

Introduction to Laravel framework

Laravel is an open-source PHP framework, which is robust and easy to understand. It follows a model-view-controller design pattern. Laravel reuses the existing components of different frameworks which helps in creating a web application. The web application thus designed is more structured and pragmatic.

Laravel offers a rich set of functionalities which incorporates the basic features of PHP frameworks like CodeIgniter, Yii and other programming languages like Ruby on Rails. Laravel has a very rich set of features which will boost the speed of web development.

If you are familiar with Core PHP and Advanced PHP, Laravel will make your task easier. It saves a lot of time if you are planning to develop a website from scratch. Moreover, a website built in Laravel is secure and prevents several web attacks.

Advantages of Laravel

Laravel offers you the following advantages, when you are designing a web application based on it –

- The web application becomes more scalable, owing to the Laravel framework.
- Considerable time is saved in designing the web application, since Laravel reuses the components from other frameworks in developing web applications.
- It includes namespaces and interfaces, thus helping to organize and manage resources.

Composer

Composer is a tool which includes all the dependencies and libraries. It allows a user to create a project with respect to the mentioned framework (for example, those used in Laravel installation). Third party libraries can be installed easily with help of composer.

All the dependencies are noted in **composer.json** file which is placed in the source folder.

Artisan

Command line interface used in Laravel is called **Artisan**. It includes a set of commands which assist in building a web application. These commands are incorporated from the Symfony framework, resulting in add-on features in Laravel 5.1 (latest version of Laravel).

Features of Laravel

Laravel offers the following key features which makes it an ideal choice for designing web applications –

Modularity

Laravel provides 20 built in libraries and modules which helps in enhancement of the application. Every module is integrated with Composer dependency manager which eases updates.

Testability

Laravel includes features and helpers which helps in testing through various test cases. This feature helps in maintaining the code as per the requirements.

Routing

Laravel provides a flexible approach to the user to define routes in the web application. Routing helps to scale the application in a better way and increases its performance.

Configuration Management

A web application designed in Laravel will be running on different environments, which means that there will be a constant change in its configuration. Laravel provides a consistent approach to handle the configuration in an efficient way.

Query Builder and ORM

Laravel incorporates a query builder which helps in querying databases using various simple chain methods. It provides **ORM** (Object Relational Mapper) and **ActiveRecord** implementation called Eloquent.

Schema Builder

Schema Builder maintains the database definitions and schema in PHP code. It also maintains a track of changes with respect to database migrations.

Template Engine

Laravel uses the **Blade Template** engine, a lightweight template language used to design hierarchical blocks and layouts with predefined blocks that include dynamic content.

E-mail

Laravel includes a **mail** class which helps in sending mail with rich content and attachments from the web application.

Authentication

User authentication is a common feature in web applications. Laravel eases designing authentication as it includes features such as **register**, **forgot password** and **send password reminders**.

Redis

Laravel uses **Redis** to connect to an existing session and general-purpose cache. Redis interacts with session directly.

Queues

Laravel includes queue services like emailing large number of users or a specified **Cron** job. These queues help in completing tasks in an easier manner without waiting for the previous task to be completed.

Event and Command Bus

Laravel 5.1 includes **Command Bus** which helps in executing commands and dispatch events in a simple way. The commands in Laravel act as per the application's lifecycle.

Laravel - Installation

For managing dependencies, Laravel uses **composer**. Make sure you have a Composer installed on your system before you install Laravel. In this chapter, you will see the installation process of Laravel.

You will have to follow the steps given below for installing Laravel onto your system –

Step 1 – Visit the following URL and download composer to install it on your system.

<https://getcomposer.org/download/>

Step 2 – After the Composer is installed, check the installation by typing the Composer command in the command prompt as shown in the following screenshot.

