

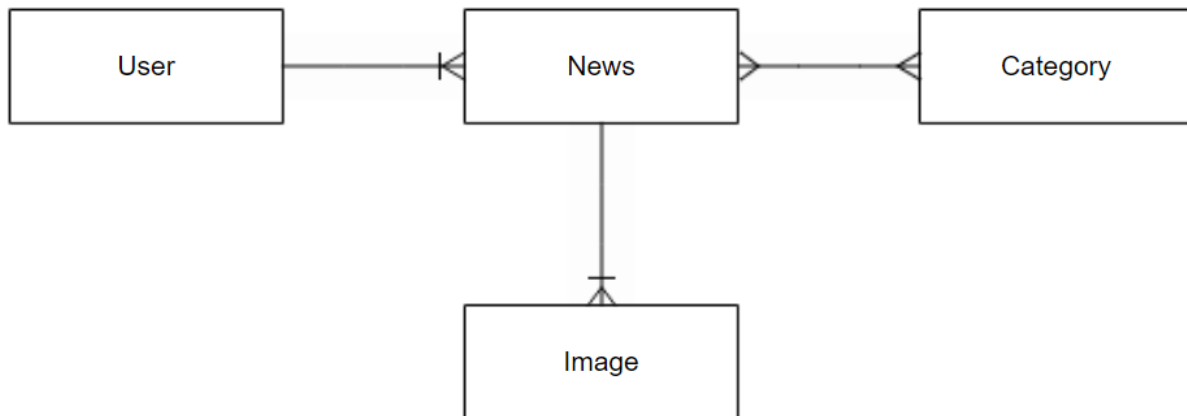
Day-2: Building News Platform with Laravel API - CRUD for News and Images

Prepared by: Rana M. Fakeeh

Application overview:

Our goal is to create a News Platform's CRUD APIs with the following entities:

- **Category**: To categorize news articles.
- **News**: To store news articles.
- **Image**: To associate image album with news articles.
- **User**: To manage user accounts.



Training program:

Day-1	Setup. CRUD for Category entity.
Day-2	CRUD for News, and Image entities. Upload image files to server. One-to-Many and Many-to-Many associations.
Day-3	Recap. Filtering, Sorting, and Pagination of list results.
Day-4	Laravel Passport authentication (OAuth2 token-based) Register, Login, Logout, and Profile APIs for User entity. Public vs. Protected endpoints.
Day-5	Localization. Deployment (Docker).

Learning objectives:

- Create News and Image tables.
- Create RESTful API for CRUD operations.
- Handle one-to-many and many-to-many associations.
- Upload image files to server.
- Hands-on.

Steps:

1- Environment

- Run XAMP control panel.
- Start Apache server
- Start MySQL server
- Open VS Code on previous project.

2- News Module (Migration -> Model -> Factory -> Seeder -> Resource)

- Create News model, migration, and factory.
- `php artisan make:model News -mf`
- The flag `-mf` creates factory, and migrations files in addition to model file.
- **Migration**
- Open `database/migrations/timestamp_create_news_table.php` file.
- Add fields **title**, **description**, **thumbnail**, **completed**, and **visible**.
- `$table->string('title');`
- `$table->text('body')->nullable();`
- `$table->string('thumbnail')->nullable();`
- `$table->boolean('completed')->default(false);`
- `$table->boolean('visible')->default(false);`
- **Model**
- Open `app/Models/News.php` file.
- Add fields **title**, **description**, **thumbnail**, **completed**, and **visible**.
- `protected $fillable = [`
- `'title',`
- `'body',`
- `'thumbnail',`
- `'completed',`
- `'visible'`
- `];`
- **Factory**
- Open `database/factories/NewsFactory.php` file.
- Create fake values for fields **title**, **description**, **thumbnail**, **completed**, and **visible**.
- `'title' => ucwords(fake()->words(rand(3, 6), true)),`
- `'body' => fake()->text(),`
- `'thumbnail' => fake()->imageUrl(),`
- `'completed' => fake()->randomElement([true, false]),`
- `'visible' => fake()->randomElement([true, false])`
- **Seeder**
- Open `database/seeder/DatabaseSeeder.php` file.
- Populate database with 20 dummy news.
- `News::factory(20)->create();`
- Migrate changes to database schema.
- `php artisan migrate`
- [if migration failed due to an existing table, drop it from the database then re-migrate].
- Run seeding

- `php artisan db:seed`
- Check database changes in MySQL (phpMyAdmin)
- **Resource**
- Create News resource.
- `php artisan make:resource NewsResource`
- Open `app/Http/Resources/NewsResource.php` file.
- Return only the article's *id*, *title*, and *thumbnail* (e.g. for the slider on the home page).
- `[`
- `'id' => $this->resource->id,`
- `'title' => $this->resource->title,`
- `'thumbnail' => $this->resource->thumbnail`
- `];`
- Create News collection.
- `php artisan make:resource NewsCollection --collection`
- Open `app/Http/Resources/NewsCollection.php` file.
- Return the objects as defined in the resource.
- `[`
- `'count' => $this->collection->count(),`
- `'news_articles' => $this->collection,`
- `];`

