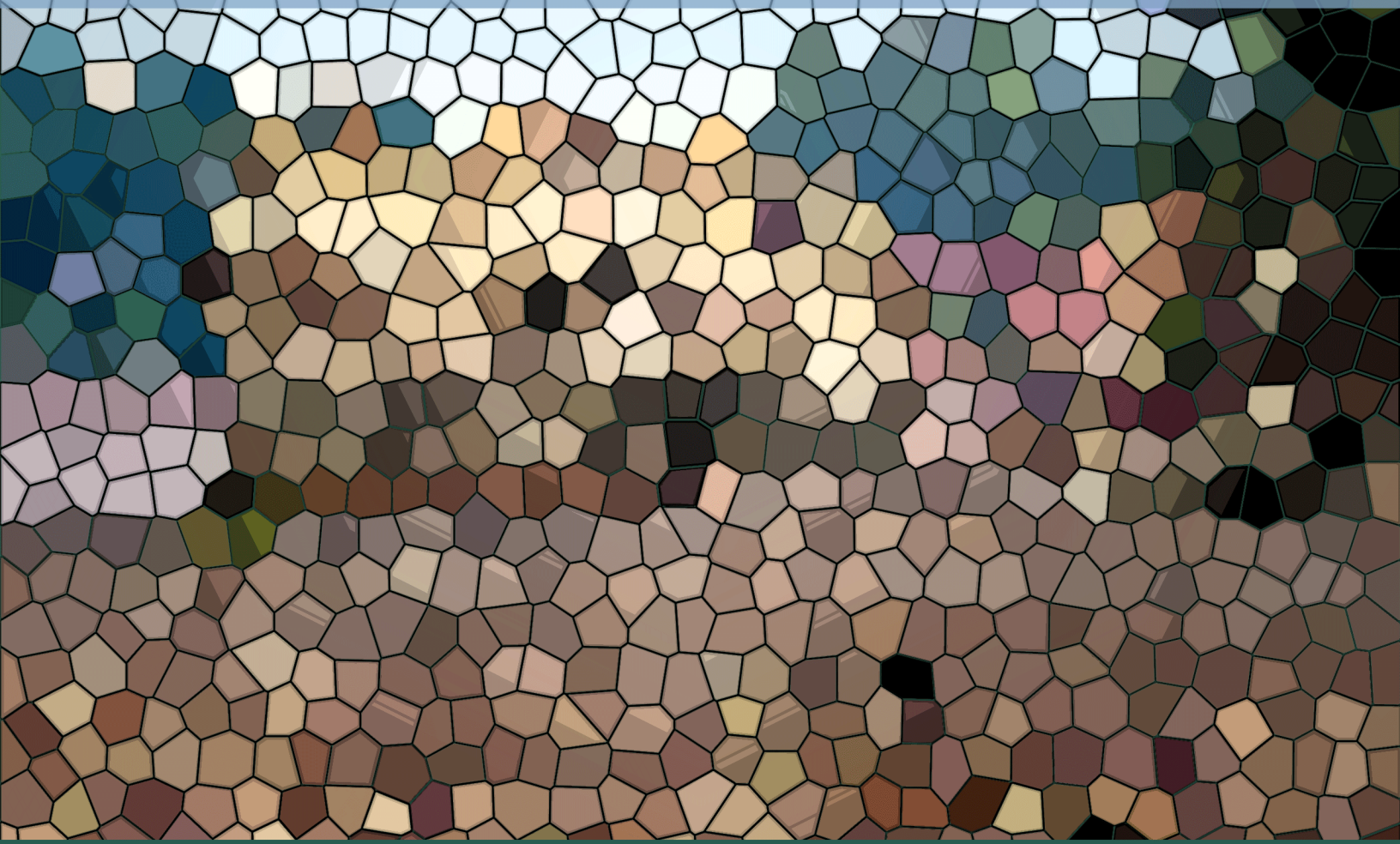
# BUILDING WEB APPS with SILEX 2



by KEVIN BOYD

# Building Web Applications with Silex 2

## ... and other useful PHP libraries, too.

Kevin Boyd

This book is for sale at http://leanpub.com/silexwebapps

This version was published on 2017-02-08

*For Kieran & Declan*

*Go Forth And Code*

# Contents

# Introduction

Building web applications is easier than ever these days - or is it? A growing number of technologies are involved in creating even a simple web application. The choices can be overwhelming, and that's before considering popular methodologies like Test-Driven Development (TDD), Continuous Integration, and Cloud Computing.

