

# Laravel, The PHP Framework

## Documented by

Mobin Ahmed

BSc in Computer Science and Engineering (2021),

North East University Bangladesh

[mobinahmed102345@gmail.com](mailto:mobinahmed102345@gmail.com)

01314890701

\* After creating the project folder in the htdocs folder, do the command as Shift + right click and open the powershell window, then write the command for installing Laravel by using composer =

`composer create-project laravel/laravel {project folder name goes here}`

\* **What's the framework?** = Framework is a predefined structure.

\* **What's the Laravel framework?** = Laravel framework is predefined structure by using php.

## Designation of software companies:

- Intern developer
- Junior developer/ Junior software engineer
- Developer/ Software engineer
- Senior developer/ Senior software engineer
- Project manger
- System analyst

## History of the PHP and Laravel framework:

\* There was no php framework before the year 2004.

\* PHP started its revolution after the year 2000 and the big companies started to use the php eagerly. The reason behind that is;

- It's an open-source programming language (advantage).
- No specific pattern of coding (disadvantage).
- It has no security, but if one wants, can use security personally (advantage, but partially).
- It runs very fast in server (advantage).
- The maintenance cost of it is low (advantage).

- Can't manage huge scale project by using it (disadvantage).

\* Father of php is Rasmus Lerdorf.

\* The compiler & server of the php is Apache.

\* Laravel came in 2004 to diminish disadvantages.

\* Father of Laravel is Taylor Otwell.

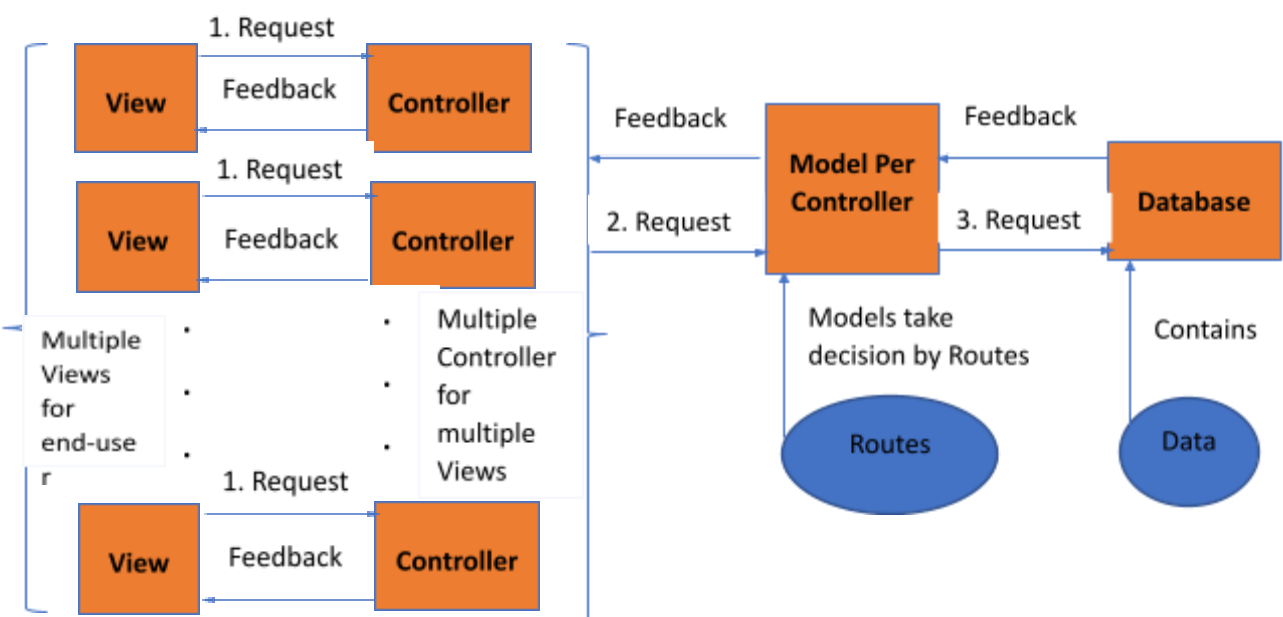
### Two big things to know:

- OPP (Object Oriented Programming) = Thinking pattern of projects
- MVC (Model View Controller) = Coding pattern of projects
  - Model (classes of php) = Data
  - View = End-user view
  - Controller (classes of php) = Logic

\* Coding pattern of every framework such as Laravel is Model View Controller (MVC)

\* **How does MVC work?** = One project may have many views. These views are controlled by Controller.

Views request Controller for data, but data is in the database, Controller can't request database directly, for this Controller request Model and then Model request database for data. Database provides data to the Model, Model provides data to the Controller, Controller provides data to the View, and finally View shows data to the end-user. For the convenience diagram of the MVC workflow is given below;



**Figure-1:** Workflow for Model View Controller (MVC)

- \* One project may have many controllers, decision of which one will run first is taken by Routes.

Routes are called by index.php file.

#### **Opening the Laravel project after installation:**

- \* Download, install, and open the XAMPP software app.
- \* Run Apache and MySQL server, open any one of the browsers.
- \* Type “localhost” in the URL and press ENTER.
- \* If the folders are not available, then go into the htdocs folder and rename the index file there.  
Otherwise, click the Laravel project folder, then click to the public folder and Laravel interface will appear there.

#### **Overview of some important features of Laravel structure:**

- \* Select one of the code editors such as VS Code and then open the Laravel project folder.
- \* app folder contains 4 subfolders, 1 php file and they are Console, Exception, Http, Providers, User.php.
  - Console/Kernel.php = We computer has 2 different kinds of port, serial port (keyboard input) and parallel port (USB input). You can do programming by parallel port, so Kernel.php contains some configurational things for this USB programming. For short, if some one works with console, then Console folder is necessary.
  - Exception/Header.php = Handles exception error.
  - **Http** folder contains 2 subfolders, 1 php file and they are **Controllers**, Middleware, Kernel.php.  
Controllers/Auth/php files (classes) = using for security purpose.  
Controllers/Controller.php = it's a mother controller, in this there is a class use traits, **Trait** is one kind of php class including properties and methods.  
Middleware/php files (classes) = using for web protection.
  - Providers/php files (classes) = these are creative files for extra advantages.
  - User.php = containing Model files (classes).
- \* bootstrap folder is responsible for handling running load of the Laravel project.
- \* config folder contains some configurational php files, of them app.php is important one, in app.php file 2 things are important such as providers (array) and aliases (array).
- \* **database** folder contains 3 subfolders, 1 gitignore file and they are factories, **migrations**, seeds,  
gitignore = using for convenience way inserting, deleting, updating in the database.
- \* **public** folder contains assets like **css**, **js** and **index.php** file (default executable file).
- \* **resource/views/view php files** = for end-user view.

- \* **routes/web.php** = manages http requests that comes through the browser URL.
- \* **storage/logs/gitignore** = responsible for line by line debugging.
- \* **tests** folder is responsible for unit test, you don't need to require any of the classes for using.
- \* **vendor**/library file's folders or package file's folders = in time of library or package installation, library or package folders will appear in the vendor folder.
- \* **.env** file is a very important file = responsible for database connection, live broadcasting etc.
- \* **composer.json** file is a very important file, in this file **require**, **require-dev** and **autoload** are important.  
If someone install new libraries, then it will be added in the **require** index automatically.
- \* **phpunit.xml** file = handles all the require of classes automatically.

### Starting practical Laravel:

- \* Open anyone of the code editors, open the Laravel project folder and open **Routes/web.php** file
- \* Here are some codes to analyze

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

In this code,

**Route** = object of a class

**get(first-argument, second argument)** = have to set the URL in the first argument and for this URL, tell what should do in the second argument. In the second argument, you can use closer function or for MVC pattern you must use Controller

**::** = scope sign = calls class's method

**get()** = method = handles URL request

**'/'** = **https://localhost/projectName/public** = Main domain or Base path

**function(){} =** Closer function or Anonymous function

**view()** = built-in global method works without object = loads one of the views

**'welcome'** = view filename or php filename (including .blade) = this file must be present in the **resource/views** folder, and **view()** method fetches this file by the help of **config/view.php** file

- **Route::get('/', function(){  
    return "Ariyan Khan";**  
**});**

In this code, **"Ariyan Khan"** is a string

- \* All the view/view file (**welcome.blade.php**) will be present in the **resource/views** folder. Here, **.blade** is a template engine and most powerful one, comes from another old php framework **'Symphony'**.

\* Laravel framework is a combine of two php framework, these two are 'Symphony' and 'Codegeyar'.

\* Now, open the CMD and change the directory to Laravel project folder.

\* Write command `php artisan` and press Enter key for the Laravel artisan list

\* Command for making the Controller/Controller class = `php artisan make:controller className`

\* If anyone works through Controller, the code in the routes/web.php looks like;

- **Route::get('/about', 'AboutController@about');**

In this kind of Route, you have to provide link like;

`<a href="<?php echo url('/about'); ?>">About</a>` or `<a href="<?php echo URL::to('/about'); ?>">About</a>`

less secure, view page can be hacked by hackers

Or.....

**Route::post('/about', 'AboutController@about');**

Here,

**post()** = method = handles form request

**AboutController** = Class name of the Controller

**@about** = method of the AboutController class

Or.....

**Route::get('/about', [**

**'uses' => 'AboutController@about';**

**'as' => '/mabout';**

**]);**

Here,

Second argument of the **get()** method is **array**, in which **'uses'** and **'as'** are the indexes of the array, where value of the **'as'** index is hide able link path that comes from the clicked link and first argument of the **get()** method is the URL.

This kind of Route is more secure and beneficial, hackers may get confused while try to hack the view pages. Use of this kind of route is recommended by most of the software companies.

In this kind of Route, you have to provide link like;

`<a href="<?php echo route('/about'); ?>">About</a>`

**route('/about')** = route calling

\* Code of the AboutController class is in the app/Http/Controllers/AboutController.php, looks like;

- **Class AboutController extends Controller{**

**Public function about(){**

**Return view('about');**

**}**

**}**

Here, **Controller** = mother controller php class

\* Some of the **syntaxes** of the **.blade** template engine are shown bellow;

- `<a href="<?php echo url('/about'); ?>">About</a>`, using for .php extension

Or.....

`<a href="{{ url('/about'); }}">About</a>`, using for .blade.php extension

Or.....

`<a href="{!! url('/about'); !!}">About</a>`, using for .blade.php extension

\* Now, time to download themes/templates from the site [www.w3layouts.com](http://www.w3layouts.com) for working with Laravel

**For example**, Enter to this site, go to the **Categories**, select **Ecommerce Online Shopping**, select **New Shop** template and download it.

\* Introducing three syntaxes/techniques such as **compact**, **->with**, and **array** for passing data from Controller to one of the Views. Lets examine these with respect to code;

- ```
class StudentController extends Controller{
    public function index(){
        $name = "Ariyan Khan";
        $age = "23"
        return view('home', compact('name', 'age'));
    }
}
```

Code for displaying these data in the view page,

`<h1><?php echo $name; ?></h1>`

`<h1><?php echo $age; ?></h1>`

**compact()** is recommended one

- ```
class StudentController extends Controller{
    public function index(){
        $name = "Ariyan Khan";
        $age = "23"
        return view('home')
            ->with('n', $name)
            ->with('a', $age);
    }
}
```

Code for displaying these data in the view page,

`<h1><?php echo $n; ?></h1>`

`<h1><?php echo $a; ?></h1>`

- ```
class StudentController extends Controller{
    public function index(){
        $name = "Ariyan Khan";
        $age = "23"
        return view('home', [
            'name' => $name,
            'age' => $age
        ]);
    }
}
```

```
}
```

Code for displaying these data in the view page,

```
<h1><?php echo $name; ?></h1>
```

```
<h1><?php echo $age; ?></h1>
```

Or.....

```
<h1>{{ $name }}</h1>
```

```
<h1>{{ $age }}</h1>
```

Or.....

```
<h1>{!! $name !!}</h1>
```

```
<h1>{!! $age !!}</h1>
```

**Array/[]** is recommended one

- \* Download **Composer.exe** from <https://getcomposer.org> and do install it including php.exe selection from the **xampp/htdocs/php/php.exe** while clicking the brows in installation.

**What's the Composer?** = Composer is an **application-level package manager** for the PHP

programming language that provides standard format for **managing dependencies of PHP software and required libraries.**

It was developed by **Nils Adermann** and **Jhordi Boggiano**.

#### Let's covering Laravel by doing project:

- \* Let the project name as NewShop

- \* Make the controller class by the command `php artisan make:controller NewShopController`.

Open **app/Http/Controllers/NewShopController.php**.

- \* Open **routes/web.php** and define the base path Route by code;

- ```
Route::get('/', [  
    'uses' => 'NewShopController@index',  
    'as' => '/'  
]);
```

- \* Open **app/Http/Controllers/NewShopController.php** and write code including index method;

- ```
class NewShopController extends Controller{  
    public function index(){  
        return view('front-end.home.home');  
    }  
}
```

Here,

**front-end.home.home** = front-end is a **folder** and first home is a **sub-folder**, last home is a **home.php** file located in the **resource/view/**.

