Department of Information Technology
IT 350 - Web Technologies
Fall - 2021

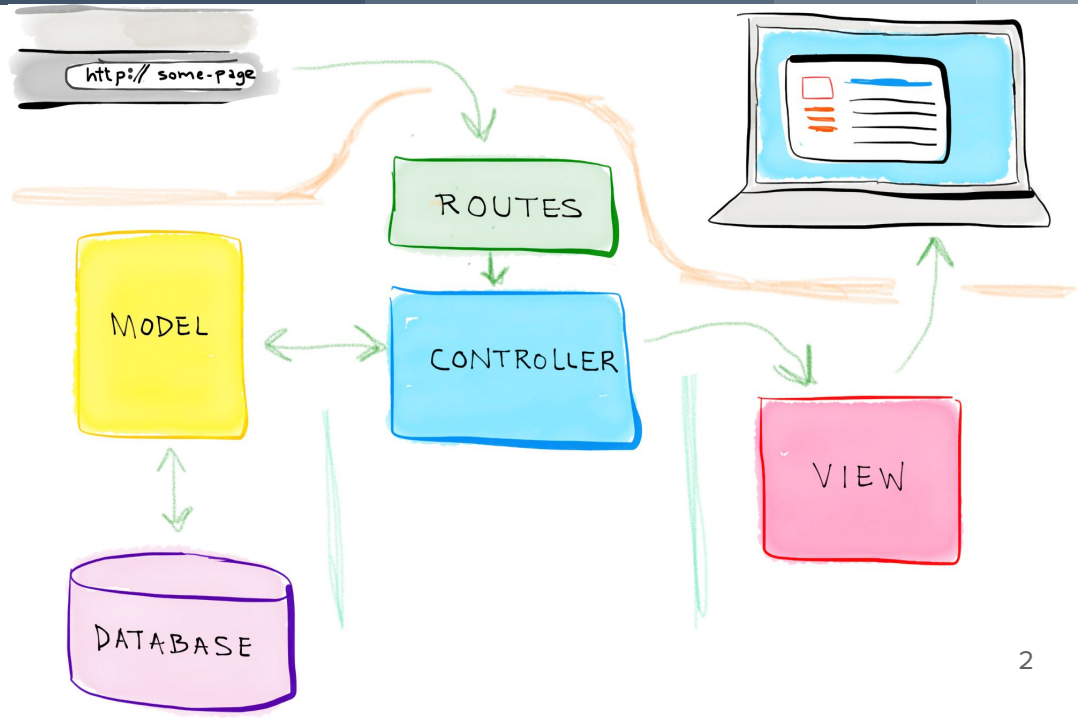
Laravel Development Environment

Lecture Two

Rebin M. Ahmed
rebin.mohammed@tiu.edu.iq

Previous Lecture

- What is Laravel?
- Why Framework?
- What is MVC?
- How does Laravel Work?
- What is Route, Controller, Model and View?



Contents

- System Requirements
- Creating a New Laravel Project
- Laravel’s Directory Structure
- Configurations
- Architecture Concepts
- Request Lifecycle

System requirements

The Laravel framework has a few system requirements. All of these requirements are satisfied by the Laragon, so it's highly recommended that you use Laragon as your local Laravel development environment.

You can find Laragon’s website in your Software and Tools Folder in Edmodo with tutorials on how to install it.

System requirements

- PHP >= 7.2.5
- BCMath PHP Extension
- Ctype PHP Extension
- Fileinfo PHP extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

Composer

Whatever machine you're developing on will need to have **Composer** installed globally. If you're not familiar with Composer, it's a tool that's at the foundation of most modern PHP development.

Composer is a dependency manager for PHP, much like NPM for Node or RubyGems for Ruby. But like NPM, Composer is also the foundation of much of our testing, local script loading, installation scripts, and much more.

You'll need Composer to install Laravel, update Laravel, and bring in external dependencies.

Creating a New Laravel Project!

Creating a New Laravel Project!

- (officially) There are two ways to create a new Laravel project, but both are run from the command line.
- The first option is to globally install the Laravel installer tool (using Composer);
- The second is to use Composer’s create-project feature.

You can learn about both options in greater detail on the Installation documentation page

https://laravel.com/docs/7.x#installing-laravel

Installing Laravel with the Laravel Installer Tool

- If you have Composer installed globally, installing the Laravel installer tool is as simple as running the following command:

  composer global require "laravel/installer"

- Once you have the Laravel installer tool installed, spinning up a new Laravel project is simple. Just run this command from your command line:

  laravel new projectName

Installing Laravel with Composer

- Composer also offers a feature called **create-project** for creating new projects with a particular skeleton. To use this tool to create a new Laravel project, issue the following command:

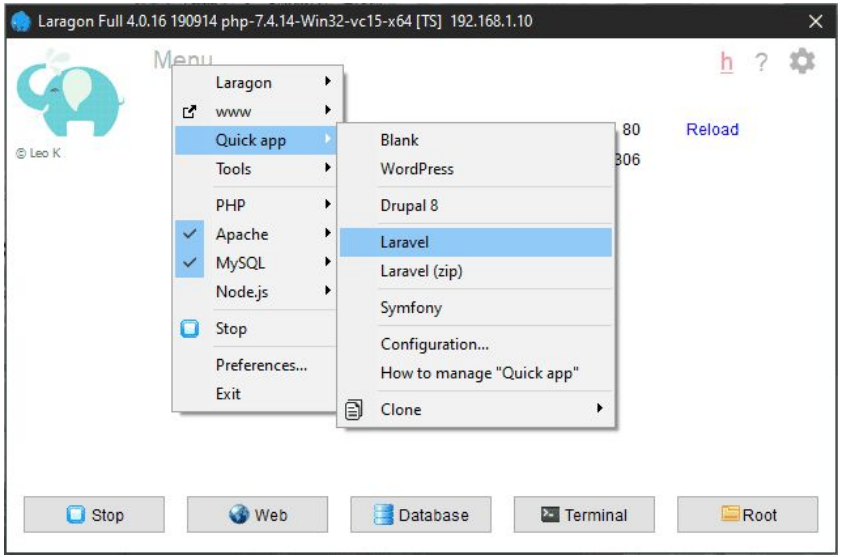
  composer create-project laravel/laravel projectName

- Just like the installer tool, this will create a subdirectory of your current directory named {projectName} that contains a skeleton Laravel install, ready for you to develop.

Installing Laravel with Laragon

- Laragon provides one of the easiest ways to install laravel without command using command line, all you have to do is

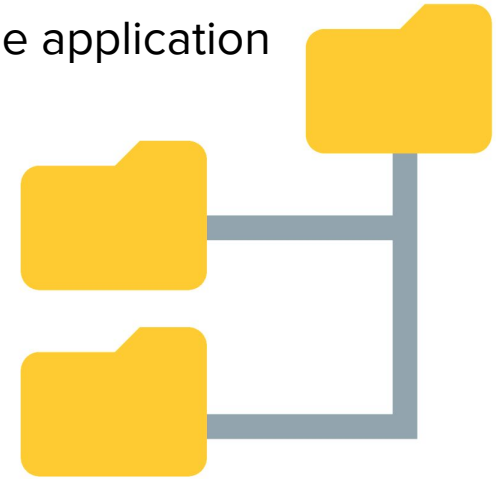
  right click-> select Quick app->Laravel



Directory Structure

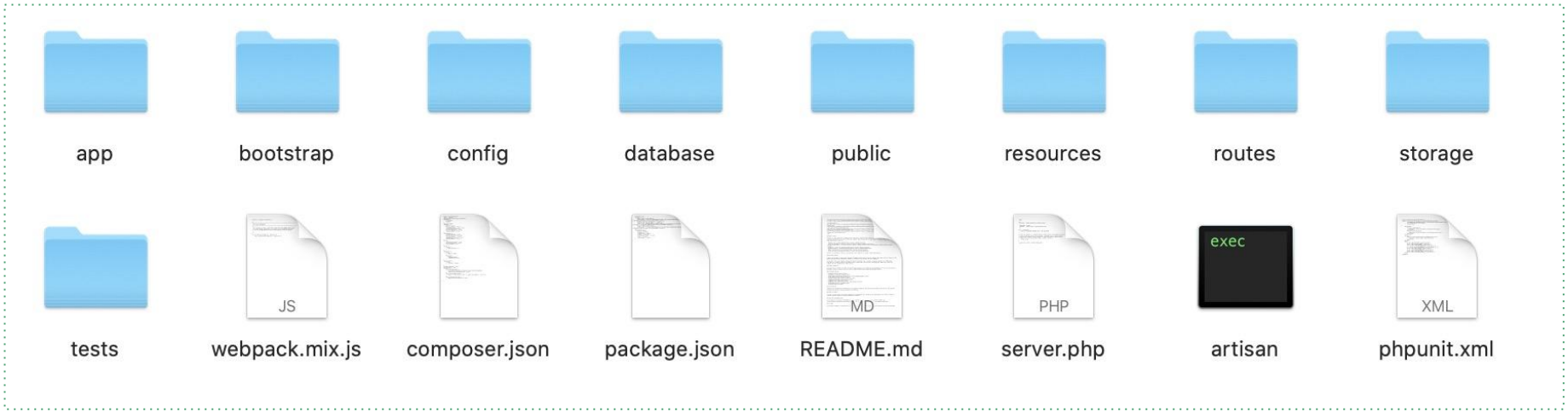
The directory structure in Laravel is basically the structure of **folders**, **sub-folders** and **files** included in a project.

Once we create a project in Laravel, we get an overview of the application structure.



