AppDividend

Laravel 5.5 Tutorial With Example From Scratch

Laravel 5.5 CRUD Tutorial With Example

By **Krunal** Last updated **Feb 22, 2018**

lt.earcurette.eu

Laravel 5.5 Tutorial With Example From Scratch is **Laravel 5.5 CRUD tutorial for beginners**. Laravel has recently launched its new version called **Laravel 5.5,** and it is also come up with a bunch of new Features. Today, we are not discussing in depth new features of Laravel 5.5, just list them but rather than focusing on the coding in **Laravel 5.5**. It's small application in **Laravel 5.5. Laravel** is most flexible and elegant PHP Framework so far and always come up with new things. Laravel is the best representation of **PHP Language** in my opinion. It is the by far most in built functionality Framework. Creator has seriously put his own heart to make it and right now maintaining. Every Laravel events, we will see there is always new and new coming ready with this Framework, and that is why It is the most trusted framework among the PHP Community. It provides us the power to whatever we can build and play with this structure, with in scope and also with out its scope if you have in-depth knowledge of **Latest PHP version.**

Content Overview [hide]

Laravel 5.5 New Features

1. **Custom validation rules object.**
2. **Returns response data from the validator.**
3. **Improvements with the default error views.**
4. **Great support for the custom error reporting.**
5. **Support for email themes in mailable.**
6. **We can render mailable in the browser.**
7. **Vendor packages have provider support.**
8. **It adds front end presets, which means we can use Vue, React.js or none of them if we want.**
9. **Laravel Migrate Fresh command.**
10. **Whoops, which was there in Laravel 4.2 is back in Laravel 5.5**
11. **Laravel Package Auto Discovery.**

Installation Requirments

- PHP >= 7.0.0
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension

- XML PHP Extension

Note: Here Laravel 5.5 requires **PHP 7** or above version. If you have **PHP version like 5.4, 5.5, 5.6**, then It will not work. Please upgrade it to **PHP 7**

Laravel 5.5 CRUD Tutorial From Scratch

Step 1: Installing The Laravel 5.5 Framework.

```
composer create-project --prefer-dist laravel/
laravel Laravel55
```

It will create a directory and install all the Laravel specific dependencies.

Note: Here, we are not going to in detail about front end dependencies like Vue.js or React.js using Laravel Mix so, we are not going to install any of the Node dependencies right now.

Step 2: Setup a MySQL database in .env file.

Create one database in MySQL and then switch to your editor and open the **.env** file.

```
// .env

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=Laravel55
DB_USERNAME=root
DB_PASSWORD=mysql
```

I have setup my local database credentials. You just need to change last three constants and you are ready to go.

Now, migrate two tables provided by Laravel 5.5.
Switch to your terminal and type following command.

```
php artisan migrate
```

It will create two tables in your database.

1. **users**
2. **password_resets**

Step 3: Create a model as well as migration file for our Products table.

Type the following command in your terminal.

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

It will create two files.

1. **Product.php** model.
2. **create_products_table** migration file.

We need to create Schema for the products table. So navigate to **Laravel55 >> database >> migrations >> create_products_table.**

```
// create_products_table

public function up()
{
    Schema::create('products', function (Bluep
rint $table) {
       $table->increments('id');
       $table->string('name');
       $table->integer('price');
       $table->timestamps();
    });
}
```

So, for our **products** table, there will be **5 columns**. **timestamps()** have 2 columns

1. created_at
2. updated_at

Now, migrate the table by the following command.

```
php artisan migrate
```

In the database, you can see the **products** table.

Step 4: Make one view file to add the form data in the database.

