Handling File Uploads and Image Storage in Laravel

File uploads are one of the most common requirements in web applications. Laravel makes handling file uploads and image storage secure, simple, and flexible. In this guide, we'll walk through uploading files, validating them, storing them locally or in cloud services like Amazon S3, and building a simple UI to manage uploaded images.

Setting Up File Upload Routes and Controller

```
// routes/web.php
use App\Http\Controllers\FileController;

Route::get('/upload', [FileController::class, 'create']);
Route::post('/upload', [FileController::class, 'store']);Code language: PHP (php)
```

Here we define two routes: one for displaying the upload form and another for processing file uploads. The `FileController` will handle the logic.

Controller for Handling Uploads

```
// app/Http/Controllers/FileController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
```

```
use Illuminate\Support\Facades\Storage;

class FileController extends Controller
{
    public function create()
    {
        return view('upload');
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            'file' => 'required|image|mimes:jpg,jpeg,png,gif|max:2048'
        ]);

        $path = $request->file('file')->store('uploads', 'public');

        return back()->with('success', 'File uploaded successfully!')
                     ->with('path', $path);
    }
}Code language: PHP (php)
```

We validate that the uploaded file is an image with a maximum size of 2MB. The file is stored in the `storage/app/public/uploads` directory using the `public` disk configuration.

Blade Template for File Upload

```
<!-- resources/views/upload.blade.php -->
<!DOCTYPE html>
<html>
<head>
    <title>File Upload</title>
```

```
</head>
<body>
    @if(session('success'))
        <p style="color:green;">{{ session('success') }}</p>
        <img src="{{ asset('storage/' . session('path')) }}"
width="200">
    @endif

    <form action="/upload" method="POST" enctype="multipart/form-
data">
        @csrf
        <input type="file" name="file" required>
        <button type="submit">Upload</button>
    </form>

    @error('file')
        <p style="color:red;">{{ $message }}</p>
    @enderror
</body>
</html>Code language: PHP (php)
```

This Blade template provides a simple upload form. If the upload succeeds, it displays the uploaded image back to the user.

Storing Files in Amazon S3

```
// app/Http/Controllers/FileController.php
$path = $request->file('file')->store('uploads', 's3');Code language: PHP
(php)
```

By changing the second parameter to `s3`, files are stored in Amazon S3. Make sure you configure your `.env` with the correct `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_BUCKET`.

Listing Uploaded Files

```
// app/Http/Controllers/FileController.php
public function index()
{
    $files = Storage::disk('public')->files('uploads');
    return view('files.index', compact('files'));
}Code language: PHP (php)
```

With `Storage::disk('public')->files('uploads')` you can list all uploaded files in a given directory. This makes it easy to create a file manager or gallery.

```
<!-- resources/views/files/index.blade.php -->
<ul>
    @foreach($files as $file)
        <li>
            <img src="{{ asset('storage/' . $file) }}" width="100">
            {{ $file }}
        </li>
    @endforeach
</ul>Code language: PHP (php)
```

This Blade view displays thumbnails of uploaded images. You can extend it with delete buttons, rename functionality, or even drag-and-drop reordering.

Security Considerations

- Always validate files with `mimes` and `max` rules.
- Never store user uploads in the `public/` root without validation.
- Use signed URLs or policies for private file access.
- Limit maximum file size to avoid performance issues.

Following these security best practices helps prevent malicious uploads and keeps your app safe.

Wrapping Up

Handling file uploads and image storage in Laravel is straightforward thanks to the powerful `Storage` facade and built-in validation. You can support local, S3, or other drivers, and even build a UI to display uploaded files. By combining validation, secure storage, and Blade templates, you can safely add file management features to any application.

What's Next

If you found file uploads useful, explore these guides to go further:

- Implementing Image Upload and Processing in Laravel
- How to Build a File Manager in Laravel
- Best Practices for Storing API Keys Securely in Laravel