\* Now go to the new\_shop theme/template and open index.html file through code editor, make it copy, And now past into the **resource/view/front-end/home/home.php** file.

\* Go to one of the browsers and run this **localhost/basicLaravel/public** base path URL and you will see the home page interface of the project without css, js, and images.

\* Go to the new\_shop theme/template and make copy of the **folders such as css, js, front, and images**. Now paste them into the **public/front-end** folder (after making the front-end folder).

\* Now, go to the **resource/view/front-end/home/home.php** file and make correct the css, js and Images links by doing select and replace;

- Select **css/**, go to the edit option of the code editor, select replace and write **{{ asset('/') }}****/front-end /css**, and then mark **replace all**.
- Select **js/**, go to the edit option of the code editor, select replace and write **{{ asset('/') }}****/front-end /js**, and then mark **replace all**.
- Select **images/**, go to the edit option of the code editor, select replace and write **{{ asset('/') }}****/front-end /images**, and then mark **replace all**.

Here,

**asset('/')** = is a function inside of blade syntax, responsible for covering the path upto index.php file of the Laravel framework projects.

\* Let's covering the **template-mastering**;

**What's the Template-mastering?** = In each page of the project; header, big-header, footer, big-footer parts remain same. So, we make a different master page for keeping these parts. Thereafter, when we start to work with individual page, we just yield the body of this by **@yield('body');**.

Moreover, we go to the individual page and include the master page by **@extends('front-end.master');**, and then past the body part code of the individual page (for example, category-product.php). By doing **template-mastering** we can make the individual **page dynamic**.

**For the convenience, see the codes given bellow;**

- For master page,  
**<?php**  
**Header**  
**Big-header**  
  
**@yield('body');**  
  
**Big-footer**  
**Footer**



Here, **@yield()** = directive of the blade template engine

- For the individual page,

```
<?php
    @extends('front-end.master')
    @section('body')
        Body part code of the individual page
        .....
        .....
        .....
    @endsection
```

Here, **@extends()**, **@section()**, **@endsection** = blade syntax = directive

**Note:** while clicking the different individual pages, the code inside of **@section('body')** ...  
**@endsection**

of individual pages will automatically yield to the master page's **@yield('body')**. That's why,  
template-mastering is dynamic for the individual pages.

\* **While clicking** any of the links of the individual page, go to this page, find the link, write the blade syntax's code **{{ url('/category-product' ) }}** in the href="" portion. Thereafter, go to routes/web.php file and **define Route** for this link by writing code as follows;

- **Route::get('/category- product', [**  
    **'uses' => 'NewShopController@categoryProduct',**  
    **'as' => 'category-product'**  
**]);**

Thereafter, go to app/Http/Controllers/NewShopController.php file and create a method for returning view of this page by writing code as follows;

- **class NewShopController extends Controller{**  
    **public function categoryProduct(){**  
        **return view('front-end.category.category-product');**  
    **}**  
**}**

**Let's discuss in detail about the Laravel built-in authentication system:**

\* You can make authentication system in Laravel in 3 ways;

**What are the advantages you have for managing login-logout? =**

1. by using row/basic PHP

2. by using Laravel built-in authentication system
3. by using third party packages such as SENTINNEL CARTALYST

\* Our concern is about Laravel authentication system only, now go to the project folder, open CMD, and write this command `php artisan make:auth`. Thereafter, there will be some changes in 3 locations and these locations are **app/Http/Controllers**, **resource/views**, and **routes/web.php**

\* Now open anyone of the browsers, write **localhost/basicLaravel/public/login** or **localhost/basicLaravel/public/register** in the URL, and then you will find login or register interface there.

**But the question is, is the '/login' URL's route defined in the routes/web.php file?**

**No**, route isn't defined directly, but there is special route **Auth::routes()**, location of this special route is in the **vendor/laravel/framework/src/Routing/Router.php** file, code written such as follows;

- **class Router implements RegistrarContract, BindingRegistrar{**  
     **public function auth(){**  
         **\$this->get('login', 'Auth/LoginController@showLoginForm')->name('login');**  
     **}**  
**}**

Now, **where is the location @showLoginForm function?** = Go to the **app/Http/Auth/**

**LoginController.php** file, thereafter you will find a trait as **use AuthenticatesUsers;**,

do **ctrl+click** to this trait, and then you will find **@showLoginForm**, see code as follows;

- **public function showLoginForm(){**  
     **return view('auth.login');**  
**}**

Here, **auth.login = resource/views/auth/login.blade.php**

You can customize your login or register in **resource/views/auth/login.blade.php** file

\* Go to the site <https://startbootstrap.com>, select **Themes**, then select **Admin & Dashboard**, download theme/template such as **SB Admin 2**

\* Link Laravel project to the Database;

1. Open any of the browsers, type **localhost/phpMyAdmin** in the URL, select Database, **create Database** by entering Database name such **new\_shop**.
2. go to **.env** file in the Laravel project, customize the **DB\_CONNECTION** portion by changing like as follows;

**DB\_DATABASE = new\_shop**

**DB\_USERNAME = root**

**DB\_PASSWORD =**

3. Go to the project folder, open CMD there, for example write `php artisan make:migration create_categories_table --create=categories` command according to **Query builder library/class** for creating **migration schema** in the **database/migrations** folder. Or write `php artisan make:model Category -m` command according to **Eloquent ORM library/class** for creating **migration schema** in the **database/migrations** folder as well as **Model** in the **/app** folder, by the way; for each of the tables there will be one migration file as well.

**Note:** when you are in the Laravel framework, for dealing with database business like create, delete, update etc., there are two libraries/class by default, and they are;

**Query builder** = don't follow the MVC pattern, works with Controller without Model, it seems itself as a Model handler while Controller doing request to it and it's a very powerful library/class. This library is useful for big job.

**Eloquent ORM** = here, **ORM** defines **Object Relation Methodology** = This library/class follows MVC pattern, works with Controller as well as Model, while creating tables of the database, for each table there will be a Model class and the properties of this Model class will be columns/attributes of this table. C# and Java developers follow this library/class and **recommended** one.

**Note:** while we creating Laravel authentication system by the command `php artisan make:auth`, there will be two migration files in the **database/migrations** folder by default, one is for **user\_table** and another is for **create\_password\_table**.

4. Open of the migration schema files, by the way; every **table** has it migration file and every **migration file** contains different **table's class** and these classes always contain **two functions** namely **up()** and **down()**.

**up()** = is responsible for creating the tables

**down()** = is responsible for destroying the tables

**Note:** In the migration file there are **three dots on top of the page**, **ctrl+click** to this dots, there are used **three classes** namely **Schema**, **Blueprint**, and **Migration**.

Now let's take a glance over the following code;

```
class CreateUserTable extends Migration{
    public function up(){
        Schema::create('users', function (Blueprint $table) {
```

```

        $table->increments('id');
        $table->string('name', 100);
        $table->string('email')->unique();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
}

```

Here,

**Schema::** is the object of Schema class

**\$table** is the object of Blueprint class

**'users'** is the table name, by the way; for each **table** such **'users'** there will be **Model class file** such as **User.php** with same name as table in the **app/** folder. The **name** of the **table** will be **plural** and the **name** of the **Model** will be **singular**.

**'id', 'name', 'email', 'password'** the **columns/attributes** of the **'users'** table.

**100** = defines the length of the column/attribute

**unique()** = a method = defines, you must use unique email each time you login

**increments(), strings()** etc. defines the types of the columns/attributes

To know about the methods of the \$table object visit the site <https://laravel.com> , select Documentation, select Database, select Migrations, scroll down, see the section **Available Column Types**

**rememberToken()** = works for forget password = generates 100 length string token which can be null as well, to see this function do ctrl+click to this function.

**timestamps()** = **make** two **columns** namely **created\_at** and **updated\_at** of the table of the Database, they can be null, to see this function do ctrl+click to this function.

**Note:** All of the above **functions mentioned above** will **find** in the **Blueprint class**, click to the Blueprint class, **ctrl+click to the Blueprint class**, then you can see these functions.

5. Here, the instructions for creating table of the Database are in the migration file of that table, so to this migration file or create this table in Database server you write the command **php artisan migrate** by opening CMD in the Laravel project folder.

**Note:** After running this command, you will find **error of email's length** of Laravel authentication system's tables specially, these errors occur **when you use unique()** method for

the email column/attribute, to solve these error go to each of the migration files and decrease the values of the lengths of the emails up to 150 (recommended).

\* Let's examine how POST data work in Laravel while submitting over FORM;

While clicking login button, action attribute of the form where **action="{{ url('/login') }}"** route is defined comes in action. Now go to the **routes/web.php** you will not find this route directly, but **this route is handled** by the special route **Auth::routes()**.

\* POST data are received by the class Request in a JSON (Javascript Object Notation) form.

To see these data, write code as follows;

- ```
public function login(Request $request){  
    return $request->all();  
}
```

To see this POST data **in good form** along with JSON, add **JSON Viewer** to the chrome as a plug-in.

\* Normally in **row php**, POST data are received **as array** form, but you can **make this** array to **object** by writing code as follows;

- ```
<?php  
    echo '<pre>';  
    $data = $_POST;  
    echo $data['name'];  
  
    $x = (object)$data;  
    echo $x->name;  
?>
```

\* Hackers can hack in three ways;

- Through URL = this is **handled by Laravel** in routes/web.php file by defining routes
- Through Form = this is also **handled by Laravel** by using **{{ csrf\_field() }}** in the Form. **csrf**(Cross Site Request Forgery) = generates **Token** along with POST data, in account of this Token hackers can't make hack **through Form** by driving their script.  
**Note:** To know details about **csrf**, visit the site <https://laravel.com>

- Through Server

\* How **'localhost/basicLaravel/public/home'** url page comes after successful login? Describe the flow.

= when run **'localhost/basicLaravel/public/login'** url, then **'/login'** route is handled by special route

**Auth::routes()**, click+ctrl to this route, you will code as follows:

- **public function auth(){**  
     **\$this->get('login', 'Auth\LoginController@showLoginForm')->name('login');**  
     **\$this->post('login', 'Auth\LoginController@login');**  
     **\$this->post('logout', 'Auth\LoginController@logout')->name('logout');**  
   **}**

Now go to **app/Http/Controllers/Auth/LoginController.php** file, here there is no **showLoginForm** method directly in this Controller class, you will a trait like **use AuthenticatesUsers;**, ctrl+click to this trait, then you will find the **showLoginForm** method along with view return such as **view('auth.login')**.

Now go to **resource/views/auth/login.blade.php** file, the login page will appear, thereafter entering email and password, and then while entering submit button **action="{{ url('/login') }}"** attribute of the login page Form comes in action, this route is also handled by special **Auth::routes()**, location of this special route is in the **vendor/laravel/framework/src/Routing/Router.php** file, ctrl+click to this special route, you will find code as follows;

- **public function auth(){**  
     **\$this->get('login', 'Auth\LoginController@showLoginForm')->name('login');**  
     **\$this->post('login', 'Auth\LoginController@login');**  
     **\$this->post('logout', 'Auth\LoginController@logout')->name('logout');**  
   **}**

Now go to **app/Http/Controllers/Auth/LoginController.php** file, here there is no **showLoginForm** method directly in this Controller class, you will see a trait like **use AuthenticatesUsers;**, ctrl+click to this trait, then you will find **@login method**, the code as follows;

- **public function login(Request \$request){**  
     **\$this->validateLogin(\$request);**  
       
     **if(\$this->hasTooManyLoginAttempts(\$request)){**  
         **\$this->fireLockoutEvent(\$request);**  
           
         **Return \$this->sendLockoutResponse(\$request);**  
     **}**

```

    }

    if($this->attemptLogin($request)){
        Return $this->sendLoginResponse($request);
    }

    $this->incrementLoginAttempts($request);

    return $this->sendFailedLoginResponse($request);
}

```

Now ctrl+click on `sendLoginResponse()`, then you will code as follows;

- **protected function sendLoginResponse(Request \$request){**  
`$request->session()->regenerate();`  
`$this->clearLoginAttempts($request);`  
  
`return $this->authenticated($request, $this->guard()->user())`  
`?: redirect()->intended($this->redirectPath());`  
**}**

Now ctrl+click on `redirectPath()`, then you will find code as follows;

- **public function redirectPath(){**  
`if(method_exists($this, 'redirectTo')){`  
`return $this->redirectTo();`  
**}**  
  
`return property_exists($this, 'redirectTo') ? $this->redirectTo : '/home';`  
**}**

This is how the dashboard appear after successful login.

- \* **Now the problem is that, when you do logout from the dashboard, it should redirect to login page, but it is redirecting to the base path or 'localhost/basicLaravel/public', to solve this problem you have to find out the location of the logout page? =**

Now go to **resource/views/home.blade.php** file, here you will not find logout page directly, but in this file another file namely **app.blade.php** is included by **@extends('layouts.app')**

Go to **resource/views/layouts/app.blade.php** file, here is the **logout page** as well.