Make one folder called **products** and then create a file in the **resources >> views >> products >> create.blade.php** and put the following code in it.

```
<!-- create.blade.php -->

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Laravel 5.5 CRUD Tutorial With Exam
ple From Scratch </title>
    <link rel="stylesheet" href="{{asset('css/
app.css')}}">
  </head>
  <body>
    <div class="container">
      <h2>Create A Product</h2><br  />
      <form method="post">
        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <label for="name">Name:</label>
            <input type="text" class="form-con
trol" name="name">
          </div>
        </div>
        <div class="row">
          <div class="col-md-4"></div>
            <div class="form-group col-md-4">
              <label for="price">Price:</label
>
              <input type="text" class="form-c
ontrol" name="price">
            </div>
          </div>
        </div>
        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <button type="submit" class="btn b
tn-success" style="margin-left:38px">Add Produ
ct</button>
          </div>
        </div>
      </form>
    </div>
  </body>
</html>
```

Step 5: Create one

controller and route to display the Product form

Go to the terminal and type the following command.

```
php artisan make:controller ProductController
--resource
```

It will create one controller file called **ProductController.php** and It has all the CRUD Functions, we need to seek.

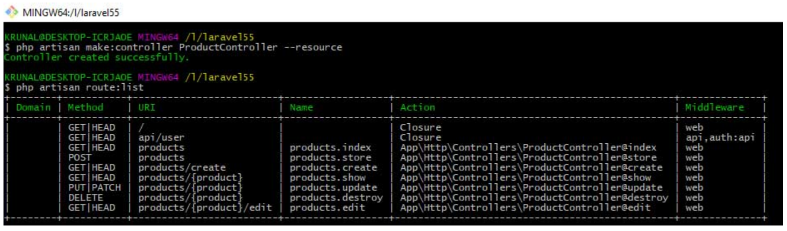
Here, we have used **resource** parameter, so by default, It provides us some routing patterns, but right now, we will not see until we register one route in **routes >> web.php** file. So let us do it.

```
// web.php

Route::resource('products','ProductController'
);
```

Now, switch to your terminal and type the following command.

```
php artisan route:list
```

You will see following route list.



```
MINGW64:/l/laravel55

KRUNAL@DESKTOP-ICRJAOE MINGW64 /l/laravel55
$ php artisan make:controller ProductController --resource
Controller created successfully.

KRUNAL@DESKTOP-ICRJAOE MINGW64 /l/laravel55
$ php artisan route:list
+--------+-----------+-------------------------+------------------+------------------------------------------------+--------------+
| Domain | Method    | URI                     | Name             | Action                                         | Middleware   |
+--------+-----------+-------------------------+------------------+------------------------------------------------+--------------+
|        | GET|HEAD  | /                       |                  | Closure                                        | web          |
|        | GET|HEAD  | api/user                |                  | Closure                                        | api,auth:api |
|        | GET|HEAD  | products                | products.index   | App\Http\Controllers\ProductController@index   | web          |
|        | POST      | products                | products.store   | App\Http\Controllers\ProductController@store   | web          |
|        | GET|HEAD  | products/create         | products.create  | App\Http\Controllers\ProductController@create  | web          |
|        | GET|HEAD  | products/{product}      | products.show    | App\Http\Controllers\ProductController@show    | web          |
|        | PUT|PATCH | products/{product}      | products.update  | App\Http\Controllers\ProductController@update  | web          |
|        | DELETE    | products/{product}      | products.destroy | App\Http\Controllers\ProductController@destroy | web          |
|        | GET|HEAD  | products/{product}/edit | products.edit    | App\Http\Controllers\ProductController@edit    | web          |
+--------+-----------+-------------------------+------------------+------------------------------------------------+--------------+
```

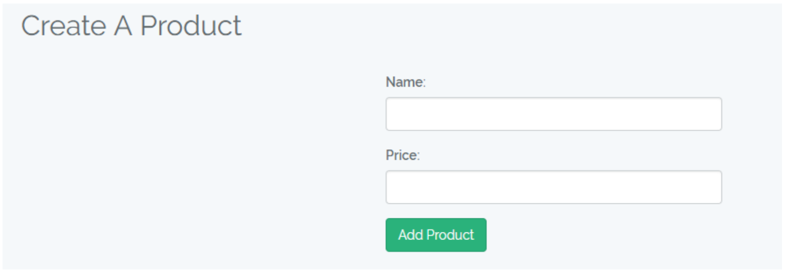
Next step would be to go to **ProductController.php** file and add into **create()** function some code.

```
// ProductController.php

  /**
     * Show the form for creating a new resour
ce.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        return view('products.create');
    }
```

After that, we need to start Laravel Development server. So in the terminal, hit the following command.

```
php artisan serve
```

Switch to the browser and hit this URL:
http://localhost:8000/products/create
You will see the following screen.



