What is Laravel ?

* Open Source PHP Framework
* Robust and Easy to Understand
* It Follows MVC Pattern
* Latest version is 8.x
* Community of Laravel is Bigger then Codeigniter as per the Github Search Results
* Comes with Artisan CLI Tool
* It comes with a simple but robust template engine like Blade. Blade template engine allows PHP programmers to optimize the performance of the web application by enhancing and manipulating views.

Laravel Installation

**Installing Laravel from the Web**

Need composer to be Installed on Device

Composer is Package Manager for Php Dependencies Like Nodejs’s NPM

composer create-project laravel/laravel example-app

### cd example-app

### php artisan serve

**Create Project by Laravel Installer**

### composer global require laravel/installer

### laravel new demo

### cd demo

### php artisan serve

File & Folder Structure in Laravel

* **app** 
  + contains core codes of Application
* **bootstrap** 
  + contains app.php and cache directory which contains framework generated files for performance optimization such as the route and cache service files
* **config** 
  + As per its name it contains application configuration files
* **Database**
  + This Directory contains database migrations or models
  + Also can be used for Holding an SQLite Database
* **public**
  + contains index.php which is entry point of App
  + can be used for housing your assets like css,js,images etc
* **resource**
  + contains views of an app and also can used for un-compiled js,css files
* **routes**
  + contains all of the route definitions for the app
  + There are some routes files comes by default including web.php, api.php, console.php, channels.php
* **storage**
  + contains logs, compiled blade templates, file based sessions, files caches
  + storage/app/public directory can be used to store user-generated avatars, profile etc
* **tests**
  + contains automated tests
* **vendor**
  + it contains composer dependencies
* *.env*
* *composer.json*
* *package.json*
* *Server.php*

Create View

* Create view on Directory **resources/views/home.blade.php**
* <div>Hello World</div>
* Then Route it by adding route on **routes/web.php file**
* **Route::view('/home', 'home');**
* Another Way of Load View
* **Route::get('/home', function () {**
* **return view('home');**
* **});**

Create Controller

* Create Controller by Command = **php artisan make:controller welcome**
* Controller Will Create on **app/Http/Controllers/welcome.php**
* Then Create Functions like this inside Main(Controller Named Class)

function welcome()

{

echo "Welcome";

}

* Then Route these Controller in Web Route

Route::get('/welcome', [welcome::class, 'welcome']);

* Pass the parameter in URI and use them in functions

Route::get('/welcome/{name}', [welcome::class, 'welcome']);

* Use them in Functions

function welcome($name)

{

echo "Welcome " . $name;

}

* Load View in Controller

**function welcome()**

**{**

**return view('welcome');**

**}**

* Pass data into views

function welcome()

{

return view('welcome', ['name' => 'Jemish']);

}

* Use Passed Data into Views

<div>{{$name}}</div>

Create Components

* Components are mainly used for Reusability
* Example : header, footer etc
* Create component by command = **php artisan make::component component\_name**
* It will Create 2 files
  + app/resources/views/**components/header.blade.php**
  + app/view/**header.php**
* Here components is nothing but views so it will act views
* So Doing Code as Views and Reusability of Components Makes it Easy and Fast
* header.php file of that component contains code of component, there we can define data and pass it through component view
* In view pages we can render components like

<x-header name="Home" />

<div>

Hello

</div>

* Here passed properties can be accessed by header.php coding file of components in \_\_construct method by passing parameters inside it

public $page = "";

public function \_\_construct($name)

{

$this->page = $name;

//

}

* And That Variables can be used in components using double {} brackets

<div>

Welcome to {{$page}} Page

</div>

Intro to Blade Template Engine

* Normal PHP Code Looks like <?php echo $name?>
* It Makes Code Little Lengthy or Too much Repeated Statements like <?php?>
* So Blade Template Makes it Easy, We can run Php Code in Other Way Like Accessing Php Variables as {{name}}
* It makes coding faster and Short Syntaxes

Some shorter syntax then core PHP in Blade Template

<div>

@if ($age > 18)

You're Eligible for Voting

@else

You're not Eligible for Voting

@endif

<br>

@for ($i = 0; $i < 5; $i++)

{{$i}}

@endfor

</div>

We can Also use Php Variables in Javascript like these

var users = @json($users);

console.table(users);

Form + CSRF

<form action="login" method="POST">

@csrf

@method('POST')

Email :

<input type="text" name="email"/>

<br>

Password :

<input type="password" name="password"/>

<br>

<input type="submit" value="submit">

</form>

Here @method is used for setting request method

We can user there , GET POST PUT DELETE

Mainly GET, POST, PUT, DELETE is Used In APIs

Here CSRF Token is Mandatory to pass inside form

Otherwise You’ll get en Error 419 || Page Expired

Why is CSRF Token Used?

