

Name: Preeti Shrestha

University Id: c7261178

Level 6

Advanced Web Engineering (Component 2)

# Episode One: MVC Architecture

M: Model

V: View

C: Controller

When we setup laravel to our desktop, we can see several files and folders on the project. After creating the laravel project inside the htdocs now we can see where the MVC lies. Thus I created the project with the name “myProject”. You can get the directory location of Model, View, Controller as follow:

Model : Xampp/htdocs/myProject/app/Models/user.php

View: Xampp/htdocs/myProject/resources/views/web.php

Controller: Xampp/htdocs/myProject/app/Http/Controllers/controllers.php

When user request the browser, route file loads the necessary controller corresponds to the URI/URL. Then, controller loads all the necessary info and generate the view/html structure to the user.

## Episode Two: Initial Environment Setup and Composer

We need to install

* PHP
* Xampp
* Xdebug
* IDE of your choice (Vs Code)
* Composer

Composer is the dependency manager of php. Since laravel has so many extensions and that is impractical to install all of them so composer does that for us. It installs all the third party extension and dependencies.

# Episode Three: The laravel installer tool

To create laravel project we can run the following command on command prompt.

1. composer create-project laravel/laravel projectName

Or,

1. composer global require ‘laravel/installer’

laravel new projectName

If we run the first command, in this case we need to call dependency again and again. But if we run the second command, we just need to call dependency one time and it will work for the rest.

To run the laravel server first of all we need to go to project directory.

* cd/
* cd xampp/htdocs/myProject
* php artisan serve [This command will run the laravel server]

To stop the server, run ctrl + c.

# Episode Four: Why do we use tools?

We use and learn tools cause they help us to accomplish something.

# Episode Five: How a route load a view

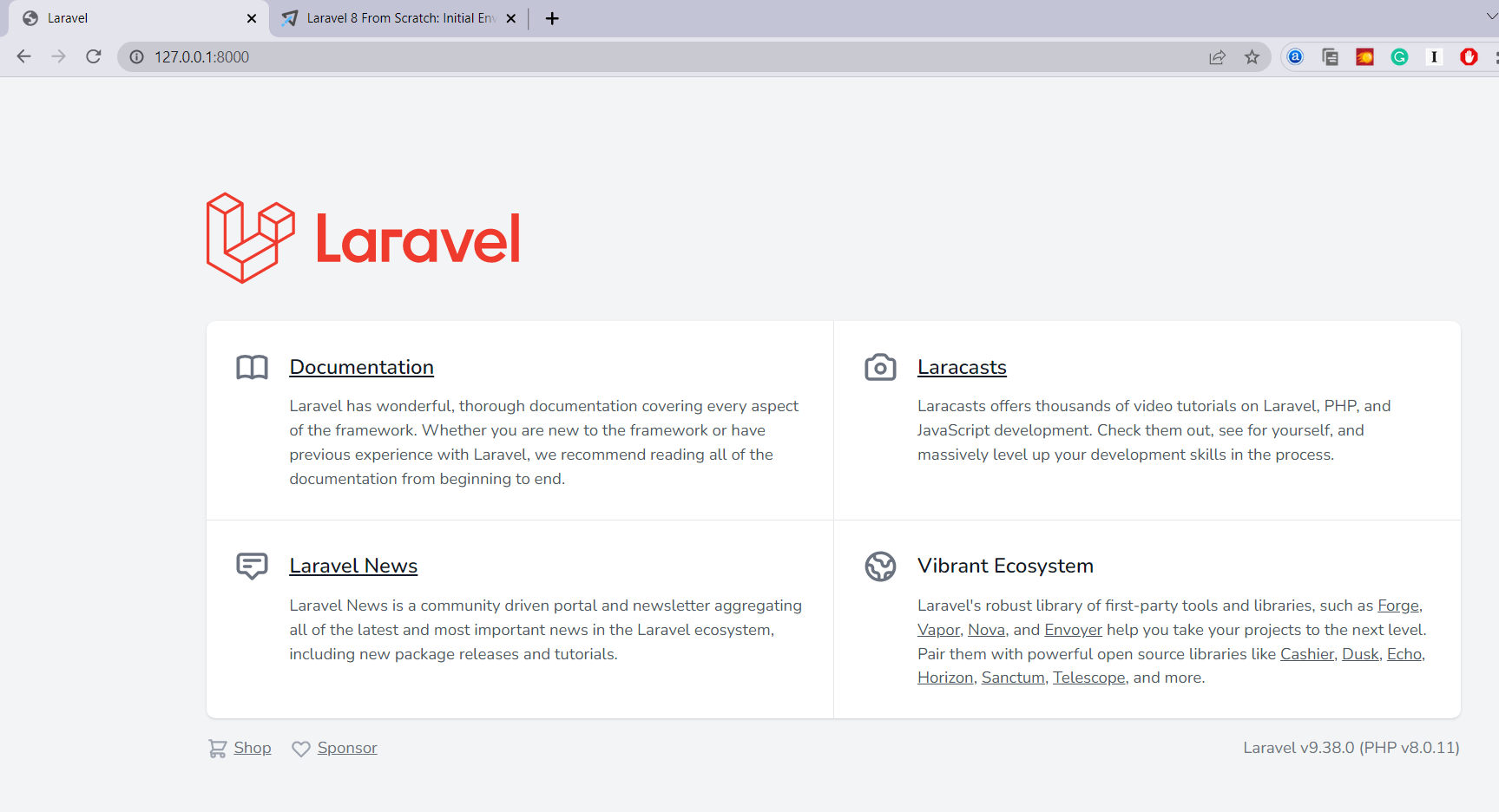
Route directory location : xampp/htdocs/route/web.php