Step 6: Put the Laravel 5.5 Validation in Product Form.

First, we need to apply an action to our product creation form.

```
<!-- create.blade.php -->

<form method="post" action="{{url('products')}
}">
```

Now, we need to handle

CSRF issue. So, put the following code in the form

```
<!-- create.blade.php -->

{{csrf_field()}}
```

We also need to handle **Mass Assignment Exception.** So we need to go to **app >> Product.php** file and in that put the **protected $fillable** property in it.

```
// Product.php

protected $fillable = ['name','price'];
```

If you see the resource routes then it has **post request** has '**/products**' route and **store** function in **ProductController.php file.** So we need to code the store function in order to save the data in the database.

One thing to keep in mind that, we need to include the **namespace** of **Product.php** model in the ProductController.php file. So type the following line at the starting of **ProductController**.**php** file.

```
use App\Product;
```

Also, we need to put the validation there.

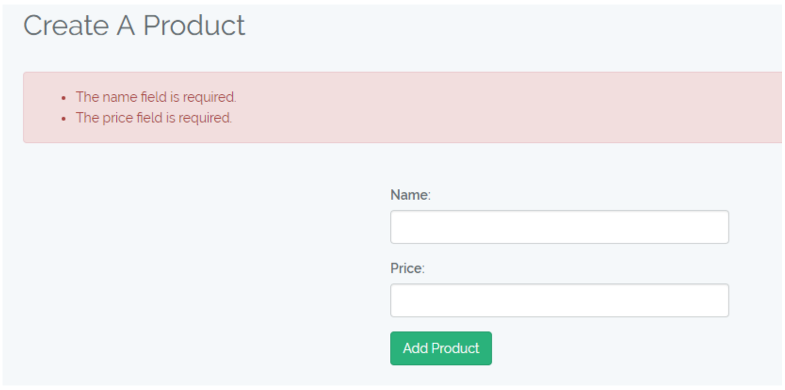
```
// ProductController.php

/**
     * Store a newly created resource in stora
ge.
     *
     * @param  \Illuminate\Http\Request  $requ
est
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $product = $this->validate(request(),
[
          'name' => 'required',
          'price' => 'required|numeric'
        ]);

        Product::create($product);

        return back()->with('success', 'Produc
t has been added');;
    }
```

Now, if validation fails then we need to show an error. So go back to **create.blade.php and** put the following code after **h2** tag.

```
<!-- create.blade.php -->

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Laravel 5.5 CRUD Tutorial With Exam
ple From Scratch </title>
    <link rel="stylesheet" href="{{asset('css/
app.css')}}">
  </head>
  <body>
    <div class="container">
      <h2>Create A Product</h2><br  />
      @if ($errors->any())
      <div class="alert alert-danger">
          <ul>
              @foreach ($errors->all() as $err
or)
                  <li>{{ $error }}</li>
              @endforeach
          </ul>
      </div><br />
      @endif
      @if (\Session::has('success'))
      <div class="alert alert-success">
          <p>{{ \Session::get('success') }}</p
>
      </div><br />
      @endif
      <form method="post" action="{{url('produ
cts')}}">
        {{csrf_field()}}
        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <label for="name">Name:</label>
            <input type="text" class="form-con
trol" name="name">
          </div>
        </div>
        <div class="row">
          <div class="col-md-4"></div>
            <div class="form-group col-md-4">
              <label for="price">Price:</label
>
              <input type="text" class="form-c
ontrol" name="price">
            </div>
          </div>
        </div>
        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <button type="submit" class="btn b
tn-success" style="margin-left:38px">Add Produ
ct</button>
          </div>
        </div>
      </form>
    </div>
  </body>
</html>
```

At last, go to the browser and hit the development Laravel URL: http://localhost:8000

If you submit the form without any value then, you can see the errors like below image.


Create A Product

- The name field is required.
- The price field is required.

