

# PHP - Introduction of Laravel PHP Framework – Industry

## 1. Introduction

The purpose of this thesis work is to learn a new PHP framework and use it efficiently to build an eCommerce web application for a small start-up freelancing company that will let potential customers check products by category and pass orders securely. To fulfil this set of requirements, a system consisting of a web application with a backend will be designed and implemented using a modern MVC framework.

It is worthwhile considering the use of a PHP framework when time is a limitation and the developer's PHP coding skills do not match the high level demanded to build a complex application. Frameworks handle all the repetitive basic tasks of a PHP project, letting the developer concentrate her/his efforts on the business logic and the general structure of the project as a whole, in doing so, frameworks are becoming an ideal tool used by said developers to rapidly build complex operational prototypes in a matter of hours with minimal time spent on coding. Frameworks offer also whole range of ready-made utilities and libraries.

The use of a robust framework is recommended when the security of the web application is an essential requirement. It even becomes a necessity when the developer lacks the necessary know-how to prevent security breaches from happening. Most of the modern frameworks have built-in security features that range from input sanitising to automatic cookie encryption.

Organised structure of the project as a whole, clear and clean code are required when working in an organisation or co-developing an application in a team of developers. Frameworks permit the organisation of said code into a logical architecture, thus facilitating its maintainability and expandability. To achieve this, modern PHP frameworks follow the Model-View-Controller (MVC) architecture pattern.

Among the highly popular PHP frameworks, Laravel stands out with its claim in its ability to produce a development process that is agreeable for the developer without losing the application's functionality. That is one of the many reasons it was chosen as the framework of choice for building an eCommerce web application for Armel Solutions freelance start-up. This thesis work will study if Laravel lives up to its claim by evaluating its ability in building an up and running secure eCommerce web application in minimal time.

## 2. Laravel's Main Features

This study will focus only on the features used during the building of the eCommerce web application, otherwise this work will not be large enough to cover the entirety of the features of the whole Laravel 4 framework.

### ➤ Architecture

Laravel is a web application framework that tries to ease the development process by simplifying repetitive tasks used in most of today's web applications, including but not limited to routing, authentication, caching and sessions.

Since it manages to do all essential tasks ranging from web serving and database management right to HTML generation, Laravel is called a full stack framework. This vertically integrated web development environment is meant to offer an improved and smooth workflow for the developer.

All the new Laravel projects come out of the box equipped with a full directory tree and also many placeholder files resulting in a structure permitting a quick start of the actual development process. This structure is nevertheless fully customizable. Here in the following figure is shown what such a structure looks like

#### ➤ MVC

The term MVC was briefly mentioned earlier in this work and it is worthwhile mentioning now that Laravel is actually a fully-fledged MVC framework. MVC rapidly became the industry's standard practice used in every modern development environment. Many frameworks such as Ruby on Rails, ASP.NET, CakePHP and CodeIgniter make use of it to separate the logic behind the application from the representation layer.

An MVC architecture pattern let the web application have many different views of a single common model. That is, in our current context of building an eCommerce web application, a Category page, for example, can have multiple views such as the Product List View or Product Gallery View. In an MVC development environment, one model for the Category table will be created and via that one model multiple views can be created.

#### ➤ Model

A Model is the mode by which the developer can manipulate the data. It consists of a layer residing between the data and the application. The data itself can be stored in various types of database systems such as MySQL or even simple XML or Excel files.

#### ➤ Views

Views are the visual representation of our web application (presentation layer), they are responsible for presenting the data that the Controller received from the Model (business logic). They can be easily built using the Blade template language that comes with Laravel or simply using plain PHP code. Blade is driven by template inheritance and sections. When Laravel renders these Views it examines first their file extension, and depending on it being either ".blade.php" or simply ".php", determines if Laravel treats our View as a Blade template or not

#### ➤ Control

The primary function of a Controller is to handle requests and pass data from the Model to the Views. Thus a Controller can be considered as the link between our Model and Views.

The developer has the option to write her/his business logic either in Routers or Controllers. Routers can be useful when dealing with a small web application, or in rapidly defining static pages. Writing Controllers for every single page of the web application is thus not necessary

#### ➤ Database

- Eloquent ORM

The Eloquent ORM provided with Laravel includes a simple PHP ActiveRecord implementation which lets the developer issue database queries with a PHP syntax where instead of writing SQL code, methods are simply chained. Every table in the database possess a corresponding Model through which the developer interact with said table.