This book introduces you to **Silex 2**, a PHP-based microframework. You will be taken on a fabulous journey of amazing wonderment, guiding you through the creation of an image gallery application.

Silex-related topics will be covered, such as Front Controllers, Routing, Dependency Injection, Lazy Loading, Service Providers, and Middleware.

Supporting libraries and technologies will be introduced one chapter at a time. Here and there, we will backtrack and refactor code to make it cleaner and more maintainable.

Unit and Functional Testing will be used to help create a solid and reliable test suite, which will verify that refactoring hasn't unintentionally broken the site or caused features to stop working as expected.

By exploring these technologies one-by-one, you'll learn how to combine them into a complete web application. This will give you valuable insight when it comes time to research and apply the up-and-coming web technologies of tomorrow.

## Baseline Assumptions

Every book has to make assumptions about the reader, and you know what they say about assumptions: They're fun!

**Assumption One**: You, the reader, are an amazing human being or benevolent machine intelligence. You are open-minded and eager to learn as much as possible.

**Assumption Two**: You have dabbled a bit in programming or scripting. You know your way around programming concepts like loops (while, for, do..while) and conditionals (if/else, switch), and have some knowledge of functions and classes/objects. You might be a junior or intermediate Wordpress developer who wants to explore more of what PHP has to offer.

**Assumption Three**: You want to make web applications and need a foothold to get started.

## Our Application: The Image Gallery

Before we begin coding our application, we should decide what we want it to do. It's helpful to have a basic outline of what features you want to have, and what problems you aim to solve.

## Initial Goal

> A simple and attractive photo gallery that can receive image uploads, generate thumbnails, and display albums.

## Feature Outline

- Homepage
  - View the Full Gallery
  - Upload an Image
- View an Individual Image

As we explore each chapter, we'll come closer to the initial goal. We'll have to make some compromises in the beginning in order to get more comfortable with Silex.

In the first few chapters, the application will be very primitive. It will look awful, and it will be built in an insecure way - but **DON'T PANIC**. Things will fall into place nicely, and we'll explore some technologies that help you build safe and pretty web applications.

Now that we've got a basic idea of what we want to build, let's get started!

# Chapter One: Getting Started

As you might have guessed from the title of the book, we're going to use Silex as the starting point for our application.

Silex is a PHP microframework derived from the Symfony framework, built by Sensio Labs.

## What is a Microframework?

Every microframework seems to have a different bone to pick with the full-stack framework philosophy. This makes the definition of "microframework" tough to nail down.

In a full-stack framework, virtually everything you would want to do in an application of any size or scale is available somewhere in the framework - it's just a matter of knowing where to find it. This often results in bulky, clunky, or slow applications. The framework constantly expands to include more functionality and more code. All of that expansion worms its way into your application, and if you're not careful it can lock you into the framework.

Microframeworks try to combat this progression of bloat and lock-in. Some set a goal to only deliver the minimum lines of code necessary for basic functionality. Others are built to only expose a basic level of functionality, but have a great deal of flexibility under the hood to draw from.

**Silex 2** reuses the components of the **Symfony Framework** to expose a basic but flexible set of features. Instead of making every decision for you, it allows you to build your application the way you want. While it won't beat other microframeworks for line-of-code minimalism, the actual amount of code being executed at any given time is surprisingly lightweight.

Because Silex lets you choose your own path, it can be easy to get tangled in spaghetti code in larger web applications. It often takes a skillful and experienced architect to reign things in and engineer a scalable codebase. Hopefully, the guidance in this book will help you avoid those pitfalls.

## The Anatomy of Silex

Silex is, at its core, a layer of glue on top of pre-existing components.

- **Pimple**

  Pimple is a simple PHP Dependency Injection Container. It is an array object that can contain your entire application, including many of the objects (called Services) you'll use when running your application. When you need to access those Services, Silex will tell Pimple to locate the object itself - or, if the object hasn't been created yet, Pimple will locate and run the code for creating the object, and then remember the object for later.