Now in this **app.blade.php** file let's examine the following code portion;

- **<li>**  
`<a href="{{ url('logout') }}"`  
`onclick="event.preventDefault()";`

```

                                Document.getElementById('logout-form').submit();">
                                Logout
                                </a>

                                <form id="logout-form" action="{{ route('logout') }}" method="POST">
                                    {{ csrf_field() }}
                                </form>
</li>

```

In this code, there are two attributes of the html tag `<a></a>` namely `href=""` and `onclick=""`

Normally, while clicking link along with html tag `<a></a>`, then data pass through URL, for this hackers can hack by writing `?logout=true` in the URL portion.

To overcome this problem, in this code, the powerful event attribute `onclick=""`, prevent the URL operation and **make it like passing form data by fetching id in a javascript code**, and `{{ csrf_field() }}` inside the form will not work before the submit button click.

Now when do click to the Logout the `action="{{ url('logout') }}"` will come in action. This route is also handled by the special route `Auth::routes()`, ctrl+click to this special route, you will find code as follows;

- ```
public function auth(){
    $this->get('login', 'Auth\LoginController@showLoginForm')->name('login');
    $this->post('login', 'Auth\LoginController@login');
    $this->post('logout', 'Auth\LoginController@logout')->name('logout');
}
```

Now go to `app/Http/Controllers/Auth/LoginController.php` file, here there is no logout method directly in this Controller class, you will see a trait like `use AuthenticatesUsers;`, ctrl+click to this trait, then you will find **@logout method**, at the bottom of the file ,the code as follows;

- ```
public function logout(Request $request){
    $this->guard()->logout();

    $request->session()->invalidate();

    return redirect('/');
}
```



From this code you change the redirect to login page such as `return redirect('/login')`, by doing you have solved the problem that was like, **when you do logout from the dashboard, it should redirect to login page, but it is redirecting to the base path or 'localhost/basicLaravel/public'.**

Let's get an overview with CRUD(Create, Read, Update, and Delete) functionalities:

Data insert/save:

\* Normally, in **row php** while data passing through **Form**, we accessed data by **POST** in the form of array. But now in the **Laravel** Framework we access Form data by **Request class** object which is `$request` in the form of JSON.

\* **Is Request a class, then what's its properties and methods? =**

There are no predefined properties of the **Request class**. That means the properties of the Request class are dynamic, while submitting the Form data, then each **input field of the Form being the property of The Request class**.

Methods of the Request class for example `all()`

\* If necessary, the command for creating Model for the Database table is like that `php artisan make:model modelName`

\* Remember, the **Request class works with Form** and **Model class works** with any of the **tables** of the Database **such as Category(singular) Model of the categories(plural) table of the Database**.

Name of the input field of the Form and the name of the columns of the table will be same.

Now go to the Model class along with the path `app/Category.php` and write code as follows:

- ```
class Category extends Model{
    protected $fillable=['category_name', 'category_description', 'publication_status'];
}
```

In this above code,

```
$fillable=['category_name', 'category_description', 'publication_status'];
```

If you add this property to the Model class as indexes of an array without values, you will find more advantages in further.

Now go the `app/Http/Controllers/CategoryController.php` and write code as follows;

- ```
class CategoryController extends Controller{
    public function saveCategory(Request $request){
        $category=new Category();
        $category->category_name=$request->category_name;
```

```

        $category->category_description=$request->category_description;
        $category->publication_status=$request-> publication_status;
        $category->save();
        return redirect('/category/add')
            ->with('message', 'Category info saved
                succesfully');
    }
}

```

In this above code,

This is an approach to save the Form data in the Database table. Recommended for **Eloquent**.

Here, **\$category=new Category()** is an object of the **Eloquent** class.

While you creating object of the Category class, the library reference of this class such as **use App\Category;** will appear automatically on the top of this page.

Then assign properties of the Request class to the \$fillable property of the Category Model class.

Thereafter, Model will save these data to the Database as well.

To show the message on the add page, write blade syntax with markup as **<h4 class="text-center text-success"> {{ Session::get('message') }} </h4>**

You have to use reference library such as **use Session;** on top of this page.

Another approach to save data in table of the Database;

**Category::create(\$request->all());** recommended for **Eloquent**

Here, **::create()** is an object of the **Eloquent** class.

Another approach to save data in table of the Database;

```

DB::table('categories')->insert([
    'category_name' => =$request->category_name,
    'category_description' => =$request->category_description,
    'publication_status' => publication_status
]);

```

This approach is recommended for **Query builder**. In this approach you have to use the library reference such as **use DB;** on the top of this page.

Here, **::table()** is an object of the **Query Builder** class.

**Note:** while you are in Laravel framework, for working with database, they have been made two built-in library 1) Query builder(a class), 2) Eloquent ORM(a class)

- 1) Query builder-----> it's a powerfull class/library, works without Model, Controller will direct ask to it for data instead Model, using for big work, PHP uses it
- 2) Eloquent ORM(Object Relationship Methodology)----> it follows the rules of the MVC, it says for each table of the database, a Model class is a must needy for that table.  
where columns of the table(table name plural) will be the properties of the Model(model name singular) class, JAVA C# use it

By running the following command can create the model class as well as migration schema class;

```
php artisan make:model model_name -m
```

\* Some **boiler code** for CRUD(Create, Read, Update, and Delete) functionalities;

For example, Add Category page:

- ```
<div class="row">
  <div class="col-md-10 col-md-offset-1">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h4 class="text-center text-success">Add Category Form</h4>
      </div>
      <div class="panel-body">
        <form action="{{ url('add-category') }}" method="POST" class="form-horizontal">
          <div class="form-group">
            <label class="control-label col-md-4">Category Name</label>
            <div class="col-md-8">
              <input type="text" name="category_name" class="form-control"/>
            </div>
          </div>
          <div class="form-group">
            <label class="control-label col-md-4">Category description</label>
            <div class="col-md-8">
              <textarea class="form-cotrol" name="category_description"></textarea>
            </div>
          </div>
          <div class="form-group">
            <label class="control-label col-md-4">Publication Status</label>
            <div class="col-md-8 radio">
              <label><input type="radio" checked name="publication_status"
                value="1"/>Published</label>
              <label><input type="radio" name="publication_status"
                value="0"/>Unpublished</label>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
```

```

        </div>
    </div>
    <div class="form-control">
        <div class="col-md-8 col-md-offset-4">
            <input type="submit" name="btn" class="btn btn-success btn-block"
                value="Save Category"/>
        </div>
    </div>
</form>
</div>
</div>
</div>
</div>

```

For example, Manage Category page:

- <br/>
 <div class="row">
 <div class="col-md-12 col-md-offset-1">
 <div class="panel panel-default">
 <div class="panel-body">
 <h3 class="text-center text-success" id="Ariyan Khan">{{ Session::get('message') }}</h3>
 <table table-bordered>
 <tr class="bg-primary">
 <th>SI No.</th>
 <th>Category Name</th>
 <th>Category Description</th>
 <th>Publication Status</th>
 <th>Action</th>
 </tr>
 @php(\$i=1)
 @foreach(\$categories as \$category)
 <tr>
 <td>{{ \$i++ }}</td>
 <td>{{ \$category->category\_name }}</td>
 <td>{{ \$category->category\_description }}</td>
 <td>{{ \$category->publication\_status == 1 ? 'Published' : 'Unpublished' }}</td>
 <td>
 @if(\$category->publication\_status == 1)
 <a href="{{ url('unpublished-category', ['id' => \$category->id]) }}" class="btn btn-info btn-xs">
 <span class="glyphicon glyphicon-arrow-up"></span>
 </a>
 </td>
 </tr>
 @endforeach
 </table>
 </div>
 </div>
 </div>
 </div>

```

        @else
        <a href="{{ url('published-category', ['id' => $category->id]) }}" class="btn btn-info btn-xs">
            <span class="glyphicon glyphicon-arrow-down"></span>
        </a>
        @endif
        <a href="{{ url('edit-category', ['id' => $category_id]) }}" class="btn btn-warning btn-xs">
            <span class="glyphicon glyphicon-edit"></span>
        </a>
        <a href="{{ url('delete-category', ['id' => $category_id]) }}" class="btn btn-danger btn-xs">
            <span class="glyphicon glyphicon-trash"></span>
        </a>
    </td>
</tr>
@endforeach
</table>
</div>
</div>
</div>
</div>

<script>
    $(document).ready(function (){
        $('#xyz').click(function (){
            $(this).text(' ');
        });
    });
</script>

```

Here, inside of the script tag is the JQuery code for text deletion for the id = 'xyz' with respect to the manage Category page.

For example, Edit Category page:

- ```

<div class="row">
    <div class="col-md-10 col-md-offset-1">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h4 class="text-center text-success">Update Category Form</h4>
            </div>
            <div class="panel-body">
                <h3 class="text-center text-success">{{ Session::get('message') }}</h3>
            </div>
        </div>
    </div>
</div>

```

```

<form action="{{ route('update-category') }}" method="POST" class="form-
    horizontal">
    {{ csrf_field() }}
    <div class="form-group">
        <label class="control-label col-md-4">Category Name</label>
        <div class="col-md-8">
            <input type="text" name="category_name" class="form-control" value="{{
                $category->name }}" />

            <input type="hidden" name="category_id" class="form-control" value="{{
                $category->id }}" />
        </div>
    </div>
    <div class="form-group">
        <label class="control-label col-md-4">Category description</label>
        <div class="col-md-8">
            <textarea class="form-control" name="category_description">{{ $category-
                >description }}</textarea>
        </div>
    </div>
    <div class="form-group">
        <label class="control-label col-md-4">Publication Status</label>
        <div class="col-md-8 radio">
            <label><input type="radio" {{ $category->publication_status == 1 ? 'checked' :
                " " }} name="publication_status" value="1" />Published</label>
            <label><input type="radio" {{ $category->publication_status == 0 ? 'checked' :
                " " }} name="publication_status" value="0" />Unpublished</label>
        </div>
    </div>
    <div class="form-control">
        <div class="col-md-8 col-md-offset-4">
            <input type="submit" name="btn" class="btn btn-success btn-block"
                value="Update Category Info" />
        </div>
    </div>
</form>
</div>
</div>
</div>
</div>

```

All of the Routes for the Category:

- Route::post('/category/add-category', [

- ```

        'uses' => 'CategoryController@addCategoryInfo',
        'as' => 'add-category'
    ));

    • Route::get('/category/unpublished-category/{id}', [
        'uses' => 'CategoryController@unpublishedCategoryInfo',
        'as' => 'unpublished-category'
    ]);

    • Route::get('/category/published-category/{id}', [
        'uses' => 'CategoryController@publishedCategoryInfo',
        'as' => 'published-category'
    ]);

    • Route::get('/category/edit-category/{id}', [
        'uses' => 'CategoryController@editCategoryInfo',
        'as' => 'edit-category'
    ]);

    • Route::post('/category/update-category', [
        'uses' => 'CategoryController@updateCategoryInfo',
        'as' => 'update-category'
    ]);

    • Route::get('/category/delete-category/{id}', [
        'uses' => 'CategoryController@deleteCategoryInfo',
        'as' => 'delete-category'
    ]);

```

All of the methods of the CategoryController class for Category:

- Public function addCategoryInfo(Request \$request){  
     \$category = new Category();  
  
     \$category->category\_name = \$request->category\_name;  
     \$category->category\_description = \$request->category\_description;  
     \$category->publication\_status = \$request->publication\_status;  
  
     \$category->save();  
   }
- public function unpublishedCategoryInfo(\$id){  
     \$category = Category::find(\$id);  
     \$category->publication\_status = 0;

- ```

$category_save();

return redirect('/category/manage')
    ->with('message', 'Category info unpublished');
}

```
- ```

public function publishedCategoryInfo($id){
    $category = Category::find($id);
    $category->publication_status = 1;
    $category_save();

    return redirect('/category/manage')
        ->with('message', 'Category info published');
}

```
  - ```

public function editCategoryInfo($id){
    $category = Category::find($id);

    return view('admin.category.edit-category', ['category' => $category]);
}

```
  - ```

public function updateCategoryInfo(Request $request){
    $category = Category::find($request->categories);

    $category->name = $request->category_name;
    $category->description = $request->category_description;
    $category->publication_status = $request->publication_status;

    $category->save();

    return redirect('/category/manage')
        ->with('message', 'Category info updated successfully');
}

```
  - ```

public function deleteCategoryInfo($id){
    $category = Category::find($id);
    $category->delete();

    return redirect('/category/manage')
        ->with('message', 'Category info deleted successfully');
}

```

\* Laravel package installation



**What is package?** = collection of classes to perform the specific task, for example, laravelcollective

Process:

1. Visit the site namely <https://laravelcollective.com> and then copy the command `composer require "laravelcollective/html": "^5.4.0"`, run the command through the CMD. After doing this the package(laravelcollective) will be installed automatically.
2. Then check **composer.json file** of the laravel, you will find change in the **require object array**. and check the **vendor folder** of the laravel, you will see one **new folder will be added** by naming laravelcollective.

**Note:** **require object array** will get change when you install new packages from outside, but **require-dev object array** will get change when internal package of the laravel will get change.

