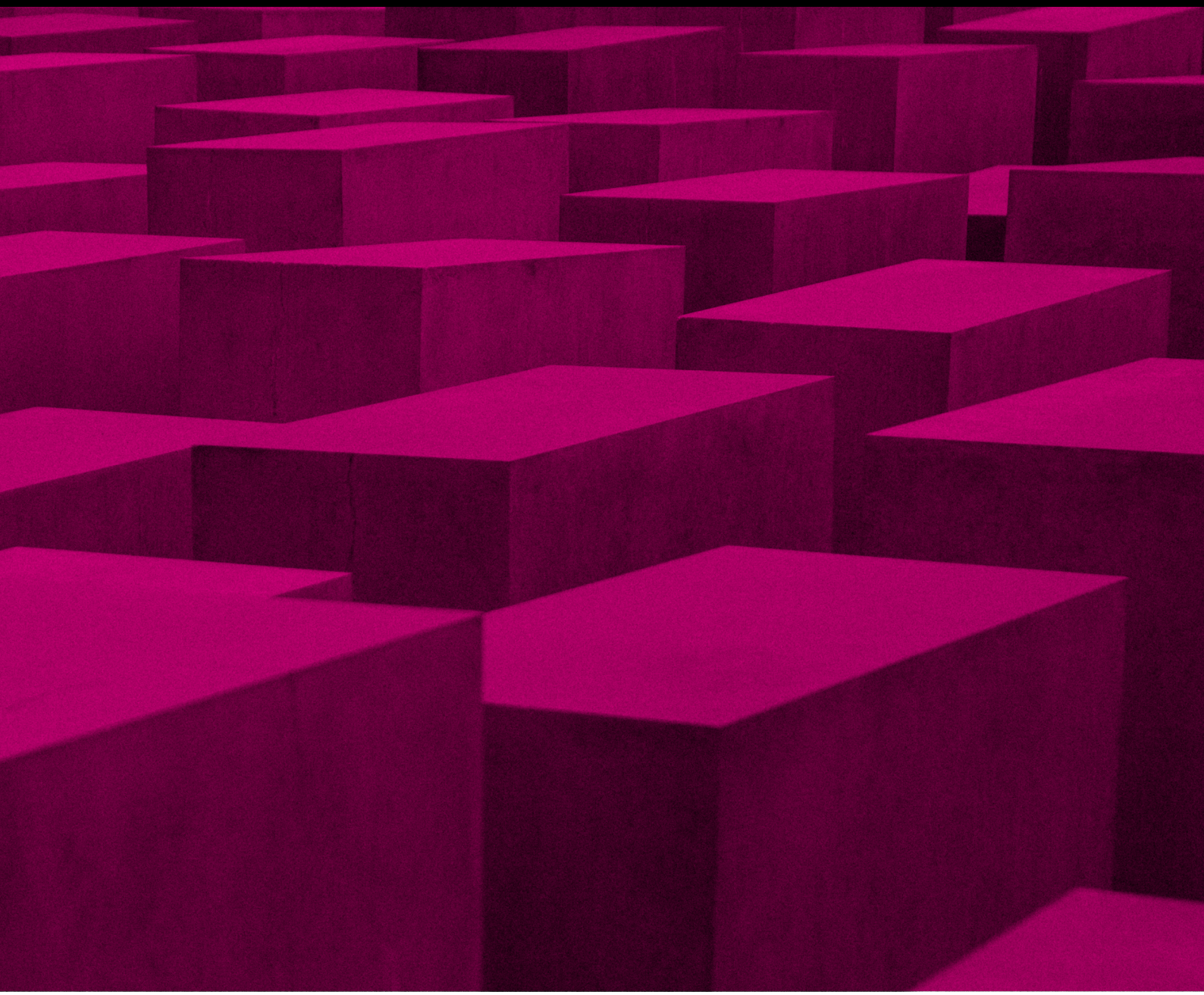

PHP Framework in One Weekend

Build your own web framework with modern PHP 7 in one weekend.

Darren Mothersele

First Edition, May 2016

Copyright ©2016 Ideation Networks Ltd.

All Rights Reserved. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher. For information, please contact Ideation Networks info@ideationnetworks.com

Although the author and publisher have made every effort to ensure the accuracy and completeness of information container in this book, we assume no responsibility for errors, innacuracies, omissions, or any inconsistency herein. Any slighting of people, places, or organizations is unintentional.

