Laravel IMP

1. How to create a route in Laravel:
2. Syntax:

1 method🡺

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

    return view('page name');

});

1. Syntax:

2 method🡺

Route::view( '/url', 'page name' );

1. Now Example of route:

//home route

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

    return view('welcome');

});

//about route

Route::view("about","about");

1. What if we create same route url with different view name:

//home route

Route::view('/',"welcome");

//contact route

Route::view('/',"contact");

**Fact: It will consider the last one (Updated one) with program flow**

1. How to create a router’s like one page pointing to other and so on… :

//home route

Route::view('/',"welcome");

//contact route

Route::view('/contact',"contact");

//about route

Route::view("/contact/about","about");

**Output:**

****

****

1. How to send data with route(including in url itself):
2. Simply echoing on the top of page itself:

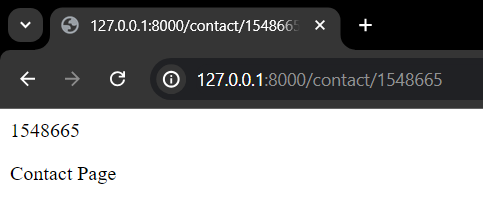
//how to send data throw route

Route::get("/contact/{contact\_number}", function($contact\_number){

    echo($contact\_number);

    return view("contact");

});



1. Now how we can provide it in blade.php page

For printing in php file we are using {{ }} braces with variable name

Syntax: {{$variable\_name}} ==> it will print date in variable

//how to send data throw route

Route::get("/contact/{contact\_number}", function($contact\_number){

    return view("contact",['ph\_no'=>$contact\_number]);

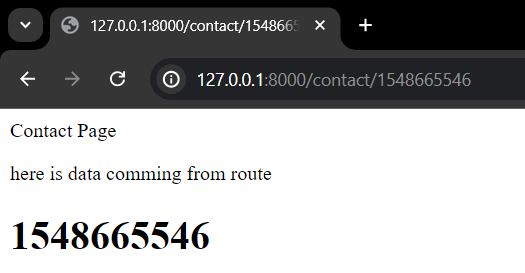
});

//contact.blade.php==>>

<p>Contact Page</p>

<p>here is data comming from route</p>

<h1>{{$ph\_no}}</h1>



1. Now how to navigate in browser without changing url (using Ancher tag):

//Web.php =>

//home route

Route::view('/',"welcome");

//contact route

Route::view('/contact', "contact");

//about route

Route::view("/about","about");

//welcome.blade.php=>

<p> Welcome Page </p>

<a href="/about">About Page</a></br>

<a href="contact">Contact Page</a>

1. How to redirect to some page:

Simply changing home page

Syntax:

Here we are passing page name not url

**redirect(‘blade\_name’);**

If user try to add this url: <http://127.0.0.1:8000/>

Then also automatically this url will open: http://127.0.0.1:8000/contact

//home route

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

    // return view('welcome');

    //how to redirecting to some other page

    return redirect('contact');

});

1. How to create Controller in Laravel
2. Using command:

Syntax:

**php artisan make:controller controller\_name**

after this controller automatically created at place **/app/Http/Controllers/here**

ex:

Laravel/first-project (main)

$ php artisan make:controller user

INFO Controller [Laravel\first-project\app\Http\Controllers\user.php] created successfully.

1. Code inside controller looks like:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class user extends Controller

{

    //code

}

?>

1. How to create function in controller

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class user extends Controller

{

    //function in controller

    function printing(){

        echo "This is controller echoing";

    }

}

?>

1. How to call controller:

To call controller we need to write code in web.php

1. Import controller class
2. Create route for that with syntax:

Route::get(“url”, [ControllerClassName::class, “function\_name”] );

//web.php

//importing controller

use App\Http\Controllers\user;

//calling controller after laravel 8

/\*

syntax:

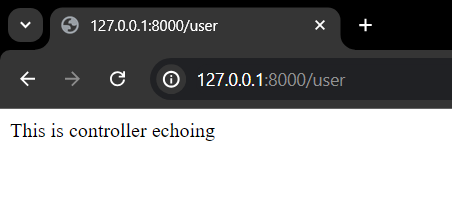
//importing controller

use App\Http\Controllers\user;

Route::get('url', [ControllerClassName::class,"FunctionWhichYouWantToCall"] );

\*/

Route::get("user",[user::class,"printing"]);



1. How does controller call before Laravel 8:
2. By creating route only like this

/\*

syntax was:

Route::get("url","ControllerClassName@FunctionName");

\*/

Route::get("laravel7","user@printing");

Why it was changed?

* **This change enhances PHP type safety features and provide better support for IDEs to offer code completion and type hinting**
* **So, this change was made un Laravel 8 to aim to provide modern and robust development experience**

1. How to send data from controller url
2. Accept variable and print it using echo

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class user extends Controller

{

    //function in controller

    function printing($comming){

        echo "$comming This is controller echoing";

    }

}

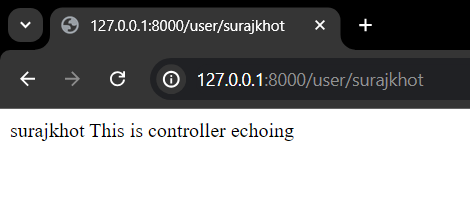
?>

1. Make change in web.php to accept data

//web.php

//passing data from url

Route::get("user/{username}",[user::class,"printing"]);



1. How to call a view from controller:
2. Create it by using cli:

**$ php artisan make:controller DemoController**

INFO Controller [D:\IMPMyWorkplace\Laravel\first-project\app\Http\Controllers\DemoController.php] created successfully.