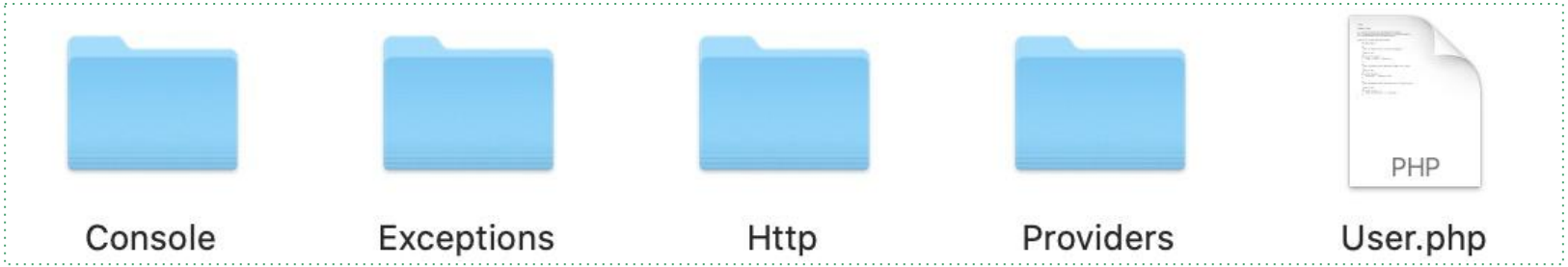
Directory Structure

The root directory contains the following folders by default:



Directory Structure: App

It is the application folder and includes the entire source code of the project. It contains Models, controllers, commands, and your PHP domain code all go in here.



Directory Structure: App

Http :

The Http folder has subfolders for **controllers** and **middleware**. As Laravel follows the MVC design pattern, this folder includes **Model** and **Controllers**

The **Middleware** sub-folder includes middleware mechanism, comprising the filter mechanism and communication between response and request.

Http

Directory Structure: Bootstrap

- Bootstrap contains the files that the Laravel framework uses to boot every time it runs.
- It contains a sub-folder namely cache, which includes all the files associated for caching a web application.
- You can also find the file app.php, which initializes the scripts necessary for running your application.

bootstrap

What is Cache?

Directory Structure: Config

- The config folder is where all the configuration files live.
- It includes various configurations and associated parameters required for the smooth functioning of a Laravel application.

config

Directory Structure: Database

As the name suggests, this directory includes various parameters for database functionalities. It includes three sub-directories as given below

- **Seeds** − This contains the classes used for unit testing database.
- **Migrations** − This folder helps in queries for migrating the database used in the web application.
- **Factories** − This folder is used to generate large numbe records.

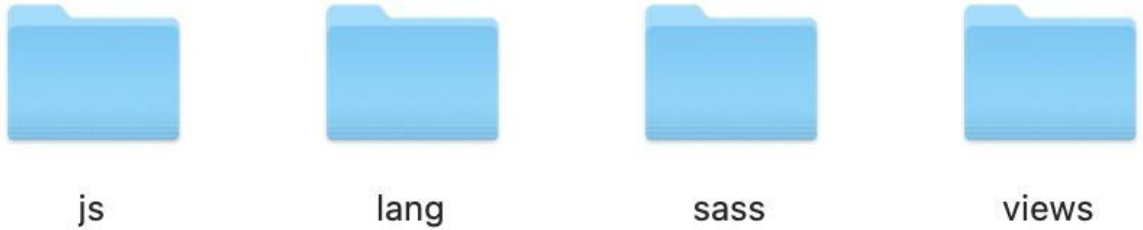
Directory Structure: Public

- The directory the server points to when it’s serving the website.
- This contains index.php, which is the front controller that kicks off the bootstrapping process and routes all requests appropriately.
- It’s also where any public-facing files like images, stylesheets, scripts, or downloads go.

public

Directory Structure: Resources

- Where files that are needed for other scripts live. Views, language files, and (optionally) Sass/Less/source CSS and source JavaScript files live here.
- **views** − Views are the HTML files or templates which interact with end users and play a primary role in **MVC** architecture.



Directory Structure: Routes

Routes directory contains the files which routes files of your web application. The files included in this directory and their purpose is explained below

- **web.php** – file defines routes that are for your web interface. These routes are assigned the web middleware group.
- **api.php** – api routes are stateless and are ass api middleware group.

routes

Directory Structure: Storage

- Storage is where **caches**, **logs**, and compiled system files live.
- This is the folder that stores all the logs and necessary files which are needed frequently when a Laravel project is running.

storage

Part Two

Environment Configuration

Environment variables are those which provide a list of web services to your web application. All the environment variables are declared in the .env file which includes the parameters required for initializing the configuration.

By default, the .env file includes following parameters −

Environment Configuration

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

Environment Configuration

Important Points, While working with basic configuration files of Laravel, the following points are to be noted :

- The **.env** file should not be committed to the application source control, since each developer or user has some predefined environment configuration for the web application.
- For backup options, the development team should include the **.env.example** file, which should contain the default configuration.