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

   return view('welcome');

});

It will renders the page welcome.blade.php



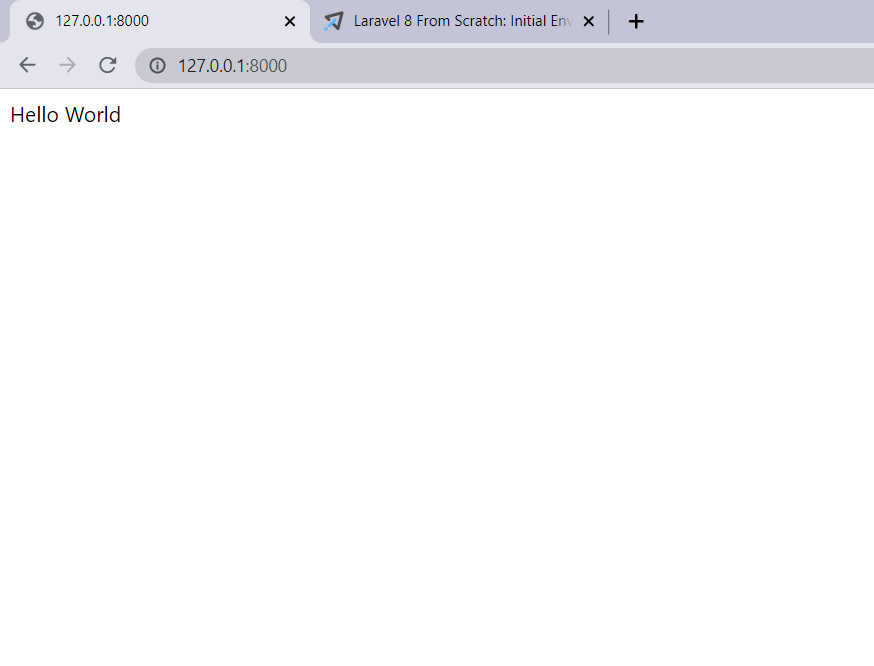
Route directory location : xampp/htdocs/route/web.php

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

   return ('Hello World');

});

This will only return string Hello World.



/ : Path (URI/URL)