- Schema builder

The Laravel Schema class provides a database agnostic (i.e. can function with a multitude of DBMS) way of managing all database related work such as creating or deleting tables and adding fields to an existing table. It works with a multitude of databases systems supported by Laravel and MySQL being the default one. The Schema class has the same API across all of these database systems.

- Managing the database with Migrations

Migrations can be considered as a form of version control for our database. They allow us to change the database schema and describe and record all those specific changes in a migration file. Each Migration is usually associated with a Schema Builder to effortlessly manage our application's database. A migration can also be reverted or “rolled back” using the same said file.

- Seeders

The Seeder class lets us seed data into our tables. This feature is very valuable since the developer may insert data into her/his database's tables every time she/he wants to test the web application

## ➤ Composer

Another feature that makes Laravel stand out from the other frameworks is that it is Composer ready. In fact Laravel is itself a mixture of different Composer components, this adds a much needed interoperability to the framework.

Composer has the ability to manage a dependency up to a given nth level, meaning that all dependencies of our project can be managed via a single tool which is a really handy option to have when we are dealing with a multitude of libraries. Another advantage of using Composer is that it generates and handles an autoload file at the root of our vendor/ directory, which will contain all the project's dependencies that wires up the autoloading of classes when it is included in a PHP script. In doing so, there is no need from the developer side to remember all dependencies' paths and include each of them on every file of the project, she/he just needs to include the autoload file provided by Composer.

Composer is installed in the form of a PHP executable that is added to the PATH environment variable. A PATH environment variable is the listing of locations that is explored when a command is run in the terminal. When Composer is installed properly, the developer can execute it through the command-line from any place in the file system using the “\$ composer” command. The project, and its dependencies, are defined within a JSON file, named composer.json.

Laravel combined with the power of Composer gives the developer more freedom in choosing what kind of packages she/he would like to use with her/his web application. For example, if she/he do not like the default Mail component that comes with Laravel, which is Swift Mailer, and she/he wants to replace it with a more preferred package like the PHPMailer component for example, which is likewise Composer ready; thus, switching between the two packages would be a very easy task. The developer can replace components at will and with ease when there is need to do so via the Composer and Laravel configuration.

## ➤ Artisan

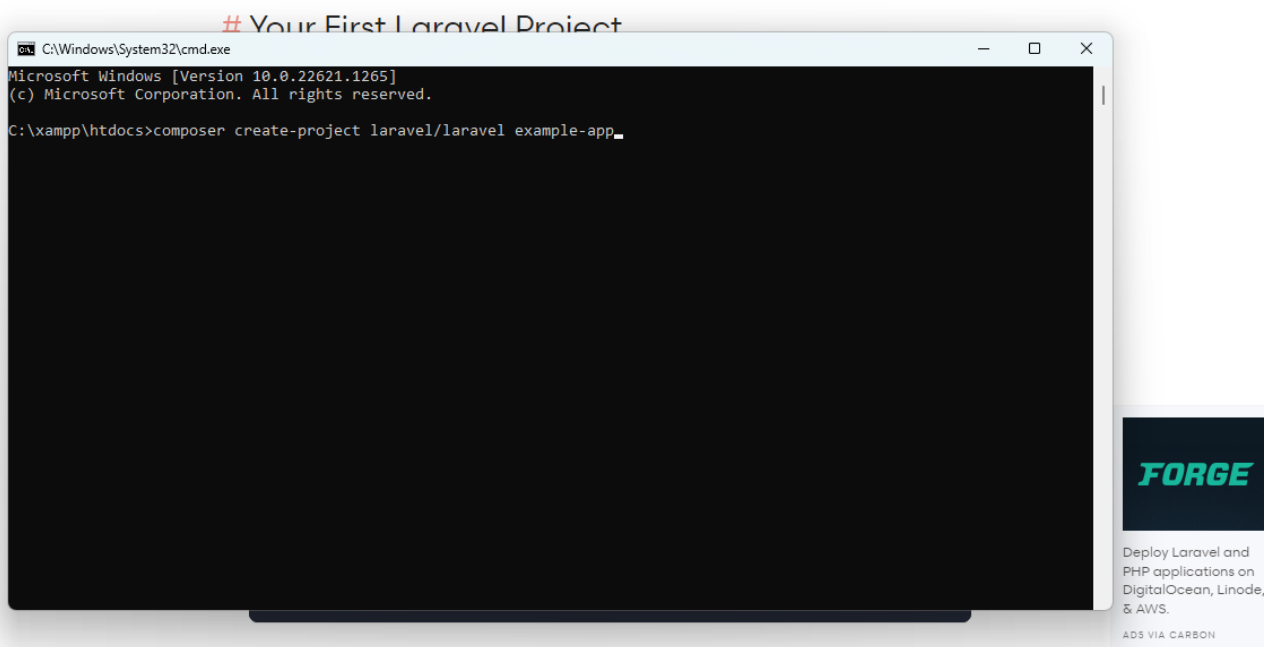
A developer would have to usually interact with the Laravel framework using a commandline utility that creates and handles the Laravel project environment. Laravel has a builtin command-line tool called Artisan. This tool allows us to perform the majority of those repetitive and tedious programming tasks that most of developers shun to perform manually

