filestack

All Posts

Search blog

A Step-By-Step Guide To Laravel File Upload

Posted on March 15, 2022 by Shamim

Uploading Files in Laravel is as easy as writing a few lines of code. All you need to do is create a file viewer where your user selects a file to upload, and a Laravel controller does the rest. There are many file upload options, but with Laravel file upload and the Filestack PHP SDK framework, you can perform the job efficiently.

Laravel is a robust, open-source PHP framework. It is well known for its simplicity and elegant toolkit for creating full-featured web applications that upload files, images, and videos. If you are familiar with PHP and want to make a Laravel file upload application, then the guide is for you.

Let's take a look at three different types of Laravel file upload, look at some examples and discuss the simple steps involved. Then we will cover how to upload files using the advanced version of Laravel 8 with database validation and storage. Finally, we will look at how the Filestack File Uploader may be your best option with Filestack PHP SDK.

Let's start!

Table of Contents

What will you learn from the post?

It will cover the following steps to Laravel file uploads:

- Installing a Laravel project
- Creating and configuring routes
- Creating a blade file
- Migrating databases in Laravel
- Implement validation on file upload components
- Displaying file upload status messages
- Enabling uploading specific file types. For example .csv, .txt, .xlx, .xls, .pdf with file size limitation max up to 2MB
- Storing uploaded files in the database.

What will be the three ways of Laravel file upload you will learn in this guide?

1. **The Simple Laravel/PHP Way:** The most straightforward way of adding a PHP uploader to your service. The benefit is that you have complete control of the uploaded files.
2. **Laravel 8 File Upload with validation and store in database:** The sensible way to add a PHP uploader to your service, and you have complete control of the uploaded files.
3. **Upload file with Filestack's PHP SDK**: This is the easiest to add PHP upload functionality. The upside here is that you don't have to manage the complex file upload infrastructure behind the scenes.

How do you implement a simple Laravel file upload?

To implement a simple laravel file uploader, you need to generate a file input in a view file. You can do this by adding the following code.

```
Form::file('file_name');
```

In Form::open(), you need to add 'files' =>' true' as shown below. This will enable the form to be uploaded in multiple parts.

```
Form::open(array('url' => '/uploadfile','files'=>'t
```

Example

Step 1 – Create a view file named resources/views/uploadfile.php and then copy the following code into that file.

```
<html>
   <body>
      <?php
         echo Form::open(array('url' => '/uploadfil
         echo 'Select the file to upload.';
         echo Form::file('image');
         echo Form::submit('Upload File');
         echo Form::close();
      ?>
   </body>
</html>
```

For more information, you can visit https://github.com/filestack/filestack-phphere:

Step 2 – Create a controller called UploadFileController by executing the following command.

```
php artisan make:controller UploadFileController --
```

Step 3 – After successfully executing the command, you will receive the output.

Step 4 – Next, copy the following lines of code in the app/Http/Controllers/UploadFileController.php file.

app/Http/Controllers/UploadFileController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\Http\Controllers\Controller;

class UploadFileController extends Controller {
   public function index() {
      return view('uploadfile');
   }
   public function showUploadFile(Request $request)
      $file = $request->file('image');

      //Display File Name
      echo 'File Name: '.$file->getClientOriginalNa
      echo '<br>';

      //Display File Extension
      echo 'File Extension: '.$file->getClientOrigi
      echo '<br>';

      //Display File Real Path
      echo 'File Real Path: '.$file->getRealPath();
      echo '<br>';

      //Display File Size
      echo 'File Size: '.$file->getSize();
      echo '<br>';

      //Display File Mime Type
      echo 'File Mime Type: '.$file->getMimeType();

      //Move Uploaded File
      $destinationPath = 'uploads';
      $file->move($destinationPath,$file->getClient
   }
}
```

Step 5 –Add the following lines in app/Http/routes.php.