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\laravel>php artisan --version
Laravel Framework version 5.1.23 <LTS>

C:\xampp\htdocs\laravel>cd\

C:\>composer

Composer version 1.0-dev <c7ed232ef42c2bd63cdba057b6c7c8043b37cd5a> 2015-10-29 09:52:59

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display this help message
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
  --ansi                    Force ANSI output
  --no-ansi                 Disable ANSI output
  -n, --no-interaction      Do not ask any interactive question
  --profile                 Display timing and memory usage information
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  -vvvv, --verbose          Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

Step 3 – Create a new directory anywhere in your system for your new Laravel project. After that, move to path where you have

created the new directory and type the following command there to install Laravel.

```
composer create-project laravel/laravel --prefer-dist
```

Now, we will focus on installation of version 5.7. In Laravel version 5.7, you can install the complete framework by typing the following command –

```
composer create-project laravel/laravel test dev-develop
```

The output of the command is as shown below –

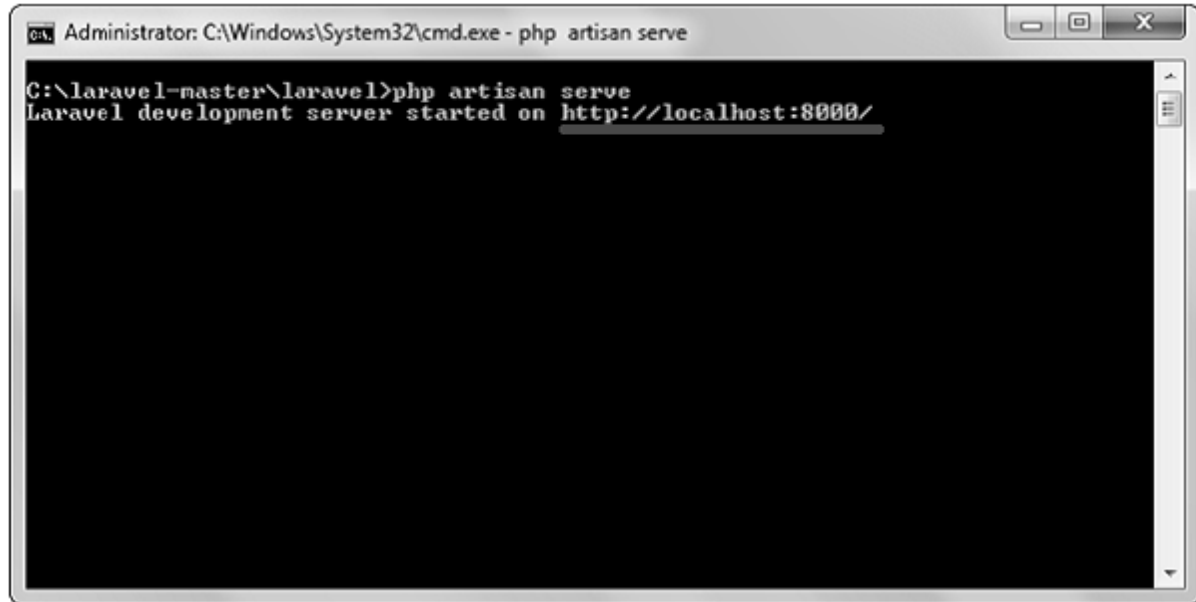
```
→ code composer create-project laravel/laravel test dev-develop
Installing laravel/laravel (dev-develop d6acad21cb2288713d9c09a31f9b4ab86f116039)
- Installing laravel/laravel (dev-develop develop): Cloning develop from cache
Created project in test
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 71 installs, 0 updates, 0 removals
- Installing vlucas/phpdotenv (v2.5.1): Loading from cache
- Installing symfony/css-selector (v4.1.3): Loading from cache
- Installing tijsverkoyen/css-to-inline-styles (2.2.1): Loading from cache
- Installing symfony/polyfill-php72 (v1.9.0): Loading from cache
- Installing symfony/polyfill-mbstring (v1.9.0): Loading from cache
- Installing symfony/var-dumper (v4.1.3): Loading from cache
- Installing symfony/routing (v4.1.3): Loading from cache
- Installing symfony/process (v4.1.3): Loading from cache
- Installing symfony/polyfill-ctype (v1.9.0): Loading from cache
- Installing symfony/http-foundation (v4.1.3): Loading from cache
- Installing symfony/event-dispatcher (v4.1.3): Loading from cache
- Installing psr/log (1.0.2): Loading from cache
- Installing symfony/debug (v4.1.3): Loading from cache
- Installing symfony/http-kernel (v4.1.3): Loading from cache
- Installing paragonie/random_compat (v9.99.99): Loading from cache
```

The Laravel framework can be directly installed with develop branch which includes the latest framework.