Artisan let us even create our very own commands and do convenient things with them such as sending pending mails to recipients or anything that can be necessary to properly run our web application. Unit tests for our web application can also be run through Artisan

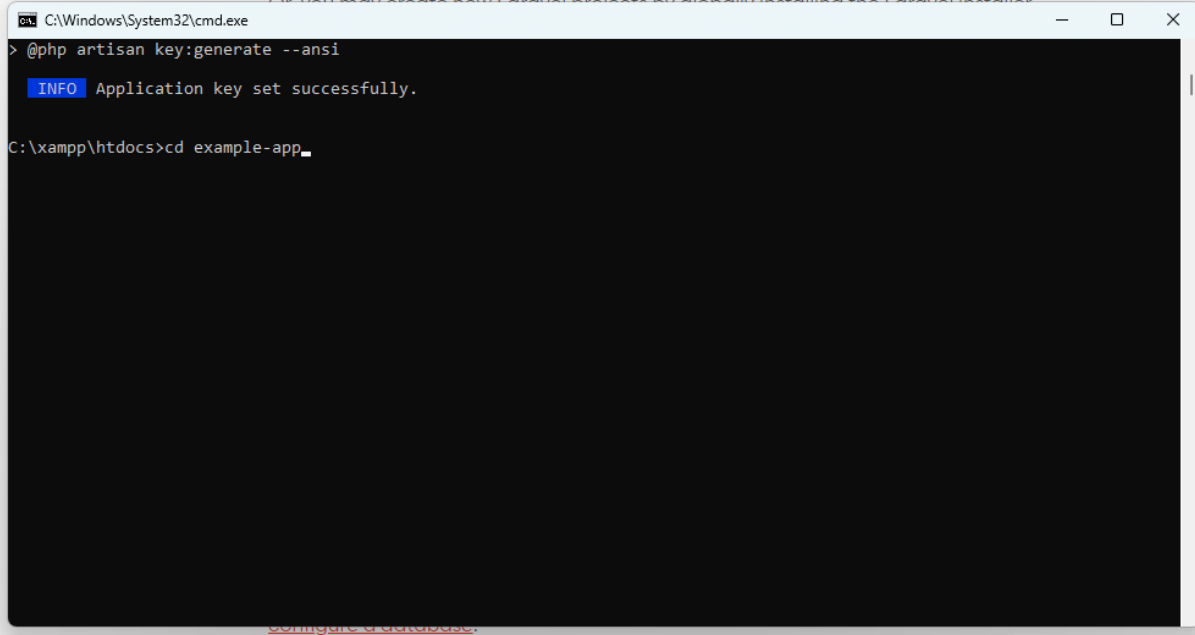
## ➤ Example

Print Welcome to Laravel

### 1. Create Project



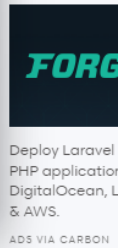
## 2. Change Dictionary



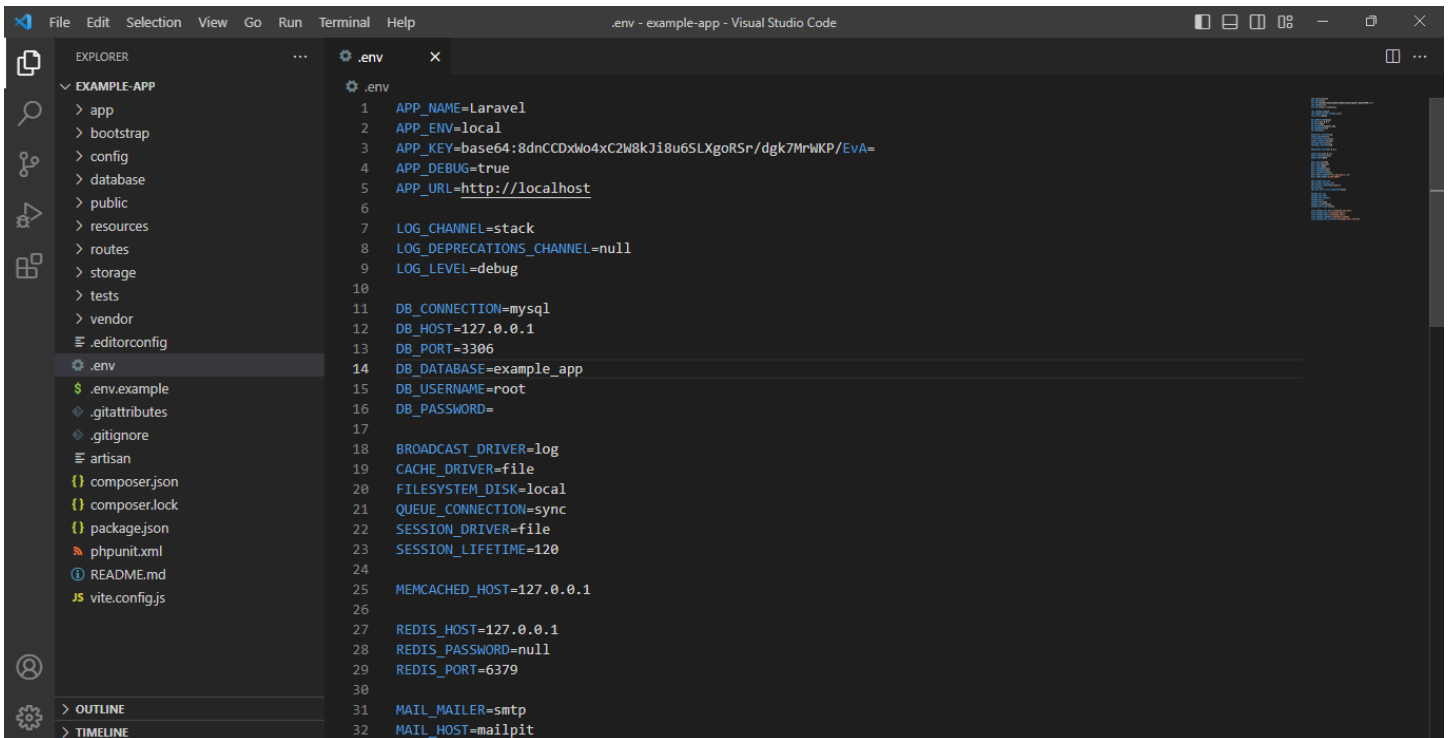
```
C:\Windows\System32\cmd.exe
> @php artisan key:generate --ansi

INFO Application key set successfully.

C:\xampp\htdocs>cd example-app
```

