

Introduction to Laravel

Lecture#25

Lecture Outline



1. Introduction to Laravel
2. Starting the framework
3. Laravel directory structure
4. Routing basics
5. Routing Options
6. View Engine (Blade)
7. Creating Views

Introduction to Laravel

Why Laravel?



- Laravel is a web application framework with expressive, elegant syntax.
- Laravel makes easy the tasks of authentication, session, caching in web projects.
- Powerful web applications can be made with Laravel.
- Laravel framework makes implementation of authentication techniques very simple. Laravel provides a very simple way to organize authorization logic and control access to various resources.
- Laravel is the best PHP framework as it has Object Oriented libraries and other pre-installed ones, which are not found in any other PHP frameworks.
- Laravel framework offers a build in a tool named Artisan. This tool allows a developer to perform the majority of those repetitive and tedious programming tasks which most of the developers avoid performing manually.



Why Laravel?

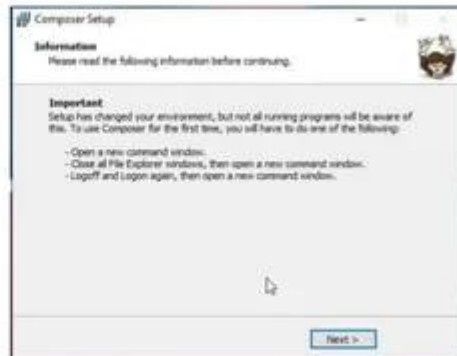
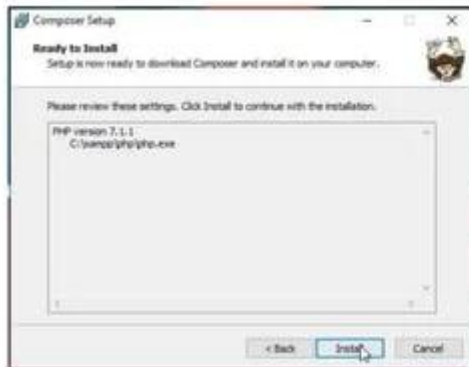
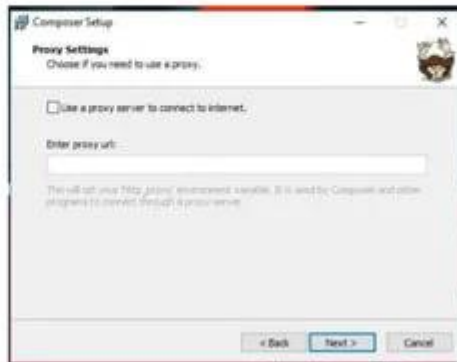
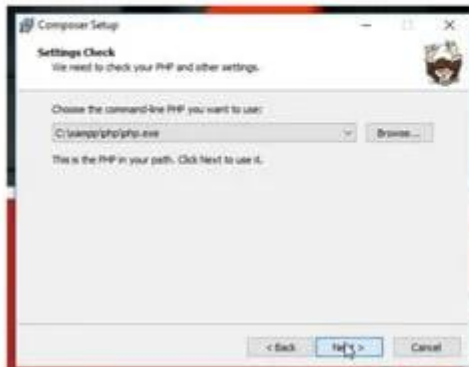
- Another reason which makes Laravel the best PHP framework is it supports MVC Architecture like Symfony, ensuring clarity between logic and presentation.
- Laravel takes care of the security within its framework. It uses the hashed password, which means that the password would never save as the plain text in the database. Laravel framework uses prepared SQL statements which make injection attacks unimaginable.
- With Laravel framework database migrations, it is extremely easy. As long as you keep all of the database work in migrations and seeds, you can easily migrate the changes into any other development machine you have.
- The Blade templating engine of the Laravel framework is very intuitive and helps to work with the typical PHP/HTML spaghetti so perfect, that's it one of the best features of the Laravel framework.