Name:

Price:

If you fill all the values then, you will redirect to this page with the success message. So, here One thing must be noticed.

```
// ProductController.php

  public function store(Request $request)
  {
        $product = $this->validate(request(),
[
          'name' => 'required',
          'price' => 'required|numeric'
        ]);
        Product::create($product);
        return back()->with('success', 'Produc
t has been added');
  }
```

In Laravel 5.5 we directly get the array of the values return by validation function and use it to insert in the database, which is new feature in Laravel 5.5

Step 7: Make an index page to list the products.

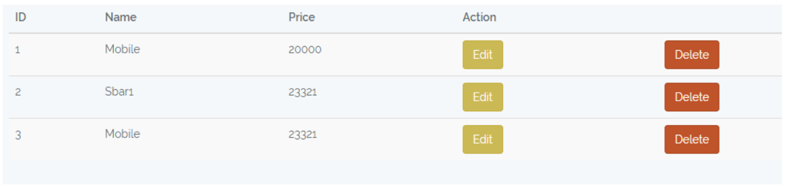
For that, first, we need to send the data to the **index.blade.php.** So, in **ProductController.php** file, we need to write the code to fetch the data and return it to the **index view.**

```
// ProductController.php

  /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $products = Product::all()->toArray();
        return view('products.index', compact(
'products'));
    }
```

In **resources >> views >> products,** create one blade file called **index.blade.php file** and put the following code in it.

```
<!-- index.blade.php -->

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Index Page</title>
    <link rel="stylesheet" href="{{asset('css/
app.css')}}">
  </head>
  <body>
    <div class="container">
    <br />
    @if (\Session::has('success'))
      <div class="alert alert-success">
        <p>{{ \Session::get('success') }}</p>
      </div><br />
     @endif
    <table class="table table-striped">
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Price</th>
        <th colspan="2">Action</th>
      </tr>
    </thead>
    <tbody>
      @foreach($products as $product)
      <tr>
        <td>{{$product['id']}}</td>
        <td>{{$product['name']}}</td>
        <td>{{$product['price']}}</td>
        <td><a href="{{action('ProductControll
er@edit', $product['id'])}}" class="btn btn-wa
rning">Edit</a></td>
        <td>
          <form action="{{action('ProductContr
oller@destroy', $product['id'])}}" method="pos
t">
            {{csrf_field()}}
            <input name="_method" type="hidden
" value="DELETE">
            <button class="btn btn-danger" typ
e="submit">Delete</button>
          </form>
        </td>
      </tr>
      @endforeach
    </tbody>
  </table>
  </div>
  </body>
</html>
```

So, when you hit the URL: **http://localhost:8000/products**



ID	Name	Price	Action	
1	Mobile	20000	Edit	Delete
2	Sbar1	23321	Edit	Delete
3	Mobile	23321	Edit	Delete

Step 8: Make an edit view for update the products.

Our step will be to add the edit function in **ProductController.php** file and put the following code in it.

```
// ProductController.php

 /**
   * Show the form for editing the specified
resource.
   *
   * @param  int  $id
   * @return \Illuminate\Http\Response
   */
  public function edit($id)
  {
    $product = Product::find($id);
    return view('products.edit',compact('p
roduct','id'));
  }
```

Now, make an **edit.blade.php** file inside **resources >> views >> products**

```
<!-- edit.blade.php -->

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Laravel 5.5 CRUD Tutorial With Exam
ple From Scratch </title>
    <link rel="stylesheet" href="{{asset('css/
app.css')}}">
  </head>
  <body>
    <div class="container">
      <h2>Edit A Product</h2><br  />
      @if ($errors->any())
      <div class="alert alert-danger">
          <ul>
              @foreach ($errors->all() as $err
or)
                  <li>{{ $error }}</li>
              @endforeach
          </ul>
      </div><br />
      @endif
      <form method="post" action="{{action('Pr
oductController@update', $id)}}">
        {{csrf_field()}}
        <input name="_method" type="hidden" va
lue="PATCH">
        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <label for="name">Name:</label>
            <input type="text" class="form-con
trol" name="name" value="{{$product->name}}">
          </div>
        </div>
        <div class="row">
          <div class="col-md-4"></div>
            <div class="form-group col-md-4">
              <label for="price">Price:</label
>
              <input type="text" class="form-c
ontrol" name="price" value="{{$product->price}
}">
            </div>
          </div>
        </div>
        <div class="row">
```

```
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <button type="submit" class="btn b
tn-success" style="margin-left:38px">Update Pr
oduct</button>
          </div>
        </div>
      </form>
    </div>
  </body>
</html>
```