1. Made Change in controller file:

//DemoController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class DemoController extends Controller

{

    //calling view from controller

    function loadView(){

        //simply returning about view

        return view('about');

    }

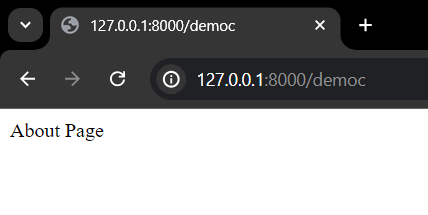
}

1. Make route for it

//Web.php

//democontroller route

Route::get("democ",[DemoController::class,"loadView"]);



1. How to pass data from controller to view:
2. demoController.php

// DemoController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class DemoController extends Controller

{

    //calling view from controller

    function loadView($arg){

        //simply returing about view

        return view('about',['argument'=>$arg]);

    }

}

1. create route with argument

//web.php

//democontroller route

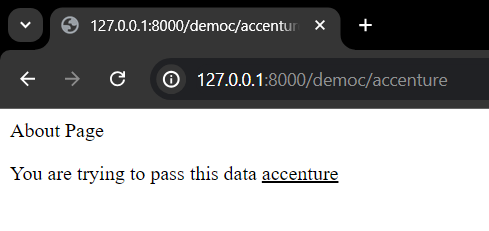
Route::get("democ/{company}",[DemoController::class,"loadView"]);

1. now display in about.blade.php

// about.blade.php

<p> About Page </p>

<p> You are trying to pass this data <u>{{$argument}}</u> </p>



1. how to create component:
2. fire command in root directory

**$ php artisan make:component DemoComponent**

INFO Component [D:\IMPMyWorkplace\Laravel\first-project\ap

p\View\Components\DemoComponent.php] created successfully.

Two files are created after it

- **componentName.php (app/view/components/xxx)**

- **componentName.blade.php**  **(resources/views/component/xxx)**

1. Here code inside component

// first-project\app\View\Components\DemoComponent.php

<?php

namespace App\View\Components;

use Closure;

use Illuminate\Contracts\View\View;

use Illuminate\View\Component;

class DemoComponent extends Component

{

    /\*\*

     \* Create a new component instance.

     \*/

    public function \_\_construct()

    {

        //

    }

    /\*\*

     \* Get the view / contents that represent the component.

     \*/

    public function render(): View|Closure|string

    {

        return view('components.demo-component');

    }

}

// first-project\resources\views\components\demo-component.blade.php

<div>

//code

</div>

1. How to use components

It will allow us to use same code in every file with on line of code

Syntax:

**<x-ComponentName/>**

1. Write code in blade.php

//demo-component.blade.php

<div>

    <p> This is code from component and this file is in resources/views/components/ directory </p>

</div>

1. Using in our views

Contact.blade.php

<p>Contact Page</p>

<x-DemoComponent/>

Welcome.blade.php

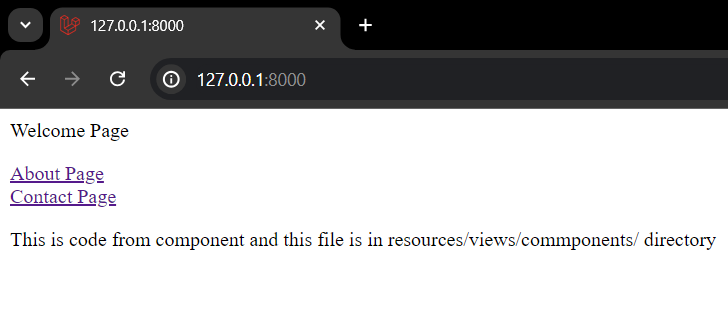
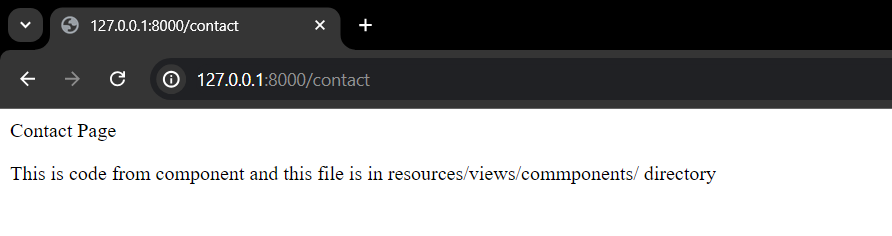
<p> Welcome Page </p>

<a href="/about">About Page</a></br>

<a href="contact">Contact Page</a>

<x-DemoComponent/>

1. And do not forget to create routes for those views and Outputs:



1. How to pass data through component
2. Create variable (use public) and use this keyword to refer to argument coming from \_\_construct

**Syntax:**

//define

public $variableName;

    public function \_\_construct($argument)

    {

       $this->variableName=$argument;

    }

// DemoComponent.php

<?php

namespace App\View\Components;

use Closure;

use Illuminate\Contracts\View\View;

use Illuminate\View\Component;

class DemoComponent extends Component

{

    /\*\*

     \* Create a new component instance.

     \*/

    //publicly defining variable

    public $title;

    public function \_\_construct($arg)

    {

        //Refers to the current object and for further use use variable \*\*\*\*title\*\*\*\*\* not arg

       $this->title=$arg;

    }

    /\*\*

     \* Get the view / contents that represent the component.

     \*/

    public function render(): View|Closure|string

    {

        return view('components.demo-component');

    }

}

1. Now print data in components blade file by using variable name not an argument

//demo-component.blade.php

<div>

//printing here a data title

    {{$title}}

    <p> This is code from component and this file is in resources/views/components/directory </p>

