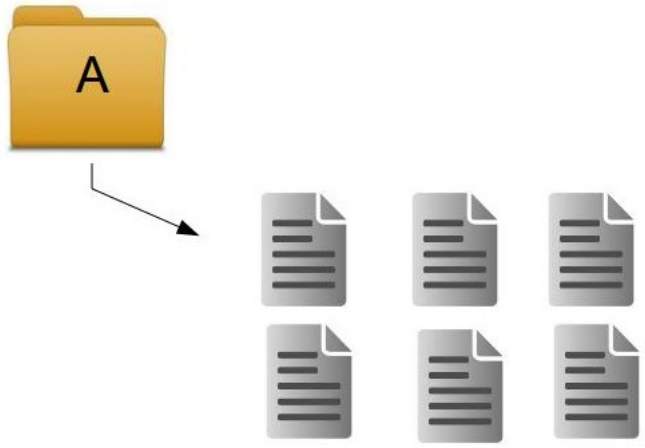
How To Upload Multiple Files in Laravel 5.4

February 23, 2017 25 Comments



How To Upload Multiple Files in Laravel 5.4

By: Povilas Korop

File upload is one of the most commonly used features in web-projects. And it seems pretty easy – form, submit, validation, store. But it gets a little more complex if you want to allow your users to upload more than one file with one input – let's see how it's done in Laravel.

1. Preparing the database

Let's have a simple and probably the most often example – product and its photos. To simplify let's take only necessary field – product will have a title and N photos. So we create models with migrations – with these commands:

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

Notice: *this* ***-m*** *parameter means that migration should be automatically created with the model – more about it here.*

File **app/Product.php**:

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    protected $fillable = ['name'];
}
```

SEARCH OUR ARTICLES

Search … SE

AUTHOR: POVILAS K

Laravel/PHP web-develop
14 years experience in web-pr
Now leading an in-house team of

See our projects/clients | Mee

Want to hire me or my team? En
povilas@laraveldaily.cor

FOLLOW ON TWITTER

Migration of **products** table:

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateProductsTable extends Migration
{
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('products');
    }
}

```

Now, a little more complicated thing – for **products_photos** table we have a model **app/ProductsPhoto.php** with a relationship method:

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class ProductsPhoto extends Model
{
    protected $fillable = ['product_id', 'filename'];

    public function product()
    {
        return $this->belongsTo('App\Product');
    }
}

```

And, finally, migration for **products_photos** table:

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateProductsPhotosTable extends Migration
{
    public function up()
    {
        Schema::create('products_photos', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('product_id')->unsigned();
            $table->foreign('product_id')->references('id')->on('products');
            $table->string('filename');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('products_photos');
    }
}

```


2. Structure: Routes, Controllers and Views

We start with a default Laravel project and will create two pages – upload form and result page. Basically, we need two routes and one controller.

File **routes/web.php** (you can choose different URLs or method names):

```
Route::get('/upload', 'UploadController@uploadForm');
Route::post('/upload', 'UploadController@uploadSubmit');

```

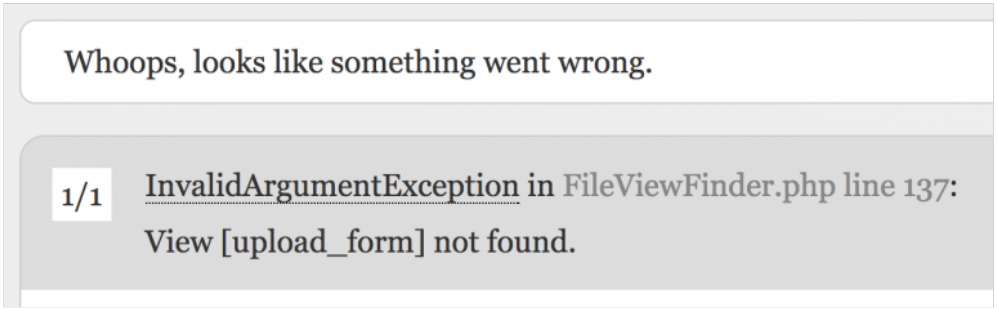
Next we generate a controller – there's an artisan command for that:

```
php artisan make:controller UploadController
```