Step 4 – The above command will install Laravel in the current directory. Start the Laravel service by executing the following command.

```
php artisan serve
```

Step 5 – After executing the above command, you will see a screen as shown below –



```
Administrator: C:\Windows\System32\cmd.exe - php artisan serve
C:\laravel-master\laravel>php artisan serve
Laravel development server started on http://localhost:8000/
```

Step 6 – Copy the URL underlined in gray in the above screenshot and open that URL in the browser. If you see the following screen, it implies Laravel has been installed successfully.



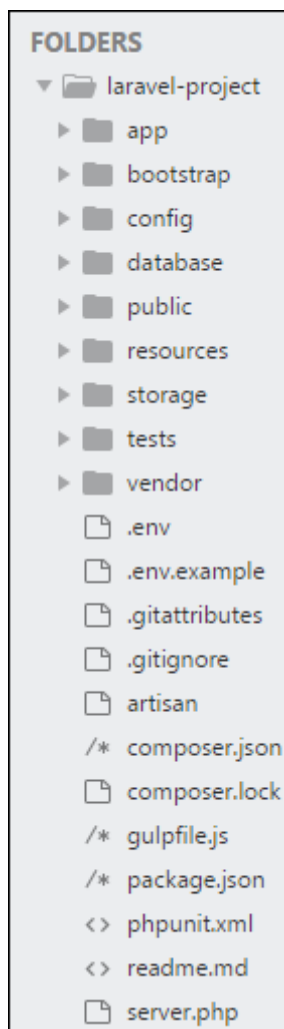
Laravel - Application Structure

[Previous](#)

[Next](#)

The application 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 as shown in the image here.

The snapshot shown here refers to the root folder of Laravel namely **laravel-project**. It includes various sub-folders and files. The analysis of folders and files, along with their functional aspects is given below –

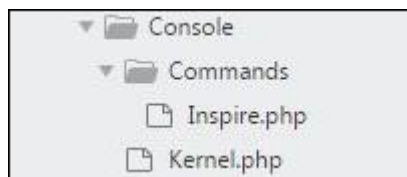


App

It is the application folder and includes the entire source code of the project. It contains events, exceptions and middleware declaration. The app folder comprises various sub folders as explained below –

Console

Console includes the artisan commands necessary for Laravel. It includes a directory named **Commands**, where all the commands are declared with the appropriate signature. The file **Kernel.php** calls the commands declared in **Inspire.php**.



If we need to call a specific command in Laravel, then we should make appropriate changes in this directory.

Events

This folder includes all the events for the project.



Events are used to trigger activities, raise errors or necessary validations and provide greater flexibility. Laravel keeps all the events under one directory. The default file included is **event.php** where all the basic events are declared.

Exceptions

This folder contains all the methods needed to handle exceptions. It also contains the file **handle.php** that handles all the exceptions.

Http

The **Http** folder has sub-folders for controllers, middleware and application requests. As Laravel follows the MVC design pattern, this folder includes model, controllers and views defined for the specific directories.

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

The **Requests** sub-folder includes all the requests of the application.

Jobs

The **Jobs** directory maintains the activities queued for Laravel application. The base class is shared among all the Jobs and provides a central location to place them under one roof.

Listeners

Listeners are event-dependent and they include methods which are used to handle events and exceptions. For example, the **login** event declared includes a **LoginListener** event.

Policies

Policies are the PHP classes which includes the authorization logic. Laravel includes a feature to create all authorization logic within policy classes inside this sub folder.

Providers

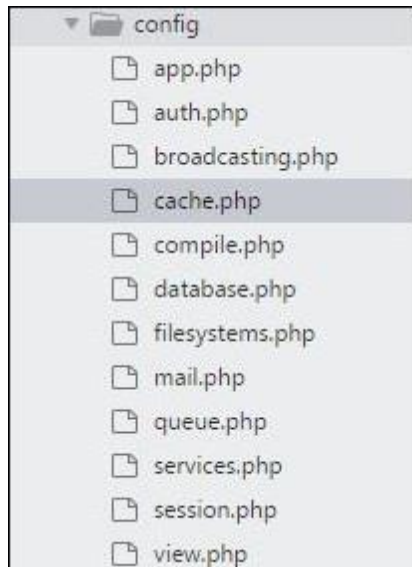
This folder includes all the service providers required to register events for core servers and to configure a Laravel application.