```
Route::get('/uploadfile','UploadFileController@inde
Route::post('/uploadfile','UploadFileController@sho
```

Step 6 – Test the upload file functionality.

How do I implement Laravel 8 File Upload with validation and store in database?

This Laravel file upload method generates two routes. First, it creates a form with the get method. The second route is for the file upload or post file upload data.

To do this, develop a simple form using Bootstrap and its Form UI component. This allows you to choose a file, photo, or video for upload to your storage > public > uploads folder. You can also configure the database model and store the file path and name in the MySQL database.

How will you install the project?

First, open the command-line tool from the start menu and execute the following command to create a Laravel project from scratch.

```
composer create-project laravel/laravel --prefer-di
```

Then, go into the freshly installed Laravel project directory.

```
cd laravel-file-upload
```

How do you connect to a database?

To upload files to storage in Laravel, use MAMP or XAMPP as a local web server, define a database name in MySQL, and add the correct configuration in the .env file.

How do you create a file model and configure migration?

To store the uploaded file information, open a migration file to define the table values in the database.

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

Then, go to database/migrations/timestamp_create_files_table file and define the table values

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
class CreateFilesTable extends Migration
{
    public function up()
    {
        Schema::create('files', function (Blueprint
            $table->id();
            $table->string('name')->nullable();
            $table->string('file_path')->nullable()
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('files');
    }
}
```

After you are done there, add the $fillable property in the File model. Open the app/Models/File.php file and place the following code.

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFacto
use Illuminate\Database\Eloquent\Model;

class File extends Model
{
    use HasFactory;
    protected $fillable = [
        'name',
        'file_path'
    ];
}
```

Now, you are set to run the migration. You can also see the update in the MySQL database.

How do you create routes in Laravel?

Go to routes: web.php and create two routes. The first route handles the form creation, whereas the second route stores the file in the MySQL database.

```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\FileUpload;
/*
|--------------------------------------------------
| Web Routes
|--------------------------------------------------
|
| Here is where you can register web routes for you
| routes are loaded by the RouteServiceProvider wit
| contains the "web" middleware group. Now create s
|
*/
Route::get('/upload-file', [FileUpload::class, 'cre
Route::post('/upload-file', [FileUpload::class, 'fi
```

How do you create a file upload controller?

To create a file upload controller that defines the business logic for file uploads and storage in Laravel, follow these steps.

Execute this command to create the controller:

```
php artisan make:controller FileUpload
```

Open the app/Http/Controllers/FileUpload.php file. Here you define the two methods that handle the file upload. The first method renders the view via FileUpload controller, and the second, the fileUpload() method, checks the validation, mime type, or file size limitation.

This method also stores the file in your storage/public/uploads folder and saves the file name and path in your database.

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\File;
class FileUpload extends Controller
{
  public function createForm(){
    return view('file-upload');
  }
  public function fileUpload(Request $req){
```

```
        $req->validate([
        'file' => 'required|mimes:csv,txt,xlx,xls,p
        ]);
        $fileModel = new File;
        if($req->file()) {
            $fileName = time().'_'.$req->file->getC
            $filePath = $req->file('file')->storeAs
            $fileModel->name = time().'_'.$req->fil
            $fileModel->file_path = '/storage/' . $
            $fileModel->save();
            return back()
            ->with('success','File has been uploade
            ->with('file', $fileName);
        }
   }
}
```

How do you create a Blade File in Laravel?

To create a blade file, start with creating a view and the file upload form.

Create a resources\views\file-upload.blade.php file, then place the following code inside it.

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-wid
    <link rel="stylesheet" href="https://stackpath.
    <title>Laravel File Upload</title>
    <style>
        .container {
            max-width: 500px;
        }
        dl, ol, ul {
            margin: 0;
            padding: 0;
            list-style: none;
        }
    </style>
</head>
<body>
    <div class="container mt-5">
        <form action="{{route('fileUpload')}}" meth
          <h3 class="text-center mb-5">Upload File
            @csrf
            @if ($message = Session::get('success')
            <div class="alert alert-success">
                <strong>{{ $message }}</strong>
            </div>
          @endif
          @if (count($errors) > 0)
            <div class="alert alert-danger">
```

```
                <ul>
                    @foreach ($errors->all() as $er
                      <li>{{ $error }}</li>
                    @endforeach
                </ul>
            </div>
          @endif
            <div class="custom-file">
                <input type="file" name="file" clas
                <label class="custom-file-label" fo
            </div>
            <button type="submit" name="submit" cla
                Upload Files
            </button>
        </form>
    </div>
</body>
</html>
```

This displays the file upload form, as well as a status message.

How can you start a Laravel application?