And for now we fill the methods with something like this:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UploadController extends Controller
{

    public function uploadForm()
    {
        return view('upload_form');
    }

    public function uploadSubmit(Request $request)
    {
        // Coming soon...
    }

}
```

Let's load URL **/upload** in the browser.

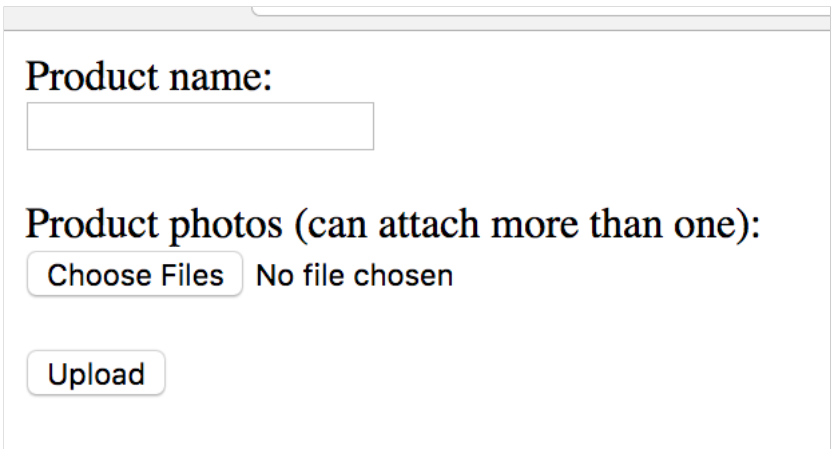


Got this error **"View [upload_form] not found."**? GOOD. It means two things:

- Route and Controller work ok;
- It's time to create the missing view

3. View with Upload Form

For our form let's use default HTML tags without any Form packages – our form will look like this:



```
<form action="/upload" method="post" enctype="multipart/form-data">
    {{ csrf_field() }}
    Product name:
    <br />
    <input type="text" name="name" />
    <br /><br />
    Product photos (can attach more than one):
    <br />
    <input type="file" name="photos[]" multiple />
    <br /><br />
    <input type="submit" value="Upload" />
</form>
```

4. Validation Request

Now let's get back to our Controller and start filling in **uploadSubmit()** method. Before actually uploading the file, we probably want to validate the form data.

Let's say that we have a **required** product name and files are images not bigger than 2 MB. We create a request file for it:

```
php artisan make:request UploadRequest
```

And then open the file **app/Http/Requests/UploadRequest.php**.

```
class UploadRequest extends FormRequest
{

    public function authorize()
    {
        return false;
    }

    public function rules()
    {
        return [
            //
        ];
    }
}
```

First, we change **authorize()** method to **return true**, otherwise we would get "unauthorized action" on submit.

Now, what we're interested in is **rules()** method. We need to fill that array with our own rules of validation.

First, we add product name as required, so we have:

```
return [
    'name' => 'required'
];
```

Now, we need to fill the rules for photos. If we had only one field **photo**, it would look like this:

```
return [
    'name' => 'required',
    'photo' => 'image|mimes:jpeg,bmp,png|size:2000'
];
```

Notice: *you can see all the validation rules (and add more) on this page of official documentation.*

But we have more than one file, right? So we can fill in the **rules()** array dynamically! With a loop, something like this:

```
    public function rules()
    {
        $rules = [
            'name' => 'required'
        ];
        $photos = count($this->input('photos'));
        foreach(range(0, $photos) as $index) {
            $rules['photos.' . $index] = 'image|mimes:jpeg,bmp,png|max:2000';
        }

