Laravel 8 Auth Login and Registration with Username or Email

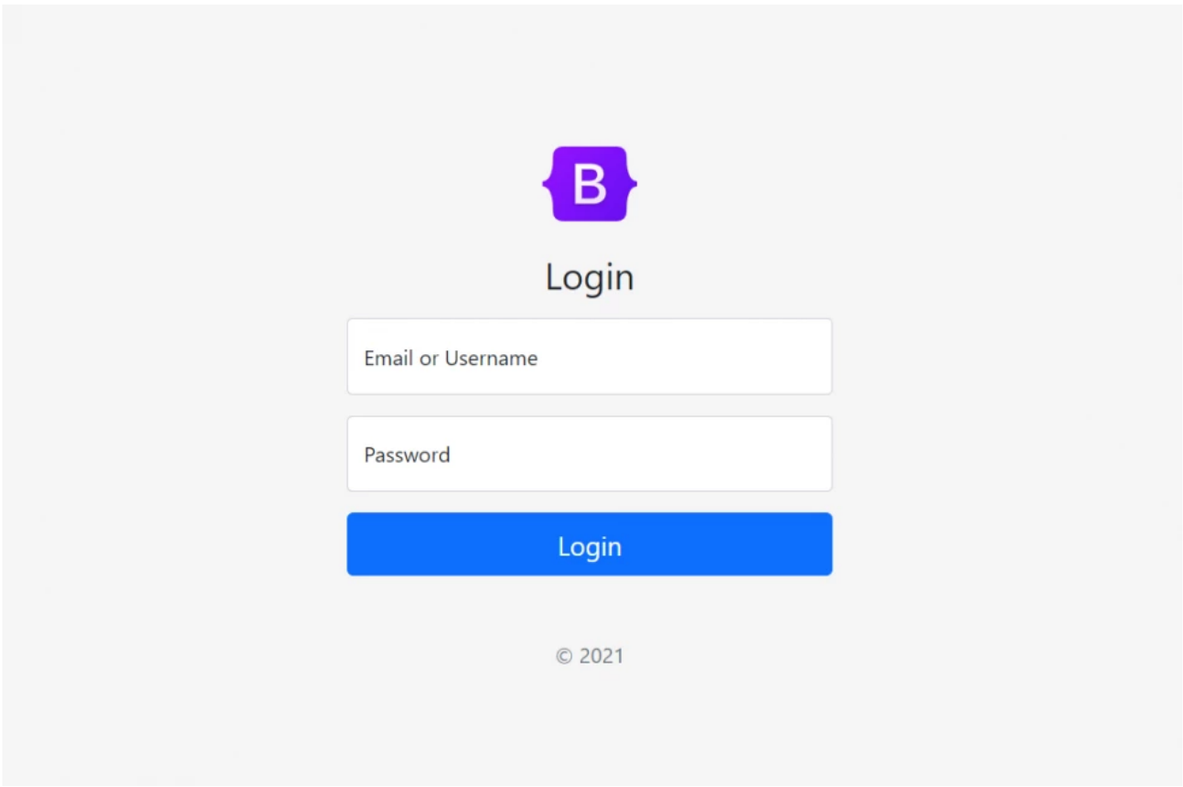
Laravel # PHP

Admin posted this 5 months ago 17 minutes to read

TABLE OF CONTENTS

In this post, I will share how to implement **Laravel 8 custom auth login and registration** with username or email. We know that Laravel Framework is one of the best PHP Frameworks because of these advanced features and development tools that help make your development fast; that's why many PHP Developers use it and recommend it. Laravel additionally helps web developers to simplify their development process with clean and reusable code.

As I said above that I will share how to create a simple and clean authentication for your Laravel app with username or email. We know that authentication is one of the most important to implement before coding your application to control your users to access sensitive data.



I will give you easy steps for you to understand clearly.