Environment Configuration

```
APP_NAME="Laravel"
```

Name of your application is written in the APP_NAME variable

```
APP_ENV=local
```

APP_ENV stores the environment type of your application **local** or **production**.

Environment Configuration

```
APP_KEY=base64:kil9YRYEyWAZNpmnuTujS0Rn23qyTtMWNl2/6ppSYtA=
```

Every time Laravel developers start or clone a Laravel app, generating the application key or APP_KEY is one of the most important first steps.

```
APP_DEBUG=true
```

Laravel uses APP_DEBUG value to determine whether to show detailed debug information for the developers, or not showing any debug information.

Environment Configuration

```
APP_URL=http://localhost
```

APP_URL contains the url of your application.

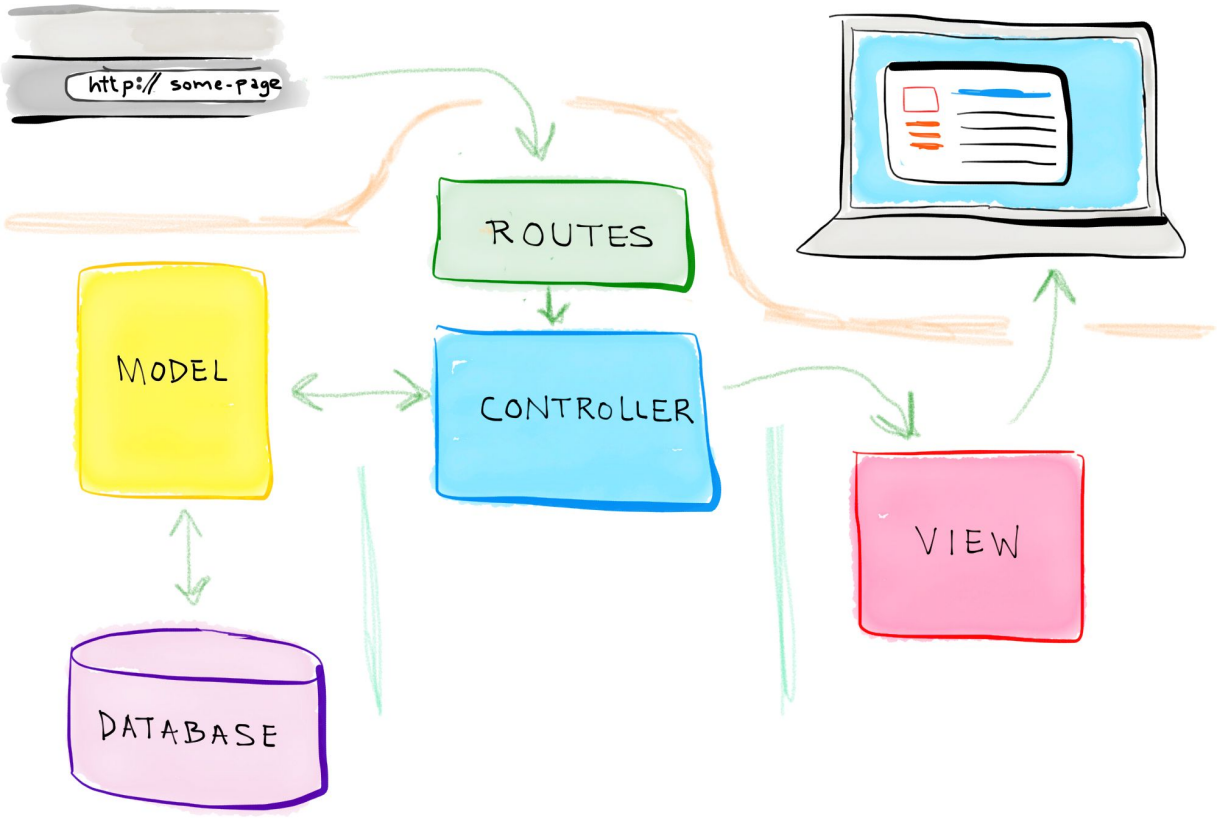
```
LOG_CHANNEL=stack
```

By default, Laravel will use the stack channel when logging messages. The stack channel is used to aggregate multiple log channels into a single channel.

Environment Configuration : Database Credentials

```
DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=laravel

DB_USERNAME=root

DB_PASSWORD=
```