</div>

1. Now pass data using argument (use without $ sign)

And use the correct parameter name

Contact.blade.php

<p>Contact Page</p>

<x-DemoComponent arg="This is argument passing here in Contact Page"/>

Welcome.blade.php

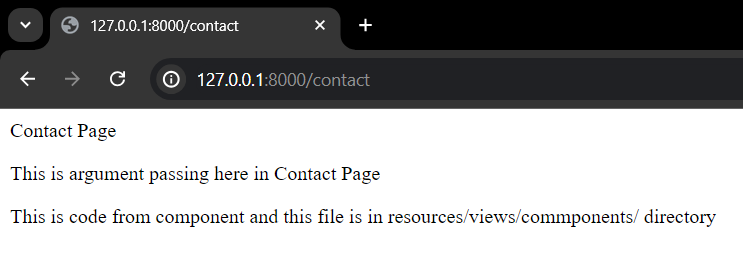
<p> Welcome Page </p>

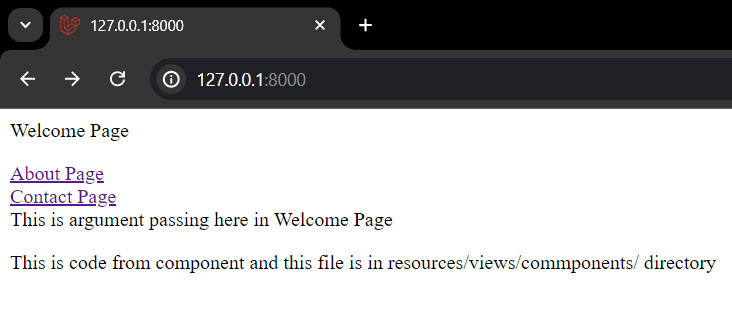
<a href="/about">About Page</a></br>

<a href="contact">Contact Page</a>

{{-- do not include $ here with variable --}}

<x-DemoComponent arg="This is argument passing here in Welcome Page"/>





1. Now It’s time to learn blade template engine:
2. How to comment in blade

Syntax:

**{{-- commenting --}}**

Same for single line and multiline

Ex:

{{-- this is comment in blade --}}

1. How we use php in blade template

We cannot directly add php in blade we have to use such thing like php tags and for printing {{ }} braces like this. 🡺

**For printing php in blade**

1. Double braces

{{ //code }}

1. Php tags

<?php

//code

?>

1. Php attribute tags

@php

//code

@endphp

Ex:

//welcome.blade.php

{{-- 1. operations --}}

10+2  {{--  not work --}}

{{10+2}}  {{-- 12 --}}

@php

echo 10+2; //12

@endphp

<?php

echo 10+2; //12

?>

1. If conditions:
2. Simply using blade format

{{-- if condition --}}

<?php

$a = 10;

$b = 20;

if ($a > $b) {

    echo "$a>$b";

} elseif ($b > $a) {

    echo "$b>$a";

} else {

    echo "$a==$b";

}

?>

@if ($a > $b)

    {{ "$a>$b" }}

@elseif ($b > $a)

    {{ "$b>$a" }}

@else

    {{ "$a==$b" }}

@endif

1. Using controller
2. Create a controller

$ php artisan make:controller PhpExampleController

1. Pass data and load view

//PhpExampleController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PhpExampleController extends Controller

{

    function loadView()

    {

        $a = 10;

        $b = 20;

        $arr = [12, 45, 65, 74, 68, 12];

        return view('php\_code', ['a' => $a, 'b' => $b, 'arr' => $arr]);

    }

}

1. Create route:

//php code controller

Route::get("php", [PhpExampleController::class, "loadView"]);

1. Code for blade

//php\_code.blade.php

{{-- data comming from controller --}}

<h1>data comming from controller</h1>

{{ "a:$a \n b:$b \n array:arr" }}

<br>

{{-- if condition  --}}

@if ($a > $b)

    {{ "$a is greater than $b" }}

@elseif($b > $a)

    {{ "$a is leass than $b " }}

@endif

<br>

<h1>length of array</h1>

{{ count($arr) }}

<br>

<h1>for loop</h1>

{{-- for loop  --}}

@php

    $len = count($arr);

@endphp

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

    <h5> {{ $arr[$i] }}</h5>

@endfor

<br>

<h1>for each loop</h1>

{{-- for each loop  --}}

@foreach ($arr as $value)

    <h5>{{ $value }}</h5>

@endforeach

1. How to use javascript in blade

Using same controller for it

Syntax:

* tag

**<script>**

**//code**

**</script>**

* for printing

**console.log(code);**

* for initializing

**var variableName=@json($phpVariableName);**

//php\_code.blade.php

<script>

    //coming array

    var x = @json($arr);

    console.log("array:" + a);

    //coming a

    var a = @json($a);

    console.log("a:" + a);

    //coming b

    var b = @json($b);

    console.log("b:" + b);

</script>

1. How to include one view in another

* Syntax:

**@include(‘viewName’)**

Ex:

//blade\_code.blade.php

@include('header')

<h1>This is Body section</h1>

@include('footer')

1. How to create header and footer in blade

* Create footer🡺

//footer.blade.php

<footer class="footer">

    <h1>

        Hi Am a footer

    </h1>

    <h3>

        @copywrite 2024

    </h3>

</footer>

<style>

    .footer{

background-color: burlywood;

position: absolute;

height: 100px;

width: 100%;

text-align: center;

color: black;

padding-bottom: 0%;

    }

</style>

* Create header🡺

//heade.blade.php

<header class="header">

    <h1>

        Hi Am a header

    </h1>

    <h3>

        Learn Laravel

    </h3>

</header>

<style>

    .header{

background-color: burlywood;

height: 100px;

width: 100%;

text-align: center;

color: black

    }

</style>

* Create blade to include it

Syntax:

**@include(‘viewName’)**

//blade\_code.blade.php

@include('header')

<h1>This is Body section</h1>

@include('footer')