Bootstrap

This folder encloses all the application bootstrap scripts. 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 bootstrap.

Config

The **config** folder includes various configurations and associated parameters required for the smooth functioning of a Laravel application. Various files included within the config folder are as shown in the image here. The filenames work as per the functionality associated with them.



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 number of data records.

Public

It is the root folder which helps in initializing the Laravel application. It includes the following files and folders –

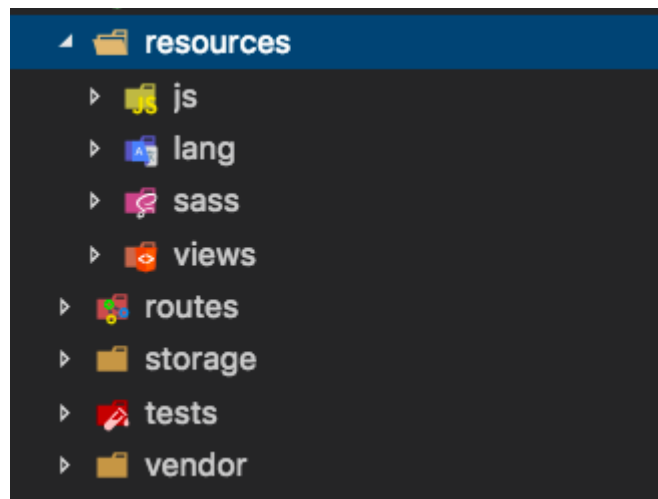
- **.htaccess** – This file gives the server configuration.
- **javascript and css** – These files are considered as assets.
- **index.php** – This file is required for the initialization of a web application.

Resources

Resources directory contains the files which enhances your web application. The sub-folders included in this directory and their purpose is explained below –

- **assets** – The assets folder include files such as LESS and SCSS, that are required for styling the web application.
- **lang** – This folder includes configuration for localization or internalization.
- **views** – Views are the HTML files or templates which interact with end users and play a primary role in MVC architecture.

Observe that the resources directory will be flattened instead of having an assets folder. The pictorial representation of same is shown below –



Storage

This is the folder that stores all the logs and necessary files which are needed frequently when a Laravel project is running. The sub-folders included in this directory and their purpose is given below –

- **app** – This folder contains the files that are called in succession.
- **framework** – It contains sessions, cache and views which are called frequently.
- **Logs** – All exceptions and error logs are tracked in this sub folder.

Tests

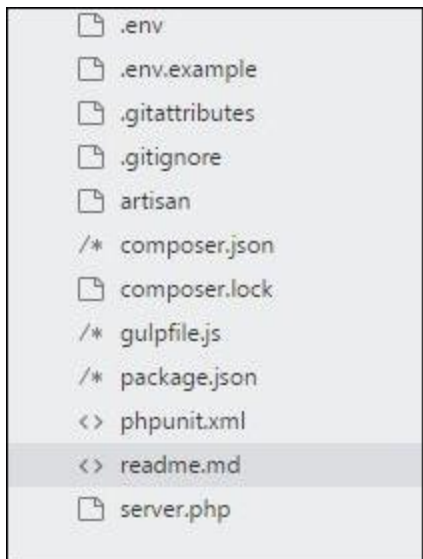
All the unit test cases are included in this directory. The naming convention for naming test case classes is **camel_case** and follows the convention as per the functionality of the class.

Vendor

Laravel is completely based on Composer dependencies, for example to install Laravel setup or to include third party libraries, etc. The Vendor folder includes all the composer dependencies.

In addition to the above mentioned files, Laravel also includes some other files which play a primary role in various functionalities such as GitHub configuration, packages and third party libraries.

The files included in the application structure are shown below –



Print Page

Laravel - Configuration

[Previous](#)

[Next](#)

In the previous chapter, we have seen that the basic configuration files of Laravel are included in the **config** directory. In this chapter, let us discuss the categories included in the configuration.

