IPL
escola superior
de tecnologia e gestão
instituto politécnico
de leiria

Laravel

Response, Storage, Files and Upload

Marco Monteiro

Contributors

- Author(s):
  - Marco Monteiro (marco.monteiro@ipleiria.pt)

Summary

1. Response
2. Storage
3. Files
4. Upload

1 – RESPONSE

Response

- With HTTP protocol, **requests** and **responses** work in **pairs**
- Laravel ensures that for every request, it returns a response
  - Response may something like 404 not found, 401 unauthorized, etc…
- Laravel builds the response from what is returned by controller
  - If it returns a view, generates a HTML page; if it returns an array, it generates a JSON string; if an exception is thrown and is not handled (by a try/catch block) it generates an error page with an appropriate status code; etc.
- Typical responses for a traditional web application, are:
  - **HTML** pages – generated by the **views**
  - **Redirects** – a redirect is a response with status code 302 Found.
  - **Error** response – when an error occurs (generated by the framework code, or by the developer), it returns a response with an error status code

Response

- **HTML** Pages - generated by the **view**

```
return view('name_view')->with('var1', 'value1);
```

  - Content-Type: `text/html`
  - Status Code: **200 OK**

- **Redirect** response

```
return redirect()->route('route.name');
```

  - Content-Type: `text/html`
  - Status Code: **302 Found**
  - Header: `Location: http://redirect_to_this_url`

- **Error** response

```
abort(401);
```

  - Content-Type: `text/html`
  - Status Code: **401 Unauthorized**

Response Object

- Function **response** returns the instance of the current response (the Response that is currently being created)
  - Attach headers to the response:

```
public function action1() {
   return response('Hello World', 200)
          ->header('Content-Type', 'text/plain');
};
```

  - Attach header to the view response:

```
   return response()->view('hello', $data, 200)
             ->header('Some-Header', 'Value');
```

  - Create a JSON response:

```
return response()->json([
         'name' => 'Abigail',
         'email' => 'abigail@mail.com',
    ])
```

Response Object - File

- Method **file** of the Response object:
  - Copy the content of a file, such as an image or a PDF, directly to the HTTP Response (also changes the content type accordingly)
  - If the browser supports that type of content, the file is viewed on the browser

```
return response()->file($pathToFile);

return response()->file($pathToFile, $headers);
```

2 – STORAGE

File Storage

- Laravel uses **Flysystem** package to abstract a file system.

  https://laravel.com/docs/filesystem

  - Supports local filesystem (default), Amazon S3, ftp server, etc.
  - We will use Flysystem (local flysystem) to store files (images, PDF, Excel, etc.) that were uploaded by the user or were dynamically generated by the application.

- Local filesystem stores files on this folder:

  `<localPrjFolder>/storage/app`

File Storage – Public Disk

- **Public disk** - for publicly accessible files.
  - Files within `storage/app/public` will be publicly available with the following URL: `http://mysite/storage/*`
  - Requires a symbolic link from **public/storage** to **storage/app/public**. To create the symbolic link:

```
php artisan storage:link
```

File Storage – Public Disk

- Example - file full path (on the web server machine):

```
<localPrjfolder>/storage/app/public/img/fileimg.jpeg
```

- Will be available as the URL:

```
http://mysite/storage/img/fileimg.jpeg
```

- Use "**asset**" function to obtain the URL

```
asset('storage/img/fileimg.jpeg');
```

This returns the complete URL. Example:

```
http://mysite.test/storage/img/fileimg.jpeg
```

File Storage

- **Private files** on storage
  - Files within **`storage/app`** but outside of **`storage/app/public`**

- Example - file full path (on the web server machine):

```
<localprjfolder>/storage/app/docs/filedoc.doc
```

- These files will not be directly available – they are private for the web server
- However, they can be accessed through code, and therefore available indirectly through a route of the web application
- This allows us to control the access (authorization) to these files

File Storage

- Storage facade class (examples):

```
use Illuminate\Support\Facades\Storage;
Storage::put('avatars/1', $fileContents);
$contents = Storage::get('file.jpg');
$file_exists = Storage::exists('file.jpg');
$size = Storage::size('file.jpg');
$url = Storage::url('file.jpg');
Storage::delete('file.jpg');
Storage::copy('old/file.jpg', 'new/file.jpg');
Storage::move('old/file.jpg', 'new/file.jpg');
```

- storage_path function (example):

```
storage_path('app/doc.pdf');
```

  - Returns the fully qualified name of the file (local file) in the storage
  - Something similar to: …/prj/storage/app/doc.pdf