- **Symfony Event Dispatcher**

  Silex uses the event dispatcher to control the flow of execution for Request/Response handling. As the user's request flows through the application, various events are triggered, eventually resulting in a response that can be sent to the user. This also gives you an event "bus" you can extend in your application, by listening for and triggering your own custom events.
- **Symfony HTTP Foundation**

  The foundation component maps the HTTP specification to objects in PHP. This saves you from having to directly call $_GET, $_POST, $_SESSION, etc., by providing a set of common interfaces to work with.
- **Symfony HTTP Kernel**

  The kernel component accepts Request objects and uses the Event Dispatcher to ferry things along until a Request is ready to be sent to the browser.
- **Symfony Routing Component**

  The routing component examines incoming requests and compares them against the Route Collection to determine which controller action should be executed. Essentially, this component decides which code to run when the browser requests a specific web page URL.

The Silex Application class extends the Pimple container class, inheriting all of Pimple's functionality, and also implements the HttpKernel interface, giving it the ability to coexist with the Symfony components. When first instantiated, it also prepares a bunch of services and parameters in the Pimple container related to Routes, Controllers, Events, Exception Handlers, etc.

All of these components work together to deliver the Silex experience.

# Installing PHP and Composer

Depending on your operating system, it can be a challenge to get a PHP development environment up and running.

On Ubuntu and Debian Linux, it's generally pretty easy:

```
1 $ sudo apt-get install php php-cli
```

Same goes for the RedHat and CentOS Linux:

```
1 $ sudo yum install php php-cli
```

OS X conveniently ships with PHP pre-installed. For our initial needs, this should be sufficient. More advanced applications may require custom PHP builds and extensions, and that's where things get challenging. Developers who need these custom extensions generally have to install utilities like "Homebrew" or "MacPorts" in order to get easy access to PHP customizations.

Installing PHP on Windows is a bit of a tricky process. The PHP website has a page that is intended to help with the process, but it's a fairly complex set of steps:

```
http://php.net/manual/en/install.windows.php
```

More recent releases can be found on the dedicated Windows section of the PHP website:

```
http://windows.php.net/download/
```

For the purposes of this book, it is preferable (although not mandatory) to use Linux instead of Windows. Luckily, the technology exists to run Linux inside of Windows, so you can have the best of both worlds.

By installing an application called VirtualBox, you can create a virtual machine inside of Windows that is capable of running other operating systems. Then, you can download a version of the Ubuntu operating system, such as Ubuntu 16.04 Xenial Xerus, to use as the OS for your virtual machine. Once installed, you can just use the same instructions as above to install PHP:

```
$ sudo apt-get install php php-cli
```

## Composing the Project (composer.json)

The next step for building a Silex application is initializing the project using Composer, a dependency management utility for PHP projects. Composer is a great tool for including open source libraries in your projects, and we'll be using it to include Silex as one of those open source libraries.

Let's begin with the basic setup and a slightly different approach to the standard "Hello World" application.

Install Composer by following the instructions at http://getcomposer.org/download/[1]. I prefer to install it globally on my systems (Linux and OS X), so I can invoke it easily and keep it up to date - this simply means that I put the file in my system path, usually storing it as `/usr/local/bin/composer`. If you use services like PuPHPet or Phansible to generate virtual machine configurations, you will most likely have composer pre-installed inside the resulting virtual machine.

Once composer is installed, let's begin by creating a folder for the project:

```
$> mkdir the-image-gallery
$> cd the-image-gallery
```

Once inside the project folder, run this command:

[1] http://getcomposer.org/download/

```
$> composer init
```

This kicks off the interactive initialization process. Give the project a name (my-username/the-image-gallery), a quick description (Simple photo gallery application), add your contact info in the format "Firstname Lastname <email@domain.com>", set minimum stability to "dev", and choose a license.

I tend to release my software under the "MIT" open-source license, so I would usually type "MIT" here and hit enter. There are many other licenses you could choose from, or you could build it with no license. Pressing enter here without selecting a license essentially means the project would be an All-Rights-Reserved closed-source project. Either "MIT" or a blank/all-rights-reserved license would be fine for this sample project.