3. Then go to the site again copy the command `Collective\Html\HtmlServiceProvider::class` from the provider part of the page and **add to the provider array** of config/app.php
4. Then go to the site again copy the commands  
`'Form' => Collective\Html\FormFacade::class,`  
`'Html' => Collective\Html\HtmlFacade::class`  
from the aliases part of the page and **add to the aliases array** of config/app.php

**Note:** every package has its own at least one provider and aliases

Now lets talk about the benefits of using laravelcollective package of the laravel framework;

Take a glance to the following boiler code of the panel body of the Form;

- ```
<h3 class="text-center text-success">{{ Session::get('message') }}</h3>
{{ Form::open(['route' => 'new-brand', 'method' => 'POST', class=> 'form-horizontal'])
}}\
    <div class="form-group">
        {{ Form::label('brand_name', 'Brand Name', ['class' => 'col-md-3']) }}
        <div class="col-md-9">
            {{ Form::text('brand_name', "", ['class' => 'form-control']) }}
        </div>
    </div>
{{ Form::close() }}
```

In the above code, no csrf field is needed for this Form, **if the code is written like this by the help of laravelcollective package, the Form will be more secure and threat free**

\* Laravel Form or field validation description:

Laravel base controller class uses **ValidateRequests treat(kind of class)**, this treat contains **various method** for incoming input data **validation**.

**Note:** public method of any class can be used by another extended class using **'\$this'** keyword.

For better understanding take a glance to the following codes of validation:

- Class BrandController extends Controller{  
     Public function saveBrand(Request \$request){  
         \$this->validate(\$request, [  
             'brand\_name'=>'required|alpha|max:15|min:5'  
         ]);  
     }  
 }

Here, in the above code;

**\$this->validate()** is a method of **ValidateRequest** treat containing two parameters. Here, we are validating the brand name, **'required|alpha|max:15|min:5'** is the validation rules. To know the various **available validation rules** have to visit the <https://laravel.com> site.

- Class BrandController extends Controller{  
     Public function saveBrand(Request \$request){  
         \$this->validate(\$request, [  
             'brand\_name'=>'required|**regex:/^[pL\s\~]+\$/u**|max:15|min:5',  
             'brand\_description' => 'required',  
             'publication\_status' => 'required'  
         ]);  
     }  
 }

Here, in the above code;

**Red color part** separated by the pip symbol is the **regular expression** part for name validation. To know the different regular expression, like for email, password etc. **Search in the google by typing 'regular expression for email/password validation laravel'**.

**ValidateRequest** treat or class provides an **object namely 'errors'** which is responsible for showing errors while one trying to input **invalid data**.

The demonstration code as follows:

- `<input type="text" name="brand_name" class="form-control"/>`  
     `<span>{{ $errors->has('brand_name') ? $errors->first('brand_name') : ' ' }}</span>`

Here, in the above code;

**has()** method is responsible for **catching the errors** and **first()** method is responsible for **printing something with respect to these errors**.

**\* Process for saving the Brand of ecommerce:**

1. Write the command `php artisan make:model Brand -m` on the CMD, this will create a model along with migration file for the database table
2. Go to the created model file namely **Brand.php** and write the following code;
  - `class Brand extends Model{`  
     `protected $fillable=['brand_name', 'brand_description', 'publication_status'];`

```
}
```

3. go the migration file of this model and write the following code;

- **public function up(){**  
    **Schema::create('brands', function(Blueprint \$table){**  
        **\$table->increments('id');**  
        **\$table->string('brand\_name');**  
        **\$table->text('brand\_description');**  
        **\$table->tinyInteger('publication\_status');**  
        **\$table->timestamps();**  
    **});**  
**}**

4. Go the CMD and write the command **php artisan migrate**, then check the database whether the required table has been completed or not

5. Go to the web.php file and write the following code;

- **Route::post('/brand/save', [**  
    **'uses' => 'BrandController@saveBrand',**  
    **'as' => 'new-brand'**  
**]);**

6. Go to the CMD and write the command **php artisan make:controller BrandController** for creating the controller for Brand of ecommerce

7. Go the BrandController.php file and write the following code;

- **public function saveBrand(Request \$request){**  
    **\$this->validate(\$request, [**  
        **'brand\_name'=>'required|regex:/^[pL\s\-\-]+\$/u|max:20|min:2',**  
        **'brand\_description' => 'required',**  
        **'publication\_status' => 'required'**  
    **]);**  
  
    **\$brand=new Brand();**  
    **\$brand->brand\_name = \$request->brand\_name;**  
    **\$brand->brand\_description = \$request->brand\_description;**  
    **\$brand->publication\_status = \$request->publication\_status;**  
    **\$brand->save();**  
  
    **return redirect('brand/add')->with('message', 'Brand info saved successfully');**  
**}**

**\* Process for saving the Product of ecommerce and covering the image saving to the database in case**

**of laravel:**

1. Go the add-product.blade.php in the view and write the following code;

- **<div class="row">**  
    **<div class="col-md-10 col-md-offset-1">**

```

<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="text-center text-success">{{ Session::get('message') }}</h3>
  </div>
  <div class="panel-body">
    {{ From::open(['route' => 'new-brand', 'method' => 'POST', 'class' => 'form-
      horizontal', 'enctype' => 'multipart/form-data']) }}

```

here, 'enctype'  
attribute used for the  
image data

```

<div class="form-group">
  <label class="control-label col-md-3">Category Name</label>
  <div class="col-md-9">
    <select class="form-control" name="category_id">
      <option>-----Select Category Name-----</option>
      @foreach($categories as $category)
        <option value="{{ $category->id }}">{{ $category->category_name
          }}</option>
      @endforeach
    </select>
    <span class="text-danger">{{ $errors->has('category_id') ? $errors-
      >first('category id error')}}</span>
  </div>
</div>
<div class="form-group">
  <label class="control-label col-md-3">Brand Name</label>
  <div class="col-md-9">
    <select class="form-control" name="brand_id">
      <option>-----Select Brand Name-----</option>
      @foreach($brands as $brand)
        <option value="{{ $ brand ->id }}">{{ $ brand -> brand_name
          }}</option>
      @endforeach
    </select>
    <span class="text-danger">{{ $errors->has('brand_id') ? $errors-
      >first('bran id error')}}</span>
  </div>
</div>
<div class="form-group">
  <label class="control-label col-md-4">Product Name</label>
  <div class="col-md-9">
    <input type="text" class="form-cotrol" name="product_name"/>
    <span class="text-danger">{{ $errors->has('product_name') ? $errors-

```

```

        >first('product name error'))}</span>
    </div>
</div>
<div class="form-group">
    <label class="control-label col-md-4">Product Price</label>
    <div class="col-md-9">
        <input type="number" class="form-control" name="product_price"/>
        <span class="text-danger">{{ $errors->has('product_price') ? $errors-
            >first('product price error')}}</span>
    </div>
</div>
<div class="form-group">
    <label class="control-label col-md-4">Product Quantity</label>
    <div class="col-md-9">
        <input type="number" class="form-control"
            name="product_quantity"/>
        <span class="text-danger">{{ $errors->has('product_quantity') ? $errors-
            >first('product quantity error')}}</span>
    </div>
</div>
<div class="form-group">
    <label class="control-label col-md-4">Short description</label>
    <div class="col-md-8">
        <textarea class="form-control" name="short_description"></textarea>
        <span class="text-danger">{{ $errors->has('short_description') ? $errors-
            >first('short description error')}}</span>
    </div>
</div>
<div class="form-group">
    <label class="control-label col-md-4">Long description</label>
    <div class="col-md-8">
        <textarea class="form-control" name="long_description"></textarea>
        <span class="text-danger">{{ $errors->has('long_description') ? $errors-
            >first('long description error')}}</span>
    </div>
</div>
<div class="form-group">
    <label class="control-label col-md-4">Product Image</label>
    <div class="col-md-8">
        <input type="file" class="form-control" name="product_image"
            accept="image/*">
        <span class="text-danger">{{ $errors->has('product_image') ? $errors-
            >first('product image error')}}</span>
    </div>
</div>

```

here, '\*' means  
all image  
format type



```

//..... // here, '/' is the
comments
->get()/first();

//here, the function of 'get()' to
//fetch the multiple row data
//from the database and 'first()'
//to fetch single row data

```

```

$brands = Brand::where('publication_status', 1)->get();

return view('admin.product.add_product', [

    'categories' => $categories,

    'brands' => $brands

]);
}

```

5. Write the command `php artisan make:model Product -m` on the CMD, this will create a model along with migration file for the database table
6. Go to the created model file namely **Product.php** and write the following code;
  - ```
class Product extends Model{
    protected $fillable=[' category_id', 'brand_id', 'product_name', 'product_price',
        'product_quantity', 'short_description', 'long_description', 'product_image',
        'publication_status'];
}
```
7. go the migration file of this model and write the following code;
  - ```
public function up(){
    Schema::create('brands', function(Blueprint $table){
        $table->increments('id');
        $table->integer('category_id');
        $table->integer('brand_id');
        $table->string('product_name');
        $table->float('product_price', 10,2);
        $table->integer('product_quantity');
        $table->text('short_description');
        $table->text('long_description');
        $table->text('product_image');
        $table->tinyInteger('publication_status');
        $table->timestamps();
    });
}
```
8. Go the CMD and write the command **php artisan migrate**, then check the database whether the required table has been completed or not
9. Go to the web.php file and write the following code;

- `Route::post('/product/save', [
 'uses' => 'ProductController@saveProduct',
 'as' => 'new-product'
]);`

10. Go the ProductController.php file and write the following code;

- `protected function productInfoValidate($request){
 $this->validate($request, [
 'product_name' => 'required'
 ]);
 //Do validation for each field in the '$request' object of 'Request' class
}`
- `protected function productImageUpload($request){
 $productImage = $request->file('product_image'); //here, 'file()' is the method
 //of request class and used
 //for fetching image field
 //input data

 $imageName = $productImage->getClientOriginalName();
 //$imageName = $productImage->getClientSize(); //here,
 // 'getClientOriginalName()' and
 // 'getClientSize()' are
 // the methods for
 // creating various
 // image name for
 // image input field data

 //return $imageName;
 $directory = 'product_image/';
 $imageUrl = $directory.$imageName;
 $productImage->move($directory, $imageName);
 return $imageUrl;

 //here, the function of
 //the 'move()' method
 //is to saved the
 //uploaded image to
 //required folder. The
 //folder is situated
 //where the 'index.php'
 //file is situated of the
 //laravel

}`
- `protected function saveProductBasicInfo($request, $imageUrl){
 $product = new Product();
 $product->product_image = $imageUrl;
 $product->brand_id = $request-> brand_id;
 $product->product_name = $request->product_name;`



```

        $product->product_price = $request->product_price;
        $product->category_id = $request->category_id;
        $product->product_quantity = $request->product_quantity;
        $product->short_description = $request->short_description;
        $product->long_description = $request->long_description;
        $product->publication_status = $request->publication_status;
        $product->save();
    }
    • public function saveProduct(Request $request){
        $this->productInfoValidate($request);
        $imageUrl = $this->productImageUpload($request);
        $this->saveProductBasicInfo($request, $imageUrl);
        return redirect('product/add')->with('message', 'Product info saved successfully');
    }

```

**Note:** Every small work should be done by using different function as done above, because when the project post on the server, the server automatically load the statements of the code primarily, then when you call the function runs once without loading the code inside it, save the time dramatically.