Get: HTTP Request

(): Argument

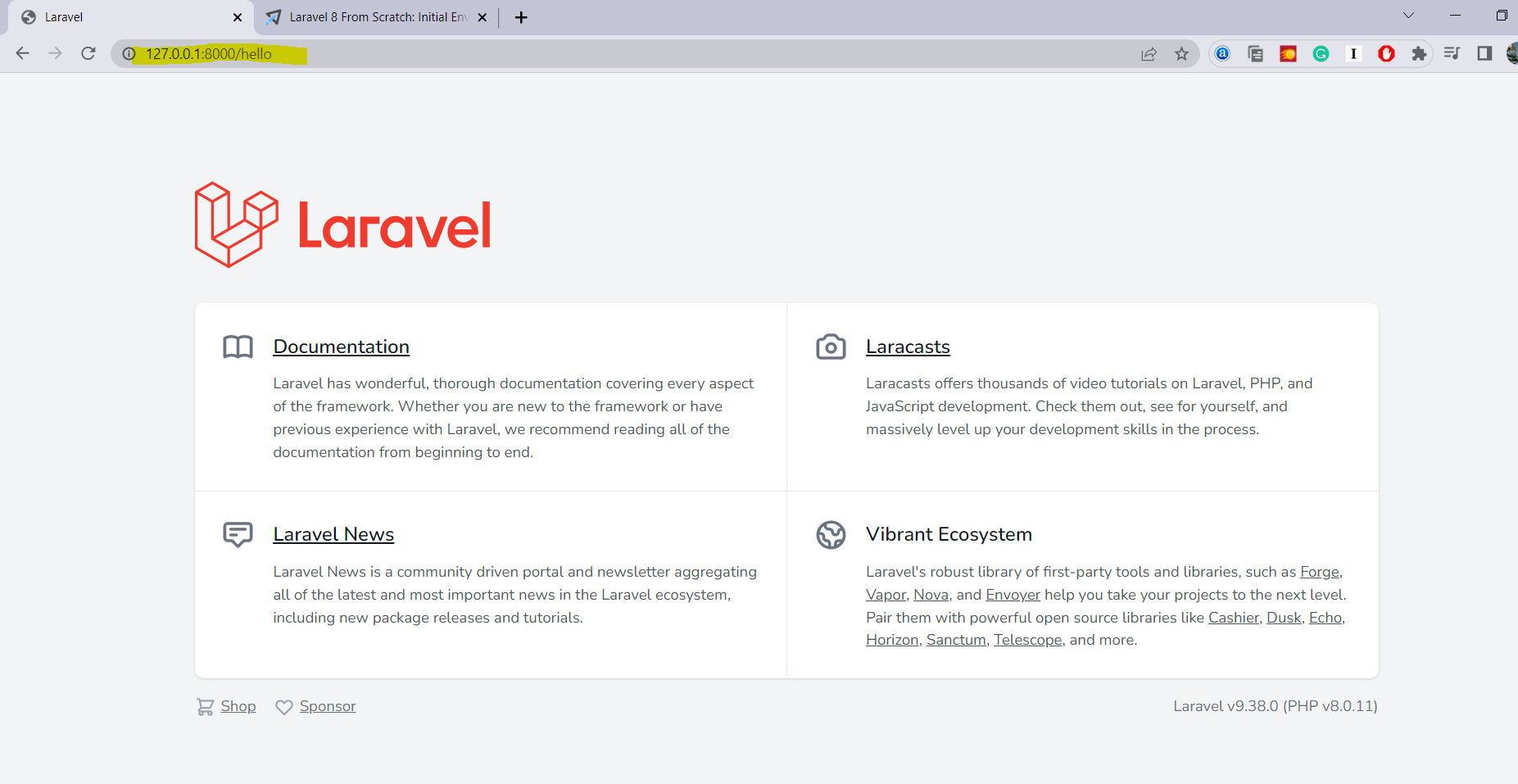
Developer got to decided where to run/route.

Route::get('/hello', function () {

   return view('welcome');

});

Then, we also need to pass /hello in the URL to get the result welcome.blade.php



# Episode Six: Include CSS and JavaScript

We can find Css and Js folder in resources folder. But if we link the CSS and JS to the resources folder we need to put the absolute path which would be quite lengthy. So, if we create it in public folder, it will be easy to navigate the path and also we can create custom code.

# Episode Seven: Make a route and link to it

In this episode we can see creating blog page with excerpt.

Excerpt: It is the paragraph shown in the websites which is not the actual content of the home page but if we click to the read more option it will redirect to its actual page.

In this episode, we changes the welcome.blade.php file to posts.blade.php file.

Replace the code with the blog normal lorem paragraph “My first post”

Since we surely have more than one blog, so made second and third blog.

And linked with “/post” and create a post.blade.php where only my first post is kept and go back option.

This will only show up for the first post but won’t work with second and third. It will show the first post even if we click to second and third. To resolve this we need to get to next episode.

