



LLO 02: Web application using PHP and MySQL

Contents:

- Intro to Web Engineering
- Technologies
- Tools
- Introduction to MVC
- Introduction to LARAVEL Framework
- Getting Started with LARAVEL
- LARAVEL Terminologies & Directory Structure
- Creating First View with Laravel
- Basic CRUD with Laravel

Introduction to Web Engineering

Web Engineering is the application of systematic and quantifiable approaches (concepts methods, techniques tools) to cost - effective requirements analysis, design, implementation, testing, operation, and maintenance of **high quality Web applications**.

Technologies to be studied

- HTML
- CSS
- JavaScript
- Bootstrap
- JQuery
- PHP
- MySQL [Database]
- Laravel [PHP FRAMEWORK]

Tools – IDEs

- Visual Studio Code
- Adobe Dreamweaver

- Visual Studio
- XAMPP

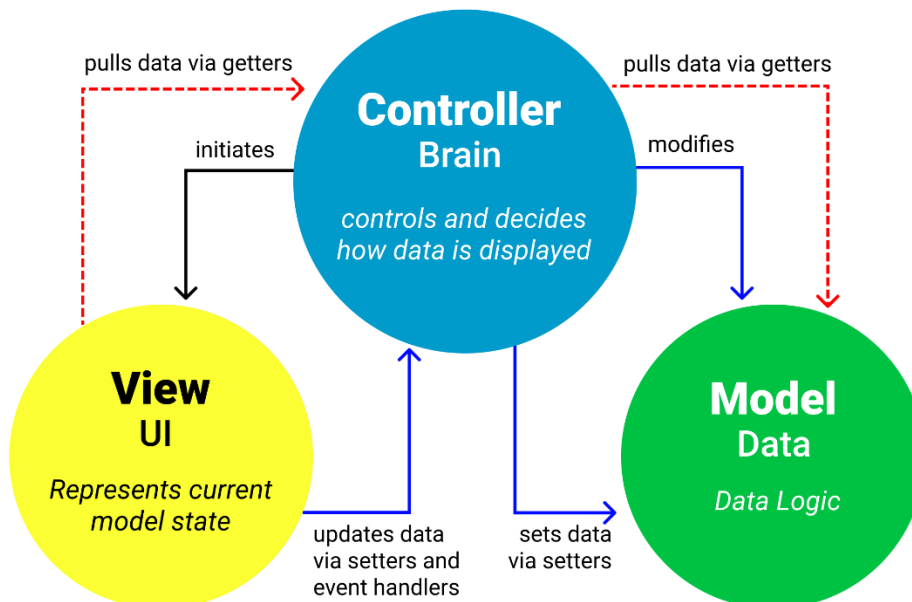
10.1 Introduction to MVC architecture:

The **Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main logical components:

1. **Model:** The backend that contains all the data logic
2. **View:** The frontend or graphical user interface (GUI)
3. **Controller:** The brains of the application that controls how data is displayed

Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

MVC Architecture Pattern



The concept of MVCs was first introduced by **Trygve Reenskaug**, who proposed it as a way to develop desktop application GUIs.

Today the MVC pattern is used for modern web applications because it allows the application to be scalable, maintainable, and easy to expand.

NEED OF MVC:

Three words: **separation of concerns**, or SoC for short.

The MVC pattern helps you break up the frontend and backend code into separate components. This way, it's much easier to manage and make changes to either side without them interfering with each other.

But this is easier said than done, especially when several developers need to update, modify, or debug a full-blown application simultaneously.

10.2 Introduction to LARAVEL Framework:

Laravel is a PHP-based web framework for building high-end web applications using its significant and graceful syntaxes. It comes with a robust collection of tools and provides application architecture. Moreover, it includes various characteristics of technologies like ASP.NET MVC, CodeIgniter, Ruby on Rails, and many more. This framework is an open-source framework. It facilitates developers by saving a lot of time and helps reduce thinking and planning to develop the entire website from scratch.

Focus Points of LARAVEL:

- **Taylor Otwell** developed Laravel in **July 2011**, and it was released more than five years after the release of the CodeIgniter framework.
- Laravel is a PHP-based web framework.
- Laravel is one of the open-source PHP frameworks.
- Laravel follows the model-view-controller (**MVC**) architectural pattern.

Features of LARAVEL:

Some essential features provided by Laravel are:

- Routing controllers.
- Configuration management.
- Testability.
- Authentication and authorization of users.
- Modularity.
- ORM (Object Relational Mapper) features.
- Provides a template engine. e.t.c

10.3 Getting Started with LARAVEL:

Before installing and configuring Laravel, there are some prior requirements that must be fulfilled to begin with:

Requirements:

- PHP version 7.3 or higher
- Composer
- MySQL

Since we have already installed XAMPP in our systems, it will be providing PHP and MySQL, we only need to install Composer and Laravel.