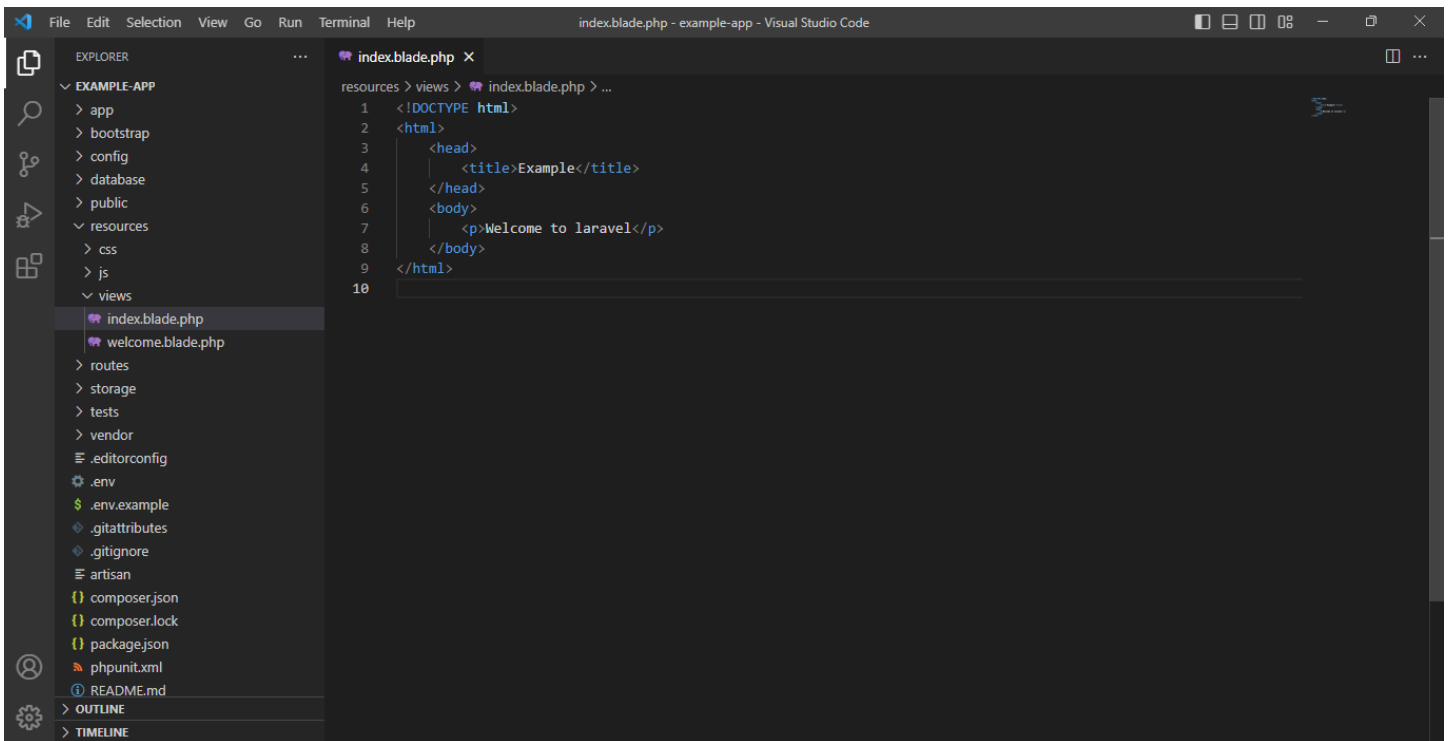


## 3. Connect Database



```
.env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:8dnCCDxW04xC2W8kJi8u6SLXgoRSr/dgk7MrWKP/EvA=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=example_app
15 DB_USERNAME=root
16 DB_PASSWORD=
17
18 BROADCAST_DRIVER=log
19 CACHE_DRIVER=file
20 FILESYSTEM_DISK=local
21 QUEUE_CONNECTION=sync
22 SESSION_DRIVER=file
23 SESSION_LIFETIME=120
24
25 MEMCACHED_HOST=127.0.0.1
26
27 REDIS_HOST=127.0.0.1
28 REDIS_PASSWORD=null
29 REDIS_PORT=6379
30
31 MAIL_MAILER=smtp
32 MAIL_HOST=mailpit
```