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 –

APP_ENV = local

APP_DEBUG = true

APP_KEY = base64:ZPt2wmKE/X4eEhrzJU6XX4R93rCwYG8E2f8QUA7kGK8 =

APP_URL = http://localhost

DB_CONNECTION = mysql

```
DB_HOST = 127.0.0.1
DB_PORT = 3306
DB_DATABASE = homestead
DB_USERNAME = homestead
DB_PASSWORD = secret
CACHE_DRIVER = file
SESSION_DRIVER = file
QUEUE_DRIVER = sync
REDIS_HOST = 127.0.0.1
REDIS_PASSWORD = null
REDIS_PORT = 6379
MAIL_DRIVER = smtp
MAIL_HOST = mailtrap.ioMAIL_PORT = 2525
MAIL_USERNAME = null
MAIL_PASSWORD = null
MAIL_ENCRYPTION = null
```

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.

Retrieval of Environment Variables

All the environment variables declared in the **.env** file can be accessed by **env-helper** functions which will call the respective parameter. These variables are also listed into **\$_ENV** global variable whenever application receives a request from the user end. You can access the environment variable as shown below –

```
'env' => env('APP_ENV', 'production'),
```

env-helper functions are called in the **app.php** file included in the **config** folder. The above given example is calling for the basic local parameter.

Accessing Configuration Values

You can easily access the configuration values anywhere in the application using the global config helper function. In case if the configuration values are not initialized, default values are returned.

For example, to set the default time zone, the following code is used –

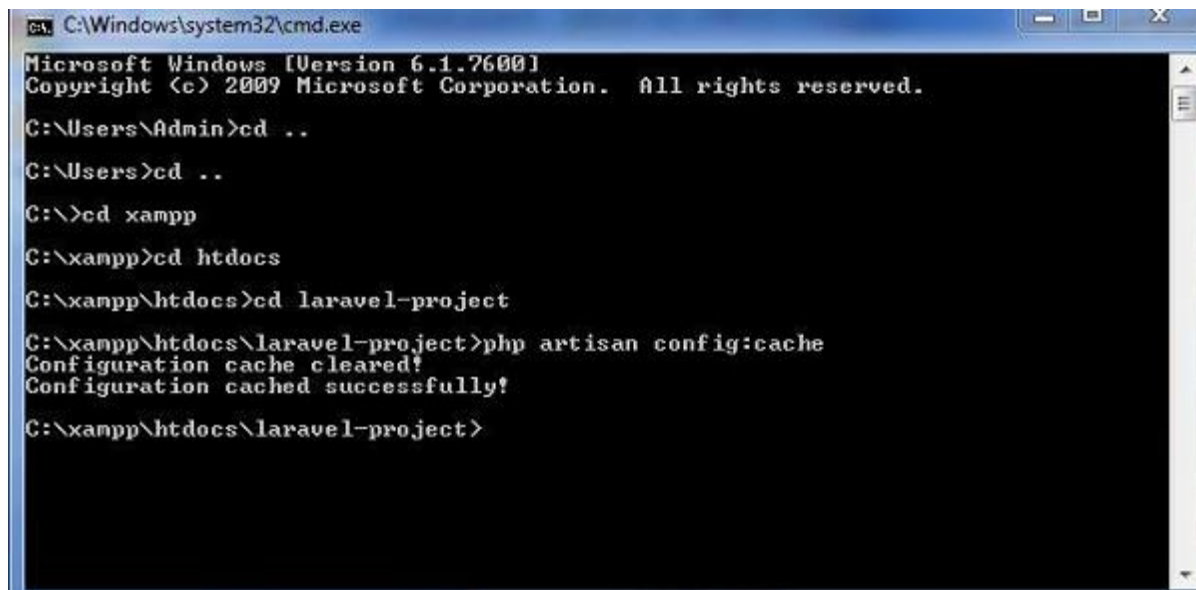
```
config(['app.timezone' => 'Asia/Kolkata']);
```

Caching of Configuration

To increase the performance and to boost the web application, it is important to cache all the configuration values. The command for caching the configuration values is –

```
config:cache
```