Composer:

Laravel implements a composer for managing dependencies within it. Hence, before using Laravel, it needs to check whether you have a composer set up on your system or not.

If you don't have Composer installed on your computer, first visit this URL to download Composer:

<https://getcomposer.org/Composer-Setup.exe>

- While installing composer, Select PHP path from XAMPP folder to configure.
- When installing the Composer, cross-check whether it is installed or not by typing in the composer command prompt. You can see the Composer screen in that CMD only.
- Open command prompt, type the following command, it will show output as follows upon successful installation of composer:

composer


```
C:\Windows\System32\cmd.e  X  +  v

Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

D:\xampp\htdocs>composer create-project laravel/laravel my_project
Creating a "laravel/laravel" project at "./my_project"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v10.0.7)
- Installing laravel/laravel (v10.0.7): Extracting archive
Created project in D:\xampp\htdocs\my_project
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 106 installs, 0 updates, 0 removals
- Locking brick/math (0.11.0)
- Locking dflydev/dot-access-data (v3.0.2)
- Locking doctrine/inflector (2.0.6)
- Locking doctrine/lexer (3.0.0)
- Locking dragonmantank/cron-expression (v3.3.2)
- Locking egulias/email-validator (4.0.1)
- Locking fakerphp/faker (v1.21.0)
- Locking filp/whoops (2.15.2)
- Locking fruitcake/php-cors (v1.2.0)
- Locking graham-campbell/result-type (v1.1.1)
```

You can also check your htdocs folder, where you'll find my_project with Laravel framework.

To make sure that Laravel has been installed completely move to your project folder by

cd my_project,

then type the following command, it will display Laravel welcome page:

php artisan serve

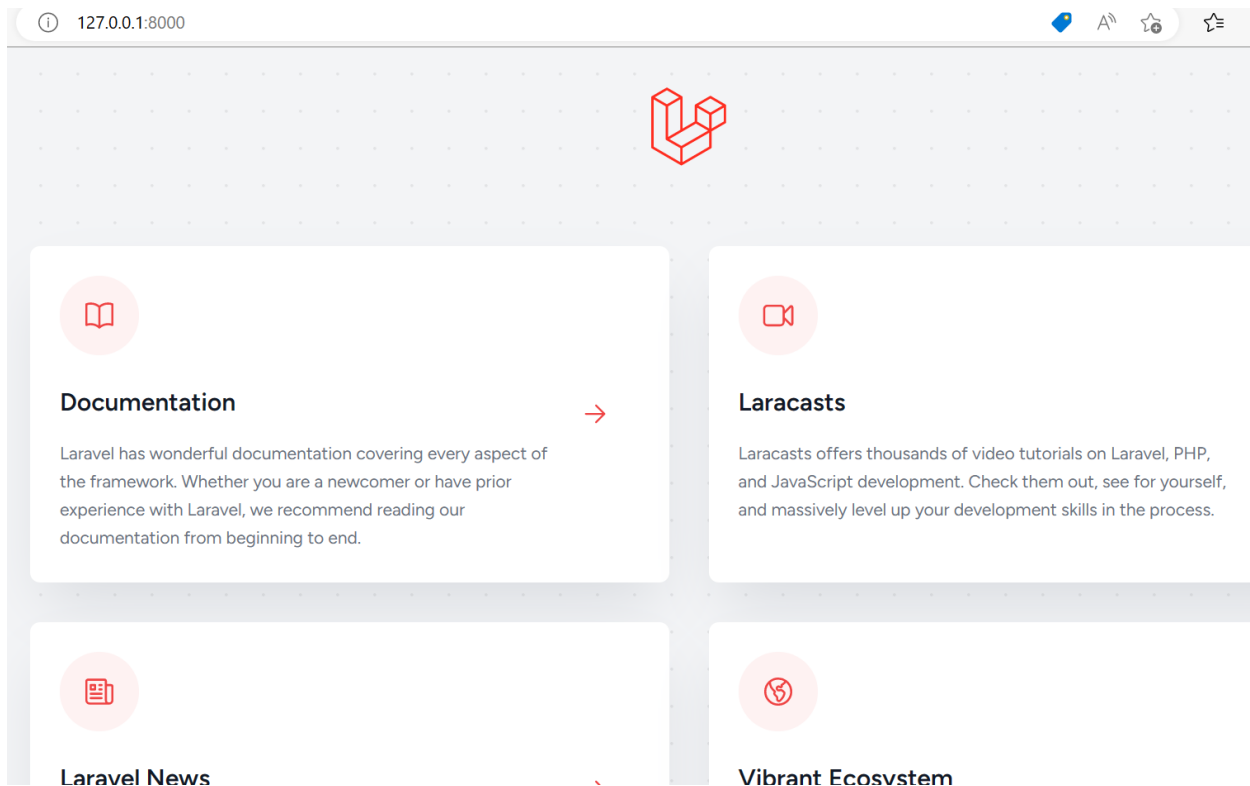
```
D:\xampp\htdocs>cd my_project

D:\xampp\htdocs\my_project>php artisan serve

INFO Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server
```

Now press Ctrl+click on **[http://127.0.0.1:8000]** to display:



Finally your Laravel project is fully functional. We will dig into the structure of Laravel now to develop its frontend and backend.