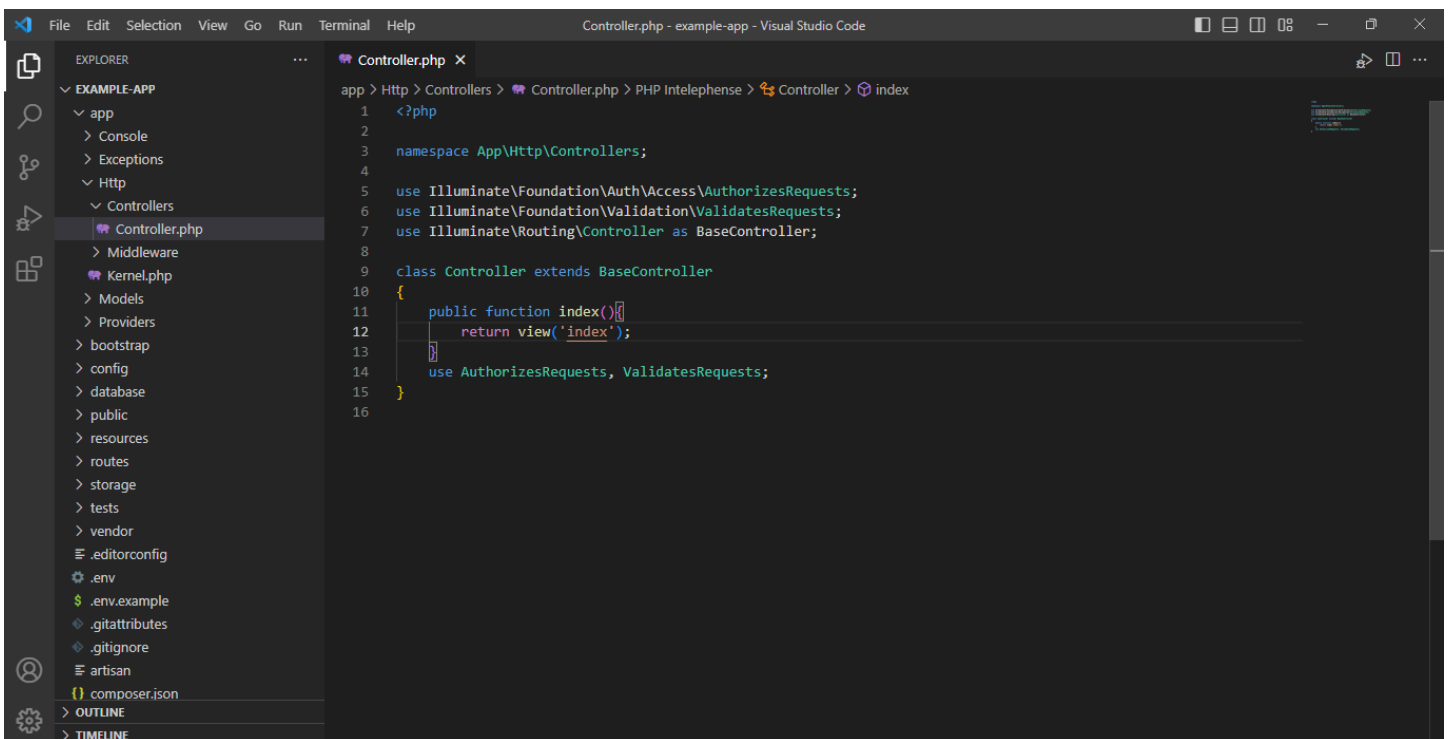
#### 4. Create Blade File



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the file structure of an 'EXAMPLE-APP' with folders like 'app', 'bootstrap', 'config', 'database', 'public', 'resources', 'css', 'js', and 'views'. The 'views' folder is expanded, showing 'index.blade.php' and 'welcome.blade.php'. The Editor panel shows the content of 'index.blade.php' with the following code:

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title>Example</title>
5     </head>
6     <body>
7         <p>Welcome to laravel</p>
8     </body>
9 </html>
10
```

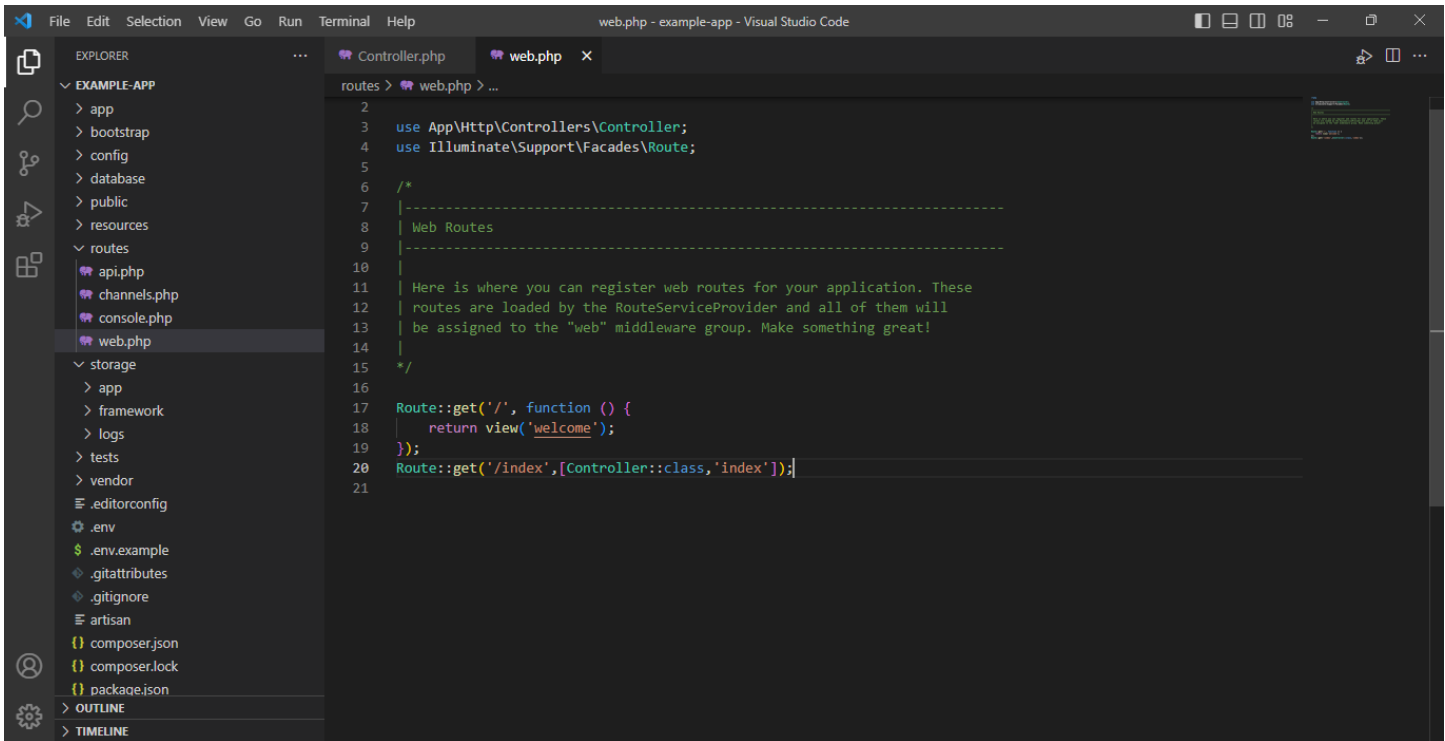
#### 5. Create a function for view in controller



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the file structure of an 'EXAMPLE-APP' with folders like 'app', 'bootstrap', 'config', 'database', 'public', 'resources', 'routes', 'storage', 'tests', and 'vendor'. The 'app' folder is expanded, showing 'Console', 'Exceptions', 'Http', 'Controllers', 'Middleware', 'Kernel.php', 'Models', 'Providers', and 'bootstrap'. The 'Controllers' folder is expanded, showing 'Controller.php'. The Editor panel shows the content of 'Controller.php' with the following code:

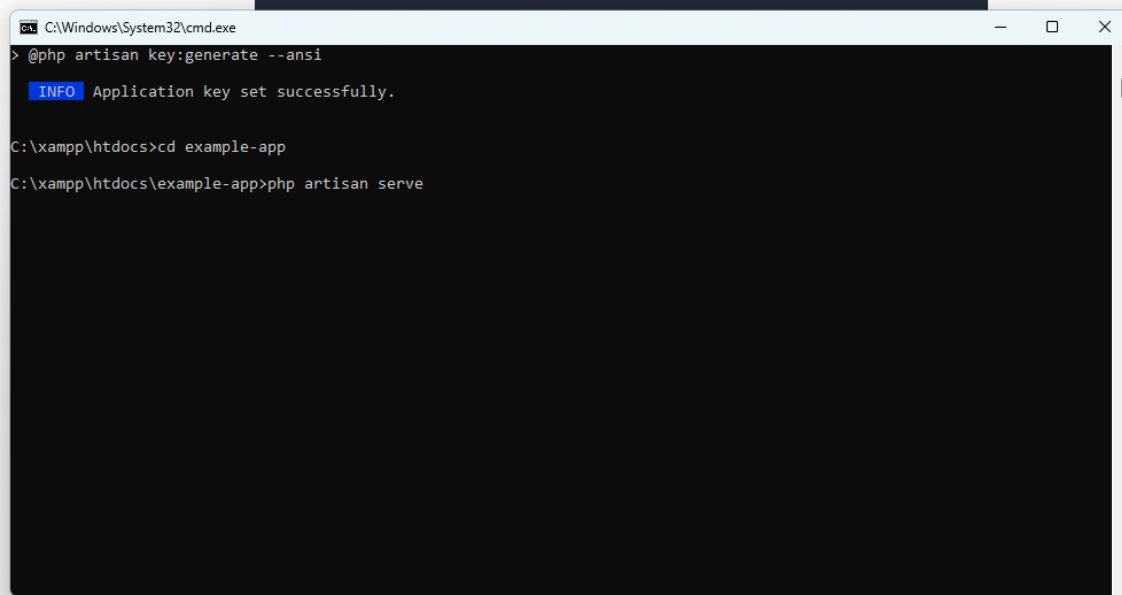
```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
6 use Illuminate\Foundation\Validation\ValidatesRequests;
7 use Illuminate\Routing\Controller as BaseController;
8
9 class Controller extends BaseController
10 {
11     public function index()
12     {
13         return view('index');
14     }
15     use AuthorizesRequests, ValidatesRequests;
16 }
```

## 6. Create a route for view in web.php



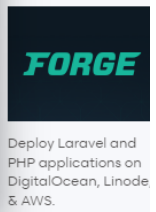
```
2
3 use App\Http\Controllers\Controller;
4 use Illuminate\Support\Facades\Route;
5
6 /*
7 |-----
8 | Web Routes
9 |-----
10 |
11 | Here is where you can register web routes for your application. These