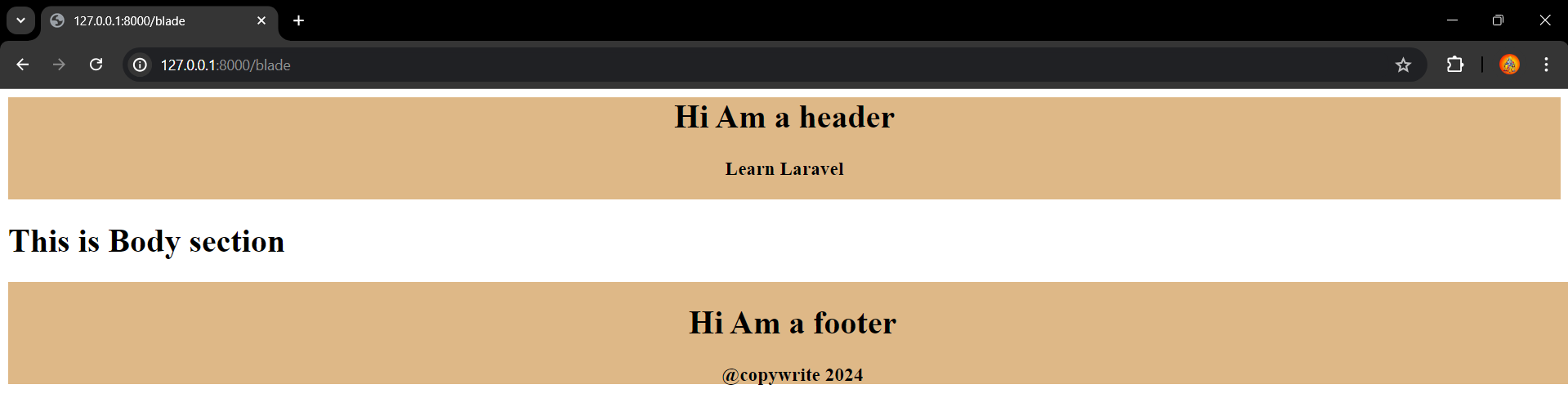
* Create a route

//Web.php

//blade code

Route::view('/blade',"blade\_code");

* Output



1. How to create a html form in laravel
2. Create a page to display form

What to add in form tag

**Action 🡺 Contollers Route urL**

**Method 🡺 POST**

<form action="controller url" method="POST">

**Donot forgot to add csrf(**Cross-Site Request Forgery) otherwise we will get error

//html\_form.blade.php

<h1>Login</h1>

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

    {{-- Cross-Site Request Forgery (CSRF) is a type of malicious exploit where unauthorized commands are performed on behalf of an authenticated user. In a CSRF attack, an attacker tricks an unsuspecting user into performing actions they didn’t intend to execute. For example, an attacker could change an authenticated user’s email address without their knowledge.

Laravel provides built-in CSRF protection to prevent such attacks. Here’s how it works:

CSRF Tokens: Laravel automatically generates a unique CSRF “token” for each active user session managed by the application.

This token is stored in the user’s session and changes each time the session is regenerated.

A malicious application cannot access this token because it’s tied to the user’s session. --}}

    {{-- if we not add csrf here then on submit "page expired" error is comming --}}

    @csrf

    <input type="text" name="username" placeholder="enter name" /> <br><br>

    <input type="password" name="password" placeholder="enter password" /> <br><br>

    <button type="submit">Login</button>

</form>

1. Create a controller for forms action and to fetch and display data

Accept a type of data as **REQUEST** data type

//HtmlFormController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HtmlFormController extends Controller

{

    //function to fetch data and return

    function fetchData(Request $request){

        return $request;

    }

}

1. Create routes for both

And use the url in controller for form tag in html blade

//web.php

//routes for html form

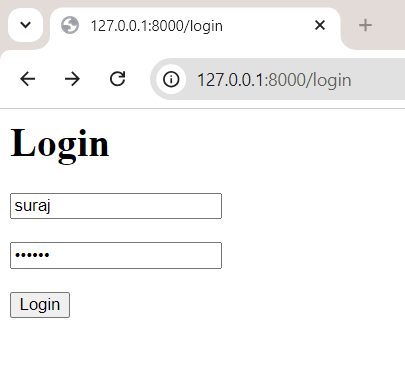
//1. one for controller

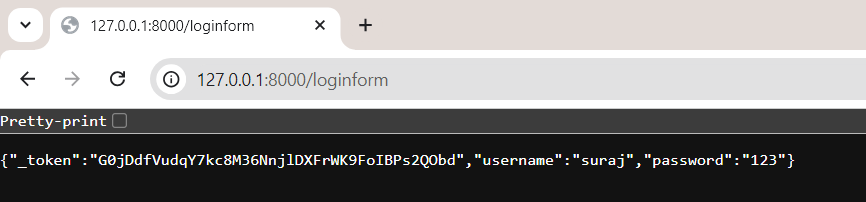
Route::post("loginform", [HtmlFormController::class, "fetchData"]);

//2. one for view

Route::view('login', 'html\_form');

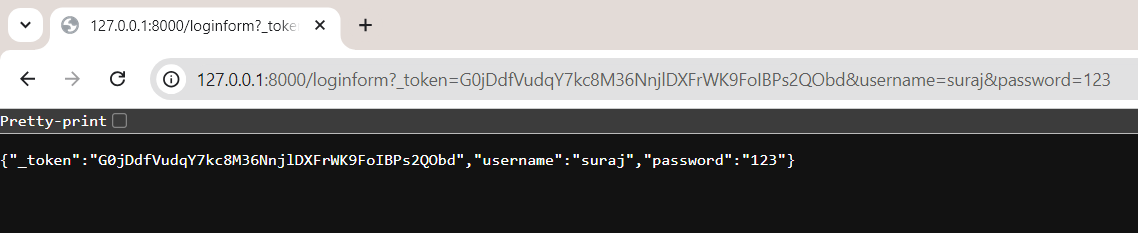
1. Output





1. What if we use **method=Get** and also define route **url type as get**

Then output looks like



1. We created a form but how to validate it:
2. Create a validate function in controller:

Validate🡺 just validtate with any condition

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HtmlFormController extends Controller

{

    //function to fetch data and return

    function fetchData(Request $request)

    {

        $request->validate([

            'username' => 'required',

            'password' => 'required'

        ]);

        return $request;

