Implementing File Uploads with S3 and Local Disk Fallback in Laravel

This implementation will allow your application to use Amazon S3 for file storage in production while falling back to local storage in development environments.

1. Install Required Packages

First, install the AWS SDK for PHP:

```
composer require league/flysystem-aws-s3-v3
```

2. Configure Filesystems

Update your `.env` file with S3 credentials:

```
FILESYSTEM_DISK=s3

AWS_ACCESS_KEY_ID=your-access-key-id
AWS_SECRET_ACCESS_KEY=your-secret-access-key
AWS_DEFAULT_REGION=your-region
AWS_BUCKET=your-bucket-name
AWS_URL=your-bucket-url
AWS_ENDPOINT= # Only needed if using a custom endpoint (like
DigitalOcean Spaces)
```

Configure `config/filesystems.php`:

PROF

```
'disks' => [
    'local' => [
        'driver' => 'local',
        'root' => storage_path('app'),
        'throw' => false,
    ],

    'public' => [
        'driver' => 'local',
        'root' => storage_path('app/public'),
        'url' => env('APP_URL').'/storage',
        'visibility' => 'public',
        'throw' => false,
    ],

    's3' => [
```

```
            'driver' => 's3',
            'key' => env('AWS_ACCESS_KEY_ID'),
            'secret' => env('AWS_SECRET_ACCESS_KEY'),
            'region' => env('AWS_DEFAULT_REGION'),
            'bucket' => env('AWS_BUCKET'),
            'url' => env('AWS_URL'),
            'endpoint' => env('AWS_ENDPOINT'),
            'use_path_style_endpoint' => false,
            'throw' => false,
        ],
    ],

    'default' => env('FILESYSTEM_DISK', 'local'),
```

3. Create a File Service

Create a service class to handle file operations with automatic fallback:

```
// app/Services/FileUploadService.php

namespace App\Services;

use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Str;

class FileUploadService
{
    public function store($file, $directory = 'uploads', $disk = null)
    {
        $disk = $disk ?? $this->getDefaultDisk();

        $filename = $this->generateFilename($file);

        $path = Storage::disk($disk)->putFileAs(
            $directory,
            $file,
            $filename
        );

        return [
            'path' => $path,
            'url' => Storage::disk($disk)->url($path),
            'disk' => $disk,
            'filename' => $filename,
        ];
    }

    public function getDefaultDisk()
    {
        // Use S3 in production, local otherwise
```

```
        return app()->environment('production') ? 's3' : 'public';
    }

    protected function generateFilename($file)
    {
        return Str::random(40) . '.' . $file->extension();
    }

    public function delete($path, $disk = null)
    {
        $disk = $disk ?? $this->getDefaultDisk();

        return Storage::disk($disk)->delete($path);
    }

    public function exists($path, $disk = null)
    {
        $disk = $disk ?? $this->getDefaultDisk();

        return Storage::disk($disk)->exists($path);
    }

    public function getUrl($path, $disk = null)
    {
        $disk = $disk ?? $this->getDefaultDisk();

        return Storage::disk($disk)->url($path);
    }
}
```

4. Create a Form Request for Validation

PROF

```
// app/Http/Requests/StoreFileRequest.php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Validation\Rules\File;

class StoreFileRequest extends FormRequest
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'file' => [
```

```
                'required',
                File::types(['jpg', 'jpeg', 'png', 'gif', 'pdf', 'doc',
'docx'])
                    ->max(10 * 1024), // 10MB
            ],
        ];
    }
}
```

5. Create a Controller

```php
// app/Http/Controllers/FileController.php

namespace App\Http\Controllers;

use App\Http\Requests\StoreFileRequest;
use App\Services\FileUploadService;

class FileController extends Controller
{
    protected $fileService;

    public function __construct(FileUploadService $fileService)
    {
        $this->fileService = $fileService;
    }

    public function store(StoreFileRequest $request)
    {
        $file = $request->file('file');

        $result = $this->fileService->store($file);

        return response()->json([
            'success' => true,
            'url' => $result['url'],
            'path' => $result['path'],
        ]);
    }

    public function destroy($path)
    {
        $deleted = $this->fileService->delete($path);

        return response()->json([
            'success' => $deleted,
        ]);
    }
}
```

