Laravel - File Uploading

Topics : Laravel
Written on December 25, 2023

In Laravel, handling file uploads is a common requirement for web applications. Laravel provides a convenient and secure way to handle file uploads through the use of the `Illuminate\Http\Request` object and the `Storage` facade. Here's a guide on how to handle file uploads in Laravel:

1. Form Setup:

In your Blade view, set up a form with the `enctype="multipart/form-data"` attribute to enable file uploads.

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

2. Controller Handling:

In your controller, use the `store` method to handle the file upload.

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

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

    $file = $request->file('file');

    // Store the file in the storage/app/public directory
    $path = $file->store('public');

    // Get the publicly accessible URL for the file
    $url = Storage::url($path);
```

```
    // Save the file information to the database or perform any other
necessary actions

    return "File uploaded successfully. URL: $url";
}
```

3. File Validation Rules:

The `validate` method can be used to define rules for file uploads. In the example above:

- `required`: The file field must not be empty.
- `mimes:jpg,jpeg,png`: Only allow files with the specified MIME types.
- `max:2048`: Limit the file size to 2048 KB.

4. Storing Files:

Laravel's `Storage` facade provides a convenient way to store files. In the example, the file is stored in the `public` disk. The `store` method automatically generates a unique filename.

5. Public Disk Configuration:

Make sure the `public` disk is configured correctly in the `config/filesystems.php` file.

```
'disks' => [
    // ...

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


6. Accessing Stored Files:

To access the publicly stored file, use the URL generated by the `Storage::url` method.

```
$url = Storage::url($path);
```

7. File Naming and Customization:

You can customize the storage path, file name, and other options as needed.

```
$path = $file->storeAs('public/images', 'custom_name.jpg');
```

8. Deleting Files:

To delete a file, use the `delete` method on the `Storage` facade.

```
Storage::delete($path);
```

9. File Download Response:

To return a file as a response for download, you can use the `response` method.

```
return response()->download(storage_path("app/$path"));
```

10. Security Considerations:

Always validate and sanitize user input. Use proper file validation rules, and consider securing file uploads by placing restrictions on file types, sizes, and storage locations.