    }

}

1. But how to display it on blade

Display errors to user

1. We can simplay print it using **$errors**

<h1>Login</h1>

<form action="loginform" method="GET">

    {{-- CSRF means Cross Site Request Forgery --}}

    {{-- if we not add csrf here then on submit "page expired" error is comming --}}

    @csrf

    {{-- to display error  --}}

    {{$errors}}

    <br>

    <input type="text" name="username" placeholder="enter name" /> <br><br>

    <input type="password" name="password" placeholder="enter password" /> <br><br>

    <button type="submit">Login</button>

</form>

Output is looks like 🡺



1. We can simply use for each loop also to display

<h1>Login</h1>

<form action="loginform" method="GET">

    {{-- CSRF means Cross Site Request Forgery --}}

    {{-- if we not add csrf here then on submit "page expired" error is comming --}}

    @csrf

    {{-- to display error  --}}

    {{-- {{$errors}} --}}

    @if ($errors->any())

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

           <li>{{$err}}</li>

        @endforeach

    @endif

    <br>

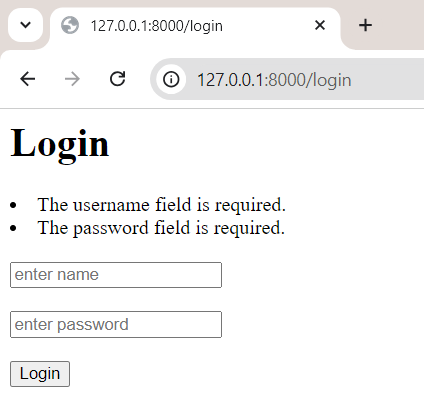
    <input type="text" name="username" placeholder="enter name" /> <br><br>

    <input type="password" name="password" placeholder="enter password" /> <br><br>

    <button type="submit">Login</button>

</form>

Output is like 🡺



1. But how to display it below of each fields

So for that we use span tag below input with

Syntax :

**@error(‘namOfFeild’) {{$message) @enderror**

**Ex:**

<span style="color: red">

        @error('username')

            {{ $message }}

        @enderror

    </span>

<h1>Login</h1>

<form action="loginform" method="GET">

    {{-- CSRF means Cross Site Request Forgery --}}

    {{-- if we not add csrf here then on submit "page expired" error is comming --}}

    @csrf

    <input type="text" name="username" placeholder="enter name" /> <br>

    <span style="color: red">

        @error('username')

            {{ $message }}

        @enderror

    </span>

    <br>

    <input type="password" name="password" placeholder="enter password" /> <br>

     <span style="color: red">

        @error('password')

            {{ $message }}

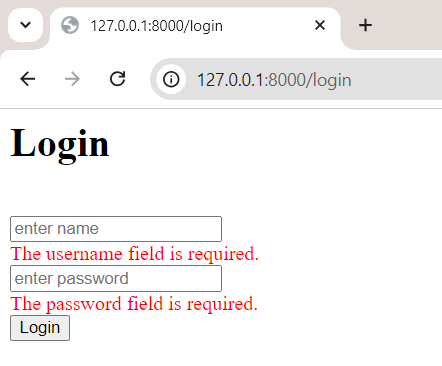
        @enderror

    </span><br>

    <button type="submit">Login</button>

</form>

**Output looks like 🡺**

****

1. But after click on button we just lost our inputed field in form if we get error so how to restore and show the fields if there are errors:

We simply use

Syntax :

**value=”{{ old(‘nameOfFeild’) }}”** in input tag in our blade

ex:

<input type="text" name="username" value="{{old('username')}}" placeholder="enter name" />

But it won’t work with password field because we set it to type=”password”

<h1>Login</h1>

<form action="loginform" method="GET">

    @csrf

<input type="text" name="username" value="{{old('username')}}" placeholder="enter name" /> <br>

    <span style="color: red">

        @error('username')

            {{ $message }}

        @enderror

    </span>

    <br>

    <input type="password" name="password" placeholder="enter password" /> <br>

     <span style="color: red">

        @error('password')

            {{ $message }}

        @enderror

    </span><br>

    <button type="submit">Login</button>

</form>

1. Now every thing is perfect just need to make password must be at least 6 character and username not contain more than 10 characters:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HtmlFormController extends Controller

{

    //function to fetch data and return

    function fetchData(Request $request)

    {

        $request->validate([

            'username' => 'required  | max:10',

            'password' => 'required | min:6'

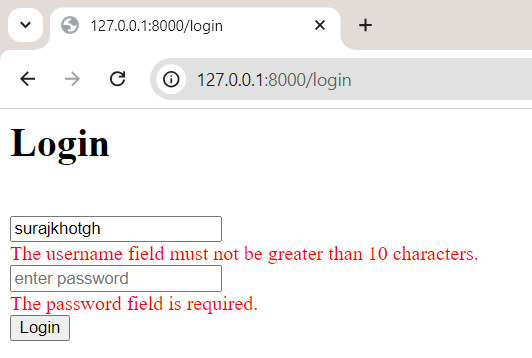
        ]);

        return $request;

    }

}

Output 🡺



1. Let’s dive into middleware

🡺it just handle request and responses between user and server

How to create a middleware in Laravel:

Syntax:

**$Php artisan make:middleware MiddlewareName**

**Ex:**

**$php artisan make:middleware DemoMiddelware**

**INFO Middleware [\app\Http\Middleware\DemoMiddelware.php] created successfully.**

3 types of middleware in Laravel

1. Global middleware :

Are the middleware which we can use through out the project (means it can used in any page)

1. Create middleware:

**$php artisan make:middleware DemoMiddelware**

1. Register it in kernel.php(app/http/)

**Now keep in mind for registration**

**Global middleware 🡺**

**protected $middleware[**

**//register here**

**]**

**Group middleware 🡺**

**protected $middlewareGroups = [**

**//register here**

**]**

**Route middleware 🡺**

**protected $middlewareAliases = [**

**//register here**

**]**

<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel

{

    /\*\*

     \* The application's global HTTP middleware stack.

     \*

     \* These middleware are run during every request to your application.