A further step would be to code the update function.

```
// ProductController.php

  /**
     * Update the specified resource in storag
e.
     *
     * @param  \Illuminate\Http\Request  $requ
est
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $
id)
    {
        $product = Product::find($id);
        $this->validate(request(), [
          'name' => 'required',
          'price' => 'required|numeric'
        ]);
        $product->name = $request->get('name')
;
        $product->price = $request->get('price
');
        $product->save();
        return redirect('products')->with('suc
cess','Product has been updated');
    }
```

Step 9: Delete the product.

```
// ProductController.php

  /**
     * Remove the specified resource from stor
age.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        $product = Product::find($id);
        $product->delete();
        return redirect('products')->with('suc
cess','Product has been  deleted');
    }
```

Our **Laravel 5.5 CRUD Tutorial With Example From Scratch** is over. I put the Github Link over here.

DOWNLOAD ON GITHUB

Github Steps:

1. Clone the repository.
2. Type this command: **composer update**
3. Configure the database in the **.env** file.
4. Start the Laravel development server: **php artisan serve**
5. Switch to the URL: **http://localhost:8000/products/create**

Laravel | Laravel 5.5 | Laravel 5.5 CRUD | Laravel 5.5 Tutorial | Laravel Tutorial

Krunal

I am Web Developer and Blogger. I

have created this website for the web developers to understand complex concepts in an easy manner.

35 Comments

Christopher Pecoraro Says *1 year ago*

Remember that we can now use “php artisan make:model -a” to create “everything” (factory, migration, model, and resource controller, all in one. It’s not even documented yet 🙂
Here’s the relevant code from ModelMakeCommand.php in the laravel/framework core.

```
if ($this->option(‘all’)) {
$this->input->setOption(‘factory’, true);
$this->input->setOption(‘migration’, true);
$this->input->setOption(‘controller’, true);
$this->input->setOption(‘resource’, true);
}
```

Dirk Bertels Says *1 year ago*

Wow, a very useful tutorial for newcomers – This has helped me a lot. Thanks for putting in the effort!

Global Says *1 year ago*

Thanks / great tutorial hopwfully you make advanced as we learned this

Fredy Sanchez Says *1 year ago*

Hello, I get the following error: “Class ‘App\Http \Controllers\Product’ not found”

Krunal Says *1 year ago*

write this in the controller : use AppProduct;

Linish Says *1 year ago*

Nice tutorial. Worked Perfectly 🙂 Thanks for putting in the effort!

Not Says *1 year ago*

man, it’s very awesome tutorial. thank you so much

Andres Lozano Says *1 year ago*

i am getting this error when i put the name and the price and click on Add product button, what could be the problem?

```
(1/1) ErrorException
Argument 1 passed to Illuminate\Database
\Eloquent\Builder::create() must be of the type
array, null given, called in C:\Users\Andres
\Desktop\laravel\Laravel55\vendor\laravel
\framework\src\Illuminate\Database\Eloquent
\Model.php on line 1374 and defined
```

Samuel Says *11 months ago*

Change the store function into

```
public function store(Request $request)
{
$product = $this->validate(request(), [
‘name’ => ‘required’,
‘price’ => ‘required|numeric’
]);
Product::create($request->all());
return back()->with(‘success’, ‘Product
has been added’);
```

}

Valeriy Kalinov Says *1 year ago*

Thank you. Very good and easy.

Sandy Says *1 year ago*

Hi, I am getting error like this "InvalidArgumentException View [products.create] not found."

My file name and code like this tutorial, any solutin everyone ? Thx

Krunal Says *1 year ago*

you need to make one directory inside resources/views folder called products. Inside make one file called create.blade.php

Luigi Says *11 months ago*

same error.nothing changed and I already had create.blade.php. any suggestion?

Kenobi8701 Says *1 year ago*

How about a delete confirm box before actually deleting?

Jumar Says *1 year ago*

why my edit is blank ?