10.4 LARAVEL Terminologies & Directory Structure:

Terminologies:

Many modern PHP frameworks are object-oriented. It is good to understand some of the basics of classes and objects before using them.

1. MVC Architecture:

The Model-View-Controller (MVC) architectural pattern is a set of rules that explains how to construct successful web apps easily and straightforwardly. This main Laravel framework pattern helps bring order to the unstructured code. Thanks to MVC support, the process of web applications development is quick and convenient, no matter if they are large or small.

2. Artisan CLI:

The Artisan Command-Line Interface (CLI) is a tool for web developers which helps them migrate data, manage databases, create the basic code, controllers, models, and other parts of an app. With Laravel's CLI, web developers can create MVC files and issue their commands. It significantly simplifies and shortens the development process.

3. Blade Template Engine:

The Blade Template Engine is a simple but essential and powerful templating engine. It allows for connecting data models, processing the application's code in the source templates, directing the output to a certain text file or stream. That makes the whole process much more convenient. This templating engine is fast, secure, and easy to use.

4. Eloquent ORM:

The Eloquent Object-Relational Mapper (ORM) is a tool for working with databases. It helps establish and maintain essential communication between a web application's architecture and its databases. The ORM uses an expressive PHP Syntax that is integral to the language of the framework, which means saving time and effort.

5. Redis:

Redis is an open-source, advanced key-value store. Because keys can contain strings, hashes, lists, sets, and sorted sets, it is commonly known as a data structure server. Laravel uses Redis to connect to an existing session and general-purpose cache. Redis interacts with the session directly.

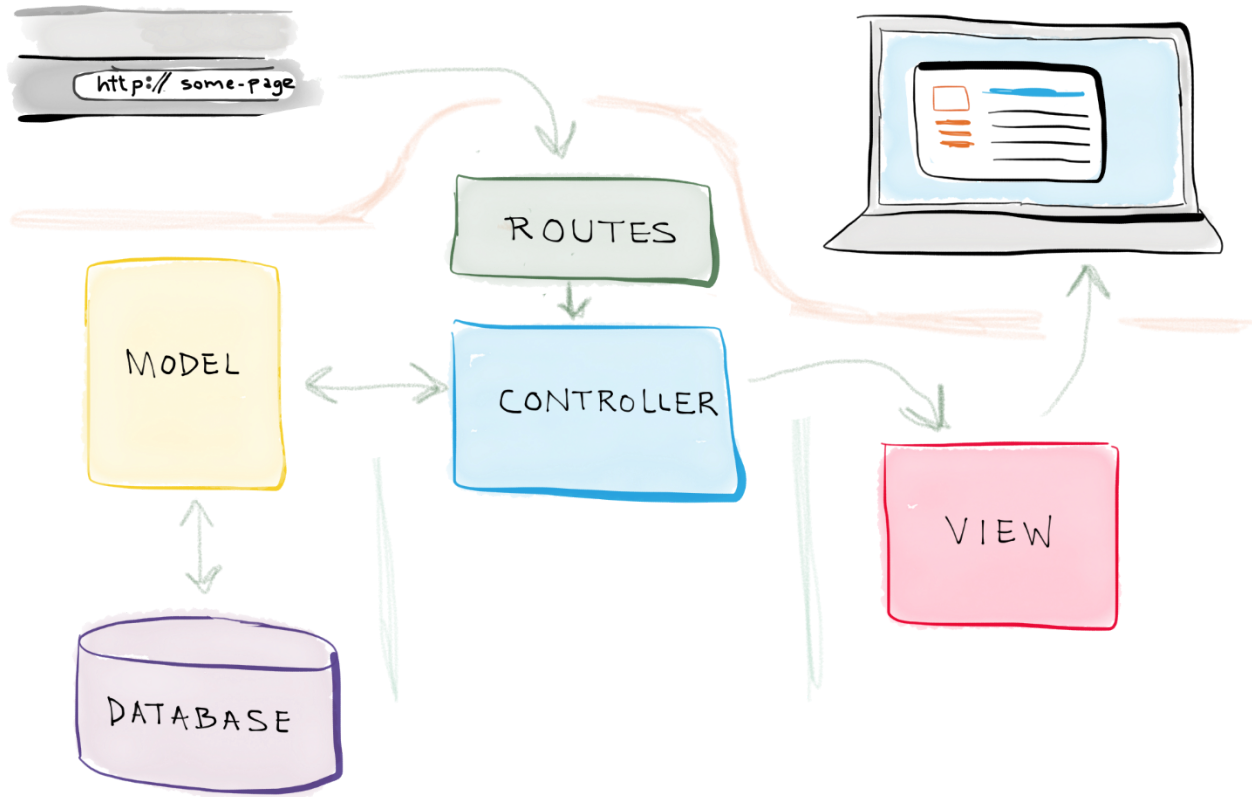
Directory Structure:

After installing the framework, take a glance around the project to familiarize yourself with the directory structure. The **app** directory contains folders of **models** and **HTTP** which has **controllers**. **Resources** directory have **Views** and **Routes** directory have respective routing files.

Laravel applications follow the traditional Model-View-Controller design pattern, where you use:

- **Controllers** to handle user requests and retrieve data, by leveraging Models
- **Models** to interact with your database and retrieve your objects' information
- **Views** to render pages

Additionally, **routes** are used to map URLs to designated controller actions, as shown below.



So...

- A request is made — say, when a user enters a URL associated with your application.
- A **route** associated with that URL maps the URL to a controller action.
- That **controller** action leverages the necessary **model(s)** to retrieve information from the database, and then passes that data off to a view.
- And that **view** renders the final page.

10.5 Creating First View with Laravel:

First Sample View:

Goto **views** folder and create '**test.blade.php**' and write following code:

In my case my views folder path

"D:\xampp\htdocs\my_project\resources\views"

test.blade.php

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
ul {
```

```
    list-style-type: none;
```

```
    margin: 0;
```

```
    padding: 0;
```

```
    overflow: hidden;
```

```
    background-color: #333;
```

```
}
```

```
li {
```

```
    float: left;
```

```
}
```

```
li a {
```

```
    display: block;
```

```
    color: white;
```

```
    text-align: center;
```

```
    padding: 14px 16px;
```

```

    text-decoration: none;
}

li a:hover {
    background-color: #111;
}
</style>
</head>
<body>

<ul>
    <li><a class="active" href="#home">Home</a></li>
    <li><a href="#news">News</a></li>
    <li><a href="#contact">Contact</a></li>
    <li><a href="#about">About</a></li>
</ul>

</body>
</html>

```

Setting Route For test.blade.php

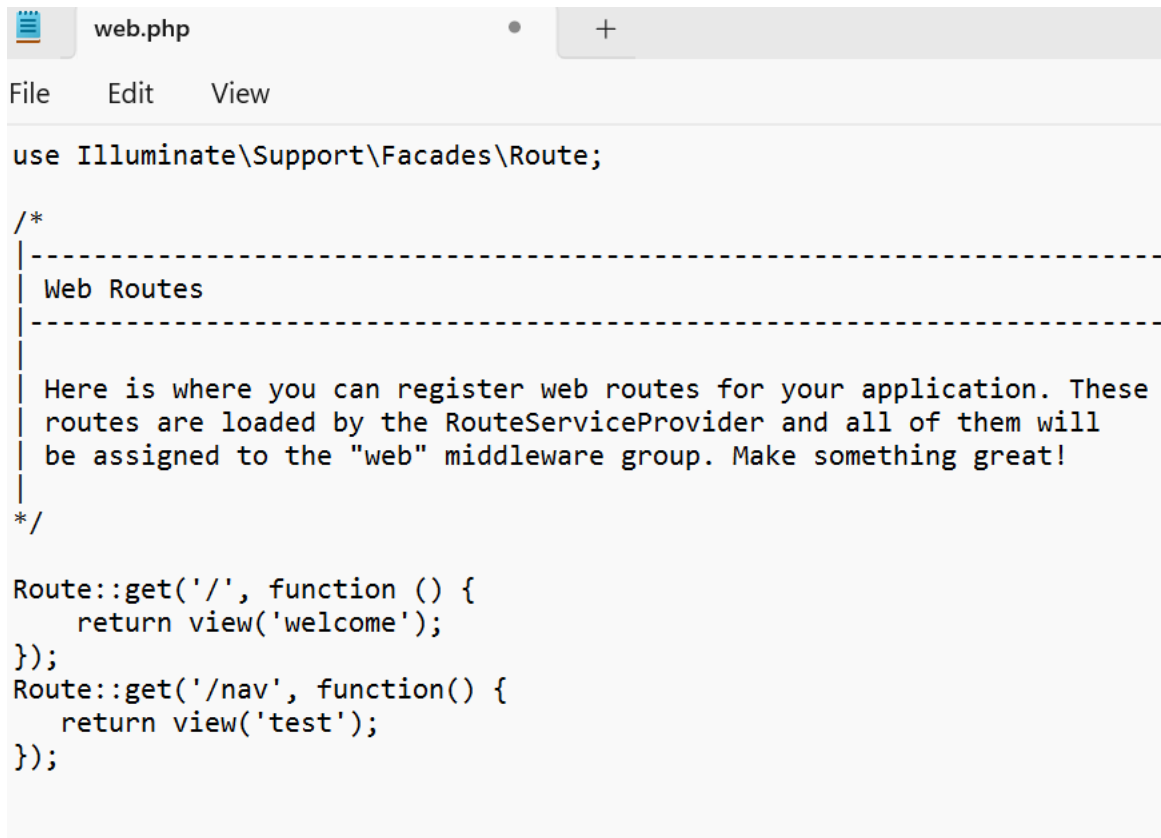
Now we need to set route for our first view, goto **routes** folder and edit the **web.php** file, in my case the path is **D:\xampp\htdocs\my_project\routes web.php**

add following code to the existing file:

```
Route::get('/nav', function() {
```

```
    return view('test');  
});
```

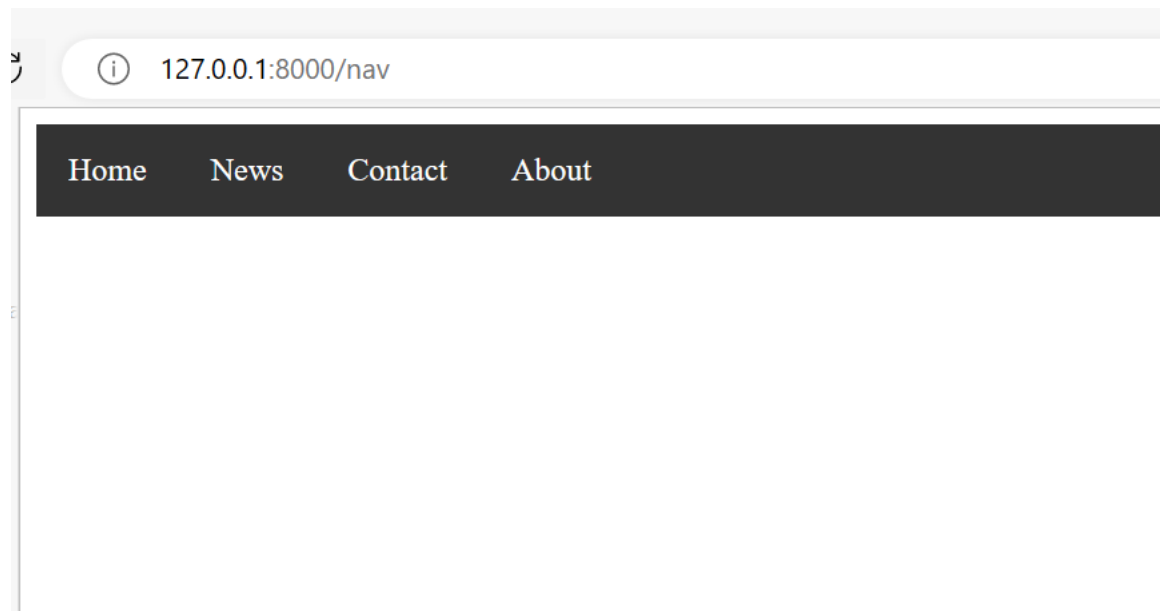
Such that:



```
use Illuminate\Support\Facades\Route;  
  
/*  
|-----  
| Web Routes  
|-----  
|  
| Here is where you can register web routes for your application. These  
| routes are loaded by the RouteServiceProvider and all of them will  
| be assigned to the "web" middleware group. Make something great!  
|  
*/  
  
Route::get('/', function () {  
    return view('welcome');  
});  
Route::get('/nav', function() {  
    return view('test');  
});
```

Now to view your file, type following address in your browser, make sure your artisan server is running:

127.0.0.1:8000/nav



10.6 Basic CRUD with Laravel:

In this example, we will create a product crud application using laravel 10. we will create a products table with name and detail columns using laravel 10 migration, then we will create routes, controller, view, and model files for the product module. we will use bootstrap 5 for design now. so let's follow the below step to create a crud operation with laravel 10.

Step 1: Install Laravel 10 App

Let us begin the tutorial by installing a new laravel 10 application. if you have already created the project, then skip the following step.