Route::get('/post', function () {

   return view('post');

});

# Episode 8: Store blog posts as Html file.

Instead of making it hard coded we need to make it dynamic as we can go to every blog page.

To do that now we need to change the code in post.blade.php

Now we erase the first blog post with the variable <? $post; ?>

And then we need to pass the post variable through view to get the result.

Route::get('post', function() {

      return view('post',[

'post' => ‘<h1>Hello World</h1>’

]);

   });

With this function it will return the variable value hello world, but we need to fetch the each blog so for that we need to work more on it and pass the slug and other function to get the post.

Then we create the posts folder with html file for each post like my-first-post.html…

Route::get('posts/{post}', function($slug) {

   $path = \_\_DIR\_\_ ."/../resources/posts/{$slug}.html";

   if(! file\_exists($path)){

      // dd("File doesnot exist.");

      // abort(404);

      return redirect('/');

   }

   $post = file\_get\_contents($path);

   return view('post',[

      'post' => $post

   ]);

});

First post = URL/WILDCARD

Second post = Name of the blade file

Third post = variable

## Episode 9: Route Wildcard Constraints

Used at the end of the code

Route::get('posts/{post}', function($slug) {

   $path = \_\_DIR\_\_ ."/../resources/posts/{$slug}.html";

      //ddd($path);

   if(! file\_exists($path)){

      // dd("File doesnot exist.");

      // abort(404);

      return redirect('/');

   }

   $post = file\_get\_contents($path);

   return view('post',[

      'post' => $post

   ]);

}) -> where('post', '[A-z\_\-]+');

# Episode 10: Using Caching for expensive operation

Traditional method

Route::get('posts/{post}', function($slug) {

   $path = \_\_DIR\_\_ ."/../resources/posts/{$slug}.html";

      //ddd($path);

   if(! file\_exists($path)){

      // dd("File doesnot exist.");

      // abort(404);

      return redirect('/');

   }

   $post = cache()->remember("posts.{$slug}", 3600, function () use ($path){

      vardump('file\_get\_contents');

      return file\_get\_contents($path);

      // $post = file\_get\_contents($path);

   });

   return view('post',[

      'post' => $post

   ]);

}) -> where('post', '[A-z\_\-]+');

Shortly,

Route::get('posts/{post}', function($slug) {

   $path = \_\_DIR\_\_ ."/../resources/posts/{$slug}.html";

   if(! file\_exists($path)){

      return redirect('/');

   }

   $post = cache()->remember("posts.{$slug}", 3600, fn() =>file\_get\_contents($path));

   return view('post',[

      'post' => $post

   ]);

}) -> where('post', '[A-z\_\-]+');

# Episode 10: Use File System Class to read a directory

Post.php in Models

>Created class named Post and used caching here for making code look short and reuse it easily for other function as well

<?php

namespace App\Models;

use Illuminate\Support\Facades\File;

class Post{

    public static function all()

    {

        return File::files(resource\_path("posts/"));

        //file fascade : class that gives you static access to all sort of functionality

    }

    public static function find ($slug)

   {

    if(! file\_exists($path = resource\_path("posts/{$slug}.html"))){

        // throw new ModelNotFoundException();

        Abort (404);

     }

     return cache()->remember("posts.{$slug}", 3600, fn() =>file\_get\_contents($path));

    }

}

Web.php

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

       $posts = Post::all();

        ddd($posts);

         return view('posts',[

        'posts' => $posts

       ]);

});

Route::get('posts/{post}', function($slug) {

        //Find a post by its slug and pass it to a view called "post"

        return view('post',[

         'post' => Post::find($slug)

        ]);

}) -> where('post', '[A-z\_\-]+');

class Post{

    public static function all()

    {

        $files = File::files(resource\_path("posts/"));

        //file fascade : class that gives you static access to all sort of functionality

        array\_map(function($file){

        return array\_map(fn($file) => $file -> getContents(), $files);

        }, $files);

    }

# Episode 12: Find a composer package for a meta data.

A to the point front matter parser. Front matter is metadata written in yaml, located at the top of a file wrapped in ---'s.

Remember: Only three --- for the metadata with paragraph tag content

[---

title: Example

---

Lorem ipsum.]