6. Set Up Routes

```
// routes/web.php

use App\Http\Controllers\FileController;

Route::middleware(['auth'])->group(function () {
    Route::post('/files', [FileController::class, 'store']);
    Route::delete('/files/{path}', [FileController::class, 'destroy'])
        ->where('path', '.*');
});
```

7. Create a Frontend Upload Component (Example with Alpine.js)

```
<div x-data="fileUpload()">
    <input type="file" @change="handleFileChange"
accept="image/*,.pdf,.doc,.docx">
    <div x-show="uploading">Uploading...</div>
    <div x-show="error" x-text="error"></div>

    <template x-if="url">
        <div>
            <img x-show="isImage" :src="url" class="max-w-full h-auto">
            <a x-show="!isImage" :href="url" target="_blank">Download
File</a>
            <button @click="removeFile">Remove</button>
        </div>
    </template>
</div>

<script>
function fileUpload() {
    return {
        file: null,
        url: null,
        path: null,
        uploading: false,
        error: null,
        isImage: false,

        handleFileChange(e) {
            this.file = e.target.files[0];
            this.error = null;

            if (!this.file) return;

            this.uploadFile();
        },
```

```
        async uploadFile() {
            this.uploading = true;

            const formData = new FormData();
            formData.append('file', this.file);

            try {
                const response = await fetch('/files', {
                    method: 'POST',
                    body: formData,
                    headers: {
                        'X-CSRF-TOKEN':
document.querySelector('meta[name="csrf-token"]').content,
                    },
                });

                const data = await response.json();

                if (data.success) {
                    this.url = data.url;
                    this.path = data.path;
                    this.isImage = this.file.type.startsWith('image/');
                } else {
                    this.error = 'Upload failed';
                }
            } catch (err) {
                this.error = 'An error occurred';
            } finally {
                this.uploading = false;
            }
        },

        async removeFile() {
            if (!this.path) return;

            try {
                const response = await
fetch(`/files/${encodeURIComponent(this.path)}`, {
                    method: 'DELETE',
                    headers: {
                        'X-CSRF-TOKEN':
document.querySelector('meta[name="csrf-token"]').content,
                    },
                });

                const data = await response.json();

                if (data.success) {
                    this.url = null;
                    this.path = null;
                }
            } catch (err) {
                this.error = 'Failed to delete file';
```

```
            }
        }
    };
}
</script>
```

8. Configure CORS for S3 (If Needed)

If you're getting CORS errors when accessing files directly from S3:

1. Go to your S3 bucket → Permissions → CORS configuration
2. Add a policy like this:

```
[
    {
        "AllowedHeaders": ["*"],
        "AllowedMethods": ["GET", "HEAD"],
        "AllowedOrigins": ["https://yourdomain.com"],
        "ExposeHeaders": []
    }
]
```

9. Testing the Fallback Mechanism

To test the local fallback:

1. Set `FILESYSTEM_DISK=public` in your `.env` file
2. Create a symbolic link from `public/storage` to `storage/app/public`:

```
php artisan storage:link
```

PROF

10. Additional Considerations

1. **File Visibility**: For private files, use:

   ```
   Storage::disk('s3')->putFileAs($path, $file, 'private');
   ```

2. **Temporary URLs**: For private files, generate temporary URLs:

   ```
   Storage::disk('s3')->temporaryUrl($path, now()->addMinutes(30));
   ```

3. **File Size Limitations**: Adjust Nginx/Apache configuration if uploading large files:

```
; php.ini
upload_max_filesize = 20M
post_max_size = 20M
```

This implementation provides a robust file upload system that automatically uses S3 in production and falls back to local storage in development, with a clean service layer abstraction for easy maintenance.