```
composer create-project laravel/laravel example-app
```

Step 2: Database Configuration

In the second step, we will make a database configuration, we need to add the database name, MySQL username, and password. So let's open the .env file and fill in all details like as below:

.env

```
DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=here your database name(blog)
```

Step 3: Create Migration

Here, we will create a "products" table using laravel migration. so let's use the following command to create a migration file. First change your directory to your project folder, cd example-app

```
php artisan make:migration create_products_table --  
create=products
```

After this command you will find one file in the following path "database/migrations" and you have to put below code in your migration file for creating the products table.

```
<?php  
  
use Illuminate\Database\Migrations\Migration;  
use Illuminate\Database\Schema\Blueprint;  
use Illuminate\Support\Facades\Schema;  
  
return new class extends Migration
```

```
/**  
 * Run the migrations.  
 */
```

```

    * @return void

    */

    public function up()

    {

        Schema::create('products', function (Blueprint
$table) {

            $table->id();

            $table->string('name');

            $table->text('detail');

            $table->timestamps();

        });

    }

```

```

/**

 * Reverse the migrations.

 *

 * @return void

 */

    public function

        Schema

        'products'

```

```
};
```

Now you have to run this migration by the following command:

```
php artisan migrate
```

Step 4: Create Controller and Model

In this step, now we should create new resource controller as ProductController. So run below command and create new controller. below controller for create resource controller.

```
php artisan make:controller ProductController --resource --model=Product
```

After the below command, you will find a new file in this path "app/Http/Controllers/ProductController.php".

In this controller will create seven methods by default as below methods:

- 1)index()
- 2)create()
- 3)store()
- 4)show()
- 5)edit()
- 6)update()
- 7)destroy()

So, let's copy below code and put on ProductController.php file.

app/Http/Controllers/ProductController.php

```
<?php
```

```
namespace App
```



```

use App\Models\Product;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\Http\Response;
use Illuminate\View\View;

class ProductController extends Controller

```

```
{
```

```
    /**
```

```
     * Display a listing of the resource.
```

```
    */
```

```
    public function view()
    {
        return view('products.index', [
            'products' => Product::paginate(5),
            'page' => request('page', 1)
        ]);
    }

    /**
```

```
1
```

```
1
```

```
5
```

```

    * Show the form for creating a new resource.
    */

    public function create(): View
    {
        return view('products.create');
    }

    /**
     * Store a newly created resource in storage.
     */

    public function store(Request $request):
RedirectResponse
    {

```

```

        'name'         'required'
        'detail'       'required'

```

```

        Product

```

```

        return redirect('products.index')
            with('success', 'Product created
successfully.')

```

```

}

/**
 * Display the specified resource.
 */
public function show(Product $product): View
{
    return view('products.show', compact('product'));
}

/**
 * Show the form for editing the specified resource.
 */
public function edit(Product $product): View
{
    return view('products.edit', compact('product'));
}

/**
 * Update the specified resource in storage.
 */

```

```

    public function update(Request $request, Product
$product): RedirectResponse
    {
        $request->validate([
            'name' => 'required',
            'detail' => 'required',
        ]);

        $product->update($request->all());

        return redirect()->route('products.index')
            ->with('success','Product updated
successfully');

```

```

/**
 * Remove the specified resource from storage.
 */

    public function          Product
RedirectResponse

        delete

```

```
        return redirect()->route('products.index')
        ->with('success','Product deleted
successfully');
```

Don't forget to add database column to fillable property to allow mass assignment.

app/Models/Product.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    protected $fillable = array('name' , 'detail' );
}
```

Step 5: Add Resource Route

Here, we need to add resource route for product crud application. so open your "routes/web.php" file and add following route.

routes/web.php

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductController;

/*
|-----
|
| Web Routes
|-----
|
| Here is where you can register web routes for your
| application. These
|
| routes are loaded by the RouteServiceProvider within a
| group that
| contains the "web" middleware group. Now create
| something great!
|
*/

Route::get('products', ProductController::class);
```

Step 6: Add Blade Files

In last step. In this step we have to create just blade files. So mainly we have to create layout file and then create new folder "products" then create blade files of crud app. So finally you have to create following below blade file:

1) layout.blade.php

2) index.blade.php

3) create.blade.php

4) edit.blade.php

5) show.blade.php

So let's just create following file and put below code.

resources/views/products/layout.blade.php

```
<!DOCTYPE html>

<html>

<head>

    <title>Laravel 10 CRUD Application</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/c
ss/bootstrap.min.css" rel="stylesheet">

</head>

<body>
```

```
<div class "container">
```

```
</div>
```

```
</body>
```

```
</html>
```

resources/views/products/index.blade.php

```
@extends('products.layout')
```

```
@section('content')
```

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