Use Spatie\YamlFrontMatter\YamlFrontMatter;

[$object = YamlFrontMatter::parse(file\_get\_contents(\_\_DIR\_\_.'/example.md'));

$object->matter('title'); // => 'Example';

$object->body(); // => 'Lorem ipsum.'

// Or retrieve front matter with a property call...

$object->title; // => 'Example';]

## >Created constructor

Switch to Collection

public static function all()

    {

      //Collection is the helper function called collect that will collect an array and wrap it with collection object

      //file fascade : class that gives you static access to all sort of functionality

      collect (File::files(resource\_path("posts"))

        )

        ->map(fn($file) => YamlFrontMatter::parseFile($file))

        ->map(fn($document) => new Post(

                          $document->title,

                          $document->excerpt,

                          $document->date,

                          $document->body(),

                          $document->slug

        ));

    }

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

         return view('posts',[

         'posts' =>  Post::all()

         ]);

});

<?php

namespace App\Models;

use App\Models\Post;

use Illuminate\Support\Facades\File;

use Spatie\YamlFrontMatter\YamlFrontMatter;

use Illuminate\Database\Eloquent\ModelNotFoundException;

class Post{

    public $title; // My First Post => my-first-post

    public $excerpt;

    public $date;

    public $body;

    public $slug;

    public function \_\_construct($title, $excerpt, $date, $body, $slug)

    {

        $this->title = $title;

        $this->excerpt = $excerpt;

        $this->date = $date;

        $this->body = $body;

        $this->slug = $slug;

    }

    public static function all()

    {

      //Collection is the helper function called collect that will collect an array and wrap it with collection object

      //file fascade : class that gives you static access to all sort of functionality

      collect (File::files(resource\_path("posts"))

        )

        ->map(fn($file) => YamlFrontMatter::parseFile($file))

        ->map(fn($document) => new Post(

                          $document->title,

                          $document->excerpt,

                          $document->date,

                          $document->body(),

                          $document->slug

        ));

    }

    public static function find ($slug)

   {

   return static::all()->firstWhere('slug', $slug);

   }

}

<?php

namespace App\Models;

use Illuminate\Support\Facades\File;

use Spatie\YamlFrontMatter\YamlFrontMatter;

use Illuminate\Database\Eloquent\ModelNotFoundException;

class Post{

    public $title; // My First Post => my-first-post

    public $excerpt;

    public $date;

    public $body;

    public $slug;

    public function \_\_construct($title, $excerpt, $date, $body, $slug)

    {

        $this->title = $title;

        $this->excerpt = $excerpt;

        $this->date = $date;

        $this->body = $body;

        $this->slug = $slug;

    }

    public static function all()

    {

      //Collection is the helper function called collect that will collect an array and wrap it with collection object

      //file fascade : class that gives you static access to all sort of functionality

      collect (File::files(resource\_path("posts"))

        )

        ->map(fn($file) => YamlFrontMatter::parseFile($file))

        ->map(fn($document) => new Post(

                          $document->title,

                          $document->excerpt,

                          $document->date,

                          $document->body(),

                          $document->slug

        ));

    }

    public static function find ($slug)

   {

    if(! file\_exists($path = resource\_path("posts/{$slug}.html"))){

        throw new ModelNotFoundException();

        Abort (404);

     }

     return cache()->remember("posts.{$slug}", 3600, fn() =>file\_get\_contents($path));

    }

}

# Episode 14: Blade The absolute Basics

he reason why we use "filename.blade.php" , the "blade.php" at the end of the file is used to make laravel know that it's a

blade file and blade files are the laravel's templating engine hence, if the file name consists of blade.php suffix,

laravel internally knows that it needs to compile the file to vanilla.php.

storage->logs->laravel.log for any errors

storage-> framework-> views [here are the compiled vanilla php verions of the blade files we created]

However, if there are any html tags, we have to use {{!! !!}} so as to escape. Escaping so that <p></p> tag does its

work instead of just outputting as <p></p>

Use blade directives for more efficient coding.