Architecture Concepts

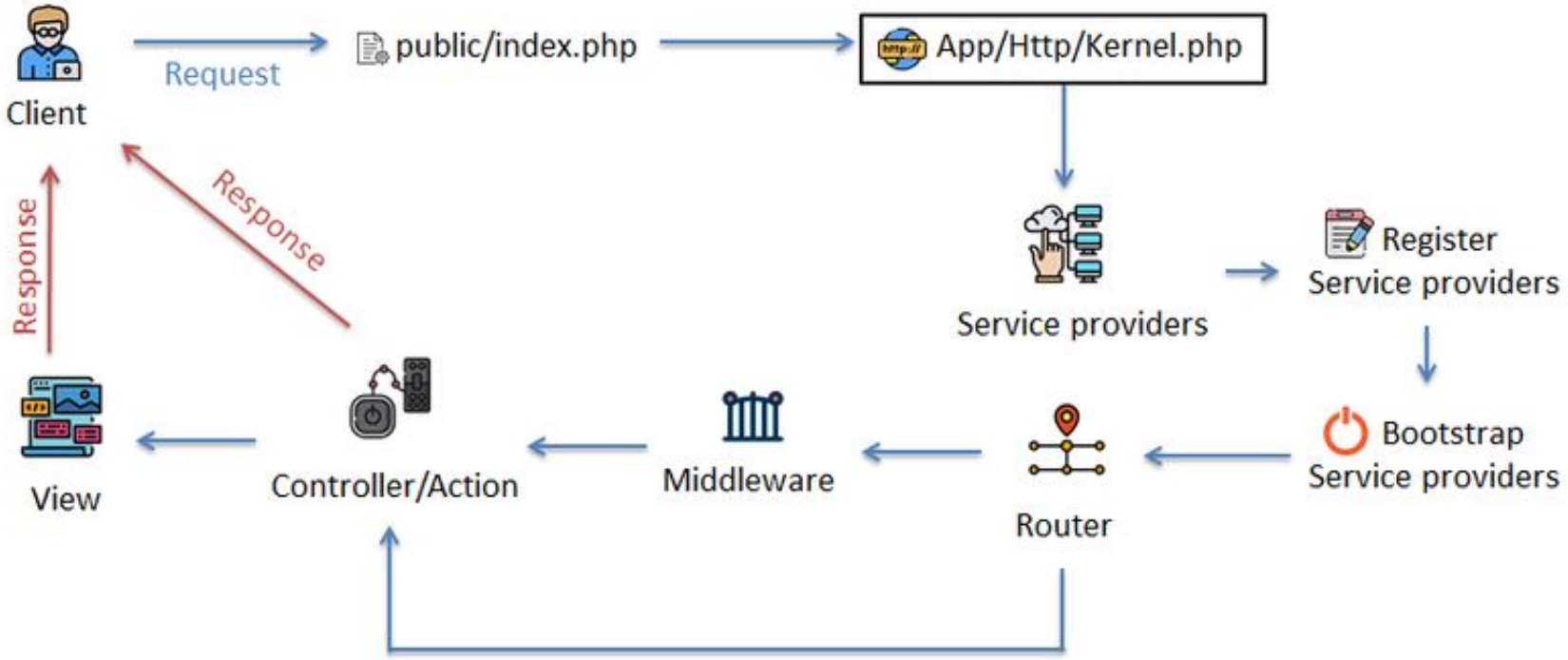


Request Lifecycle

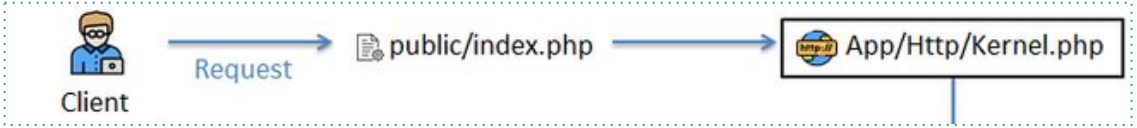
The goal is to give you a good, **high-level** overview of how the Laravel framework works. By getting to know the overall framework better, everything feels less "**magical**" and you will be more confident building your applications.

Just try to get a basic grasp of what is going on, and your knowledge will grow as you explore other sections of laravel.

Request Lifecycle



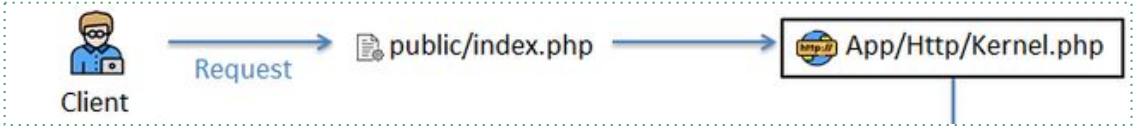
Request Lifecycle



HTTP / Console Kernel

Next, the request will be sent to either the HTTP Kernel or the Console Kernel, depending on where the request was sent from. Currently, we are only interested in the HTTP Kernel located in the file **app/Http/Kernel.php**.