        return $rules;
    }

```

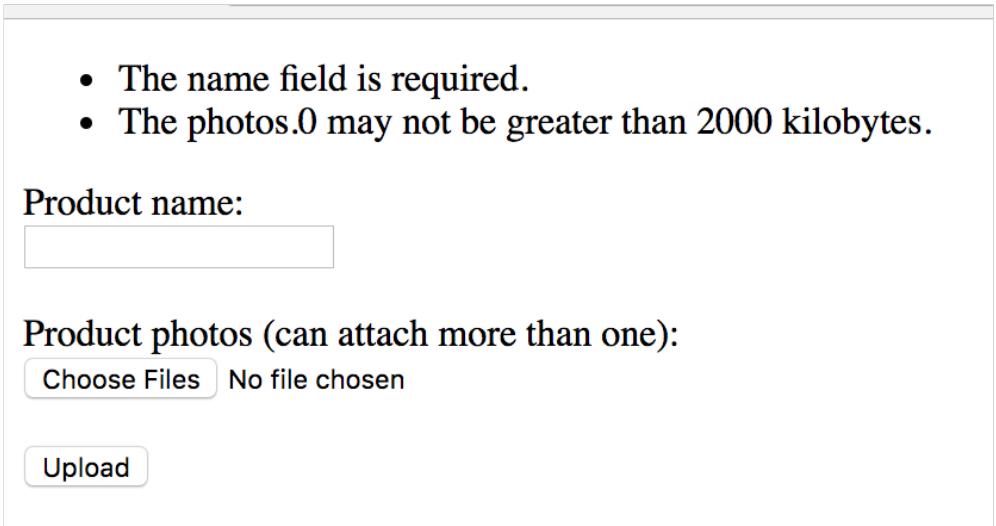
Next step – we need to apply this Request class to the Controller, so we change the parameter of the method. Also don't forget to include it in **use** on top – PhpStorm did it automatically for me.

```
namespace App\Http\Controllers;

use App\Http\Requests\UploadRequest;

class UploadController extends Controller
{

    public function uploadForm()
    {
        return view('upload_form');
    }

    public function uploadSubmit(UploadRequest $request)
    {
        // Coming soon...
    }

}

```

Finally, let's show the validation errors, if there are any. Almost copy-paste from the original Laravel documentation:

```
    @if (count($errors) > 0)
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    @endif

    <form action="/upload" method="post" enctype="multipart/form-data">
    ...

```

Now, if I don't enter product name and upload an image which is too big – here's what I will see:



5. Storing Data and Files

After all this hard work – let's finally upload the data. We have two things to take care of – database and file storage. Let's start with the files, and here we need to know about **filesystems** in Laravel.

There is a file **config/filesystems.php** where you specify the locations for your file storage. It allows you to easily configure external storage like Amazon S3, but we won't do it in this tutorial – we will stick with default parameters:

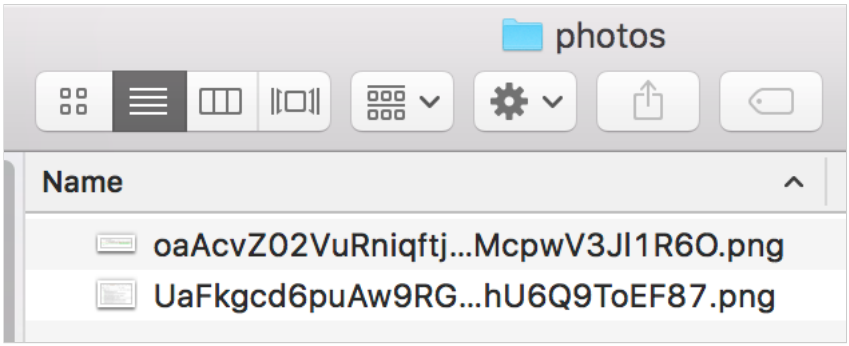
```
return [

    'default' => 'local',

    'disks' => [

        'local' => [
            'driver' => 'local',
            'root' => storage_path('app'),
        ],