\* **Plugin in the Laravel project:**

**Two plugin** names for example, **tinymce** and **ckeditor**(ckeditor.com, download) for using in the textarea

**Plugin setup process:**

- i. download the plugin and do unzip, copy it and paste to laravel project, path as **public/admin/** or **public/**
- ii. open the index.html file from the plugin and copy css and javascript path link, paste them to the bottom of the head of the html of the master.blade.php file, customize the path link by your project
- iii. identify the main thing of the plugin, for example **id="editor"**, copy it and paste to destination, for example textarea
- iv. finally to initialize the plugin, you must copy code as follows;

```
<script>
```

```
    initSample();
```

```
</script>
```

from the bottom of the index.html file of the plugin, and paste it to the bottom of the body of the html of the master.blade.php file

\* **another laravel package installation for image uploading to database:**

Visit <http://image.intervention.io>, will find package named intervention image

See the different API of the package, for example `resize()`

### Installation process:

1. write command composer require intervention/image on project terminal
2. check to vendor folder and composer.json's require
3. fill the provider and aliases to config/app.php
4. copy configuration from the site and paste it on project terminal, after that one file will be added to config/

### Now upload the image the package as code as follows:

- ```
protected function productImageUpload($request){  
    $productImage = $request->file('product_image');  
    $filetype = $productImage->getClientOriginalExtension();  
    $imageName = $request->product_name.'.'.$filetype;  
    $directory = 'product_image/';  
    $imageUrl = $directory.$imageName;  
    Image::make($productImage->resize(200, 200)->save($imageUrl);  
    return $imageUrl;  
}
```

### \* Manage product A to Z:

#### 1. in menu.blade.php:

- ```
<li>  
    <a href="{{ route('manage-product') }}">Manage Product</a>  
</li>
```

#### 2. in web.php:

- ```
Route::get('/product/manage', [  
    'uses' => 'ProductController@manageProduct',  
    'as' => 'manage-product'  
]);
```

#### 3. in ProductController.php:

- ```
public function manageProduct(){  
    return view('admin.product.manage-product');  
}
```

#### 4. in manage-product.blade.php:

- ```
<div class="panel-body">  
    <h3 class="text-center text-success">{{ Session::get('message') }}</h3>  
    <div class="table-responsive">  
        <table class="table table-bordered">  
            <tr class="bg-primary">  
                <td>SL NO.</td>  
                <td>Category Name</td>  
                <td>Brand Name</td>  
                <td>Product Name</td>  
                <td>Product Image</td>  
                <td>Product Price</td>
```

```

        <td>Product Quantity</td>
        <td>Publication Status</td>
        <td>Action</td>
    </tr>

    @php($i=1)
    @foreach($products as $product)
    <tr>
        <td>{{ $i++ }}</td>
        <td>{{ $product->category_name }}</td>
        <td>{{ $product->brand_name }} </td>
        <td>{{ $product->product_name }}</td>
        <td>
            </img>
        </td>
        <td>{{ $product->product_price }}</td>
        <td>{{ $product->product_quantity }}</td>
        <td>{{ $product->publication_status }}</td>
        <td>
            <a href="" class="btn btn-info btn-xs">
                <span class="glyphicon glyphicon-zoom-in"></span>
            </a>
            <a href="" class="btn btn-primary btn-xs">
                <span class="glyphicon glyphicon-arrow-up"></span>
            </a>
            <a href=" route('edit-product', ['id'=>$product->id])"
                class="btn btn-success btn-xs">
                <span class="glyphicon glyphicon-edit"></span>
            </a>
            <a href="" class="btn btn-danger btn-xs">
                <span class="glyphicon glyphicon-trash"></span>
            </a>
        </td>
    </tr>
    @endforeach
</table>
</div>
</div>

```

5. in ProductController.php:

- ```
public function manageProduct(){
    //$products = Product::all();
    //return $products;
    $products = DB::table('products')
        ->join('categories', 'products.category_id', '=', 'categories_id')
        ->join('brands', 'products.brand_id', '=', 'brands_id')
        ->join('products.*', 'categories.category_name',
            'brands.brand_name')

    return view('admin.product.manage-product', ['products' => $products]);
}
```

6. in web.php:

- ```
Route::get('/product/edit/{id}', [
    'uses' => 'ProductController@editProduct',
    'as' => 'edit-product'
]);
```

7. in ProductController.php:

- ```
public function editProduct($id){
    $product = Product::find($id);
    $categories = Category::where('publication_status', 1)->get();
    $brands = Brand::where('publication_status', 1)->get();
    return view('admin.product.edit-product', [
        'product'=>$product,
        'categories'=>$categories,
        'brands'=>$brands
    ]);
}
```

8. in edit-product.blade.php:

- ```
<div class="row">
    <div class="col-md-10 col-md-offset-1">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="text-center text-success">{{ Session::get('message') }}</h3>
```

```

</div>
<div class="panel-body">
  {{ From::open(['route' => 'update-brand', 'method' => 'POST', 'class' => 'form-
    horizontal', 'enctype' => 'multipart/form-data', 'name' => 'editProductForm'])
    }}

  <div class="form-group">
    <label class="control-label col-md-3">Category Name</label>
    <div class="col-md-9">
      <select class="form-control" name="category_id">
        <option>-----Select Category Name-----</option>
        @foreach($categories as $category)
          <option value="{{ $category->id }}">{{ $category->category_name
            }}</option>
        @endforeach
      </select>
      <span class="text-danger">{{ $errors->has('category_id') ? $errors-
        >first('category id error')}}</span>
    </div>
  </div>
  <div class="form-group">
    <label class="control-label col-md-3">Brand Name</label>
    <div class="col-md-9">
      <select class="form-control" name="brand_id">
        <option>-----Select Brand Name-----</option>
        @foreach($brands as $brand)
          <option value="{{ $ brand ->id }}">{{ $ brand -> brand _name
            }}</option>
        @endforeach
      </select>
      <span class="text-danger">{{ $errors->has('brand_id') ? $errors-
        >first('brand id error')}}</span>
    </div>
  </div>
  <div class="form-group">
    <label class="control-label col-md-4">Product Name</label>
    <div class="col-md-9">
      <input type="text" value="{{ $product-product_name }}" class="form-cotrol"
        name="product_name"/>
      <span class="text-danger">{{ $errors->has('product_name') ? $errors-
        >first('product name error')}}</span>
    </div>
  </div>
  <div class="form-group">

```

```

<label class="control-label col-md-4">Product Price</label>
<div class="col-md-9">
  <input type="number" value="{{ $product-product_price }}" class="form-
    control" name="product_price"/>
  <span class="text-danger">{{ $errors->has('product_price') ? $errors-
    >first('product price error')}}</span>
</div>
</div>
<div class="form-group">
  <label class="control-label col-md-4">Product Quantity</label>
  <div class="col-md-9">
    <input type="number" value="{{ $product-product_quantity }}" class="form-
      control" name="product_quantity"/>
    <span class="text-danger">{{ $errors->has('product_quantity') ? $errors-
      >first('product quantity error')}}</span>
  </div>
</div>
<div class="form-group">
  <label class="control-label col-md-4">Short description</label>
  <div class="col-md-8">
    <textarea class="form-cotrol" name="short_description">{{ $product-
      >short_description }}</textarea>
    <span class="text-danger">{{ $errors->has('short_description') ? $errors-
      >first('short description error')}}</span>
  </div>
</div>
<div class="form-group">
  <label class="control-label col-md-4">Long description</label>
  <div class="col-md-8">
    <textarea class="form-cotrol" name="long_description">{{ $product-
      >long_description }}</textarea>
    <span class="text-danger">{{ $errors->has('long_description') ? $errors-
      >first('long description error')}}</span>
  </div>
</div>
<div class="form-group">
  <label class="control-label col-md-4">Product Image</label>
  <div class="col-md-8">
    <input type="file" class="form-control" name="product_image"
      accept="image/*">

    <br/>

```



'as' => 'update-product'

});

10. in ProductController.php:

- protected function **productImageUpload**(\$request){  
    \$productImage = \$request->file('product\_image');  
    \$fileType = \$productImage->getClientOriginalExtension();  
    \$imageName = \$request->product\_name.'.'.\$fileType;  
    \$directory = 'product-image/';  
    \$imageUrl = \$directory.\$imageName;  
    Image::make(\$productImage)->resize(200, 200)->save(\$imageUrl);  
    return \$imageUrl;  
}  
  
protected function **productBasicInfoUpdate**(\$product, \$request, \$imgUrl=null){  
    \$product->category\_id = \$request->category\_id;  
    \$product->brand\_id = \$request->brand\_id;  
    //Including product\_name, price, quantity, short des, long des, publication\_status  
    if(\$imgUrl){  
        \$product->product\_image = \$imgUrl;  
    }  
    \$product->save();  
}  
  
public function **updateProduct**(Request \$request){  
    //return \$request->all();  
    //\$productImage = \$\_FILES['product\_image'];  
    //echo '<pre>';  
    //print\_r(\$productImage);  
  
    \$productImage = \$request->file('product\_image');



```

//echo $productImage->getClientOriginalName();
//echo $productImage->getClientOriginalExtension();
//echo $productImage->getClientOriginalMimeType();
$product = Product::find($request->product_id);

if($productImage){
    unlink($product->product_image);
    $imageUrl = $this->productImageUpload($request);
    $this->productBasicInfoUpdate($product, $request, $imageUrl);
}
else{
    $this->productBasicInfoUpdate($product, $request);
}
return redirect('/product/manage')->with('message', 'Product Info Updated');
}

```

**\* Fontend viewing of Homepage A to Z:**

1. see the frontend part of the project

2. in web.php:

- `Route::get('/', [
 'uses' => 'NewShopController@index',
 'as' => '/'
]);`

3. in NewShopController.php:

- `public function index(){
 return view('front-end.home');
}`

4. in home.blade.php:

You will see extends master in the above as follows

```
@extends('front-end.master')
```

5. in master.blade.php:

You will find the navbar menu

Firstly, create a folder inside the front-end folder named includes, now create a file named header.blade.php, paste the header part code of the master.blade.php, then include the header.blade.php to master.blade.php as **@include('front-end.includes.header')** now do the same for the footer part of the master.blade.php

6. in NewShopController.php:

- ```
public function index(){  
    $categories = Category::where('publication_status', 1)->get();  
    return view('front-end.home.home', [  
        'categories' = $categories  
    ]);  
}
```

7. in header.blade.php:

- ```
<div class="collapse navbar-collapse" id="bs-megadropdown-tabs">  
    <ul class="nav navbar-nav">  
        @foreach($categories as $category)  
            <li class="active"><a href="{{ route('/') }}" class="act">{{ $category-  
                >category_name }}</a></li>  
        @endforeach  
    </ul>  
</div>
```

8. in NewShopController.php:

- ```
public function index(){  
    $categories = Category::where('publication_status', 1)->get();
```

```

        $newProducts = Product::where('publication_status', 1)
                                ->orderby('id', 'DESC')
                                //->orderby('id' , 'ASC')
                                //->skip(2)
                                ->take(8)
                                ->get();

        return view('front-end.home.home', [
            'categories' => $categories,
            'newProducts' => $newProducts
        ]);
    }
}

```

9. in home.blade.php:

- ```

<h2 class="tittle">New Arrivals</h2>

<div class="arrivals-grids">

    @foreach($newProducts as $newProduct)

        <div class="col-md-3 arrival-grid simpleCart_shelfItem">

            <div class="grid-arr">

                <div class="grid-arrival">

                    <figure>

                        <a href="" calss="new-gri" data-toggle="modal" data-
                                target="">

                            <div class="grid-img">

                            </div>

                            <div class="grid-img">

                            </div>

                        </figure>

                    </div>

                </div>

            </div>

        </div>

    @endforeach

</div>

```

```

        </a>
    </figure>
</div>
<div class="ribben">
    <p>New</p>
</div>
<div class="ribben1">
    <p>Sale</p>
</div>
<div class="block">
    <div class="starbox small ghosting"></div>
</div>
<div>
    <h6><a href="single.html">{{$newProduct->product_price
    }}</a></h6>

    <span>XL / XXL / S</span>
    <p><del>100.00</del><em
        class="item_price">TK. {{$newProduct->product_price
        }} </em></p>
    <a href="" data-text="Add To Cart" class="my-cart-b
        item_add"></a>
</div>
</div>
</div>
<div>
    @endforeach
<div class="clearfix"></div>
</div>

```

10. in header.blade.php;

- ```
<div class="collapse navbar-collapse" id="bs-megadropdown-tabs">
    <ul class="nav navbar-nav">
        @foreach($categories as $category)
            <li class="active"><a href="{{ route('category_product', ['id' => $category->id])
                }}" class="act">{{ $category->category_name }}</a></li>
        @endforeach
    </ul>
</div>
```

11. in web.php:

- ```
Route::get('/category-product/{id}', [
    'uses' => 'NewShopController@categoryProduct',
    'as' => 'category-product'
]);
```

12. in NewShopController.php:

- ```
public function categoryProduct($id){
    $categoryProducts = Category::where('category_id', $id)
        ->where('publication_status', 1)
        ->get();

    return view('front-end.category-content', [
        'categoryProducts' => $categoryProducts
    ]);
}
```

13. in view/front-end/category/category-content.blade.php:

```
@foreach($categoryProduct as $categoryProduct)
<div class="col-sm-4 product-tab-grid simpleCart_shelfItem">
    <div class="grid-arr">
        <div class="grid-arrival">
            <figure>
                <a href="#" class="new-gri" data-toggle="modal" data-target="#m">
                    <div class="grid-img">
                        
                    </div>
                </a>
            </figure>
        </div>
    </div>
</div>
```

```

        <div class="grid-img">
            
        </div>
    </a>
</figure>
</div>
<div>
    <h6><a>{{ $categoryProduct->product_name }}</a>/h6>
    <span>XL/ XXL/ S</span>
    <p>{{ $categoryProduct->short_description }}</p>
    <p><em>TK. {{ $categoryProduct->product_price }} </em></p>
    <a href="">Add To Cart</a>
</div>
</div>
<div>

```

**14. when you click to any product for details information:**

I. in home.blade.php;

```

<figure>
    <a href="{{ route('product-details', ['id' => $categoryProduct->id]) }}"
        class="new-gri" data-toggle="modal" data-target="#m">
        <div class="grid-img">
            
        </div>
        <div class="grid-img">
            
        </div>
    </a>
</figure>

```

II. in web.php;

```

Route::get('/product-details/{id}', [
    'uses' => 'NewShopController@productDetails',
    'as' => 'product-details'
]);

```

III. in NewShopeController.php;

```

public function productDetails($id){
    $product = Product::find($id);
    return view('front-end.product.product-details', [

```

```
'product' = $product
```

```
]);
```

IV. in product-details.blade.php:

do the page dynamic as

product\_name, product\_price, product\_description, etc.