* For Security Reason
* It removes Cross Site Requests
* It is used for Handle Request from pages of Framework itself

MiddleWare

* Middleware is Type of Functions which will
  + Run before any function call of Controller , and
  + Run before any view rendering
* It has Request Information Access like GET,POST etc.
* Specially used for Validation or Giving Restrictions of View or Controller Functions

# Group Middleware

* Create Middleware by Command = **php artisan make:middleware middlewareName**
* Register middleware in **app/Http/Kernel.php**
  + Inside => protected $middlewareGroups
  + 'checkAge' => [
  + \App\Http\Middleware\ageCheck::class,
  + ]
* Code in Middleware set Validation which type you want **app/Http/Middleware/middlewareName.php**
  + public function handle(Request $request, Closure $next)
  + {
  + if ($request->age && $request->age > 18) {
  + return $next($request);
  + } else {
  + return redirect('welcome');
  + }
  + }
* Set Middleware on Routes **routes/web.php**

Route::group(['middleware' => ['checkAge']], function () {

Route::get('/home', [welcome::class, 'home']);

Route::view('/login', 'login');

});

* Explanation
  + Here Router::group will set checkAge middleware on declared routes inside function

# Route Middleware

* Create Middleware by command via artisan
* Register Route Middleware in **app/Http/Kernel.php**
  + Like this
  + Inside => protected $routeMiddleware
  + 'checkAge' => \App\Http\Middleware\ageCheck::class
* Set route middleware on specific routes

Route::view('/login', 'login')->middleware('checkAge');

Database

* Set Configuration of Database in **.env** file of your Project Root Folder

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=laravel

DB\_USERNAME=root

DB\_PASSWORD=

* Import Database into Controller
  + use Illuminate\Support\Facades\DB;
* Simple Select Query
  + function getUserData()
  + {
  + return DB::select('select \* from user');
  + }

Model

* Model can be Created by Command = php artisan make:model modelName
* Therefor Model will be created in **app/Models/modelName.php**
* By default model model refers to database table like these
  + user model refers to users table
  + employee refers to employees table
  + payment refers to payments table
* By default user model comes with Laravel with useful auth functionality
* But we can modify table name inside model

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class user extends Model

{

use HasFactory;

public $table = 'users';

}

* Import model in controller

use App\Models\user;

* Use model

function getUserData()

{

return user::all();

}

HTTPClient

* HTTPClient is Laravel Support Library for use HTTP Requests in Php Code (Controller)
* Easier then Curl Request
* Import Http in Controller

use Illuminate\Support\Facades\Http;

* Making a request

function getUserData()

{

return Http::get('https://reqres.in/api/users?page=1');

}

Session

* As we know, HTTP is **Stateless**, so We need something by which server can understand the source of HTTP Requests,
* For that time, sessions provide a way to store information about the user across multiple **requests** .
* That user information is typically placed in a **persistent** store / backend that can be accessed from subsequent requests.
* Store a data into Sessions

function login(Request $req)

{

$data = $req->input();

$req->session()->put('user', $data['email']);

echo session('user');

return redirect('welcome');

}

* Get data from Session

function welcome(Request $req)

{

if (session()->has('user')) {

echo $req->session()->get('user');

} else {

return view('welcome');

}

}

* Flash Sessions

$req->session()->flash('user', $data['email']);

* Flash Sessions can be used for Validation
* Flash Sessions will Automatically delete after one use

File Uploading

* Here is the Example of File Uploading

<form action="upload" method="POST" enctype="multipart/form-data">

File Uploading

@csrf

<br>

<input type="file" name="file" >

<br>

<input type="submit" value="submit" name="submit">

</form>

* Controller Function

function upload(Request $req)

{

$path = $req->file('file')->store('avatars');

echo $path;

}

* File Will Upload by **Store** function on defined Folder
* Therefor Uploaded File Directory Should be **app/storage/app/avatars/file**
* We can use **putFile** functions of **Storage** Facade

$path = Storage::putFile('file', $req->file('avatar'));

* Store a File with custom file name

$path = $req->file('file')->storeAs(

'avatars',

'filename.png'

);

Another way

$path = Storage::putFileAs(

'avatars',

$req->file('file'),

'filename.png'

);

* So, storeAs and putFileAs of Storage Facade are the functions of Upload a file via Custom File Name
* Get File name and File Extension

$name = $req->file('file')->getClientOriginalName();

$extension = $req->file('file')->extension();

echo $name . ' and extension is ' . $extension;

getClientOriginalName() & extension() are used for the get filename and Extension

* Some Useful Methods of Storage Facade
  + Storage::delete('file.jpg');
  + Storage::delete(['file.jpg', 'file2.jpg']);
  + $files = Storage::files($directory);
  + $files = Storage::allFiles($directory);
  + $directories = Storage::directories($directory);
  + $directories = Storage::allDirectories($directory);
  + Storage::makeDirectory($directory);
  + Storage::deleteDirectory($directory);