The next stage is choosing dependencies. While some projects can start out with dozens of dependencies, we'll stick to just one for this chapter:

```
Search term: silex/silex
Version: ~2
```

Then, after saying "no" to dev requirements and "yes" to confirm generation, run the install command to download the initial dependencies:

```
$> composer install
```

The libraries will be downloaded into `the-image-gallery/vendors/`, and Composer will automatically create an `autoload.php` file for making it easy to work with each dependency.

Now that the composition is finished, we can begin getting things wired up.

### The Easy Button

Sample code for each chapter is available on GitHub. You can obtain the code by cloning the repository:

```
git clone https://github.com/beryllium/the-image-gallery.git
```

This will automatically pull down the samples into a directory called `the-image-gallery`. Each chapter has its own folder inside this directory, and you can run "composer install" in each of the chapter folders to get it ready to play with.

## Bootstrapping (bootstrap.php)

Next, we create a Bootstrap file called `bootstrap.php`. This file will eventually control most of the internal wiring of your application, but for now it will only be a few lines long:

**Chapter One: bootstrap.php**

```
<?php

require __DIR__ . '/vendor/autoload.php';

$app = new Silex\Application();

$app['debug'] = true;

return $app;
```

This includes the composer autoloader, and initializes the Silex core in Debug mode.

The `return $app` line allows us to store our entire application in a single variable, which will become very useful when we explore ways of testing our code.

# Controlling the Front (web/index.php)

You may have heard of the "Model-View-Controller" design pattern for applications. Many web frameworks use this concept to guide developers toward creating maintainable applications. Silex, inspired by the Symfony philosophy, strives to make that concept even more web-friendly by advocating a more complete Separation of Concerns. It's often called the "Request/Response" design pattern.

HTTP requests come in from the web server and are handed to a controller action, which then returns a response. Any juggling of views and models happens as a separate concern elsewhere in the code, often in magical places like service providers. We'll explore those magical lands in later chapters.

Naturally, for our initial implementation, we're going to get down and dirty and forget about separated concerns.

One thing we do need to learn about is the "Front Controller". In our project, the front controller will be the main publicly-accessible script that exposes our application to the world. It will define the routes (sometimes called pages, or "endpoints" if they don't return pages) that give our application its functionality.

Go ahead and create a subfolder in our project called `web/`, and create an `index.php` file inside there. It should look like this:

**Chapter One: web/index.php**

```
<?php

require __DIR__ . '/../bootstrap.php';

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

// Declare our primary action
$app->get('/', function() use ($app) {
    return new Response('Mr Watson, come here, I want to see you.', 200);
});

$app->run();
```

The first parameter is the `path` or `pattern` of the route. Incoming requests are compared against this string to see which part of the application they belong to.

You might not recognize the `function () use ($app) {}` syntax used in this example. That's OK. We're using something called an anonymous function (otherwise known as a closure or lambda function), which lets us create a reusable chunk of logic without a name. This kind of callable/passable function is common in Javascript and other languages, and was added to PHP in version 5.3.0.

In this case, we pass the anonymous function as the second parameter to our application's "get" method. The function gets added to Silex's internal collection of routes, where incoming requests that match the path will cause it to be executed.

A typical small web application could have a few dozen registered routes like our `'/'` example. This could include standard webpage routes ("GET /contact-us"), and it can also include dynamic routes or routes that fetch specific data items by ID ("POST /comments" or "GET /comment/{id}").