Step 1: Create Laravel App

I assume that you have already set up your composer on your system. Run the following coding to install the new Laravel app. However, you can skip this step if you have the Laravel app installed already.

```
composer create-project --prefer-dist laravel/laravel login-and-r
```

Next, navigate the **login-and-registration** folder with the following command.

```
cd login-and-registration
```

Step 2: Setup Database Credentials

PHPMyAdmin. Then once created navigate the **.env** file and update your database credentials.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=your_db_name
DB_USERNAME=your_db_username
DB_PASSWORD=your_db_password
```

The Laravel default comes with a User model and migration files. But before running the migrate command we need to update your user table migration which can be found here > {project_folder}\database\migrations\2014_10_12_000000_create_users_table.php and add the username field then update the **name** field to nullable so that in registration we require the email, username, and password. See the updated migration code below.

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name')->nullable();
    $table->string('email')->unique();
    $table->string('username')->unique();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password');
    $table->rememberToken();
    $table->timestamps();
});
```

Once updated our migrations are now ready to migrate just run to your terminal the following command:

```
php artisan migrate
```

Step 3: Setup Model

Next, we need to modify our **App\Models\User.php** model and add the username as fillable. See the updated code in the following:

```
/**
 * The attributes that are mass assignable.
 *
 * @var array
*/
protected $fillable = [
  'name',
  'email',
  'username',
  'password',
];
```

registered, the **password** will always be **encrypted**. This is called Laravel mutators to know more about it just visit their documentation here.

```
/**
 * Always encrypt the password when it is updated.
 *
  * @param $value
 * @return string
 */
public function setPasswordAttribute($value)
{
   $this->attributes['password'] = bcrypt($value);
}
```

Don't worry to see the complete code of this **User.php** model just download the full source code of this tutorial below.

Step 4: Setup Registration Controller

To set up the registration controller just use your terminal and run the following command.

```
php artisan make:controller RegisterController
```

Now, you already generated our RegistrerController which you can found it here > **App\Http\Controllers\RegisterController.php** now open it and see the following code below:

```
<?php

namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;
use App\Http\Requests\RegisterRequest;

class RegisterController extends Controller
{
    /**
     * Display register page.
     *
     * @return \Illuminate\Http\Response
     */
    public function show()
    {
        return view('auth.register');
    }

    /**
```

```
     * @param RegisterRequest $request
     *
     * @return \Illuminate\Http\Response
     */
    public function register(RegisterRequest $request)
    {
        $user = User::create($request->validated());

        auth()->login($user);

        return redirect('/')->with('success', "Account successfu
    }
}
```

Step 5: Setup Registration Request

We need to separate our request validation for our registration process. So that our code is clean in our controller and not bloated. So we will create our **RegisterRequest** just follow the following command below:

```
php artisan make:request RegisterRequest
```

Now you have already created the RegisterRequest which you can find it here > **App\Http\Requests\RegisterRequest.php** then next we will add our validation rules. Just see the following code below:

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class RegisterRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
```

```
    {
        return [
            'email' => 'required|email:rfc,dns|unique:users,email
            'username' => 'required|unique:users,username',
            'password' => 'required|min:8',
            'password_confirmation' => 'required|same:password'
        ];
    }
}
```

Now you have a registration validation already.

Step 6: Setup Login Controller

To set up the login controller just use your terminal and run the following command.

```
php artisan make:controller LoginController
```

Now, you already generated our LoginController which you can found it here > **App\Http\Controllers\LoginController.php** now open it and see the following code below:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests\LoginRequest;
use Illuminate\Support\Facades\Auth;

class LoginController extends Controller
{
    /**
     * Display login page.
     *
     * @return Renderable
     */
    public function show()
    {
        return view('auth.login');
    }

    /**
     * Handle account login request
     *
     * @param LoginRequest $request
     *
     * @return \Illuminate\Http\Response
     */
```

```
    {
        $credentials = $request->getCredentials();

        if(!Auth::validate($credentials)):
            return redirect()->to('login')
                ->withErrors(trans('auth.failed'));
        endif;

        $user = Auth::getProvider()->retrieveByCredentials($crede

        Auth::login($user);

        return $this->authenticated($request, $user);
    }

    /**
     * Handle response after user authenticated
     *
     * @param Request $request
     * @param Auth $user
     *
     * @return \Illuminate\Http\Response
     */
    protected function authenticated(Request $request, $user)
    {
        return redirect()->intended();
    }
}
```

Step 7: Setup Login Request

Next, we will create our **LoginRequest** just follow the following command below:

```
php artisan make:request LoginRequest
```

Now you have already created the LoginRequest which you can find it here > **App\Http\Requests\LoginRequest.php** then next we will add our validation rules. Just see the following code below:

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Contracts\Validation\Factory as ValidationFactory;

class LoginRequest extends FormRequest
{
    /**
```

```
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'username' => 'required',
            'password' => 'required'
        ];
    }

    /**
     * Get the needed authorization credentials from the request
     *
     * @return array
     * @throws \Illuminate\Contracts\Container\BindingResolutionE
     */
    public function getCredentials()
    {
        // The form field for providing username or password
        // have name of "username", however, in order to support
        // logging users in with both (username and email)
        // we have to check if user has entered one or another
        $username = $this->get('username');

        if ($this->isEmail($username)) {
            return [
                'email' => $username,
                'password' => $this->get('password')
            ];
        }

        return $this->only('username', 'password');
    }

    /**
     * Validate if provided parameter is valid email.
     *
     * @param $param
     * @return bool
     * @throws \Illuminate\Contracts\Container\BindingResolutionE
     */
```

```
        {
            $factory = $this->container->make(ValidationFactory::clas

            return ! $factory->make(
                ['username' => $param],
                ['username' => 'email']
            )->fails();
        }
    }
```

As you can see above our **LoginRequest.php** we have our additional method called **getCredentials()** this function will support the username/email login as you can see we have username checking above if email or not.

Step 8: Setup Logout Controller

To set up the logout controller just use your terminal and run the following command.

```
php artisan make:controller LogoutController
```

Now, you already generated our LogoutController which you can found it here > **App\Http\Controllers\LogoutController.php** now open it and see the following code below:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Session;

class LogoutController extends Controller
{
    /**
     * Log out account user.
     *
     * @return \Illuminate\Routing\Redirector
     */
    public function perform()
    {
        Session::flush();

        Auth::logout();

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

Step 9: Setup Home Controller

To set up the logout controller just use your terminal and run the following command.

```
php artisan make:controller HomeController
```

Now, you already generated our **HomeController** which you can found it here > **App\Http\Controllers\HomeController.php** now open it and see the following code below:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{
    public function index()
    {
        return view('home.index');
    }
}
```

Step 10: Setup Routes

Next, we will set up our routes which we can find here **routes/web.php** since we have already set up our **controllers** and **validators**.

```
<?php

use Illuminate\Support\Facades\Route;

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

Route::group(['namespace' => 'App\Http\Controllers'], function()
{
    /**
     * Home Routes
     */
    Route::get('/', 'HomeController@index')->name('home.index');
```

```
    Route::group(['middleware' => ['guest']], function() {
        /**
         * Register Routes
         */
        Route::get('/register', 'RegisterController@show')->name(
        Route::post('/register', 'RegisterController@register')->

        /**
         * Login Routes
         */
        Route::get('/login', 'LoginController@show')->name('login
        Route::post('/login', 'LoginController@login')->name('log

    });

    Route::group(['middleware' => ['auth']], function() {
        /**
         * Logout Routes
         */
        Route::get('/logout', 'LogoutController@perform')->name(
    });
});
```

As you can see above in this line "**Route::group(['middleware' => ['auth']], function() {"** we protected the logout route that can be access only if user authenticated. If you have other routes that need to protect just add on this route group.

Step 11: Setup Our Views

Next, we will set up our views using Bootstrap with a simple theme layout. To know more about bootstrap documentation just visit here.

You need to create a **layouts** folder inside **resources/views** then create a file **resources/views/layouts/auth-master.blade.php** see the following code below:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-sc
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and
    <meta name="generator" content="Hugo 0.87.0">
    <title>Signin Template · Bootstrap v5.1</title>

    <!-- Bootstrap core CSS -->
    <link href="{!! url('assets/bootstrap/css/bootstrap.min.css'
    <link href="{!! url('assets/css/signin.css') !!}" rel="style

    <style>
```

```
        font-size: 1.125rem;
        text-anchor: middle;
        -webkit-user-select: none;
        -moz-user-select: none;
        user-select: none;
      }

      @media (min-width: 768px) {
        .bd-placeholder-img-lg {
          font-size: 3.5rem;
        }
      }
    </style>


    <!-- Custom styles for this template -->
    <link href="signin.css" rel="stylesheet">
</head>
<body class="text-center">