Research about blade directives [each blade directive has a corresponding php code, as blad directives are just

simpler way to write the php ]

Eg: @foreach, @dd,

# Episode 15: Blade Layouts Two ways

There are two ways to create blade layouts:

1. Bottom-up approach

Take only the wrapping part (common part) and keep it in "layout.blade.php" and @yield('content')

Then in other smaller pages, we can then use @extends('layout') and @section('content')

[Note: one way is not better than the other, they are just differnt]

2. Top-down approach

Using blade components to wrap a html component

Create a "components" folder inside views and put layout.blade.php there and then use {{$content}}

After that in other smaller pages (i.e blade files), we can just use

<x-layout>

<x-slot name="content">

//here you can write the codes for individual pages

</x-slot>

</x-layout>

but instead of "content", we can use "slot"

$slot is the default slot for our component

Hence in components-> layout.blade.php-> write $slot

Then, in other smaller blade files, you can use :

<x-layout>

//here u can write the codes for individual pages

</x-layout>

Posts.blade.php

<!-- Blade Layout

@extends ('layout')

@section('banner')

<h1>My Blog</h1>

@section('content')

@foreach ($posts as $post)

        <article>

          <h1>

            <a href="/posts/{{$post->slug}} ">

             {{ $post->title }}

            </a>

          </h1>

          <div>

          {{ $post->excerpt }}

          </div>

        </article>

        @endforeach

@endsection -->

Post.blade.php

Blade Layout

@extends ('layout')

@section('content')