Aboudi Says *1 year ago*

thanks

Raizercrow Says *11 months ago*

This is a very useful tutorial, simple but clear enough for beginners (like me)!
Thanks a lot!

Rendy Says *11 months ago*

i’m beginner, its just i cant see create button or code so i add them for my self, cheers, and not a problem

Thoai Says *11 months ago*

Thank you for helping us. Very useful

John Says *10 months ago*

Am grateful, very grateful

Vikash Kumar Singh Says *9 months ago*

nice tutorial

Sebastian Says *9 months ago*

I’ve this problem: Undefined variable: id
when I want to Update in edit.blade.php
I put the same code :
In my controller I´ve the same code of tutorial.

Krunal Says *9 months ago*

please pass the id like this: return

view('products.edit',compact('product','id'));
You can access id variable inside edit.blade.php file.

Aashar Muneer Says *9 months ago*

Helllo bro you Doing Wonder Full Work ..
i have a one problem to follow your tutorial ..
when i am submit form this error display:
" The page has expired due to inactivity.
Please refresh and try again. "

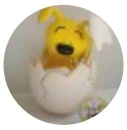
what should i do now..

Krunal Says *9 months ago*

You need to include CSRF token. If your laravel version is 5.6 then after form tag writes like this: @csrf. It will help you to get rid of that error.

Budi Says *9 months ago*

incredible tut, thanks a lot..helpfull for newbi like me...good job



Bibi_the_froggie Says *9 months ago*

Good job bro, that's what I call a tutorial.
A pure check list of things to do and that you must always do, that is what a tutorial always must be.
What ever is complicated your aplication, in fact a web application is just a formular to create datas, validation of the created datas, insert datas, show datas, modify datas, delete datas, nothing less, nothing more.

I am new in Laravel, and my OS is W7 and I use different WAMP such as Laragon or Wampserver64.
At Step 2, with Laragon I did not have any

problem with the command :
>php artisan migrate
But with Wampserver64 (who is probably the WAMP environement who is the most used on Windows)
I received this bad answer :
Illuminate\Database\QueryException : SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length is 1000 bytes (SQL: alter table \`users\` add unique \`users_email_unique\`(\`email\`))

To fix this bug, I edited config\database.php and changed
‘mysql’ => [
…
‘charset’ => ‘utf8mb4’,
‘collation’ => ‘utf8mb4_unicode_ci’,
…
],
To
‘mysql’ => [
…
‘charset’ => ‘utf8’,
‘collation’ => ‘utf8_unicode_ci’,
…
],
After I DROP manually the users TABLE because the command
>php artisan migrate:reset
says : Nothing to rollback
and relaunched the command :
>php artisan migrate
and evrething was fine.

PS : If you want to modify your tutorial, I can send you the complete result of the two commands php artisan migrate.

PS2 : All my apologies for my english, I learnt it in a far far away galaxy named french public school.

Nazim Says *9 months ago*

Krunal,

It’s good job for a intermediate developer!

Nazim

Donald E Brady Says *8 months ago*

A real nice tutorial. Never touched laravel before this…

Calabar Lion Says *7 months ago*

You are amazing man!
Thanks so much for this tutorial.

Leo B Says *7 months ago*

I get the following error. tried all config:clear, cache:clear, composer install and composer update.
But not yet fixed. What should I do? Thanks

Warning: require_once(C:\xampp\htdocs\Laravel55/../bootstrap/app.php): failed to open stream: No such file or directory in C:\xampp\htdocs\Laravel55\index.php on line 38

Fatal error: require_once(): Failed opening required ‘C:\xampp\htdocs\Laravel55/../bootstrap/app.php’ (include_path=’C:\xampp\php\PEAR’) in C:\xampp\htdocs\Laravel55\index.php on line 38

Mohammad Etemaddar Says *6 months ago*

Best tutorial I’ve found yet. Thanks a lot.

Krunal Says *6 months ago*

Thanks, keep learning and sharing!!

Mohamad Zahili Says *5 months ago*

Nice Tutorial

Harjit Says *4 months ago*

Hi Krunal,
When we click on the add product button in create.blade.php, how it call to store method of ProductController.

This site uses Akismet to reduce spam. Learn how your comment data is processed.