     \*

     \* @var array<int, class-string|string>

     \*/

    protected $middleware = [

        // here we register our middleware

        \App\Http\Middleware\DemoMiddelware::class,

//more middleware

    ];

//code

//code

}

1. Now we can put any code in middleware:

First check it registered and working fine add echo statement here

// app\Http\Middleware\DemoMiddelware.php

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Symfony\Component\HttpFoundation\Response;

class DemoMiddelware

{

    /\*\*

     \* Handle an incoming request.

     \*

     \* @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next

     \*/

    public function handle(Request $request, Closure $next): Response

    {

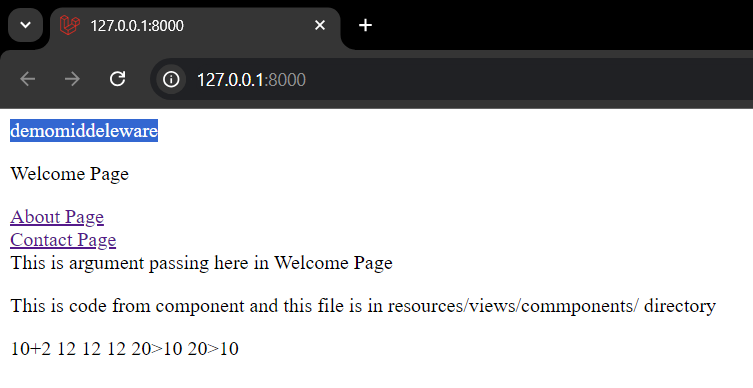
        echo "demomiddeleware";

        return $next($request);

    }

}

Output🡺



1. Task 🡺 check age of coming user

// app\Http\Middleware\DemoMiddelware.php

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Symfony\Component\HttpFoundation\Response;

class DemoMiddelware

{

    /\*\*

     \* Handle an incoming request.

     \*

     \* @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next

     \*/

    public function handle(Request $request, Closure $next): Response

    {

        // echo "demomiddeleware";

        //we get age and check here

        if ($request->age && $request->age < 18) {

            //just returning custom view by redirect

            return redirect('redirect');

        }

        return $next($request);

    }

}

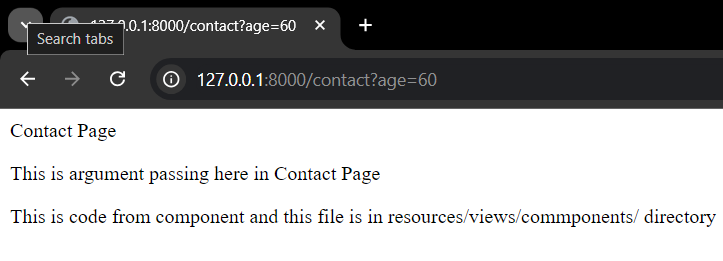
Now while accessing any route add url like

<http://127.0.0.1:8000/contact?age=60>

<http://127.0.0.1:8000/?age=60>

<http://127.0.0.1:8000/home?age=60>

output:



1. Group middleware:

It is used for specific pages not all the pages present In project

1. Create middleware:

**$php artisan make:middleware DemoMiddelware**

1. Register it in kernel.php(app/http/)

**Now keep in mind for registration**

**Global middleware 🡺**

**protected $middleware[**

**//register here**

**]**

**Group middleware 🡺**

**protected $middlewareGroups = [**

**//register here**

**]**

**Route middleware 🡺**

**protected $middlewareAliases = [**

**//register here**

**]**

1. Now register it in kernel.php

<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel

{

    //code

    /\*\*

     \* The application's route middleware groups.

     \*

     \* @var array<string, array<int, class-string|string>>

     \*/

    protected $middlewareGroups = [

//register here

        'demoGroupMiddleware'=>[

            \App\Http\Middleware\DemoMiddelware::class,

        ]

//other middleware

    ];

//code

//code

}

1. Now add code to middleware

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Symfony\Component\HttpFoundation\Response;

class DemoMiddelware

{

    /\*\*

     \* Handle an incoming request.

     \*

     \* @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next

     \*/

    public function handle(Request $request, Closure $next): Response

    {

        // echo "demomiddeleware";

        //we get age and check here

        if ($request->age && $request->age < 18) {

            //just returning custom view by redirect

            return redirect('redirect');

        }

        return $next($request);

    }

}

1. Now create route for it

Syntax:

***Route::group([‘middleware’ => [ ‘registeredMiddlewareName’]] , function(){***

***//define any route you want to add middleware***

***}***

//Web.php

//route for group middleware

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

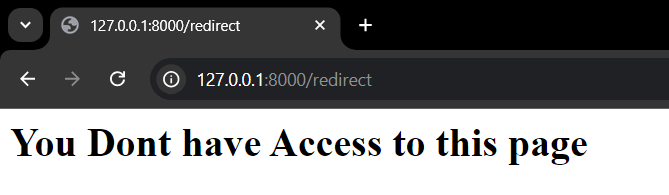
    //add any routes to add middleware to here

    Route::view('demo', 'groupmiddleware');

    Route::view('login', 'html\_form');

});

1. Output:



1. Route middleware:

Means we can simply add a middleware for one route

1. Create route

Same as first two examples

1. Register it in kernel.php

**Now keep in mind for registration**

**Route middleware 🡺**

**protected $middlewareAliases = [**

**//register here**

**]**

**Syntax:**

**‘nameForRegistration’ => path\to\it**

<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel

{

//code

//code

    /\*\*

     \* The application's middleware aliases.

     \*

     \* Aliases may be used instead of class names to conveniently assign middleware to routes and groups.