```
        <div class="col-lg-12 margin-tb">
```

```
            <div class="pull-left">
```

```
                <h2>Laravel 10 CRUD Example from  
scratch</h2>
```

```
            </div>
```

```
            <div class="pull-right">
```

```
                <a class="btn btn-success" href="{{  
route('products.create') }}"> Create New Product</a>
```

```
            </div>
```

```
        </div>
```

```
    </div>
```

```
    <div class "alert alert-success">
```

```
        <p>
```

```
        </p>
```

```
    </div>
```



```

@endif

<table class="table table-bordered">

  <tr>

    <th>No</th>

    <th>Name</th>

    <th>Details</th>

    <th width="280px">Action</th>

  </tr>

  @foreach ($products as $product)

    <tr>

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

      <td>

        <td>

          </td>

          <td>

            <td>

              <form action "{{
route('products.destroy', $product->

                <a class "btn btn-info" href "{{
route('products.show', $product->

                  <a class "btn btn-primary" href "{{
route('products.edit', $product->

```

```

        @csrf

        @method('DELETE')

        <button type="submit" class="btn btn-
danger">Delete</button>

        </form>

    </td>

</tr>

</table>

```

resources/views/products/create.blade.php

```

@extends('products.layout')

@section('content')

<div class "row">

    <div class "col-lg-12 margin-tb">

        <div class "pull-left">

```

```

        <h2>Add New Product</h2>

    </div>

    <div class="pull-right">

        <a class="btn btn-primary" href="{{
route('products.index') }}"> Back</a>

    </div>

</div>

@if ($errors->any())

    <div class="alert alert-danger">

        <strong>Whoops!</strong> There were some problems
with your input.<br><br>

        <ul>

            <li>

            </li>

        </ul>

    </div>

<form action "{{ route('products.store') }}"
method "POST">

```

```

@csrf

<div class="row">

  <div class="col-xs-12 col-sm-12 col-md-12">

    <div class="form-group">

      <strong>Name:</strong>

      <input type="text" name="name"
class="form-control" placeholder="Name">

    </div>

  </div>

  <div class="col-xs-12 col-sm-12 col-md-12">

    <div class="form-group">

      <strong>Detail:</strong>

      <textarea class "form-control"
style "      150px" name "detail"
placeholder "Detail"></textarea>

    </div>

  </div>

  <div class "col-xs-12 col-sm-12 col-md-12 text-
center">

    <button type "submit" class "btn btn-
primary">    </button>

  </div>

</div>

```

```
</form>
```

resources/views/products/edit.blade.php

```
@extends('products.layout')

@section('content')

    <div class="row">

        <div class="col-lg-12 margin-tb">

            <div class="pull-left">

                <h2>Edit Product</h2>

            </div>

            <div class="pull-right">

                <a class="btn btn-primary" href="{{
route('products.index') }}"> Back</a>

            </div>

        </div>

    </div>

    <div class="alert alert-danger">

        <strong>                </strong>
        <br><br>

    </div>
```

```

        <ul>

            @foreach ($errors->all() as $error)

                <li>{{ $error }}</li>

            @endforeach

        </ul>

    </div>

@endif

    <form action="{{ route('products.update', $product->id) }}" method="POST">

        @csrf

        @method('PUT')

```

```

    <div class "row">

        <div class "col-xs-12 col-sm-12 col-md-12">

            <div class "form-group">

                <strong>        </strong>

                <input type "text" name "name"
value "{{ $product->

            </div>

        </div>

    </div>

    <div class "col-xs-12 col-sm-12 col-md-12">

```

```

        <div class="form-group">

            <strong>Detail:</strong>

            <textarea class="form-control"
style="height:150px" name="detail"
placeholder="Detail">{{ $product->detail }}</textarea>

        </div>

    </div>

    <div class="col-xs-12 col-sm-12 col-md-12
text-center">

        <button type="submit" class="btn btn-
primary">Submit</button>

    </div>

</div>

</form>

```

resources/views/products/show.blade.php

```

@extends('products.layout')

@section('content')

    <div class="row">

        <div class "col-lg-12 margin-tb">

            <div class "pull-left">

                <h2>                </h2>

```

```

        </div>

        <div class="pull-right">

            <a class="btn btn-primary" href="{{
route('products.index') }}"> Back</a>

        </div>

    </div>

</div>

```

```

<div class="row">

    <div class="col-xs-12 col-sm-12 col-md-12">

        <div class="form-group">

            <strong>Name:</strong>

```

```

        </div>

    </div>

    <div class="col-xs-12 col-sm-12 col-md-12">

        <div class="form-group">

            <strong>          </strong>

        </div>

    </div>

</div>

```



```
@endsection
```

Run Laravel App:

All the required steps have been done, now you have to type the given below command and hit enter to run the Laravel app:

```
php artisan serve
```

Now, Go to your web browser, type the given URL and view the app output:

```
http://localhost:8000/products
```

You will see layout as like below:

List Page:

Laravel 10 CRUD Example from scratch

Create New Product

No	Name	Details	Action
1	Laravel 10 Multiple Image Upload Example	Laravel 10 Multiple Image Upload Example Tutorial	Show Edit Delete
2	Laravel 10 Multiple File Upload Example	Laravel 10 Multiple File Upload Example	Show Edit Delete
3	Laravel 10 File Upload Example	Laravel 10 File Upload Example Tutorial	Show Edit Delete
4	Laravel 10 Create PDF File Example	Laravel 10 Create PDF File Example Tutorial	Show Edit Delete
5	Laravel 10 Pagination Example	Laravel 10 Pagination Example Tutorial	Show Edit Delete

1 2 >

Add Page:

Add New Product

[Back](#)

Name:

Detail:

[Submit](#)

Edit Page:

Edit Product

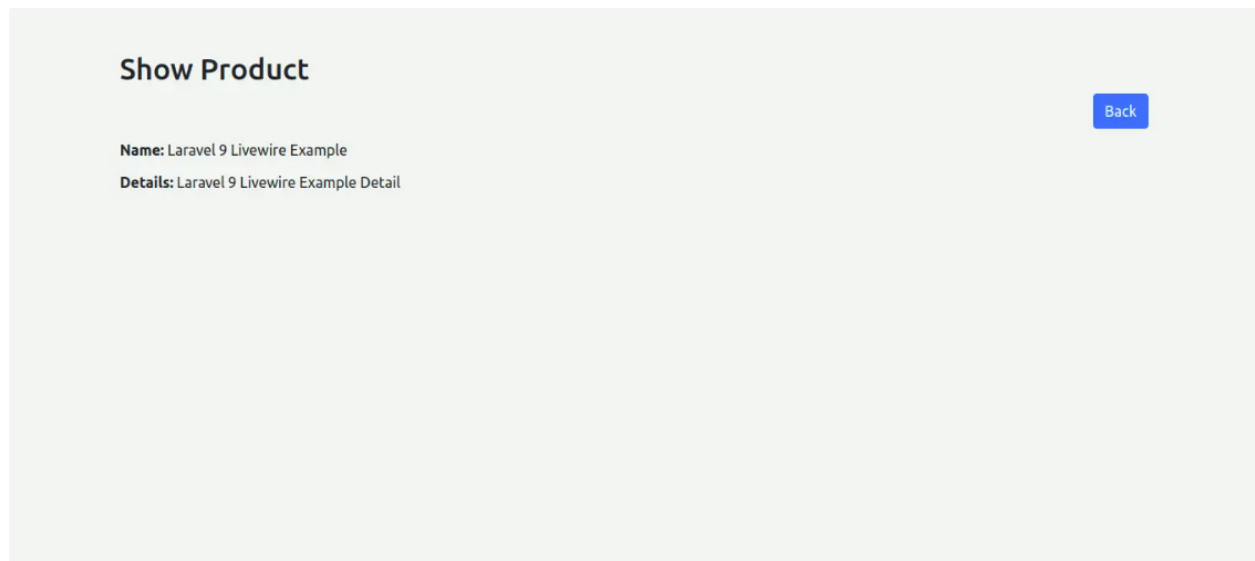
[Back](#)

Name:

Detail:

[Submit](#)

Show Page:



LAB TASKS:

1. Install Laravel:

Install Laravel on your local machine using Composer and create a new Laravel project. Verify that the installation is successful by accessing the default Laravel home page.

2. Routing:

Create a new route in the web.php file that points to a new controller method. Test the route by accessing it in the browser and verify that the correct response is returned.

3. Controllers:

Create a new controller that returns a view with some basic HTML. Test the controller by accessing the URL associated with the controller and verifying that the correct HTML is displayed.

4. Models:

Create a new model that represents a database table. Use Laravel's built-in migration feature to create the table in the database. Test the model by creating new instances of the model and saving them to the database.

5. Views:

Create a new view that displays data from the database. Use the model you created in step 4 to retrieve data from the database and pass it to the view. Test the view by accessing the URL associated with the view and verifying that the data is displayed correctly.

6. Forms:

Create a new form that allows users to submit data to the database. Use Laravel's built-in form validation features to validate the data before saving it to the database. Test the form by submitting valid and invalid data and verifying that the correct response is returned.

7. Authentication:

Implement user authentication using Laravel's built-in authentication features. Create a login page and a registration page, and test the authentication system by logging in and out of the application.