3 – FILES

Response – Download files

- Generate a response that forces the user's browser to download the file at the given path
  - File is copied directly to the response, but it is always downloaded (not viewed) by the browser.
  - Examples:

```
return response()->download($pathToFile);

return response()->download($pathToFile, $name, $headers);

return response()->download($pathToFile)
                 ->deleteFileAfterSend(true);
```

  - Examples using Storage facade:

```
return Storage::download('file.jpg');

return Storage::download('file.jpg', $name, $headers);
```

Response – View files

- Display a file, such as an image or PDF, directly in the user's browser instead of initiating a download
  - File is copied directly to the response and viewed on the browser (if the browser supports the type of content of the file)
  - Examples:

```
return response()->file($pathToFile);

return response()->file($pathToFile, $headers);
```

  - Examples using "storage_path" function:

```
$path = storage_path('app/docs/doc.zip');

return response()->file($path);
```

Response – Restrict Access to Files

- How to restrict the access to a specific file (e.g. Image, PDF document, etc.) to a user or a subset of users?

- Store the file on a private local storage folder
- Create a "standard" route - with GET method. Examples:

```
route::get('users/{id}/photo', …)
route::get('compras/{id}/fatura', …)
route::get('users/{user}/docs/{doc}', …)
route::get('documents/{filename}', …)
```

- Using Laravel authorization mechanisms, protect the route as you would protect any other route
- Controller's method (action) must return the file – using:

```
response()->download()        or:
response()->file()
```

4 – UPLOAD

File Upload

- How to **upload a file** - copy file from client to the server?

1. Browser creates an HTTP Request with the file content
   - method = **POST**
   - enctype="**multipart/form-data**"

2. Web Server receives the HTTP Request with the file content, and saves the content on a temporary file

3. Typically, server application moves the temporary file to a known location and name – the name of the file is usually associated with data on the database

File Upload - HTML

- On the HTML, uploading a file requires:
  - A form with method **POST** and enctype="**multipart/form-data**"
  - One (or more) input element with **type="file"**
    - Add 1 input for each file to upload
  - Example (using blade):

```
<form action="route('name.route')" method="POST"
      enctype="multipart/form-data">
   @csrf
. . .
   <input type="file" name="photo">
. . .
   <input type="submit>
</form>
```

Request – UploadedFile class

- Retrieving Uploaded Files
  - **file method** returns an instance of `Illuminate\Http\`**`UploadedFile`** that allows interaction with uploaded file

```
$file = $request->file('photo');
// or:
$file = $request->photo;
```

photo – the name of field

Request – UploadedFile class

- To determine if a file is present on the request:

```
if ($request->hasFile('photo')) {
    //
}
```

- Validating Successful Uploads
  - **isValid method** returns true if there were no problems uploading the file:

```
if ($request->file('photo')->isValid()) {
    //
}
```

Storing Uploaded File

- Coping the uploaded file to the Storage.
  - Name of the form field = photo
  - Folder where file is copied to: <prj>/storage/app/doc/x
  - Return the path (full name) of the file created on the storage

- Using UploadedFile class

```
$path= $request->file('photo')->store('doc/x');

$path= $request->photo->store('doc/x');

$path= $request->photo->storeAs('doc/x', 'filename');
```

- Using Storage facade class:

```
$path= Storage::putFile('doc/x', $request->file('photo'));

$path= Storage::putFileAs('doc/x', $request->photo,
                          'filename');
```

File upload – public disc

- Example: Upload file to "storage/app/**public/img**"

```
if ($request->file('photo')->isValid()) {
  $path= Storage::putFile('public/img', $request->file('photo'));
  // Save $path on DB
}
```

- Example - access the file through the browser (public URL). (name will be unique):

```
http://mysite/storage/img/bMnei6MeWyQ4.jpeg
```

- Examples: use a file within a blade view file (HTML)

```
<img src="{{ Storage::url("img/$path") }}">
<img src="{{ asset("storage/img/$path") }};
```

File upload – private location

- Example: Upload file to "storage/app**/img**"

```
if ($request->file('photo')->isValid()) {
    $path= Storage::putFile('img', $request->file('photo'));
    // Save $path on DB
}
```

  - File is not accessible through the browser (no public URL)
  - We can access the file through a route. By using a route, code is executed, which means that you can protect the file depending on the user or the context of usage. Example:

```
Route::get('photo/{path}', [Ctr::class, 'getfile']);
```

```
public function getfile($path) {
    return response()->file(storage_path('app/img/' . $path));
}
```