Request Lifecycle

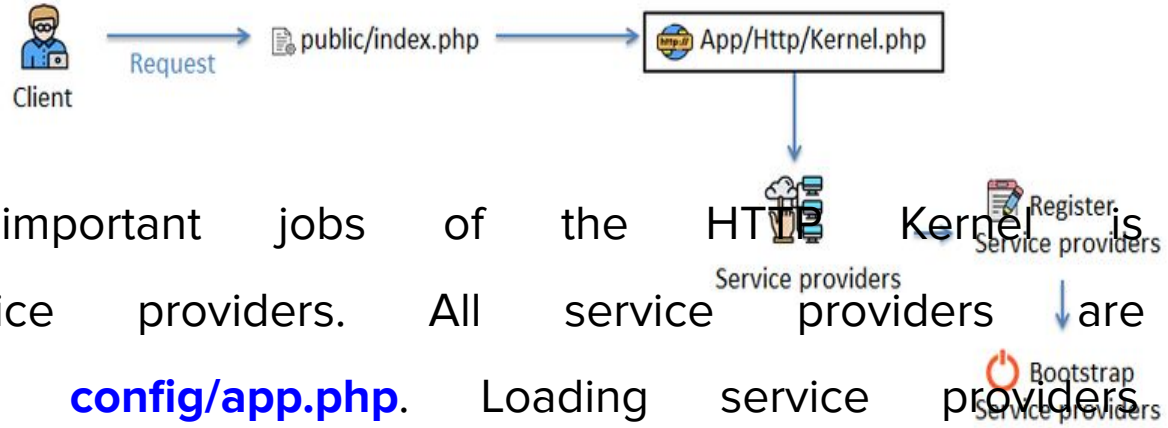


HTTP / Console Kernel

The HTTP Kernel inherits the class Illuminate\Foundation\Http\Kernel, which will do all the work before the request is executed, such as configuring error handling, configuring the logger, defining the application environment, and so on.

The HTTP Kernel is like an application's "black box", and works by a simple mechanism: **receiving requests and returning responses**.

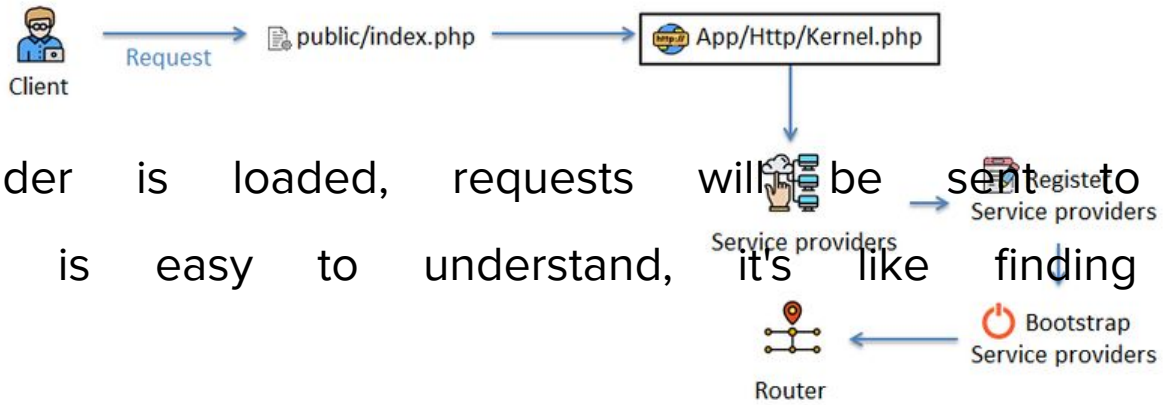
Request Lifecycle



One of the most important jobs of the HTTP Kernel is the loading of service providers. All service providers are configured in the file **config/app.php**. Loading service providers will go through two processes:

1. Register service provider
2. Start the service provider (Bootstrap service provider).

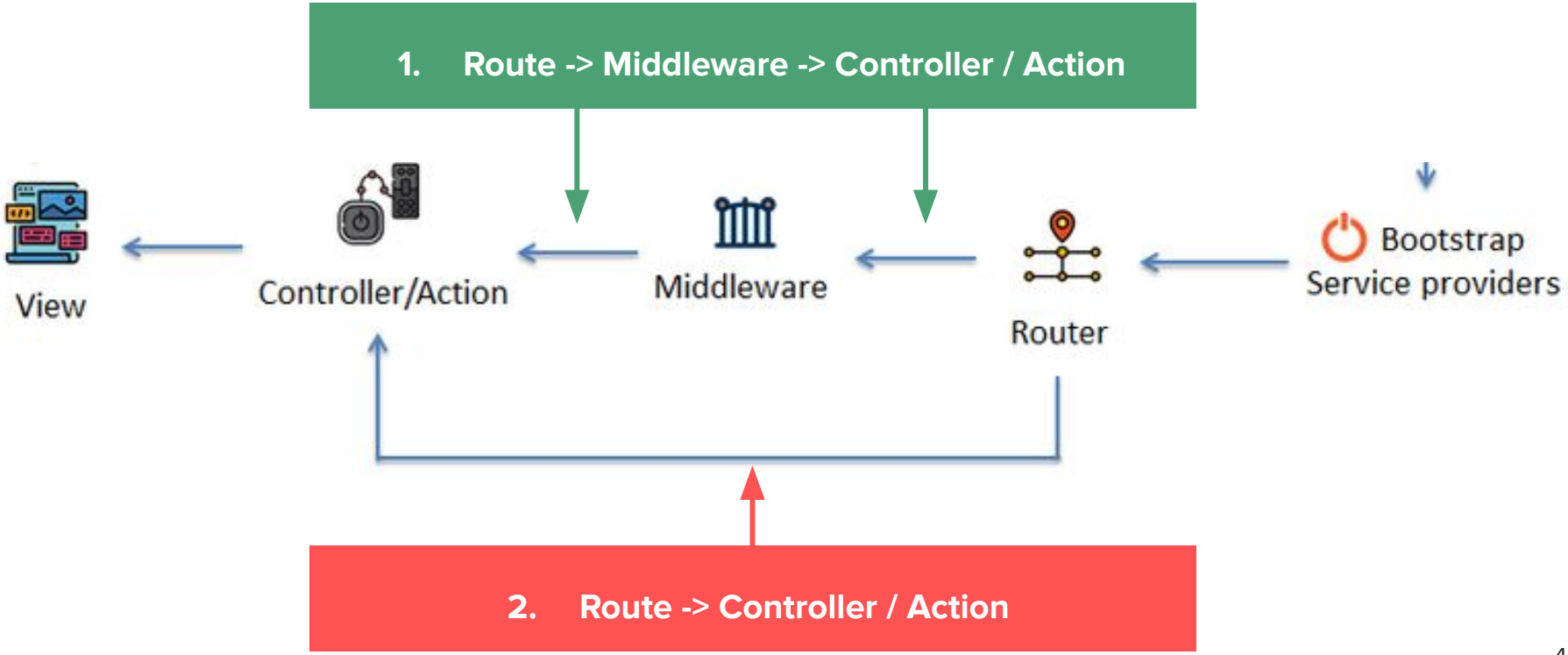
Request Lifecycle



After the service provider is loaded, requests will be sent to the **router**. This block is easy to understand, it's like finding a home for a lost child.

The router's job will check all the routes declared in the files in the directory routes against the incoming request. If it matches perfectly with a particular route, there are two processing directions.

Request Lifecycle

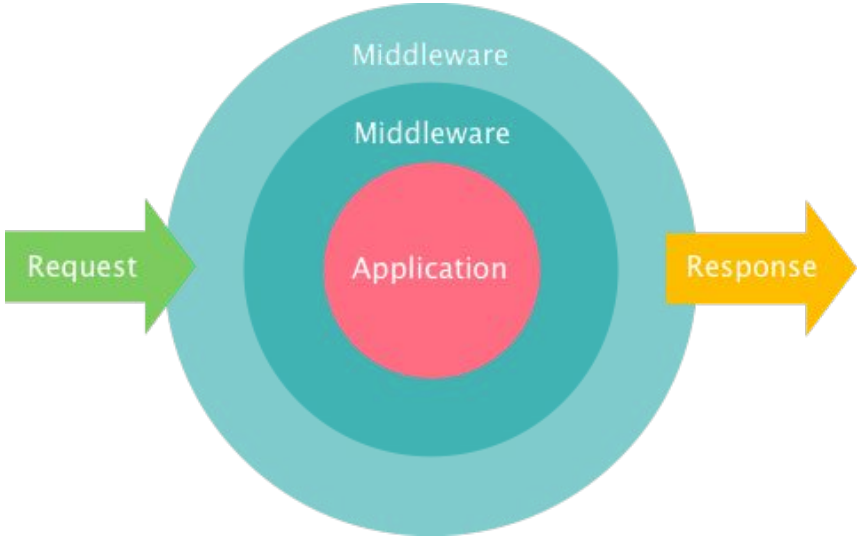


Request Lifecycle

Why are there two ways to handle such a thing?

When declaring a route, Laravel allows us to bind a passing request using our custom middleware. Therefore, depending on whether or not each route is bound by middleware, we divide two processing directions to cover.

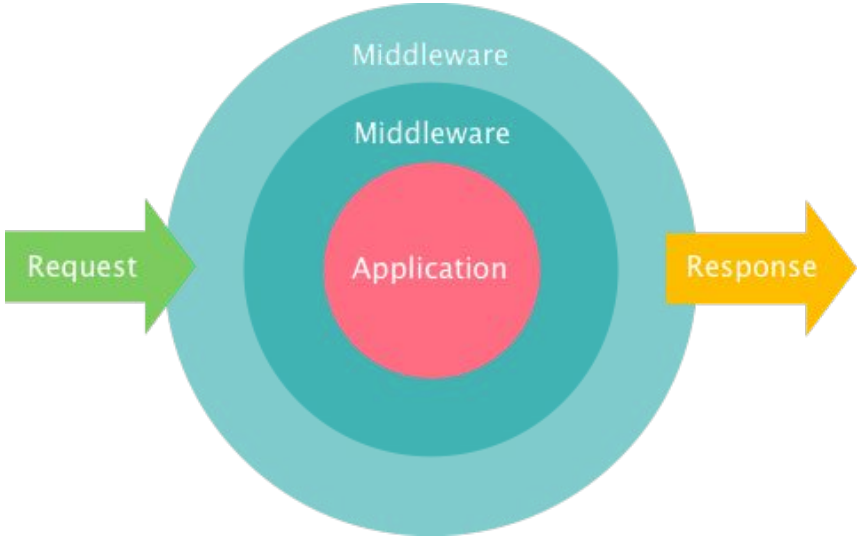
Request Lifecycle



Middleware

As mentioned above, in order for the application to process the request for which the route is registered with the middleware, there is no other way to pass it. Here, the middleware will process logic according to the constraints the coder sets to decide whether the request can go forward or not.