Cover photo by Little Visuals is CC0 (public domain) licensed.
`https://stocksnap.io/photo/67EAB2F9C2`

All source code listings from this book are released under the The MIT License (MIT). Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

Preface **4**

0 Getting Started **7**

1 Managing Complexity **8**

2 Testing **11**

3 HTTP **13**

4 Templating **16**

5 Content **20**

6 Routing **23**

7 Controllers **31**

8 Dependency Injection **37**

9 Design and Layout **47**

10 Reuse **57**

Where Next? **62**

Links **64**

Preface

"a weekend, the fundamental unit of coding self-improvement"

Peter Shirley

I recently enjoyed working through Peter Shirley's *Ray Tracing in One Weekend*[21]. Over the course of a few chapters, Peter guides you through the process of building a basic ray tracing application. It's not a comprehensive tool, but in one weekend you cover enough material to learn the fundamental concepts. Peter says he included "the most primal set of these assignments so that it's very doable in a weekend, the fundamental unit of coding self-improvement."

About this project

I have followed Peter's approach here, and will guide you through a short project that should be easily doable over the course of a weekend. The project is based on a 2 day training course I have run with small groups of PHP developers. I have refined the content to a simple set of examples, each one building on the last.

It is not intended to be a tool you can immediately go and use on real websites, but as a starting point to learn the basic concepts used in modern PHP web applications. It is also not intended to be an introductory course in web technology. Some familiarity with HTML, CSS, JavaScript, and PHP is assumed.

Another objective of this course is to teach modern PHP. There are a lot of old books, tutorials, and guides that teach PHP that would not pass code review today. In this project we will aim to use modern PHP best practises. But, one weekend is not enough time to cover the whole of modern PHP. You are encouraged to read through PHP The Right Way[16], for a more detailed overview of the modern way to use PHP.

This weekend project is about exploring modern PHP and the ecosystem of PHP packages and libraries. Over the course of this weekend you will learn some of the fundamental concepts that modern PHP apps are built on. Projects such as Drupal, phpBB, Laravel, eZ Publish, Joomla!, Magento, Piwik, and many more are using Symfony Components[25] as a foundation on which to build. We will use these, and more, to build our own PHP Framework.

About the author

I wrote and delivered my first course in PHP at the University of Sheffield in 2001. After a brief stint working at a record label, I co-founded a digital music startup and was CTO until 2007 when I went on to start Ideation Networks. As a Drupal specialist, I provided software development and training for Drupal agencies, large Drupal deployments and flagship Drupal websites. Most recently, this has included helping developers switch from Drupal 7 to Drupal 8, and teaching modern PHP.

Conventions

Text in `monospace` font is either a snippet of code, or a command to be entered at the command line. I use iTerm2 on OSX, but any environment where PHP runs should work.

I use the `$` character to indicate the command prompt. If you see a line that begins with this character, followed by a space, then it is a command to be entered at the terminal.

```
$ pwd
/Users/darren/dev/book
```

Any lines in the shell block that do not begin with a `$` character are output returned by the command. You should expect to see this appear in your terminal as a result of running the preceding command.

Other uses of `monospace` font include code examples, PHP keywords, filenames, classes, functions, and commands.

You will see partial code examples, included inline, like this:

```
$name = $_GET['name'];
```

You will also see full code listings of entire source files. These will include PHP, CSS, JavaScript, Twig templates, and JSON configuration files,

for example listing 1. The caption alongside a code listing is provided to indicate which file it relates to. The line numbers on the left of the source code indicate which lines from the source file are quoted. As you can see in this example, lines 2-6 have been provided. A listing may list the whole file, or it may list a selected number lines only that highlight the issue being discussed.

```
2   require __DIR__.'/../vendor/autoload.php';
3   $name = "World";
4 ?>
5 <!DOCTYPE html>
6 <html lang="en">
```

Listing 1: public/index.php

Note The source files are provided as examples. They are meant to illustrate the concepts being discussed. You are expected to write the code yourself. Please don't copy and paste. Read the code, understand it, but then write your own version of it. You will learn more if you write code yourself, improvise, and adapt to your liking. You are welcome to use the provided code for whatever means you wish, but it is only supposed to be used as a reference.

If you get stuck check the GitHub repository[23] or contact the author[2].