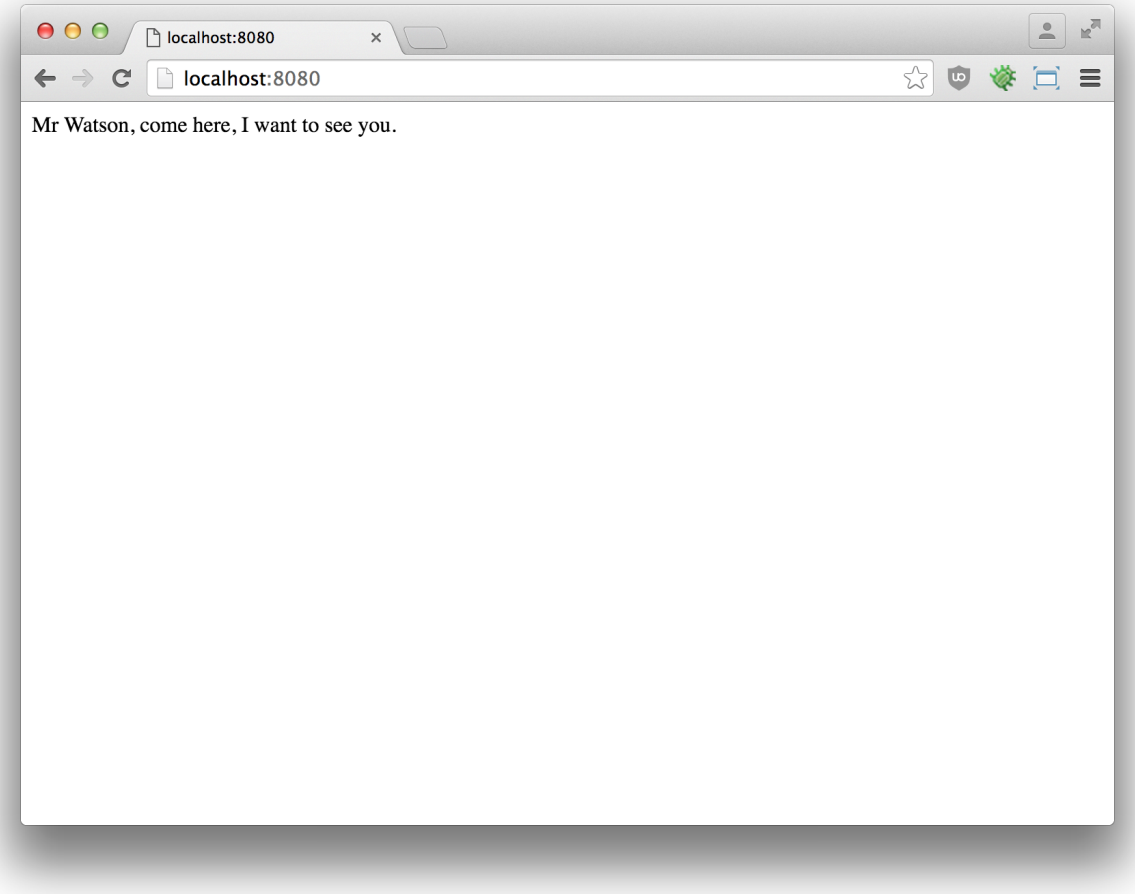
As you might imagine, the `index.php` file on these sorts of applications could get rather long - later in the book, we'll explore ways to simplify routes by handling the business logic in Services and the display logic in Templates. Silex also allows us to bundle our routes into organized collections, further streamlining the `index.php` file.

# Running Our Application

At this point, if you load `index.php` from your web server, it should respond in plain text:

```
1 Mr Watson, come here, I want to see you.
```

To try it out in the PHP development web server included in PHP 5.4+, run `php -S 0.0.0.0:8080 web/index.php` and then load `http://localhost:8080/` in your browser.




**Chapter 1 (in Chrome)**

As the script executes, the `bootstrap.php` file is included - this initializes the application and gets it ready to run.

The code then adds our first route to the collection. This tells the application that HTTP GET requests to the URL `'/'` (the base of the site) should be sent to the provided anonymous function.

When `$app->run()` is called, Silex takes the server environment (including the user's requested URL) and creates a Request object. This object contains all kinds of information about the incoming request, including the user's IP address and their web browser's User Agent. This Request object is then sent to Silex's EventDispatcher. The `routing.listener` service, which is subscribed to the EventDispatcher, receives the event, and matches it to our route's lambda function. Then, our route returns a Response object that contains the output string and an HTTP status of "`200`".

Congratulations, we've built a tiny web app! In the next chapter, we will make it more useful by adding the ability to upload and view photos.

## Version Everything!

Version Control is an important part of any modern development process. Tools like `git` make it easy to keep track of the entire development history of a project.

By tracking all the changes in a project, maintainers can ensure that all the code is reviewed and tested before being put "live".

Starting a git repository is fairly easy.

```
1  $ git init
```

That will initialize a blank repository. At this point, you can begin adding files to be tracked.

```
1  $ git add bootstrap.php composer.json web/index.php
```

Now that git is tracking these files, you have to tell it that you're ready to store your progress on the tracked files. This will be your first commit.

```
1  $ git commit -m "Make it so."
```

Each chapter will conclude with the git commands to record your progress.