    <main class="form-signin">

        @yield('content')

    </main>


</body>
</html>
```

Next, create a file **resources/views/layouts/app-master.blade.php** see the following code below:

```
<!doctype html>
<html lang="en">
    <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-sc
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and E
    <meta name="generator" content="Hugo 0.87.0">
    <title>Fixed top navbar example · Bootstrap v5.1</title>

    <!-- Bootstrap core CSS -->
    <link href="{!! url('assets/bootstrap/css/bootstrap.min.css')

    <style>
      .bd-placeholder-img {
        font-size: 1.125rem;
        text-anchor: middle;
```

```
        -moz-user-select: none;
        user-select: none;
      }

      @media (min-width: 768px) {
        .bd-placeholder-img-lg {
          font-size: 3.5rem;
        }
      }
    </style>


    <!-- Custom styles for this template -->
    <link href="{!! url('assets/css/app.css') !!}" rel="styleshee
</head>
<body>

    @include('layouts.partials.navbar')

    <main class="container">
        @yield('content')
    </main>

    <script src="{!! url('assets/bootstrap/js/bootstrap.bundle.m

  </body>
</html>
```

Next, create a new folder inside **resources/views/layouts** called **partials** folder then create a file **resources/views/layouts/partials/navbar.blade.php** see the following code below:

```
<header class="p-3 bg-dark text-white">
  <div class="container">
    <div class="d-flex flex-wrap align-items-center justify-conte
      <a href="/" class="d-flex align-items-center mb-2 mb-lg-0 t
        <svg class="bi me-2" width="40" height="32" role="img" ar
      </a>

      <ul class="nav col-12 col-lg-auto me-lg-auto mb-2 justify-c
        <li><a href="#" class="nav-link px-2 text-secondary">Home
        <li><a href="#" class="nav-link px-2 text-white">Features
        <li><a href="#" class="nav-link px-2 text-white">Pricing<
        <li><a href="#" class="nav-link px-2 text-white">FAQs</a>
        <li><a href="#" class="nav-link px-2 text-white">About</a
      </ul>

      <form class="col-12 col-lg-auto mb-3 mb-lg-0 me-lg-3">
        <input type="search" class="form-control form-control-dar
      </form>
```

```
        @auth
          {{auth()->user()->name}}
          <div class="text-end">
            <a href="{{ route('logout.perform') }}" class="btn btn-
          </div>
        @endauth

        @guest
          <div class="text-end">
            <a href="{{ route('login.perform') }}" class="btn btn-o
            <a href="{{ route('register.perform') }}" class="btn bt
          </div>
        @endguest
      </div>
    </div>
  </header>
```

Then next, create a file **resources/views/layouts/partials/messages.blade.php** see the following code below:

```
@if(isset ($errors) && count($errors) > 0)
    <div class="alert alert-warning" role="alert">
        <ul class="list-unstyled mb-0">
            @foreach($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

@if(Session::get('success', false))
    <?php $data = Session::get('success'); ?>
    @if (is_array($data))
        @foreach ($data as $msg)
            <div class="alert alert-warning" role="alert">
                <i class="fa fa-check"></i>
                {{ $msg }}
            </div>
        @endforeach
    @else
        <div class="alert alert-warning" role="alert">
            <i class="fa fa-check"></i>
            {{ $data }}
        </div>
    @endif
@endif
```

Then next, create our **auth** folder inside **resources/views** then create a file **resources/views/auth/register.blade.php** see the following code below:

```
@section('content')
    <form method="post" action="{{ route('register.perform') }}">

        <input type="hidden" name="_token" value="{{ csrf_token()
        <img class="mb-4" src="{!! url('images/bootstrap-logo.svg

        <h1 class="h3 mb-3 fw-normal">Register</h1>

        <div class="form-group form-floating mb-3">
            <input type="email" class="form-control" name="email"
            <label for="floatingEmail">Email address</label>
            @if ($errors->has('email'))
                <span class="text-danger text-left">{{ $errors->f
            @endif
        </div>

        <div class="form-group form-floating mb-3">
            <input type="text" class="form-control" name="usernam
            <label for="floatingName">Username</label>
            @if ($errors->has('username'))
                <span class="text-danger text-left">{{ $errors->f
            @endif
        </div>