**Note:** the process for the brand details is same as we have already done for  
for the product-details

**\* Advertise with images:**

Its an individual part to do, it includes ad upload, delete, update with different table over database

**\* Introducing the Providers of the laravel:**

**Note:** when one making request to the website **firstly** work **middleware**, **secondly**

**AppServiceProvider.php** from Providers folder of the laravel, **thirdly Controllers** of the laravel.

We can the pass the data to the frontend by returning with view, but there is a another way of doing that and that is using public function **boot(){} of AppServiceProvider.php** of **Providers**.

For better understanding, code as follows;

In app/Providers/AppServiceProvider.php;

```
• public function boot(){  
    //view::share('name', 'Mobin Ahmed');  
    View::composer('front-end.includes.header', function($view){  
        $view->with('categories', Category::where('publicatiion_status', 1))->get();  
    });  
}
```

**Note:** To know better about provider's visit [laravel.com](https://laravel.com) site

**\* Another laravel package namely glodemans shopping card installation:**

Do search in google, **a simple shopping card laravel**, click the first github link

### Package installation process:

- i. install latest xampp, then uninstall php composer, while installing composer again you have provide php.exe path of the latest xampp server
- ii. write command, composer require glodemans/shoppingcart, in your project folder using CMD
- iii. go to composer.json of laravel, check require array
- iv. go to config/app.php of laravel, set the provider on provider array and set the alice on alice array

### **\* Add to cart A to Z process:**

#### 1. in product.details.blade.php:

- ```
{{ Form::open(['route' => 'add-to-card', 'method' => 'POST']) }}
```

```
<div class="color-quantity">
```

```
    <h6>Quantity</h6>
```

```
    <div class="quntity">
```

```
        <input type="number" name="qty" value="1" min="1">
```

```
        <input type="hidden" name="id" value="{{ $product->id }}" min="1">
```

```
    </div>
```

```
</div>
```

```
<div>
```

```
    <span class="size"> XL/XXL/S </span>
```

```
    <input type="submit" name="btn" value="Add To Card" class="my-card-b item_add">
```

```
</div>
```

```
{{ Form::close() }}
```

#### 2. in web.php:

- ```
Route::post('/cart/add', [  
    'uses' => CartController@addToCart,  
    'as' => 'add-to-cart'  
]);
```

#### 3. in CMD:



Write command, **php artisan make::controller CartController**

4. in CartController.php:

- ```
public function addToCart(Request $request){  
    //return $request->all();  
  
    $product = Product::find($request->id);  
    Cart::add([  
        'id' => $request->id,  
        'name' => $product->product_name,  
        'price' => $product->product_price,  
        'qty' => $request->qty,  
        'options' => [  
            'image' => $product->product_image  
        ]  
    ]);  
    return redirect('cart/show');  
}
```

**Note:** before use of the Cart object you must use it first on the top as use Cart;

The content of the Cart::add() will save in the browser storage not in the database

5. in web.php:

- ```
Route::get('/cart/show', [  
    'uses' => CartController@showCart,  
    'as' => 'show-cart'  
]);
```

6. in CartController.php:

- ```
public function showCart(){  
    $cartProducts = Cart::content();  
  
    //return $cartProducts;  
  
    return view('front-end.cart.show-cart', ['cartProducts' => $cartProducts]);
```

7. in show-cart.blade.php;

- `<table class="table table-bordered">`

```
<tr>
    <th>SL No.</th>
    <th>Name</th>
    <th>Images</th>
    <th>Price TK.</th>
    <th>Quantity</th>
    <th>Total Price TK.</th>
    <th>Action</th>
</tr>

@php($i=1)
@php($sum=0)
@foreach($cartProdcuts as $cartProduct)

<tr>
    <td>{{ $i++ }}</td>
    <td>{{ $cartProduct->name }}</td>
    <td></td>
    <td>{{ $cartProduct->price }}</td>
    <td>
        {{ Form::open(['route' => 'update-cart', 'method' => 'post']) }}
            <input type="number" name="qty" value="{{ $cartProduct->qty }}" min="1">
            <input type="hidden" name="rowId" value="{{ $cartProduct->rowId }}" min="1">
            <input type="submit" name="btn" value="Update">
        {{ Form::close() }}
    </td>
    <td>{{ $cartProduct->price*$cartProduct->qty }}</td>
</tr>
</tbody>
</table>
```

```

        <td>

            <a href="{{ route('delete-cart-items', ['rowId' => $cartProduct->rowId]) }}" class="btn
            btn-danger">

                <span class="glyphion glyphion-trash"></span>

            </a>

        </td>

    </tr>

<?php $sum = $sum+$total; ?>

@endforeach

</table>

<table class="table table-bordered">

    <tr>

        <th>Item total (TK.)</th>

        <td>{{ $sum }}</td>

    </tr>

    <tr>

        <th>Vat total (TK.)</th>

        <td>{{ $vat=0 }}</td>

    </tr>

    <tr>

        <th>Grand total (TK.)</th>

        <td>{{ $orderTotal = $sum+$vat }}</td>

        <?php

            Session::get('orderTotal', $orderTotal);

        ?>

    </tr>

</table>

```

8. in web.php;

- Route::get('/cart/delete/{id}', [

```

        'uses' => CartController@deleteCart',

        'as' => 'delete-cart-item'

    ]);

```

9. in CartController.php:

- ```
public function deleteCart($id){
    Cart::remove($id);
    return redirect('cart/show');
}
```

10. in font-end/includes/header.blade.php:

- ```
<a href="{{ route('show-cart') }}">Menu Cart button</a>
```
- And
- ```

<li><a href="{{ route('/') }}">New Shop</a></li>

```

11. in web.php:

- ```
Route::post('/cart/update/{id}', [
    'uses' => CartController@updateCart',
    'as' => 'update-cart'

]);
```

12. in CartController.php:

- ```
public function updateCart(Request $request){
    Cart::update($request->rowId, $request->qty);
    return redirect('cart/show');
}
```

13. in show-cart.blade.php:

- ```
<div class="row">

    <div class="col-md-11 col-md-offset-1">

        @if(Session::get('customerId') && Session::get('shippingId'))
            <a href="{{ route('checkout-payment') }}" class="btn btn-success pull-right">Checkout</a>
        @elseif(Session::get('customerId'))
            <a href="{{ route('checkout-shipping') }}" class="btn btn-success pull-right">Checkout</a>
        @else
            <a href="{{ route('checkout') }}" class="btn btn-success pull-right">Checkout</a>
        @endif
    </div>
</div>
```

```

        <a href="" class="btn btn-success">Continue Shopping</a>
    </div>
</div>

```

14. in web.php:

- `Route::get('/checkout', [
 'uses' => CheckoutController@index',
 'as' => 'checkout'
 ]);`

15. in CMD:

write command as `php artisan make:controller CheckoutController`

16. in CheckoutController.php:

- `public function index(){
 return view('front-end.checkout.checkout-content');
 }`

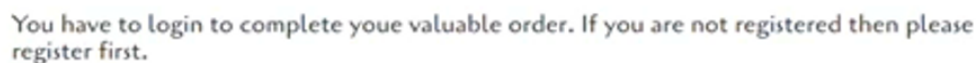
16. in checkout-content.blade.php:

In this checkout page there will be registration form and sign in form for the customer.

Registration form includes firstname, lastname, email, password, phone number, address and register button.

Sign in or login form includes email, password and sign in button.

And top of the forms there will be a message like;



You have to login to complete your valuable order. If you are not registered then please register first.

**\* Registration and login in checkout page to complete the order(front end):**

In checkout-content.blade.php:

```
{{ Form::open(['route' => 'customer-sign-up', 'method' => 'POST']) }}
```

In web.php:

- `Route::post('/customer/registration', [`

1);

1. In cmd;

2. In database/migrations/customerTable schema file;

3. In cmd;

In CheckoutController.php;

- ```
public function customerSingUp(Request $request){  
    $this->validate($request, [  
        'email_address' =>'email|unique:customers, email_address'  
    ]);  
  
    $customer = new Customer();  
  
    $customer->first_name = $request->first_name;  
  
    $customer->last_name = $request-> last_name;  
  
    $customer->email_address = $request-> email_address;  
  
    $customer->password = bcrypt($request-> password);           //here, 'bcrypt()'   
  
                                                                    //randomly generates  
                                                                    //one time key with  
                                                                    //respect to the  
                                                                    //password in the  
                                                                    //database  
  
    $customer->phone_number = $request-> phone_number;
```

```

$customer->address = $request-> address;

$customer->save();

$customerId = $customer->id;
Session::put('customerId', $customerId);
Session::put('customerName', $customer->first_name.' '.customer->last_name);

$data = $customer->toArray();

Mail::send('front-end.mails.confirmation-mail', $data, function($message) use ($data){

//here, $message is an in-built
//object of Mail packet class

$message->to($data['email_address']);
$message->subject('Confirmation mail');

});
}

```

In the above code, **Mail::send()** is used to send the email for confirmation

Note: there are various third party packages in php to send email, laravel also includes mail package in **config/app.php**

To forward the mail we need mail server, so here two factors includes 1) while sending email from your project which is in live server and the server contains mail server as well 2) mail sending testing in local server

Before sending mail you have to configure mail server first, you can do that by two ways

**1) from config/mail.php file 2) from .env file**

**Mail server configuration in laravel;**

1. go to config/mail.php;

now configure driver, host, port, from, encryption, username, password, sendmail, markdown  
these configuration will be same as the configuration of .env file

**Or**

2. go to .env;

configure as follows;

**MAIL\_DRIVER=smt**

```
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=mobinahmed102345@gmail.com
MAIL_PASSWORD=*****
MAIL_ENCRYPTION=tls/ssl
```

here, mail\_driver, mail\_host, mail\_port, mail\_encryption will be provided by the live server

mail cannot hold the objects for sending, you must make the objects to array before sending

3. go to gmail.com, enter my account, click on sign-in & security, scroll down, turn on "Allow less secure apps"

**4. socket connection:**

search in google as `ErrorException stream_socket_enable_crypto():SSL operation failed wit` for laravel enter over first link, now copy the following lines;

```
$options['ssl']['verify_peer']=FALSE;
$options['ssl']['verify_peer_name']=FALSE;
```

Now follow path as `vendor/swiftmailer/lib/classes/swift/SreamBuffer.php` and scroll the function namely private function `establishSocketConnection()`, paste the above line inside this function after `$options` array's `if()`

5. in front-end/mails/confirmation mail.blade.php;

```
<h1>Hello Dear {{ $first_name.' '.$last_name }}</h1>
Thanks for join our community ...
Your email address: {{ $email_address }}
Your mobile number: {{ $phone_number }}
```

Thanks By,  
Mobin Ahmed

6. in web.php;

```
Route::get('/checkout/shipping', [
    'uses' => 'CheckoutController@shippingForm',
    'as' => 'checkout-shipping'
]);
```

7. CheckoutController.php;

```
public function shippingForm(){
    $customer = Customer::find(Session::get('customerId'));
    return view('front-end-checkout.shipping', ['customer' => $customer]);
}
```

8. In shipping.blade.php;

```
{{ Form::open(['route' => 'new-shipping', 'method' => 'POST']) }}
```



Shipping Info goes here...



```
{{ Form::close() }}
```

write code with respect to above picture with following headline;

“Dear Mobin Ahmed, you have to give us product shipping info to complete your valuable order. If your billing and shipping info are same then, just click on save shipping info button”

**Note:** in the above picture, value of each input file will come from customer table

9. In cmd;

Write command as **php artisan make:model Shipping -m**

&

In migration file;

```
public function up()
{
    Schema::create('shippings', function (Blueprint $table) {
        $table->increments('id');
        $table->string('full_name');
        $table->string('email_address');
        $table->string('phone_number');
        $table->text('address');
        $table->timestamps();
    });
}
```

&

In cmd again;

Write command as **php artisan migrate**

10. In web.php;

```
Route::get('/shipping/save', [
    'uses' => 'CheckoutController@saveShippingInfo',
    'as' => 'new-shipping'
]);
```

11. In CheckoutController.php:

```
public function saveShippingInfo(Request $request) {  
    $shipping = new Shipping();  
    $shipping->full_name = $request->full_name;  
    $shipping->email_address = $request->email_address;  
    $shipping->phone_number = $request->phone_number;  
    $shipping->address = $request->address;  
    $shipping->save();  
  
    Session::put('shippingId', $shipping->id);  
    return redirect('/checkout/payment');  
}
```

12. In web.php:

```
Route::get('/checkout/payment', [  
    'uses' => 'CheckoutController@paymentForm',  
    'as' => 'checkout-payment'  
]);
```

13. In CheckoutController.php:

```
public function paymentForm(){  
    return view('front-end.checkout.payment');  
}
```

**Note: the briefing of the process of online or virtual order is given bellow:**

1. Info in the browser, Product choosing, add to the cart, manage the cart = shopping complete
2. Info in the database, then, checkout includes registration or login of orderer, we know who orders bill pay by him/her, so we have billing info as well = we tracked who the orderer  
Put customer or orderer id in the session
3. Info in the database, then, take shipping information for the orderer  
here, shipping = the info of where the will be delivered  
Put shipping id in the session
4. Info in the database, then, completing payment of the order by orderer or customer