The following screenshot shows caching in a systematic approach –



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

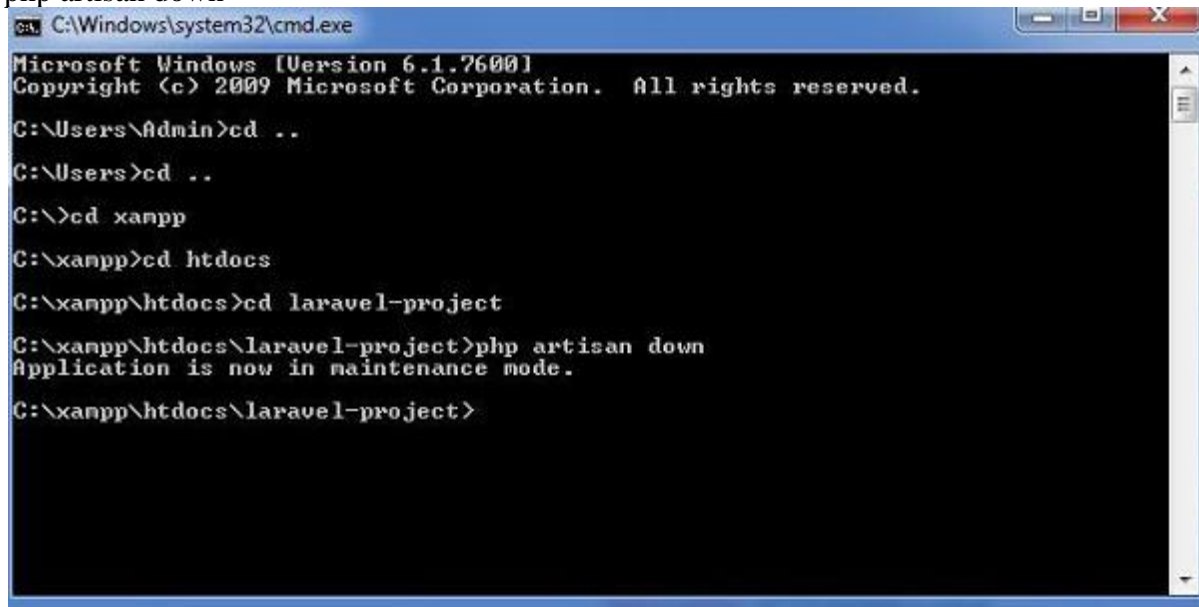
C:\Users\Admin>cd ..
C:\Users>cd ..
C:\>cd xampp
C:\xampp>cd htdocs
C:\xampp\htdocs>cd laravel-project
C:\xampp\htdocs\laravel-project>php artisan config:cache
Configuration cache cleared!
Configuration cached successfully!
C:\xampp\htdocs\laravel-project>
```

Maintenance Mode

Sometimes you may need to update some configuration values or perform maintenance on your website. In such cases, keeping it in **maintenance mode**, makes it easier for you. Such web applications which are kept in maintenance mode, throw an exception namely **MaintenanceModeException** with a status code of 503.

You can enable the maintenance mode on your Laravel web application using the following command –

php artisan down



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

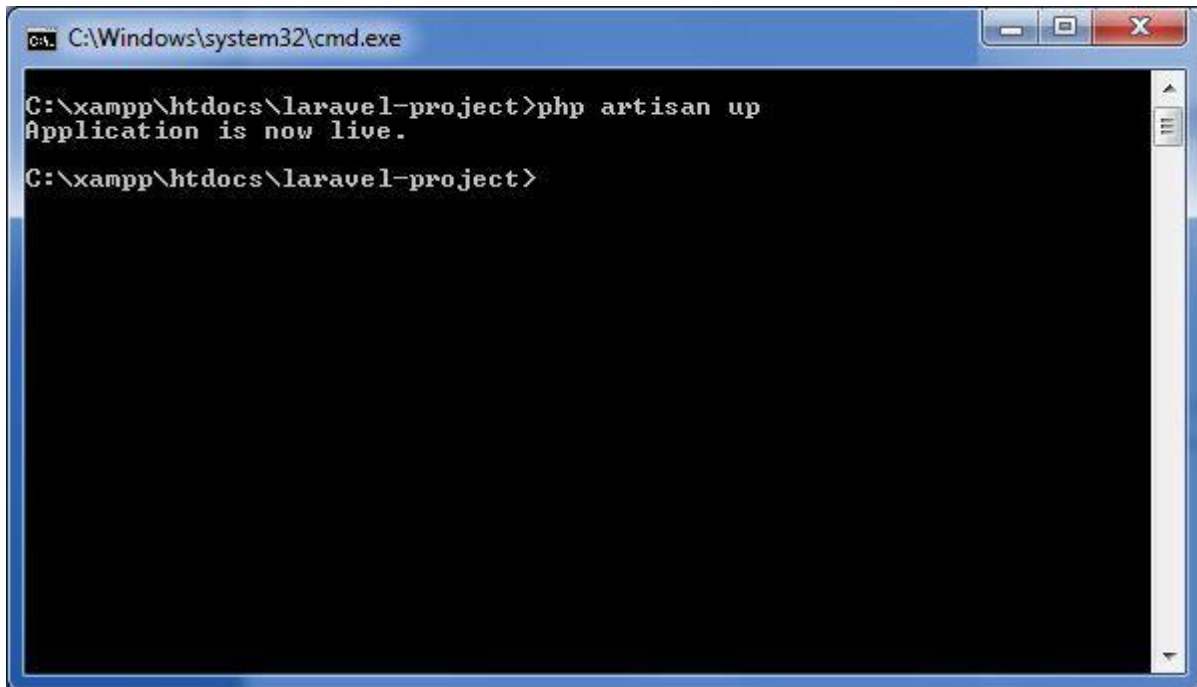
C:\Users\Admin>cd ..
C:\Users>cd ..
C:\>cd xampp
C:\xampp>cd htdocs
C:\xampp\htdocs>cd laravel-project
C:\xampp\htdocs\laravel-project>php artisan down
Application is now in maintenance mode.
C:\xampp\htdocs\laravel-project>
```