        <div class="form-group form-floating mb-3">
            <input type="password" class="form-control" name="pas
            <label for="floatingPassword">Password</label>
            @if ($errors->has('password'))
                <span class="text-danger text-left">{{ $errors->f
            @endif
        </div>

        <div class="form-group form-floating mb-3">
            <input type="password" class="form-control" name="pas
            <label for="floatingConfirmPassword">Confirm Password
            @if ($errors->has('password_confirmation'))
                <span class="text-danger text-left">{{ $errors->f
            @endif
        </div>

        <button class="w-100 btn btn-lg btn-primary" type="submit

        @include('auth.partials.copy')
    </form>
@endsection
```

Then next, create a file **resources/views/auth/login.blade.php** see the following code below:

```
@section('content')
    <form method="post" action="{{ route('login.perform') }}">

        <input type="hidden" name="_token" value="{{ csrf_token()
        <img class="mb-4" src="{!! url('images/bootstrap-logo.svg

        <h1 class="h3 mb-3 fw-normal">Login</h1>

        @include('layouts.partials.messages')

        <div class="form-group form-floating mb-3">
            <input type="text" class="form-control" name="usernam
            <label for="floatingName">Email or Username</label>
            @if ($errors->has('username'))
                <span class="text-danger text-left">{{ $errors->f
            @endif
        </div>

        <div class="form-group form-floating mb-3">
            <input type="password" class="form-control" name="pas
            <label for="floatingPassword">Password</label>
            @if ($errors->has('password'))
                <span class="text-danger text-left">{{ $errors->f
            @endif
        </div>

        <button class="w-100 btn btn-lg btn-primary" type="submit

        @include('auth.partials.copy')
    </form>
@endsection
```

Then create a **partials** folder inside **auth** folder. Then create a file **resources/views/auth/partials/copy.blade.php** see the following code below:

```
<p class="mt-5 mb-3 text-muted">&copy; {{date('Y')}}</p>
```

Then last create a **home** folder inside the **resources/views** folder then create a file **resources/views/home/index.blade.php** see the following code below:

```
@extends('layouts.app-master')

@section('content')
    <div class="bg-light p-5 rounded">
        @auth
        <h1>Dashboard</h1>
        <p class="lead">Only authenticated users can access this
        <a class="btn btn-lg btn-primary" href="https://codeandde
        @endauth
```

```
        @guest
        <h1>Homepage</h1>
        <p class="lead">Your viewing the home page. Please login
        @endguest
    </div>
@endsection
```

Now our views are already set up. Next, we need to download bootstrap and save it inside the **public/assets** directory.

This is the bootstrap files example in this tutorial:
public/assets/bootstrap/css/bootstrap.min.css
public/assets/bootstrap/js/bootstrap.bundle.min.js

Or download the source code of this tutorial below.

Now our Laravel authentication with username or email login is ready but we have another small customization the redirect default after login. Which we can found in App\Providers\RouteServiceProvider.php then change the original code below:

```
/**
* The path to the "home" route for your application.
*
* This is used by Laravel authentication to redirect users after

* @var string
*/
public const HOME = '/home';
```

to this code.

```
/**
* The path to the "home" route for your application.
*
* This is used by Laravel authentication to redirect users after
*
* @var string
*/
public const HOME = '/';
```

But take note it's up to you to change it. I just give you an idea of how to modify it that is suitable to your need. If you need to redirect after login to /dashboard or /admin then it's up to you to change it.

Step 12: Run The Development Server

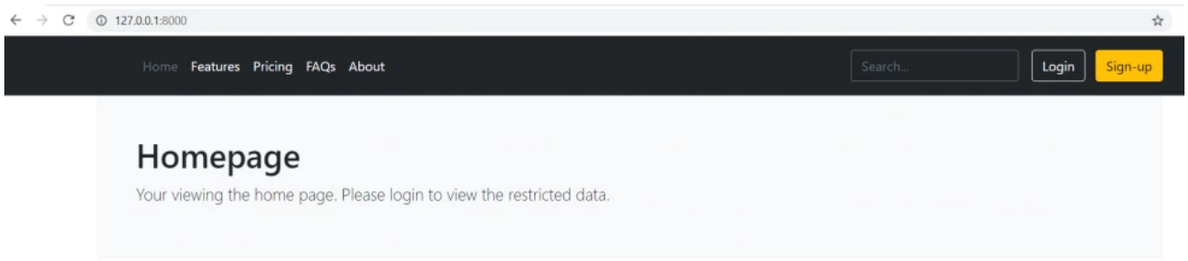
command to your terminal to run your server and test if our code is working.