12 | routes are loaded by the RouteServiceProvider and all of them will
13 | be assigned to the "web" middleware group. Make something great!
14 |
15 */
16
17 Route::get('/', function () {
18     return view('welcome');
19 });
20 Route::get('/index',[Controller::class,'index']);
21
```

## 7. Start Server using artisan command



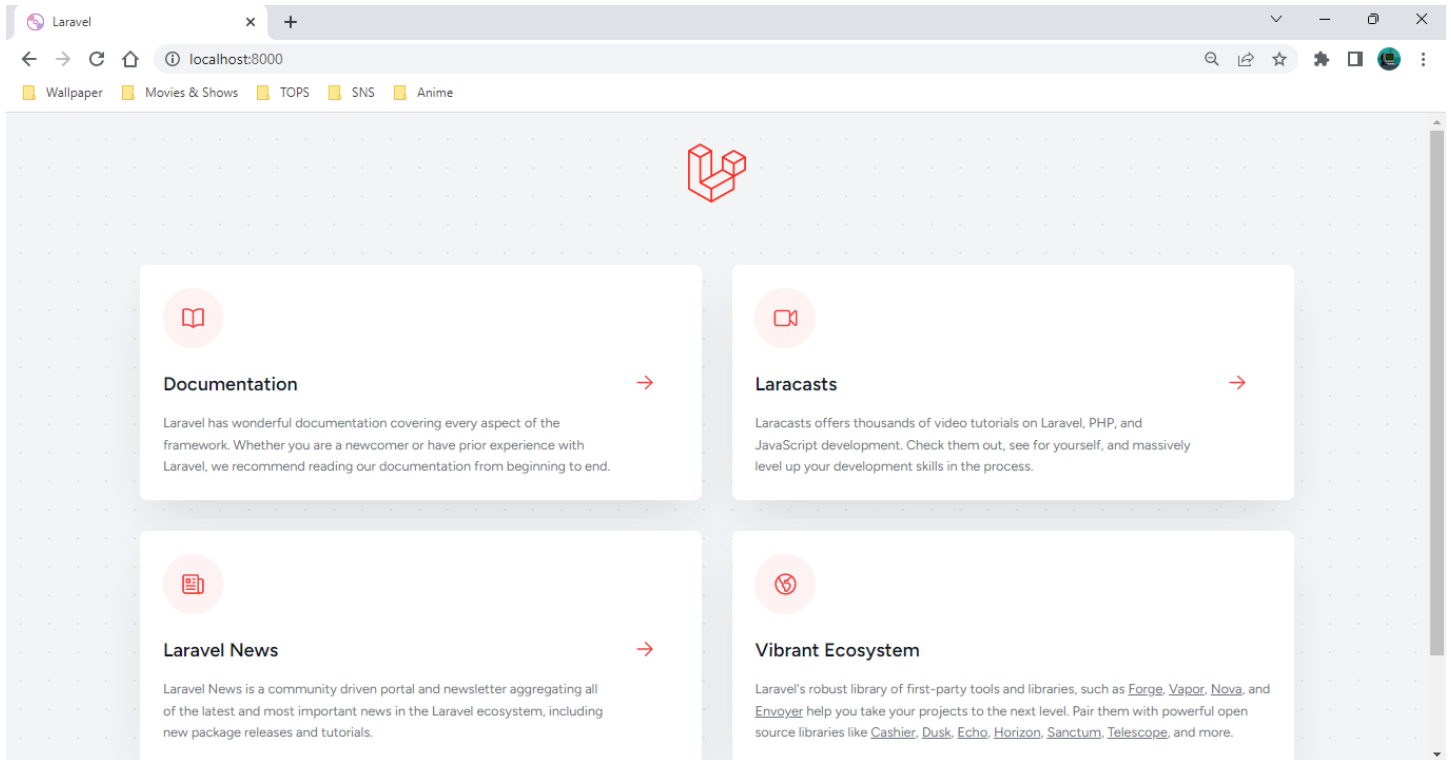
```
C:\Windows\System32\cmd.exe
> @php artisan key:generate --ansi
INFO Application key set successfully.

C:\xampp\htdocs>cd example-app
C:\xampp\htdocs\example-app>php artisan serve
```



starter kits provide backend and frontend authentication

## 8. Laravel Home Screen



## 9. Welcome to Laravel View.