The following screenshot shows how the web application looks when it is down –



Once you finish working on updates and other maintenance, you can disable the maintenance mode on your web application using following command –

php artisan up



```
C:\Windows\system32\cmd.exe
C:\xampp\htdocs\laravel-project>php artisan up
Application is now live.
C:\xampp\htdocs\laravel-project>
```

Now, you can find that the website shows the output with proper functioning and depicting that the maintenance mode is now removed as shown below –



Laravel - Routing

[Previous](#)

[Next](#)

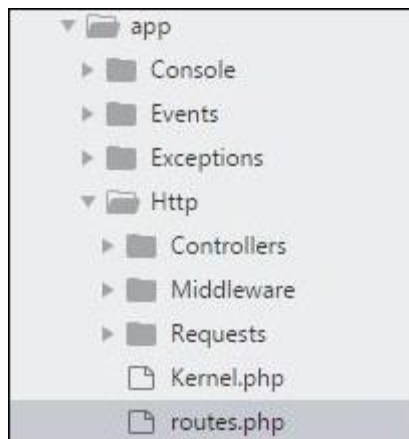
In Laravel, all requests are mapped with the help of routes. Basic routing routes the request to the associated controllers. This chapter discusses routing in Laravel.

Routing in Laravel includes the following categories –

- Basic Routing
- Route parameters
- Named Routes

Basic Routing

All the application routes are registered within the **app/routes.php** file. This file tells Laravel for the URIs it should respond to and the associated controller will give it a particular call. The sample route for the welcome page can be seen as shown in the screenshot given below –



```
Route::get('/', function () {  
    return view('welcome');});
```

Example

Observe the following example to understand more about Routing –

app/Http/routes.php

```
<?php  
Route::get('/', function () {  
    return view('welcome');  
});
```