     \*

     \* @var array<string, class-string|string>

     \*/

    protected $middlewareAliases = [

        //register here

        'routemiddleware' => \App\Http\Middleware\DemoMiddelware::class,

//other middlewares

}

1. now simply use it in routes

in web.php

//web.php

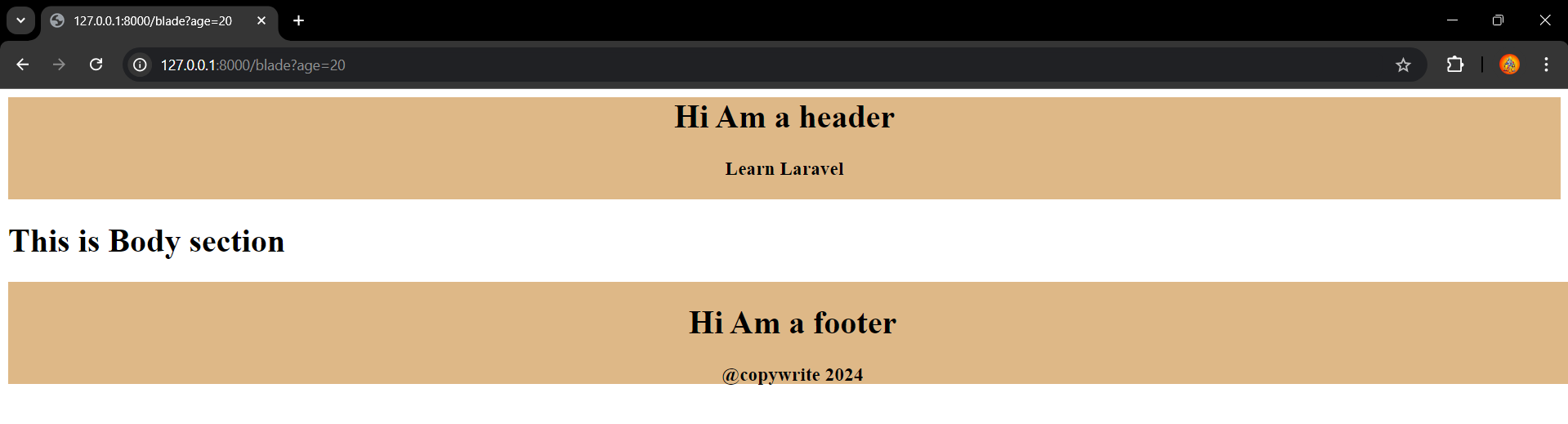
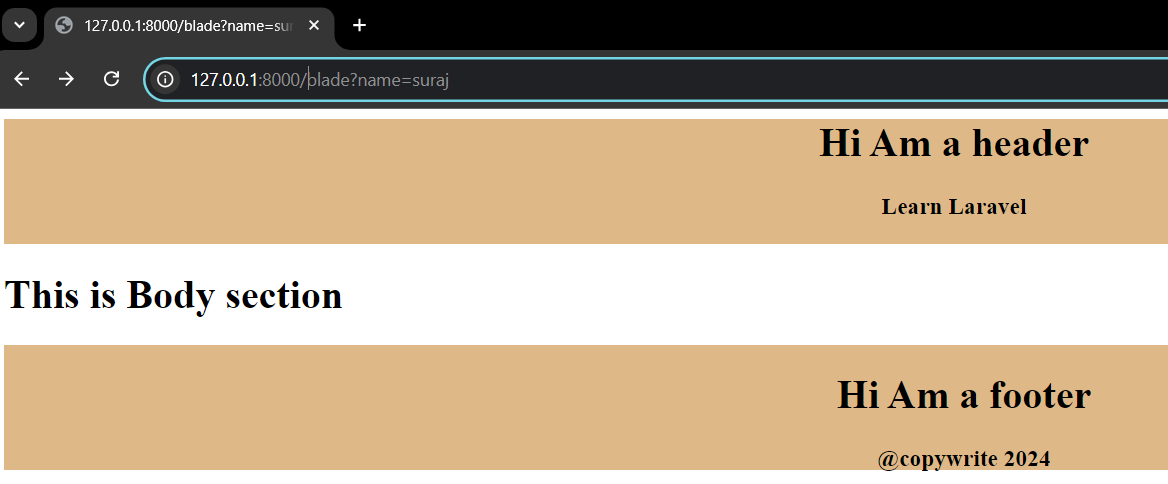
//route midd..

Route::get("php", [PhpExampleController::class, "loadView"])->middleware('routemiddleware1');

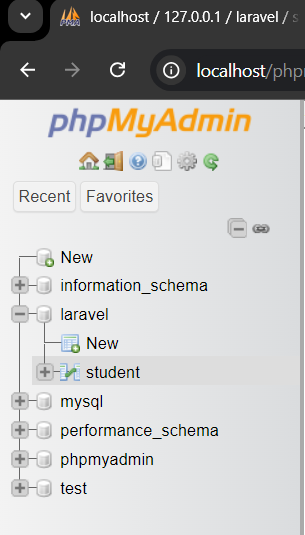
//can we add two middlewares for one view==>yessss

Route::view('/blade', "blade\_code")->middleware('routemiddleware1', 'routemiddleware2');

Output 🡺



1. Now let’s dive into database:
2. Create a database using xampp
3. Now change .env file



//.env

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=laravel

DB\_USERNAME=root

DB\_PASSWORD=

1. Now create a controller

**$ php artisan make:controller mysqldb**

INFO Controller [first-project\app\Http\Controllers\mysqldb.php] created successfully.

//mysqldb.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\DB;

class mysqldb extends Controller

{

    function db()

    {

       return DB::select('select \* from student');