    // ...
```

Basically, it means that all your files will be stored in **/storage/app** folder. Not in **/public**, so safely and not accessible directly from browser URL. But you can change it here, if needed.

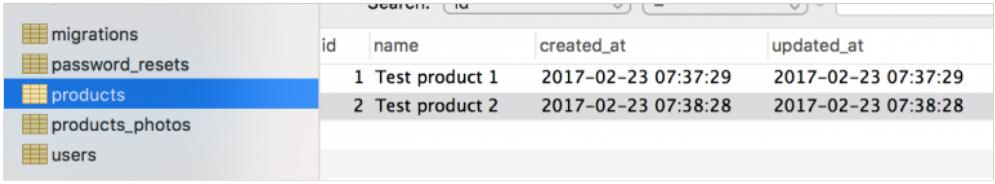
The file upload itself is incredibly simple in Laravel. Our whole Controller method will look like this:

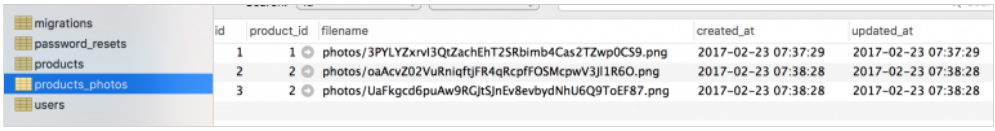
```
    public function uploadSubmit(UploadRequest $request)
    {
        $product = Product::create($request->all());
        foreach ($request->photos as $photo) {
            $filename = $photo->store('photos');
            ProductsPhoto::create([
                'product_id' => $product->id,
                'filename' => $filename
            ]);
        }
        return 'Upload successful!';
    }
```

As you can see, the only method for uploading file is **$photo->store('photos')**. Parameter means what sub-folder to use for storage, so in this case it will be **/storage/app/photos**. Filename will be created dynamically to a random string. Like this:



And here's how the result looks in database:





Basically, that's it! Of course, you can go a step further and add more validation rules, image resize, thumbnails and more functions. But that's a topic for another article, I can recommend a few packages to work with images:

- Intervention Image
- Spatie / Image

Finally, if you want to play around with the code listed here, I've prepared you the archive to download. Enjoy!

25 thoughts on "How To Upload Multiple Files in Laravel 5.4"

Peter says:
February 23, 2017 at 8:53 am

Question: instead of writing validation by counting inputs and making foreach loop, isn't better to use validation of array based input? See https://laravel.com/docs/5.4/validation#validating-arrays

REPLY

Povilas Korop says:
February 23, 2017 at 9:30 am

Good question, Peter. For some reason it didn't work for me when I tried photos.* => XXX. Probably with files it works differently. Maybe you would write the code and I would update the article? Thanks a lot!

REPLY

Michael Stivala says:
February 23, 2017 at 8:56 pm

Array validation wouldn't work with just "photos.*", but you could use "photos.*.photo" instead, and rename the file input to "photos[][photo]"

REPLY

Povilas Korop says:
February 24, 2017 at 5:58 am

Thanks a lot Michael! I guess I will leave article the same, and people who would read comments will see both ways.

REPLY

Nemanja says:
April 7, 2017 at 7:53 am

This is also good way to validate various number of photos in array , that you get back from input request. It worked for me. Cheers.

```
$validate = count($photo);
$photos = $input['photos'];

$this->validate(request(),[
'photos' => 'required|min:'.$validate
]);
```

REPLY

Ken A. Fu says:
February 26, 2017 at 4:25 am

Thanks for the detailed write up! Just one minor thing I'd argue, I feel that the naming of the second model and table would make a lot more sense as "ProductPhoto" and "product_photos" respectively, instead of using an awkward convention. It really makes no sense to use plural forms for models, as your model basically describes an object (singular), any relation to another model which might be plural is described by a property/method of the model, not the name itself. And in this case, only your product model is "really" aware that there is a 1 to n relation, so it makes even less sense, on top of that, if you're going plural, it should've been ProductPhotos, because you have more photos per product, but again, that should not be conveyed through the name.

For your own sanity while coding, using names that are semantically and grammatically correct will lead to less confusion and code that reads naturally.

REPLY

Povilas Korop says:
February 26, 2017 at 7:36 am

Thank you, Ken, for the detailed opinion. In general, I agree, that is my personal preference to have database tables "***S" and ***S_***S" therefore I've used models like this.

Don't have time to change the article now, including screenshots and code files, but will use this logic for the future articles.

REPLY

Adrian says:
February 26, 2017 at 9:43 am

Better validate is use 'photos.*.required' etc. For what you did a loop for this?

REPLY

Povilas Korop says:
February 26, 2017 at 11:13 am

Please try the code as you written "photos.*.required" and tell me if it works with file upload fields. And see other comments above.

REPLY

fulgorit says:
February 26, 2017 at 4:57 pm

from the archive
I modified web.php
Route::get('/', function () {
return view('welcome');
});
to have the upload form
I submited the form but error:

TokenMismatchException in VerifyCsrfToken.php line 68:

How can I correct this ?

REPLY

Povilas Korop says:
February 26, 2017 at 5:00 pm

You probably don't have this line in your form:
{{ csrf_field() }}

Or any other CSRF token.

REPLY

Tek Raj Shrestha says:
February 27, 2017 at 3:19 pm

Thank you for the code and help here. But I was in need of code to access the stored pictures for that particular product.
Say i have 10 products and each product have 5 or more images then i need to show the images when that particular product is clicked.That is at home page we have only the list of product with thumbnails and when that thumbnails is clicked the details of that product should come with the related images, Can you help me with that? Please... 🙂

REPLY

Povilas Korop says:
February 27, 2017 at 3:31 pm

Hi Tek Raj,

How exactly can I help you? What code have you tried and it didn't work? Or you want me to write a full tutorial for this? I think it's pretty easy to find some related tutorials on this topic.

REPLY

Tek Raj Shrestha says:
February 27, 2017 at 3:59 pm

I tried a lot. I couldn't find it even in stack overflow. Yours is the first tutorial i found for storing image/files using relationship. Up to uploading, I completed and worked fine I just needed is the accessing part. You could help me either by providing me the technique following your uploading tutorial or you could write a sample code. Either way its fine for me. I would be very grateful to you. I am stuck with this problem since 3 days. And finally i found yours tutorial so 🙂

REPLY

Tek Raj Shrestha says:
February 27, 2017 at 5:48 pm

This is my Route:

```
Route::get('/upload', 'UploadController@uploadForm');
Route::post('/upload', 'UploadController@uploadSubmit');
Route::get('/upload/{id}','UploadController@show');
```

This is my show function inside UploadController:

```
public function show($id)
{
// $data=Product::find($id);
$data =ProductsPhoto::with('Product')->find($id);
return view('showpage',compact('data'));
}
```

This is my showpage code:

```
{{$data->name}}
name}}"/>
{{– images}}" alt="a" title="a" id="wows1_0"/>
images}}" alt="a" title="a" id="wows1_0"/> –}}
```

REPLY

Tek Raj Shrestha says:
February 27, 2017 at 5:49 pm

This is showpage :

{{$data->name}}
name}}"/>
{{– images}}" alt="a" title="a" id="wows1_0"/>
images}}" alt="a" title="a" id="wows1_0"/> –}}

REPLY

Tek Raj Shrestha says:
February 27, 2017 at 5:53 pm

in controller in show function when i use "$data=Product::find($id);" I can access the product name but if "$data =ProductsPhoto::with('Product')->find($id);" is used i cant get name nor image.I think my img src is wrong. what is the img src for the image stored in storage/app/photos in laravel.

REPLY

Povilas Korop says:
February 27, 2017 at 8:13 pm

This thread will help you – to get to /storage/app/photos folder you need to create a separate route. Which, by the way, you can protect by some middleware or other logic.
http://stackoverflow.com/questions/30191330/laravel-5-how-to-access-image-uploaded-in-storage-within-view

REPLY

Tek Raj Shrestha says:
February 28, 2017 at 4:58 am

okay I will give it a shot. Thank you 🙂

REPLY

Tek Raj Shrestha says:
February 28, 2017 at 7:36 am

please provide a tutorial for displaying all images related to particular product. We couldn't do it.

REPLY

Povilas Korop says:
February 28, 2017 at 7:41 am

Sorry, we don't plan to write tutorial about this, have other topics on the list.

REPLY

jamalcoder says:
March 16, 2017 at 12:18 am

if you use just image mime type for validation in your code then this code can be exploited by uploading a crafted image with php extension

search in google for image mime type exploit

REPLY

david says:
April 4, 2017 at 1:11 pm

I was gonna write the same, I think this code is insecure as a user can upload a .php file crafted that appears to be an image, and because this code keeps the original extension

REPLY

David says:
April 4, 2017 at 1:36 pm

It seems the validator check the extension too…
http://stackoverflow.com/questions/29842625/laravel-5-mime-validation

REPLY

sara says:
May 11, 2017 at 1:11 pm

iam getting this error

"FileException in UploadedFile.php line 251:
The file "Lighthouse.jpg" was not uploaded due to an unknown error. "

REPLY

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Receive Email Notifications?

yes, replies to my comment ▾ instantly ▾

Or, you can subscribe without commenting

POST COMMENT

Brand of **Web Coder Pro Ltd** company

London office:
Unit 210, No.11 Burford Road
London E15 2ST
United Kingdom
+44 20 3286 4847

Lithuanian office:
Ozo st. 4-407, Vilnius
Lithuania
+370 622 18617

Email: info@laraveldaily.com
Skype: LaravelDaily