```
php artisan serve
```

And add the following URL to your web browser.

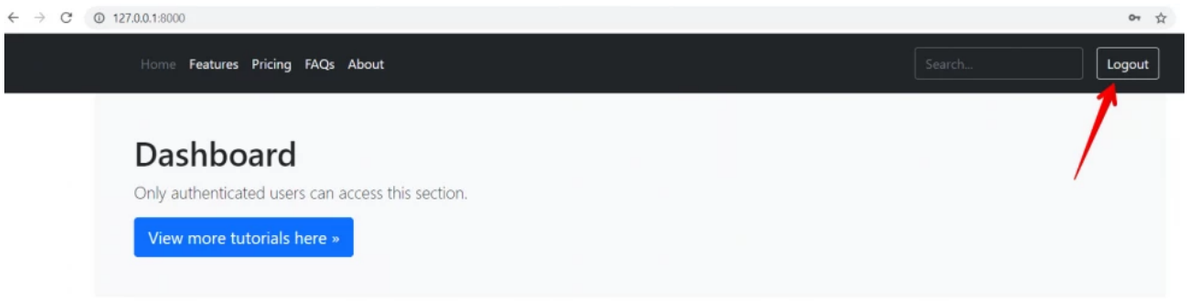
```
http://127.0.0.1:8000/
```

You will see the homepage if not yet authenticated as shown below:



As you can see you will see a Login and Sign-up menu above.

But if you already logged in. Then you will see this show below:



As you can see now you already see the Logout menu above.

Now you have a custom Laravel registration and login with username/email that can apply to your app. I hope this simple tutorial may help you.

NOTE: Don't forget to import the classes uses in a specific class or controller when copying my code above.

Download Source Code

Thanks for reading. If you think this tutorial is helpful to you. Please share it with your friends.

Happy coding :)

Read next

Combine the Nwidart Laravel Modules Assets to Public using Laravel Mix

How To Add Remember Me Functionality To Your Laravel 8 Login?

5 months ago 2 minutes to read

How To Implement Remember Me with Custom Expiration To Your Laravel 8 Login?

5 months ago 2 minutes to read

Laravel 8 Logout For Your Authenticated User

5 months ago less than a minute to read

How To Implement Laravel 8 Email Verification?

5 months ago 5 minutes to read

How To Change The Laravel Redirect URL When Not Authenticated?

5 months ago less than a minute to read

How To Queue the Laravel 8 Email Verification?

5 months ago 1 minute to read

Laravel 8 - Simple Custom Validation Rules Example

5 months ago 3 minutes to read

✨ Popular posts

How To Return JSON Response in PHP & MySQL using Ajax and jQuery

last read 1 second ago

How To Add Remember Me Functionality To Your Laravel 8 Login?

last read 29 seconds ago

Dynamic Ajax Form Validation in PHP & MySQL using jQuery

last read 47 seconds ago

How to add Default Eloquent Model Values on Laravel 8?

last read 50 seconds ago

Laravel Maintenance Mode Example

last read 54 seconds ago

Check If Email Address Is Already Exists in The Database
last read 1 minute ago

Laravel 8 User Roles and Permissions Step by Step Tutorial
last read 2 minutes ago

Delete Record using jQuery Ajax in Laravel 8
last read 3 minutes ago

How to Install PHP GD Extension on Windows WAMP and XAMPP Server
last read 3 minutes ago

Complete Laravel 8 Soft Delete & Restore Deleted Records Tutorial
last read 6 minutes ago

Edit or Update Data with Bootstrap Modal in PHP & MySQL Using Ajax
last read 8 minutes ago

Provides a programming tutorial for aspiring web & software developers to help them understand PHP, Laravel Framework, CSS3, HTML5, MySQL, Bootstrap, and many more. We hope that our tutorials can help you grow your career as a programmer.

ABOUT

About Us Contact

OTHER PAGES

Terms of Service Disclaimer Privacy Policy