    }

}

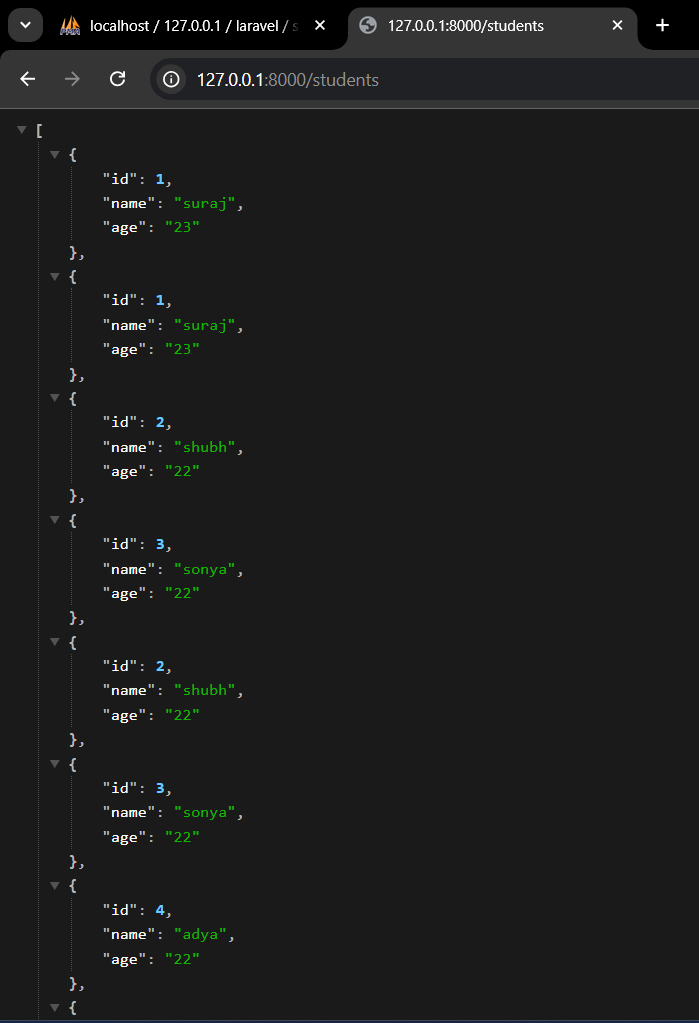
1. Create route for it:

//web.php

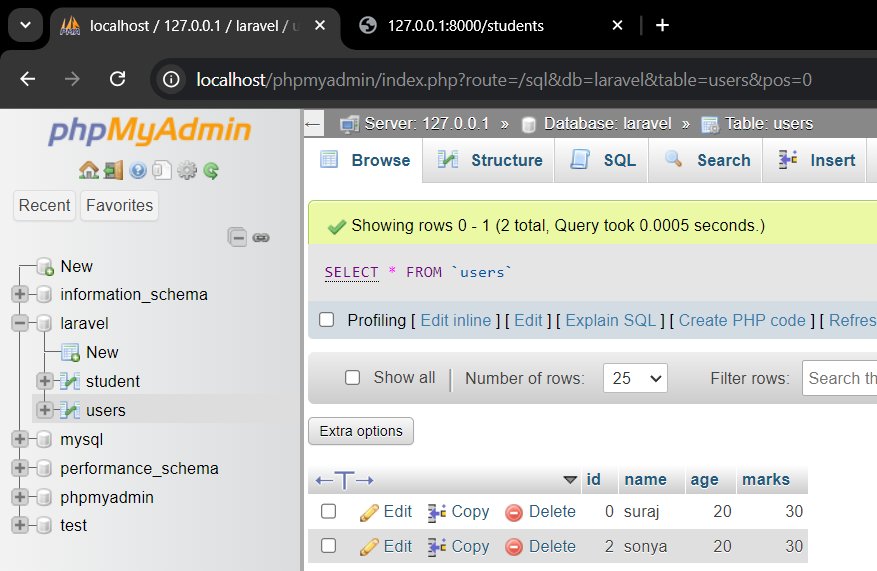
//myysqldb (controller)

Route::get('students',[mysqldb::class,'db']);

1. Output:



1. Now there is another method to connect with db using model
2. Create a db:



1. Edit .env file

//.env

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=laravel

DB\_USERNAME=root

DB\_PASSWORD=

1. Create a model

Now keep in mind

|  |  |
| --- | --- |
| **Db Table** | **Model Name** |
| users | User |
| students | Student |

Our database table name is users then we create a model named User

**$ php artisan make:model User**

INFO Model [ app\Models\User.php] created successfully.

//User.php

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class User extends Model

{

    use HasFactory;

}

1. Now create a controller

**$ php artisan make:controller mysqldb2**

INFO Controller [\app\Http\Controllers\mysqldb2.php] created successfully.

// mysqldb2.php

<?php

namespace App\Http\Controllers;

use App\Models\User;

use Illuminate\Http\Request;

class mysqldb2 extends Controller

{

    function db(){

        //User==> App\Models\User

        //all==> Get all of the models from the database.

        return User::all();

    }

}

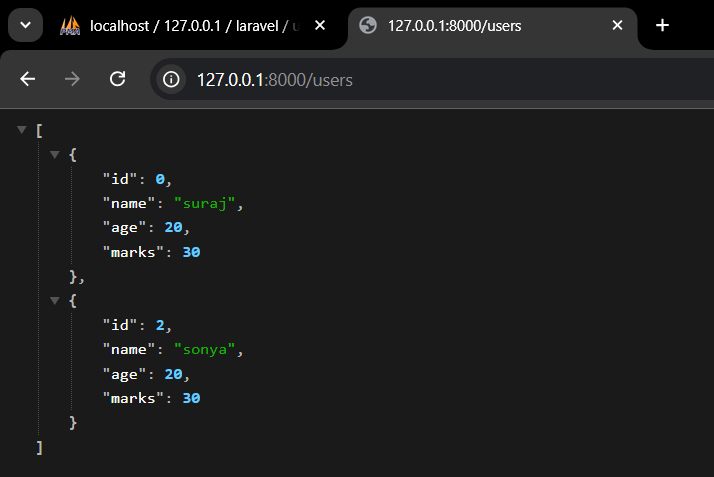
1. To check create a route for it

//web.php

//myysqldb2 (controller)

Route::get('users', [mysqldb2::class, 'db']);

1. Output



1. Now what if our table name is like student and also need to call that table

Now keep in mind

|  |  |
| --- | --- |
| **Db Table** | **Model Name** |
| users | User |
| students | Student |

Our database table name is users then we create a model named User

Because this condition is not satisfy:

So for that we need to make a change in model we created

//User.php

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class User extends Model

{

//add this to call specific table

    public $table='student';

    use HasFactory;

}