Installation Process

- We will be using windows operating system.
- Laravel utilizes Composer to manage its dependencies.
- At first install XAMPP which supports apache and mysql(MariaDB)
- Download composer from <https://getcomposer.org/download/>
- Git bash can be used.
- Install composer.
- Choose your text editor.(Notepad++,VSCode,Sublime etc.)

Installation Process





What is composer?

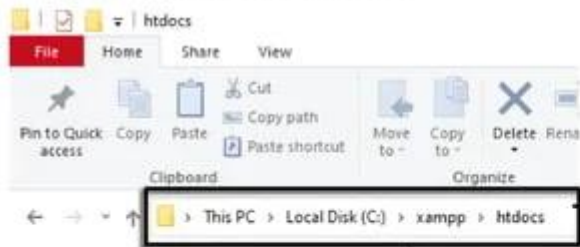
- Composer is a tool which is used for dependency management for the framework
- It allows you to declare the libraries your project depends on and it will manage (install/update) them for you.
- It manages libraries on a per-project basis, installing them in a directory (e.g. vendor) inside your project.
- You can update all your dependencies in one command with composer
- Composer requires PHP 5.3.2+ to run.
- Composer is multi-platform and we strive to make it run equally well on Windows, Linux and macOS.
- With composer its very easy to create and manage projects and update dependencies.

Starting the Framework

Creating the project



- PHP codes are to be placed in htdocs folder.
- If you have git bash installed you can use it for running commands.
- You can also use `cmd` application of windows.
- Open cmd in htdocs folder.



Click here
Type `cmd`
Hit enter



Running the project

- The Laravel project can be run by 2 ways
 - With apache (Like other php apps we studied)
 - With built in development server
- With apache
 - Open Xampp
 - Start apache
 - Open browser and type http://localhost/demo_project_laravel/
 - You will get the application directory
 - Click on public folder the default project interface will appear
 - You can also directly browse http://localhost/demo_project_laravel/public/
 - **If you have running apache in any customized port like 8082 then don't forget to mention the port like**
http://localhost:8082/demo_project_laravel/public/



Running the project with apache

localhost:8080/project_laravel/public/



Laravel

[DOCS](#)

[LARACASTS](#)

[NEWS](#)

[BLOG](#)

[MOVA](#)

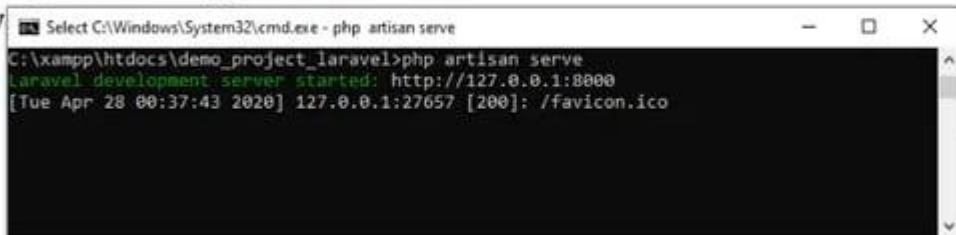
[FORGE](#)

[VAPOR](#)

[GITHUB](#)

Running the project with built in development server

- In Laravel, we can run Laravel Development server by typing ***php artisan serve*** command on terminal or command line.
- Open cmd in project directory and type
`php artisan serve`
- By default it will run in 8000 port. If 8000 is used then use another port.
`php artisan serve --port=8082`
- The server will run now type
`localhost:8000` [for default]
`localhost:8082` [for customized port]
- This is the URL of your project.
- To stop the server type `ctrl+c` in the cmd.
- It w

A screenshot of a Windows command prompt window. The title bar reads "Select C:\Windows\System32\cmd.exe - php artisan serve". The command prompt shows the following text:

```
C:\xampp\htdocs\demo_project_laravel>php artisan serve
Laravel development server started: http://127.0.0.1:8000
[Tue Apr 28 00:37:43 2020] 127.0.0.1:27657 [200]: /favicon.ico
```