14. In payment.blade.php:

```
{{ Form::open(['route' => 'new-order', 'method' => 'POST']) }}
```

Cash On Delivery	<input checked="" type="radio"/> Cash On Delivery
Paypal	<input type="radio"/> Paypal
BKash	<input type="radio"/> Bkash
<input type="button" value="Confirm Order"/>	

**{{ Form::close() }}**

Write code as above picture with heading as “Dear Mobin Ahmed, you have to give product payment method”

**Note:** if you want to make payment methode as bKash or Paypal([paypal.com](https://www.paypal.com)) or etc, you must buy the payment gateway for the individual, either from the country or abroad or website like [sslcommerce.com](https://www.sslcommerce.com)

15. In web.php:

```
Route::get('/checkout/order', [
    'uses' => 'CheckoutController@newOrder',
    'as' => 'new-order'
]);
```

16. In CheckoutController.php:

```
public function newOrder(Request $request){
    return $request->all();
}
```

17. In CMD:

Write commands one after one as follows;

```
php artisan make:model order -m
php artisan make:model payment -m
php artisan make:model orderdetail -m
```

**then,**

in migration files;

```

public function up()
{
    Schema::create('orders', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('customer_id');
        $table->integer('shipping_id');
        $table->float('order_total', 10, 2);
        $table->string('order_status')->default('Pending');
        $table->timestamps();
    });
}

public function up()
{
    Schema::create('payments', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('order_id');
        $table->string('payment_type');
        $table->string('payment_status')->default('Pending');
        $table->timestamps();
    });
}

public function up()
{
    Schema::create('order_details', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('order_id');
        $table->integer('product_id');
        $table->string('product_name');
        $table->float('product_price', 10, 2);
        $table->integer('product_quantity');
        $table->timestamps();
    });
}

```

then,

in cmd again:

write command as **php artisan migrate**

then,

check the tables on database

18. In CheckoutController.php:

```

public function newOrder(Request $request){
    $paymentType = $request->payment_type;
    If($paymentType == 'Cash'){

```

```

$order = new Order();
$order->customer_id = Session::get('customerId');
$order->shipping_id = Session::get('shippingId');
$order->order_total = Session::get('orderTotal');
$order->save();

$payment = new Payment();
$payment->order_id = $order->id;
$payment->payment_type = $paymentType;
$payment->save();

$cartProducts = Cart::content();
foreach ($cartProducts as $cartProduct) {
    $orderDetail = new OrderDetail();
    $orderDetail->order_id = $order->id;
    $orderDetail->product_id = $cartProduct->id;
    $orderDetail->product_name = $cartProduct->name;
    $orderDetail->product_price = $cartProduct->price;
    $orderDetail->product_quantity = $cartProduct->qty;
    $orderDetail->save();
}
Cart::destroy();
return redirect('/complete/order');

```

```

}
else if($paymentType == 'Paypal'){

}
else if($paymentType == 'Bkash'){

}

}

```

19. In web.php:

```

Route::get('/complete/order', [
    'uses' => 'CheckoutController@completeOrder',
    'as' => 'complete-order' ]
);

```

20. In CheckoutController.php:

```

public function completeOrder(){
    return 'success';
}

```

\* **Customer login check:**

In checkout.blade.php:

```
{{ Form::open(['route' => 'customer-login', 'method' => 'POST'])}}
```

In web.php:

- Route::post('/checkout/customer-login', [
 'uses' => 'CheckoutController@customerLoginCheck',
 'as' => 'customer-login'
 ]);

In CheckoutController.php:

- ```
public function customerLoginCheck(Request $request){  
    $customer = Customer::where('email_address', $request->email_address)->first();  
  
    //using row php  
    If(password_verify($request->password, $customer->password)){ //here,  
  //‘password_ve  
  //rify()’ is a php  
  //built-in  
  //function used  
  //to match the  
  //password  
  
        Session::put('customerId', $customer->id);  
        Session::put('customerName',$customer->first_name.' '.$customer->last_name);  
  
        Return redirect('/checkout/shipping');  
    }  
    else{  
        return redirect('/checkout')->with('message', 'invalid password');  
    }  
}
```

In checkout-content.blade.php:

```
<h3 class="text-center text-danger">{{ Session::get('message') }}</h3>
```

**\*\*>**

In header.blade.php page:

- ```
@if(Session::get('customerId'))  
    <li><a href="#"
```

```

        onclick="document.getElementById('customerLogoutForm').submit()">Logout
    <

    /a></li>

    {{ Form::open([ 'route' => 'customer-logout', 'method' => 'POST', 'id' =>
        'customerLogoutForm']) }}

    {{ Form::close() }}

    @else

        <li><a href="{{ route('new-customer-login') }}">Login</a></li>

    @endif

```

Two types of login or logout;

1. Normally customer can visit your site as a visitor without login, sometime he or she does login without the intension of buying anything, if doing so you can not redirect them to the checkout or shipping page. But when they are not login you must prevent them accessing from checkout page as well as shipping page.
2. Now if any one in the shipping page and intention to do logout, after logout you redirect him or her to the home page not in the checkout page
3. If any in the cart page and want go the checkout page, then he or she must login

In web.php;

- `Route::post('/checkout/customer-logout', [
 'uses' => 'CheckoutController@customerLogout',
 'as' => 'customer-logout'
 ]);`

In CheckoutController.php;

- `public function customerLogout(){
 Session::forget('customerId');
 Session::forget('customerName');
 return redirect('/');
 }`

In web.php;

- `Route::post('/checkout/new-customer-login', [
 'uses' => 'CheckoutController@newCustomerLogin',`

```
'as' => 'new-customer-login'
```

```
]);
```

In CheckoutController.php:

- ```
public function newCustomerLogin(){  
    return view('front-end.customer.customer-login');  
}
```

In customer-login.blade.php:

Copy all the code content from the **checkout-content.blade.php**, paste them on current file, after that change all the route in the **Form** and make clear all of the login and registration messages, if now any one make login send them to customer home quarter not in the shipping page

**\* Manage order A to Z coding(admin side):**

1. Admin header page.

Go to admin panel menu and define route for the manage order button as follows;

```
<li><a href="{{ route('manage-order') }}">Manage order</a></li>
```

2. Route on web.php page.

- ```
Route::get('/order/manage-order', [  
    'uses' => 'OrderController@manageOrderInfo',  
    'as' => 'manage-order'  
]);
```

3. Command.

```
php artisan make:model OrderController -m
```

4. OrderController.php page.

- ```
Public function manageOrderInfo(){  
    $order = DB::table('order')  
        ->join('customers', 'orders.customer_id', '=', 'customers_id')  
        ->join('payments', 'orders.id', '=', 'payments_id')  
        ->select('orders.*', 'customers.first_name', 'customers.last_name',  
            'payments.payment_type',  
            'payments.payment_status')  
        ->get();  
  
    return view('admin.order.manage-order', [orders => $order]);  
}
```

5. Manage-order.blade.php page.



- @php(\$i=1)  
@foreach(\$orders as \$order)  
    <tr>  
        <td>{{ \$i++ }}</td>  
        <td>{{ \$order->first\_name.' '.\$order->last\_name }}</td>  
        <td>{{ \$order->order\_total }}</td>  
        <td>{{ \$order->created\_at }}</td>  
        <td>{{ \$order->order\_status }}</td>  
        <td>{{ \$order->payment\_type }}</td>  
        <td>{{ \$order->payment\_status }}</td>  
        <td>  
            <a href="{{ route('view-order-detail', ['id' => \$order->id]) }}"  
                class="btn btn-info btn-xs" title="View Order Details">  
                <span class="glyphicon glyphicon-zoom-in"></span>  
            </a>  
            <a href="{{ route('view-order-invoice', ['id' => \$order->id]) }}"  
                class="btn btn-warning btn-xs" title="View Order Invoice">  
                <span class="glyphicon glyphicon-zoom-out"></span>  
            </a>  
            <a href="{{ route('download-order-invoice', ['id' => \$order->id]) }}"  
                class="btn btn-primary btn-xs" title="download Order Invoice">  
                <span class="glyphicon glyphicon-download"></span>  
            </a>  
            <a href="{{ route('unpublished-category', ['id' => \$order->id]) }}"  
                class="btn btn-success btn-xs" title="Edit Order">  
                <span class="glyphicon glyphicon-edit"></span>  
            </a>  
            <a href="{{ route('unpublished-category', ['id' => \$order->id]) }}"  
                class="btn btn-danger btn-xs" title="Delete Order">  
                <span class="glyphicon glyphicon-delete"></span>  
            </a>  
        </td>  
    </tr>  
  
@endforeach

#### 6. On web.php,

- Route::get('/order/view-order-detail/{id}', [  
    'uses' => 'OrderController@ViewOrderDetail',  
    'as' => 'view-order-detail'  
]);

#### 7. OrderController.php page,

- Public function ViewOrderDetails(\$id){  
    \$order = Order::find(\$id);

```

$customer = Customer::find($order->customer_id);
$shipping = Shipping::find($order->shipping_id);

$payment = Payment::where('order_id', $order->id)->first();

$orderDetails = OrderDetail::where('order_id', $order->id)->get();

return('admin.order.view-order', [
    'order' => $order,
    'customer' => $customer,
    'shipping' => $shipping,
    'payment' => $payment,
    'orderDetails' => $orderDetails
]);
}

```

8. view-order.blad.php

- @extends('admin.master')

```
@section('body')
```

```
<br/>
```

```
<div class="row">
```

```
<div class="col-sm-12">
```

```
<div class="panel panel-default">
```

```
<div class="panel-body">
```

```
<h3 class="text-center text-success">
```

```
Order Details Info For This Order</h3>
```

```
<table class="table table-bordered">
```

```
<tr>
```

```
<td>Order No</td>
```

```
<td>{{ $order->id }}</td>
```

```
</tr>
```

```
<tr>
```

```
<td>Order Total</td>
```

```
<td>{{ $order->total }}</td>
```

```
</tr>
```

```
<tr>
    <td>Order Status</td>
    <td>{{ $order->status }}</td>
</tr>
<tr>
    <td>Order Date</td>
    <td>{{ $order->created_at }}</td>
</tr>
</table>
</div>
</div>
</div>
</div>
```

```
.....
<div class="row">
    <div class="col-sm-12">
        <div class="panel panel-default">
            <div class="panel-body">
                <h3 class="text-center text-success">
                    Customer Info For This Order</h3>
                <table class="table table-bordered">
                    <tr>
                        <td>Customer Name</td>
                        <td>{{ $customer->first_name.'
                            '.$customer.last_name }}</td>
                    </tr>
                    <tr>
```

```

        <td>Phone Number</td>
        <td>{{ $customer->phone_number
                                }}</td>

    </tr>
    <tr>
        <td>Email Address </td>
        <td>{{ $customer->email_address
                                }}</td>

    </tr>
    <tr>
        <td>Address </td>
        <td>{{ $customer->address
                                }}</td>

    </tr>
</table>
</div>
</div>
</div>
</div>
.....
<div class="row">
    <div class="col-sm-12">
        <div class="panel panel-default">
            <div class="panel-body">
                <h3 class="text-center text-success">
                    Shipping Info For This Order</h3>
                <table class="table table-bordered">
                    <tr>

```

```

        <td>Full Name</td>
        <td>{{ $shipping->full_name
            }}</td>
    </tr>
    <tr>
        <td>Phone Number</td>
        <td>{{ $shipping->phone_number
            }}</td>
    </tr>
    <tr>
        <td>Email Address </td>
        <td>{{ $shipping->email_address
            }}</td>
    </tr>
    <tr>
        <td>Address </td>
        <td>{{ $shipping->address
            }}</td>
    </tr>
</table>
</div>
</div>
</div>
</div>
.....
<div class="row">
    <div class="col-sm-12">
        <div class="panel panel-default">
```

```

<div class="panel-body">
    <h3 class="text-center text-success">
        Payment Info For This Order</h3>
    <table class="table table-bordered">
        <tr>
            <td>Payment Type</td>
            <td>{{ $paymet->payment_type
                }}</td>
        </tr>
        <tr>
            <td>Payment Status</td>
            <td>{{ $shipping->payment_status
                }}</td>
        </tr>
    </table>
</div>
</div>
</div>
</div>
</div>
.....
<div class="row">
    <div class="col-sm-12">
        <div class="panel panel-default">
            <div class="panel-body">
                <h3 class="text-center text-success">
                    Product Info For This Order</h3>
                <table class="table table-bordered">
                    <tr>

```

```

                <th>SI No</th>
                <th>Product Id</th>
                <th>Product Name</th>
                <th>Product Price</th>
                <th>Product Quantity</th>
                <th>Total Price</th>

            </tr>

            @php($i=1)
            @foreach($orderDetails as $orderDetail)
            <tr>

                <td>{{ $i++ }}</td>

                <td>{{ $orderDetail->product_id
                    }}</td>

                <td>{{ $orderDetail->product_name
                    }}</td>

                <td>TK. {{ $orderDetail->product_price}}</td>

                <td>{{ $orderDetail->product_
                    quantity }}</td>

                <td>TK. {{ $orderDetail->product_price*$order
                    Detail->product Quanti
                    ty }}</td>

            </tr>

        </table>

    </div>

</div>

</div>

</div>

</div>

@endsection

```

9. In web.php page;

- `Route::get('/order/view-order-invoice/{id}', [`  
    `'uses' => 'OrderController@ViewOrderInvoice',`  
    `'as' => 'view-order- invoice'`  
    `]);`

10. In OrderController.php page;

- `Public function ViewOrderInvoice($id){`  
    `$order = Order::find($id);`  
    `$customer = Customer::find($order->customer_id);`  
    `$shipping = Shipping::find($order->shipping_id);`  
  
    `$payment = Payment::where('order_id', $order->id)->first();`  
  
    `$orderDetails = OrderDetail::where('order_id', $order->id)->get();`  
  
    `return('admin.order.view-order-invoice', [`  
        `'order' => $order,`  
        `'customer' => $customer,`  
        `'shipping' => $shipping,`  
        `'payment' => $payment,`  
        `'orderDetails' => $orderDetails`  
    `]);`  
    `}`

11. In view-order-invoice.php page;

Shipping Info  
Rohin Sorkar  
Soni Aahra  
01817161514

Billing Info  
robin Bork  
3sadeidas  
21312

Invoice information shit

Shipping Customer

Some Company  
c/o Some Guy

Order

Invoice #	00002
Date	2018-02-11 05:57:17
Amount Due	TK 15000

Order details

Item	Description	Rate	Quantity	Total Price
Nokia-10	Experience Review	TK 5000	1	TK5000
OPPO-F5	Experience Review	TK 10000	1	TK10000

Total TK 15000

ADDITIONAL NOTES

A finance charge of 1.5% will be made on unpaid balances after 30 days.