resources/view/welcome.blade.php


```
<!DOCTYPE html>
<html>
  <head>
    <title>Laravel</title>
    <link href = "https://fonts.googleapis.com/css?family=Lato:100" rel = "stylesheet"
      type = "text/css">

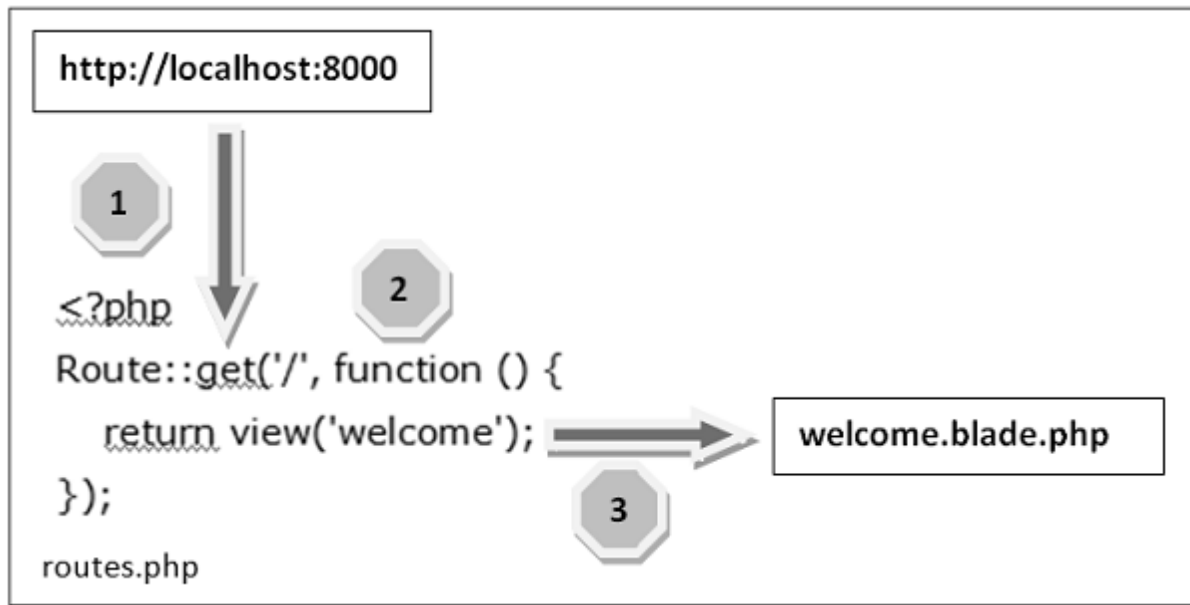
    <style>
      html, body {
        height: 100%;
      }
      body {
        margin: 0;
        padding: 0;
        width: 100%;
        display: table;
        font-weight: 100;
        font-family: 'Lato';
      }
      .container {
        text-align: center;
        display: table-cell;
        vertical-align: middle;
      }
      .content {
        text-align: center;
        display: inline-block;
      }
      .title {
        font-size: 96px;
      }
    </style>
  </head>

  <body>
    <div class = "container">

      <div class = "content">
        <div class = "title">Laravel 5.1</div>
      </div>

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

The routing mechanism is shown in the image given below –



Let us now understand the steps involved in routing mechanism in detail –

Step 1 – Initially, we should execute the root URL of the application.

Step 2 – Now, the executed URL should match with the appropriate method in the **route.php** file. In the present case, it should match the method and the root (‘/’) URL. This will execute the related function.

Step 3 – The function calls the template file **resources/views/welcome.blade.php**. Next, the function calls the **view()** function with argument ‘**welcome**’ without using the **blade.php**.

This will produce the HTML output as shown in the image below –



Route Parameters

Sometimes in the web application, you may need to capture the parameters passed with the URL. For this, you should modify the code in **routes.php** file.

You can capture the parameters in **routes.php** file in two ways as discussed here –

Required Parameters

These parameters are those which should be mandatorily captured for routing the web application. For example, it is important to capture the user's identification number from the URL. This can be possible by defining route parameters as shown below –

```
Route::get('ID/{id}',function($id) {  
    echo 'ID: '.$id;  
});
```

Optional Parameters

Sometimes developers can produce parameters as optional and it is possible with the inclusion of **?** after the parameter name in URL. It is important to keep the default value mentioned as a parameter name. Look at the following example that shows how to define an optional parameter –

```
Route::get('user/{name?}', function ($name = 'TutorialsPoint') { return $name;});
```

The example above checks if the value matches to **TutorialsPoint** and accordingly routes to the defined URL.

Named Routes

Named routes allow a convenient way of creating routes. The chaining of routes can be specified using name method onto the route definition. The following code shows an example for creating named routes with controller

```
Route::get('user/profile', 'UserController@showProfile')->name('profile');
```

The user controller will call for the function **showProfile** with parameter as **profile**. The parameters use **name** method onto the route definition.