Application Directory Structure



Name

Date modified

Type

| | | |
|---|--------------------|-------------|
|  app | 27-Apr-20 11:39 PM | File folder |
|  bootstrap | 27-Apr-20 11:39 PM | File folder |
|  config | 27-Apr-20 11:39 PM | File folder |
|  database | 27-Apr-20 11:39 PM | File folder |
|  public | 27-Apr-20 11:39 PM | File folder |
|  resources | 27-Apr-20 11:39 PM | File folder |
|  routes | 27-Apr-20 11:39 PM | File folder |
|  storage | 27-Apr-20 11:39 PM | File folder |
|  tests | 27-Apr-20 11:39 PM | File folder |
|  vendor | 27-Apr-20 11:49 PM | File folder |



Directory Structure

- **app** - The `app` directory contains the core code of your application. We'll explore this directory in more detail soon; however, almost all of the classes in your application will be in this directory.
- **bootstrap** - The `bootstrap` directory contains the `app.php` file which bootstraps the framework. This directory also houses a cache directory which contains framework generated files for performance optimization such as the route and services cache files.
- **config** - The `config` directory, as the name implies, contains all of your application's configuration files. It's a great idea to read through all of these files and familiarize yourself with all of the options available to you.
- **database** - The `database` directory contains your database migrations, model factories, and seeds. If you wish, you may also use this directory to hold an SQLite database.
- **public** - The `public` directory contains the `index.php` file, which is the entry point for all requests entering your application and configures autoloading. This directory also houses your assets such as images, JavaScript, and CSS.
- **resources** - The `resources` directory contains your views as well as your raw, un-compiled assets such as LESS, SASS, or JavaScript. This directory also houses all of your language files.



Directory Structure

- **routes** - The `routes` directory contains all of the route definitions for your application. By default, several route files are included with Laravel: `web.php`, `api.php`, `console.php` and `channels.php`.
- **storage** - The `storage` directory contains your compiled Blade templates, file based sessions, file caches, and other files generated by the framework.
 - `app` - The `app` directory may be used to store any files generated by your application.
 - `framework` - The `framework` directory is used to store framework generated files and caches.
 - `logs` - the `logs` directory contains your application's log files.

Generally `storage/app/public` is used to store files. In that case you should

create a link with `public/storage` by `php artisan storage:link` command.

- **test** - The `tests` directory contains your automated tests.
- **vendor** - The `vendor` directory contains your Composer dependencies.



app Directory Structure

- **Console** - The `Console` directory contains all of the custom Artisan commands for your application. These commands may be generated using the `make:command` command. This directory also houses your console kernel, which is where your custom Artisan commands are registered and your scheduled tasks are defined.
- **Exceptions** - The `Exceptions` directory contains your application's exception handler and is also a good place to place any exceptions thrown by your application. If you would like to customize how your exceptions are logged or rendered, you should modify the Handler class in this directory.
- **Http** - The `Http` directory contains your controllers, middleware, and form requests. Almost all of the logic to handle requests entering your application will be placed in this directory.
- **Providers** - The `Providers` directory contains all of the service providers for your application. Service providers bootstrap your application by binding services in the service container, registering events, or performing any other tasks to prepare your application for incoming requests.
- **Mail** - This `directory` does not exist by default, but will be created for you if you execute the `make:mail` Artisan command. The Mail directory contains all of your classes that represent emails sent by your application.

Laravel Routing



- Routing is one of the essential concepts in Laravel.
- Routing in Laravel allows you to route all your application requests to its appropriate controller.
- The most basic Laravel routes accept a URI and a **Closure**, providing a very simple and expressive method of defining routes.

```
Route::method(uri,closure);
```

```
Route::get('/hello', function () {  
    return 'Hello World';  
});
```

URI

Closure

- All Laravel routes are defined in your route files, which are located in the routes directory.
- These files are automatically loaded by the framework. The `routes/web.php` file defines routes that are for your web interface.