12. Visit the site <https://hdtuto.com>, know about pagination, two types, paginate and simple paginate



Generating Invoice pdf by using laravel libraries, one of the libraries namely **dom-pdf**, to know about the this library elaborately search on github, the **installation process** of this **laravel packager/library** by using the documentation from this site as follows;

- I. write the command **xampp/htdocs/project-folder>composer require barryvdh/laravel-dompdf** in CMD

config/app.php

```
<?php
return [
    ....
    'providers' => [
        ....
        Barryvdh\DomPDF\ServiceProvider::class,
    ],
    'aliases' => [
        ....
        'PDF' => Barryvdh\DomPDF\Facade::class,
    ]
]
```

II.

13. In web.php page;

- **Route::get('/order/download-order-invoice/{id}', [**  
    **'uses' => 'OrderController@downloadOrderInvoice',**  
    **'as' => 'download-order- invoice'**

**]);**

14. In OrderController.php page;

- **Public function downloadOrderInvoice(\$id){**  
    **\$order = Order::find(\$id);**  
    **\$customer = Customer::find(\$order->customer\_id);**  
    **\$shipping = Shipping::find(\$order->shipping\_id);**  
    **\$payment = Payment::where('order\_id', \$order->id)->first();**  
    **\$orderDetails = OrderDetail::where('order\_id', \$order->id)->get();**  
  
    **\$pdf = PDF::loadView('admin.order.download-order-invoice', [**  
        **'order' => \$order,**  
        **'customer' => \$customer,**  
        **'shipping' => \$shipping,**  
        **'payment' => \$payment,**  
        **'orderDetails' => \$orderDetails**  
    **]);**  
  
    **return \$pdf->stream('Invoice.pdf');**  
    **//return \$pdf->download('Invoice.pdf');**  
    **}**

15. In download-order-invoice.php page;

```

<html>
<head>
    <meta charset="utf-8">
</head>
<body>
    <h1>Billing Info</h1>
    <table>
        <tr>
            <th>Customer Name</th>
            <td>{{ $customer->first_name.' '.$customer->last_name }}</td>
        </tr>
        <tr>
            <th>Phone Number</th>
            <td>{{ $customer->phone_number }}</td>
        </tr>
    </table>
    <h1>Shipping Info</h1>
    <table>
        <tr>
            <th>Customer Name</th>
            <td>{{ $shipping->full_name }}</td>
        </tr>
        <tr>
            <th>Phone Number</th>
            <td>{{ $shipping->phone_number }}</td>
        </tr>
    </table>
</body>

```

```

<h1>Product Info</h1>
<table border="1">
    <tr>
        <th>Sl No</th>
        <th>Product Id</th>
        <th>Product Name</th>
        <th>Product Price</th>
        <th>Product Quantity</th>
        <th>Total Price</th>
    </tr>
    @php($i=1)
    @foreach($orderDetails as $orderDetail)
        <tr>
            <td>{{ $i++ }}</td>
            <td>{{ $orderDetail->product_id }}</td>
            <td>{{ $orderDetail->product_name }}</td>
            <td>TK. {{ $orderDetail->product_price }}</td>
            <td>{{ $orderDetail->product_quantity }}</td>
            <td>TK. {{ $orderDetail->product_price*$orderDetail->pr
        </tr>
    @endforeach
</table>

</body>
</html>

```

16. Download the generated PDF as;

The screenshot shows a PDF document with three main sections: Billing Info, Shipping Info, and Product Info. The Billing Info section contains fields for Customer Name (robin Bosa) and Phone Number (31231221). The Shipping Info section contains fields for Customer Name (Md. Amran Khan) and Phone Number (0181716151413). The Product Info section contains a table with 6 columns: Sl No, Product Id, Product Name, Product Price, Product Quantity, and Total Price. The table has 3 rows of data.

Sl No	Product Id	Product Name	Product Price	Product Quantity	Total Price
1	16	Apex	TK. 2000	1	TK. 2000
2	14	Nokia-10	TK. 5000	1	TK. 5000
3	15	OPPO-F5	TK. 10000	2	TK. 20000

**\* Laravel Middleware:**

**What is middleware?** = while client request to the site, response of the request depend on is there any middleware set on this request or not, so the middleware works as a guard between request and response.

**How do you work with middleware?** = create middleware, do middleware registration, and use the created middleware.

The location of middleware on laravel is **app/Http/Middleware**

**Working process with middleware:**

Middleware creation:

- I. **php artisan make:middleware middleware\_name(Mobin)**
- II. go to **app/Http/Middleware**, choose the middleware you created, and set the middleware login through handle function of the middleware

#### Middleware registration;

- i. go to **app/Http/kernel.php** file, scroll down to the **routeMiddleware** array, and set the index value as;
  - **'Mobin' => App\Http\Middleware\Mobin::class**

#### Middleware use;

- i. go to web.php file on laravel and choose the Route you want to set middleware as follows;
  - **Route::get('/order/download-order-invoice/{id}', [  
    'uses' => 'OrderController@downloadOrderInvoice',  
    'as' => 'download-order- invoice',  
    'middleware' => 'Mobin'**  
  
**]);**
- ii. Or go the Router.php file on laravel, search for auth, and set middleware auth register as follows;
  - **\$this->get('register',  
    'Auth\RegisterControlle@showRegistrationForm')->name('register')->middleware('Mobin');**

#### Multiple routes for the decided middleware or group of routes for the middleware, set as follows;

- **Route::group(['middleware' => 'Mobin'], function{  
    Route::get('/order/download-order-invoice/{id}', [  
        'uses' => 'OrderController@downloadOrderInvoice',  
        'as' => 'download-order- invoice',  
  
    });**

```
Route::get('/order/download-order-invoice/{id}', [  
    'uses' => 'OrderController@downloadOrderInvoice',  
    'as' => 'download-order- invoice',  
  
]);
```

```
Route::get('/order/download-order-invoice/{id}', [  
    'uses' => 'OrderController@downloadOrderInvoice',  
    'as' => 'download-order- invoice',  
  
]);
```

```
});
```

\* Ajax with laravel (email checking):

Interface:



The image shows a web form titled "Registration Form". It contains several input fields: "First Name", "Last Name", "Email Address", "Password", "Phone Number", and a larger "Address" field. At the bottom of the form is a green button labeled "Registration".

1. In checkout.blade.php:

```
div class="col-md-9">
    <input type="email" name="email_address" id="email_address" class="form-
    /div>
```

<script>

```
var email_address = document.getElementById('email_address');
```

```
email_address.onblur = function(){
```

```
    var email = document.getElementById('email_address').value;
    var xmlhttp = new XMLHttpRequest();
    var serverPage = 'http://localhost/php72/public/ajax-email-check/' + email;
    xmlhttp.open('GET', serverPage);
    xmlhttp.onreadystatechange = function () {
        // alert(xmlhttp.readyState);
        if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {
            document.getElementById('res').innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.send(null);
```

```
}
```

</script>

2. In web.php:

```
Route::get('/ajax-email-check/{email}', [
    'uses' => 'CheckoutController@ajaxEmailCheck',
    'as' => 'ajax-email-check';
]);
```

3. In checkoutController.php:

```
public function ajaxEmailCheck($a){
    //echo $a;

    $customer = Customer::where('email_address', $a)->first();
    if($customer) {
        echo 'Already Exist';
    } else {
        echo 'Avaliable';
    }
}
```

4. In checkout.blade.php:

```
<input type="email" name="email_address" id="email_address" class="form-control"/>
<span class="text-success" id="res"></span>
```

5. in checkoutController.php:

```
<script>
    var email_address = document.getElementById('email_address');
    email_address.onblur = function () {
        var email = document.getElementById('email_address').value;
        var xmlhttp = new XMLHttpRequest();
        var serverPage = 'http://localhost/php72/public/ajax-email-check/'+email;
        xmlhttp.open("GET", serverPage);
        xmlhttp.onreadystatechange = function () {
            // alert(xmlhttp.readyState);
            if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                document.getElementById('res').innerHTML = xmlhttp.responseText;
                if(xmlhttp.responseText == 'Already Exist') {
                    document.getElementById('regBtn').disabled = true;
                } else {
                    document.getElementById('regBtn').disabled = false;
                }
            }
        }
        xmlhttp.send(null);
    }
</script>
```

### Result:

The image displays two side-by-side screenshots of a web registration form. The form includes input fields for 'Email Address', 'Password', 'Phone Number', and 'Address'. A green 'Registration' button is positioned at the bottom of each form. In the left screenshot, the email 'khan@gmail.com' is entered and highlighted in yellow, with a red error message 'Already Exist' displayed below it. In the right screenshot, the email 'awcdasduasda@andasda.com' is entered and highlighted in green, with a green success message 'Available' displayed below it. The other fields (Password, Phone Number, Address) are empty in both screenshots.

### **\* Upload laravel project on live server(cPannel):**

cPannel is a live server, while you will be buying domain(like laravel.com) and the will be hosted in host server, afterthat you can access this domain on live server like cPannel

Basically after login on cPannel, will find the cPannel dashboard with many options, you may work with these options

While you uploading project on cPannel, there is an option as file manager, this includes project root folder namely home/ and if you want to access main domin that you bought

you have to go to the folder namely public\_html that is in the root folder as well, this public\_html folder includes a index.html/index.php/index with any extention file, this file's content will run while you will hitting on your main domain

### Uploading process:

1. **cut public folder from laravel project folder and make it as public.zip**
2. **go to cPannel dashboard, choose file manager, then home folder, then enter into public\_html folder**
3. **upload public.zip file from computer, then refresh cPannel, then extract it, then enter to the public which is extracted and select all the files/folders, then click move and delete public word from move's path and click ok**
4. **now delete default index.html and empty public folder from public\_html folder**
5. **now make laravel project folder as zip that is without public folder**

6. go to cPanel dashboard again, choose file manager, then enter into the home folder, then upload the zipped laravel project from computer, then refresh cPanel, then extract it

database import and connection:

1. go to cPanel dashboard, select MySQL database, ctrl+click, then create database with any name, click on create database button and note database name, then scroll down  
and create user for the database as follows;  
username, password, confirm password, then click create user button  
note username and password  
now scroll down to Add User To Database portion, then the user and database you created, then click Add button, select all privileges, then click make change button
2. go to cPanel dashboard, select phpMyAdmin option, you will find your database, then click on database, select import, choose database file of the laravel project from computer, click on go button
3. home folder, select .env file, click on edit, then define database name, username, password in the mysql db connection portion those were noted, then click on save change button
4. now hit your domain on url and you will get an error as currently unable to handle this request

5. to handle this open the index file in public\_html folder as edit and you will find code lines as follows;

```
require __DIR__.'../vendor/autoload.php';  
$app = require_once __DIR__.'../bootstrap/app.php';
```

now you have to customize these lines as follows;

```
require __DIR__.'../uploadedLaravelProjectFolderName/vendor/autoload.php';  
  
$app=require_once __DIR__.'../uploadedLaravelProjectFolderName/bootstrap/a  
pp.php';
```

then click save changes button

6. go to cPanel dashboard, select selectPHPVersion, select latest version or required version of php, then click on set as current button
7. now hit your domain on url, your project's home page interface will appear
8. go to the public\_html folder and delete the error\_log file