    <article>

    SEE THE ABOVE ONE I REMOVE CAUSE COMMENTING DID NOT WORK

    </article>

    <a href="/">Go Back</a>

@endsection

POST.BLADE.PHP

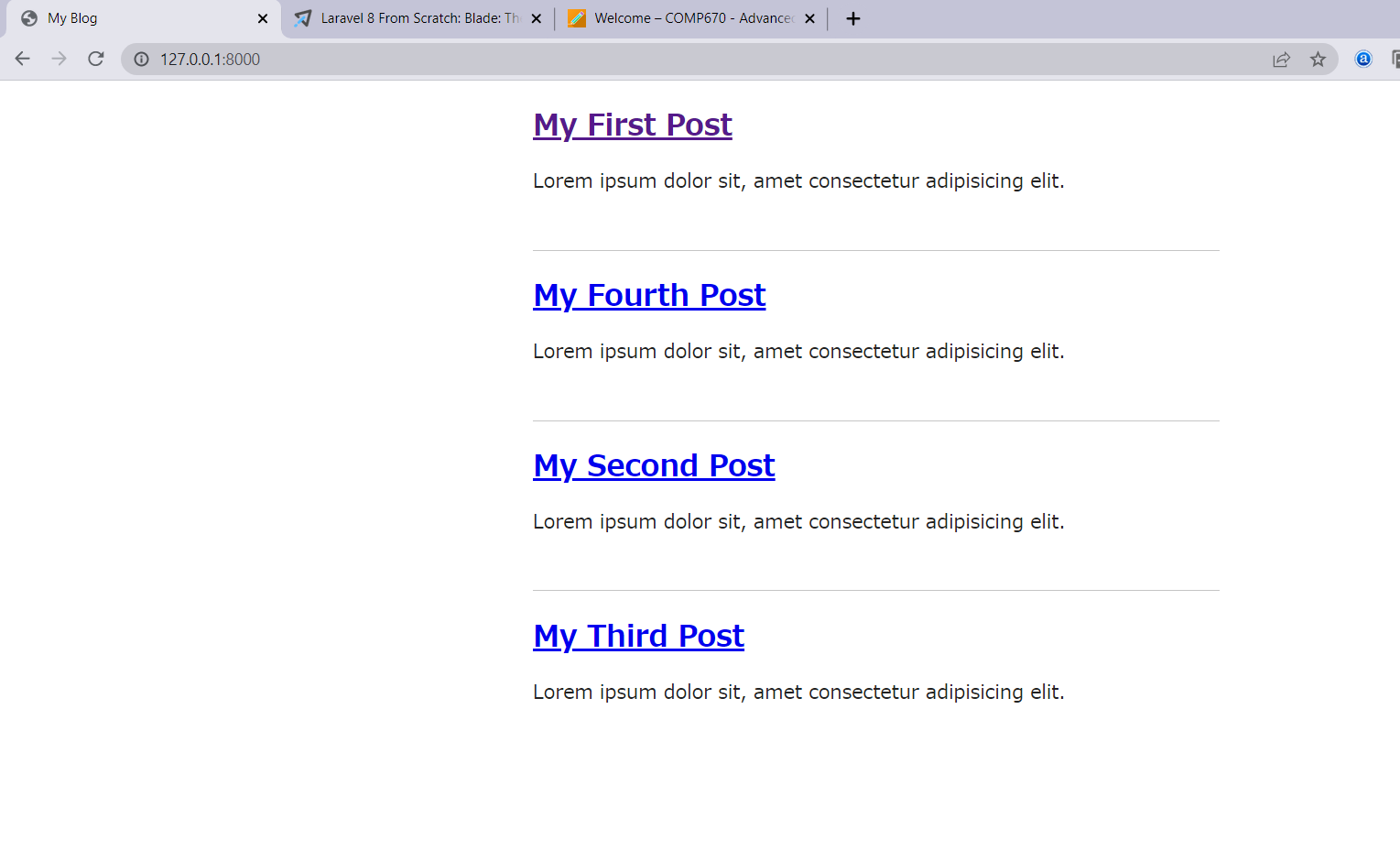
<h1><?= $post ->title; ?></h1>

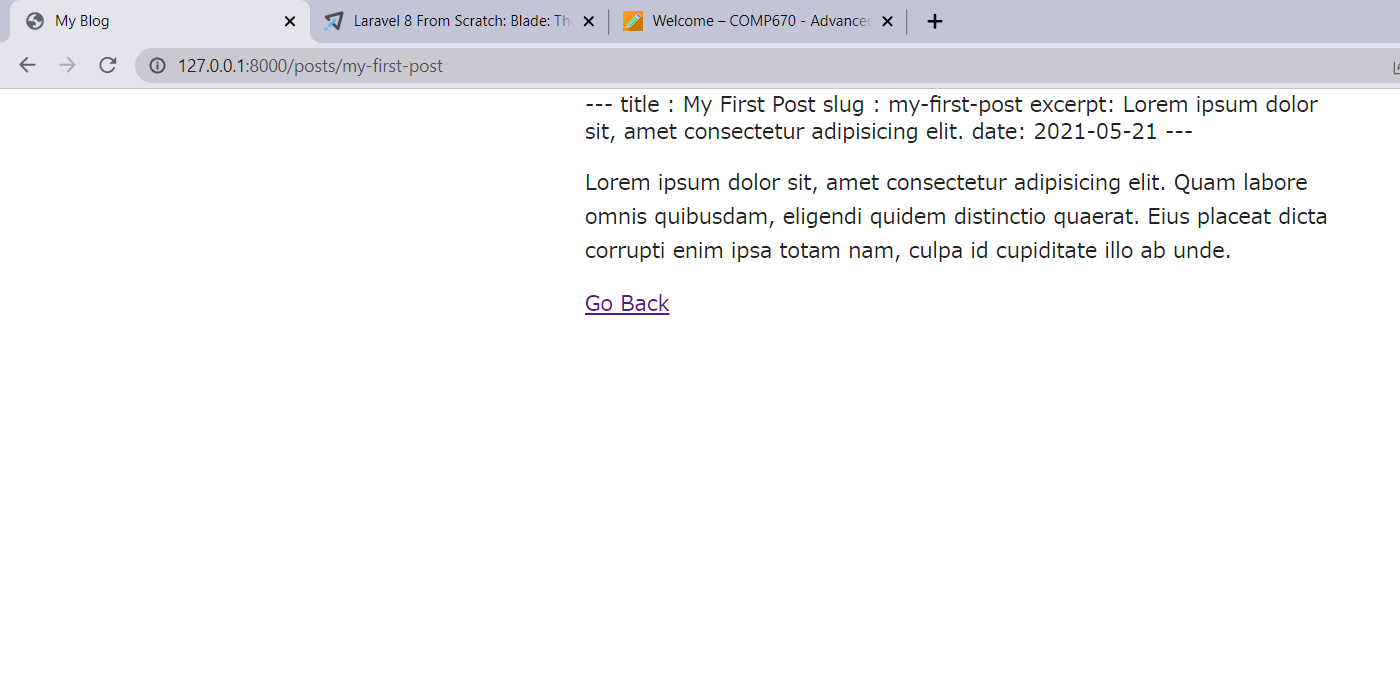
      <div>

      <?= $post ->body; ?>

      </div>

    </article>





Note: There is mistake in collection part of code so it d­­oes not let particular post to render with title and content only.