Routing

- These routes are assigned the web middleware group, which provides features like session state and CSRF protection.
- For most applications, you will begin by defining routes in your `routes/web.php` file.
- The routes defined in `routes/web.php` may be accessed by entering the defined route's URL in your browser.
 - <http://localhost:8000/hello>
- We can also write necessary functions in other files and bind them with route.
- Available routing methods

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```
- If you are defining a route that redirects to another URI, you may use the `Route::redirect` method.

```
Route::redirect('/here', '/there');
```

Creating First Route

- Open `routes/web.php` in any text editor. Here VsCode will be used.
- Write a route

```
Route::get('/hello', function () {  
    return 'Hello World';  
});
```

```
13  
14 ✓ Route::get('/', function () {  
15     return view('welcome');  
16 });  
17 ✓ Route::get('/hello',function(){  
18     return 'Hello World';  
19 });  
20
```

- Type the route in browser

- <http://localhost:8000/hello> ← → ↻ ⓘ localhost:8000/hello

Hello World

- Try this

```
Route::get('/hello/bold', function () {  
    return '<h1>Hello World</h1>';  
});
```

```
17 Route::get('/hello',function(){  
18     return "Hello World";  
19 });  
20 Route::get('/hello/bold',function(){  
21     return "<h1>Hello World</h1>";  
22 });
```

- Output

← → ↻ ⓘ localhost:8000/hello/bold

Hello World

- You can add as much routes you can and also add HTML with it.

Routing Options



In the second parameter of the route, we have the following options:

- A Closure
- An Array
- A String

Example:

Closure: `Route::get('/home', function(){return "Hello"});`

Array: `Route::get('/home', ['uses'=>'HomeController@index']);`

String: `Route::get('/home', 'HomeController@index');`

View Engine (Blade)



- Till now we can create routes and give output in HTML we can add HTML tags and design also but its not convenient to write HTML codes here.
- Views typically contain the HTML of your application and provide a convenient way of separating your controller and domain logic from your presentation logic.

```
<html>
  <body>
    <h1>Hello, Form View</h1>
  </body>
</html>
```

- For views we will be using blade engine. Blade is the simple, yet powerful templating engine provided with Laravel.
- Unlike other popular PHP templating engines, Blade does not restrict you from using plain PHP code in your views.
- Blade views are compiled into plain PHP code and cached until they are modified, meaning Blade adds essentially zero overhead to your application.



View Engine (Blade)

- Blade view files use the `.blade.php` file extension.
- Files with blade are stored in the `resources/views` directory.
- Using blade as **Layout** will be discussed later studies. Lets create our first view.

Creating Views



- Go to `resources/views`
- Create a file with `.blade.php`
- Put some html codes
- After creating views we need to bind it with Route
- Go to `web.php` and define a route or Modify existing one

```
Route::get('/hello',function(){  
    return view("hello");  
})->name('hello');
```

- `view()` function receives the blade file name.
Give only the file name without `.blade.php`

```
Route::get('/hello',function(){  
    return view("hello");  
})->name('hello');
```

```
≡ hello.blade.php ×  
resources > views > ≡ hello.blade.php > html > ≡  
1 <html>  
2     <head></head>  
3     <body>  
4         <h1>Hello Advanced Web</h1>  
5     </body>  
6 </html>
```

← → ↻ ⓘ localhost:8000/hello

Hello Advanced Web



Books

- PHP Advanced and Object-Oriented Programming, 3rd Edition; Larry Ullman; Peachpit, Press, 2013
- PHP Objects, Patterns and Practice, 5th Edition; Matt Zandstra; Apress, 2016
- Learning PHP, MySQL, JavaScript and CSS, 2nd Edition; Robin Nixon; O'Reilly, 2009
- Eloquent JavaScript: A Modern Introduction to Programming; Marijn Haverbeke; 2011
- Learning Node.js: A Hands On Guide to Building Web Applications in JavaScript; Marc Wandschneider; Addison-Wesley, 2013
- Beginning Node.js; Basarat Ali Syed; Apress, 2014



References

1. <https://laravel.com/>
2. <https://getcomposer.org/>
3. <http://www.laravelinterviewquestions.com/>



Thank You!