Chapter 0

Getting Started

You should use the latest stable version of PHP. I am using PHP 7, but this project should work without issues on PHP 5.6. You can check the installed version of PHP with the `php -v` command.

```
$ php -v
PHP 7.0.6 (cli) (built: May 12 2016 15:13:35) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

If you are on OSX you can update and install PHP using Homebrew[10]:

```
$ brew update
$ brew install homebrew/php/php70
```

We will use an application called *Composer* to download PHP packages, and make them available to our PHP code.

If you don't already have Composer installed you can find installation instructions on the Composer website[5]. You can install Composer locally to this single project, but I prefer to install it globally. I will assume you have done the same and made the `composer` executable available on your *PATH*. If you have installed it differently, you may need to update the example commands.

```
$ composer
```

If you run the command above you should see some basic help about Composer, it's options, and a list of available commands.

Chapter 1

Managing Complexity

A lot of what we do this weekend will seem like overkill. This is because the examples are simple in order to demonstrate the concepts in use on more complex systems. As projects get bigger, and complexity increases, you need ways to manage complexity. In web development, we tackle complexity by layering.

When you write code you write instruction *statements*. These *statements* are grouped into *functions* (or *methods*) which are grouped into *classes*, which are combined into *systems*. Once this all gets so big you can't fit it in your head, you can't reason about it. We need to keep the design of the system as simple as possible.

In order to simplify design, we split things into separate chunks. These chunks have many names, such as modules, packages, components, and libraries. The aim is to make them as self-contained as possible. When a component's dependencies are minimised we say it is *loosely coupled* (as apposed to *tightly coupled*). We group all the related components (i.e. those that share dependencies) so that we can think about them independently from the rest of the system. It often looks like we've split the design of the system into layers. We call this process layering.

In modern PHP we talk about *packages* a lot. These are separate blocks of reusable code with minimal dependencies and a clean, well-defined API. When we create our own apps we prefer to reuse functionality provided by packages, rather than create our own. Packages vary in size from a few functions, through to entire sub-systems.

A framework is a collection of components that work well together. Traditionally the components in a framework will have been designed and bundled together to fulfil a specific type of problem. Increasing, thanks the wide open-source ecosystem of PHP packages, we're able to pull together components from different sources and have them work together easily.

The packages that our project depends on are defined in a configuration file called `composer.json` which exists in the root folder of our codebase.

Use the Composer `init` command to create a `composer.json` file interactively. You will be prompted for some details about your project and asked if you want to define dependencies interactively. Respond with `no` to skip adding dependencies (require) and dev dependencies (require-dev) for now.

```
$ mkdir php-framework
$ cd php-framework
$ composer init
```

You should now have a project folder called `php-framework` with a bare Composer config file (`composer.json`) that looks a bit like this:

```
{
    "name": "darrenmothersele/php-framework",
    "authors": [
        {
            "name": "Darren",
            "email": "darren@darrenmothersele.com"
        }
    ],
    "require": {}
}
```

Listing 1.1: composer.json

The `require` section is where we will list all the packages that our project depends on and what versions we require. Composer will download them all into a `vendor` folder, and create an *autoloader* to make them available in our PHP code.

Autoloading

PHP doesn't need to load up all your app and dependencies on every request. Each request will use only a small fraction of the code. In modern PHP we split our code into multiple files, usually one class per file, and then only include the files we need to deal with the current task in hand. To make this process easier for the developer PHP has an autoloading mechanism. It means you don't have to specify which files to include for each request. PHP will use the autoloader to work out which files to include when they are needed. Composer creates an autoloader for us based on the configuration in our `composer.json` file.

Use the composer `install` command to download dependencies and cre-

ate the autoloader.

```
$ composer install
```

We have not specified any required packages yet, so nothing is downloaded to our project. Composer just creates the autoloader in the `vendor/composer` folder. We load this in our app by including the `vendor/autoload.php` file.

Hello World

Now let's get a simple page up and running. In modern PHP we keep our source code separate from our web accessible resources. Create a `public` folder into which we will put all publicly accessible files. In this folder create an `index.php` file as per listing 1.2.

```
<?php
  require __DIR__.'/../vendor/autoload.php';
  $name = "World";
?>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>Hello <?php print $name; ?>!</h1>
</body>
</html>
```

Listing 1.2: public/index.php

Then use PHP's built in web server to view the content:

```
$ php -S localhost:4000 -t public/
```

Now if you go to `http://localhost:4000/` in a browser you should see "Hello World!".