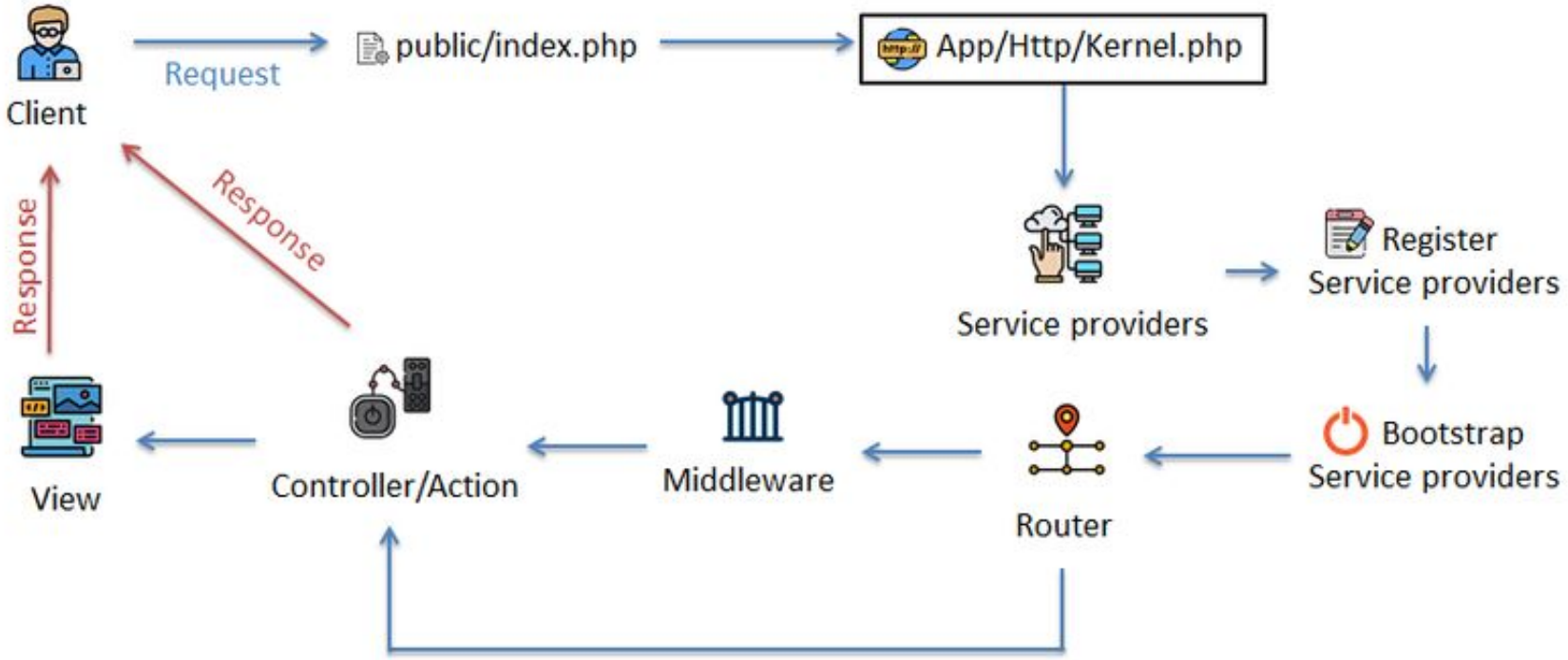
Request Lifecycle



Middleware

For example, there is a request with the path is **http://laravel.test/login** , a coder wants to bind that if there exists a client login session / cookie, when entering this request will be redirected to the homepage, otherwise still shows the login form for the client to continue. It is time to use middleware for binding.

Request Lifecycle



For more details
Read Chapter 10

Activities and Next Week Topics

This Week:

- Download and install Laravel 7.* on your Laptops
- Explore the directories and files of Laravel
- Create a new route with your name, and direct it to a view showing your profile
- Prepare your questions for the practical session in the lab.

Next Week:

- A Quick Introduction to MVC, the HTTP Verbs, and REST

References / Further Readings

- Laravel.com : Laravel’s official Documentation.
- Matt Stauffer, 2019. Laravel: Up & Running: A Framework for Building Modern PHP Apps. O’Reilly Media.
- Dayle Rees, 2016. Laravel: Code Smart.