```
php artisan serve
```

That's it!

How to implement a File upload with Filestack?

Now we will show you how to use Filestack to upload a file. Filestack is an advanced file upload API and service that securely stores files in the cloud.

Why use a third-party service like Filestack over building it yourself?

There are several reasons to use a service like Filestack over building your own service. First of all, It is risky to deal independently with scaling, security, and maintenance. Instead, you can use Filestack to scale quickly and painlessly in addition to minimizing risks. Moreover, Filestack has built-in protection and time-saving maintenance features. You don't need to worry when you use Filestack. Because of this, you can focus on developing rather than maintaining, scaling, and securing your applications.

You can also try Filestack for free. The free plan as well as handles up to 100 monthly uploads with 1GB storage and 1GB bandwidth.

So let's get started:

1. Are you registered with Filestack?

If you aren't, then you need to sign up for a Filestack account on our registration page. After registering, log in and get your API Key.

2. Are you ready to start uploading?

Now let's integrate the JavaScript file uploader widget from the Filestack library. The widget allows you to connect and upload from a variety of sources. For instance, if you want to upload from a URL or social media. Simply replace the contents of **index.html** with the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>PHP File Upload</title>
</head>
<body>
    <style>
      .picker-content{
        height:300px;
        width:200px;
      }
    </style>
    <script src="//static.filestackapi.com/filestac
    <script type="text/javascript">
      document.addEventListener("DOMContentLoaded",
        const client = filestack.init(YOUR_API_KEY)

        let options = {
          "displayMode": "inline",
          "container": ".picker-content",
          "accept": [
            "image/jpeg",
            "image/jpg",
            "image/png"
          ],
          "fromSources": [
            "local_file_system"
          ],
          "uploadInBackground": false,
          "onUploadDone": (res) => console.log(res)
```

```
        };

        picker = client.picker(options);
        picker.open();
      });
    </script>
    <div class="picker-content"></div>
</body>
</html>
```

To test, open your page and then upload a file using the upload widget. After the upload, you can log into your Filestack dashboard and see your newly uploaded file.

That's it! You don't even need the server to handle the file. Filestack is much better for scalability, security, and maintenance.

Why use the optional Filestack PHP Library?

The example above is the simplest example of a Filestack file upload, but what if you want to access the file on your server and run some kind of post-processing? That is where you use the Filestack PHP library.

You can install the Filestack PHP library using composer. If, however, you don't have Composer installed then you can install it by going to the folder you created originally and following the documentation. (see link [1] for official documentation):

```
php -r "copy('https://getcomposer.org/installer', '
php -r "if (hash_file('sha384', 'composer-setup.php
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Once you are done, run `php composer.phar.`

After that, run `require --prefer-dist filestack/filestack-php` to install the Filestack SDK.

Now you have the Filestack library, let's make a new PHP script to check your uploaded file.

To do this, create a new file called **fileUploadFilestack.php** and add the following (making sure to change the **YOUR_API_KEY**, **YOUR_SECURITY_SECRET**, and **YOUR_FILE_HANDLE** variables):

```
<?php
  require __DIR__ . '/vendor/autoload.php';
  use Filestack\FilestackClient;
  $client = new FilestackClient(YOUR_API_KEY);
  $security = new FilestackSecurity(YOUR_SECURITY_S

  $file_handle = YOUR_FILE_HANDLE;

  # get tags with client
  $result_json = $client->getTags($file_handle);

  # get tags with filelink
  $filelink = new Filelink($file_handle, YOUR_API_K

  $json_result = $filelink->getTags();

  # get safe for work flag with filelink
  $json_result = $filelink->getSafeForWork();
?>
```

When you run this script, it checks to see if your file is safe for work and saves the result in the `$json_result` variable.

That's it!

Hope you have learned the basics of Laravel file upload.

Ready to get started building excellent Laravel File Upload applications using Filestack?

Sign up for free at Filestack to upload, transform and deliver content efficiently!

[1] https://getcomposer.org/download/