3- Image Module (Migration -> Model -> Factory -> Seeder -> Resource)

- Create Image model, migration, and factory.
- `php artisan make:model Image -mf`
- The flag `-mf` creates factory, and migrations files in addition to model file.
- **Migration**
- Open `database/migrations/timestamp_create_images_table.php` file.
- Add field *url*.
- `$table->string('url');`
- **Model**
- Open `app/Models/Image.php` file.
- Add field *url*.
- `protected $fillable = [`
- `'url'`
- `];`
- **Factory**
- Open `database/factories/ImageFactory.php` file.
- Create fake values for field *url*.
- `'url' => fake()->imageUrl()`
- **Seeder**
- Open `database/seeds/DatabaseSeeder.php` file.
- Populate database with 20 dummy images
- `Image::factory(20)->create();`
- Migrate changes to database schema.
- `php artisan migrate`

- [if migration failed due to an existing table, drop it from the database then re-migrate].
- Refresh migrations to empty the tables and reseed.
- `php artisan migrate:refresh`
- Run seeding
- `php artisan db:seed`
- Check database changes in MySQL (phpMyAdmin)
- **Resource**
- Create Image resource.
- `php artisan make:resource ImageResource`
- Open `app/Http/Resources/ImageResource.php` file.
- Return only the image's *id*, and *url*.
- `[`
- `'id' => $this->resource->id,`
- `'url' => $this->resource->url`
- `];`
- Create Image collection.
- `php artisan make:resource ImageCollection --collection`
- Open `app/Http/Resources/ImageCollection.php` file.
- Return the objects as defined in the resource.
- `[`
- `'count' => $this->collection->count(),`
- `'images' => $this->collection,`
- `];`

4- Eloquent One-to-Many Relationship (Migration -> Model -> Factory -> Seeder -> Resource)

- A News article **has many** Images.
- An Image **belongs to** one News.
- Therefore, Image table (the many side) should have the foreign key from News table.
- **Image Module**
- **Migration**
- Open `database/migrations/timestamp_create_images_table.php` file.
- Add the foreign key *news_id* from News table.
- `$table->unsignedBigInteger('news_id')->nullable();`
- `$table->foreign('news_id')->references('id')->on('news')->onDelete('cascade');`
- **Model**
- Open `app/Models/Image.php` file.
- Add foreign key field *news_id*.
- `protected $fillable = [`
- `'url',`
- `'news_id'`
- `];`
- Create function to retrieve the associated record (singular).
- `public function news() {`
- `return $this->belongsTo(News::class);`
- `}`

- **Factory**
- Open **database/factories/ImageFactory.php** file.
- Create fake values for foreign key field **news_id**.
- **'news_id' => fake()->numberBetween(1,15)**
- **Seeder**
- Open **database/seeders/DatabaseSeeder.php** file.
- Check that News fake records are created before Image recodes.
- Migrate changes to database schema.
- **php artisan migrate**
- [if migration failed due to an existing table, drop it from the database then re-migrate].
- Refresh migrations to empty the tables and reseed.
- **php artisan migrate:refresh**
- Run seeding
- **php artisan db:seed**
- Check database changes in MySQL (phpMyAdmin)
- **Resource**
- Open **app/Http/Resources/ImageResource.php** file.
- Return only the image's **id, url** and the associated **article_title**.
- **[**
- **'id' => \$this->resource->id,**
- **'url' => \$this->resource->url,**
- **'article_title' => \$this->resource->news->title**
- **];**
- **News Module**
- **Model**
- Open **app/Models/News.php** file.
- Create function to retrieve the associated records (plural).
- **public function images() {**
- **return \$this->hasMany(Image::class);**
- **}**
- **Seeder**
- Open **database/seeders/DatabaseSeeder.php** file.
- Check that News fake records are created before Image recodes.
- **Resource**
- Open **app/Http/Resources/NewsResource.php** file.
- Return only the article's **id, title, thumbnail** and the associated **album_size**.
- **[**
- **'id' => \$this->resource->id,**
- **'title' => \$this->resource->title,**
- **'thumbnail' => \$this->resource->thumbnail,**
- **'album_size' => \$this->resource->images()->count(),**
- **'album' => ImageResource::collection(\$this->images)**
- **];**

5- Eloquent Many-to-Many Relationship (Migration -> Model -> Factory -> Seeder -> Resource)

- A News article **belongs to many** Categories.
- A Category **belongs to many** News articles.
- Therefore, a third table called the **pivot table** should have two foreign keys one from News table and the another from Category table. The name of the pivot table should be the concatenation of the two table names **in the alphabetical order** i.e. CategoryNews
- **CategoryNews Module (Migration -> Model -> Factory -> Seeder)**
- Create Image model, migration, and factory.
- `php artisan make:model CategoryNews -mf`
- The flag -mf creates factory, and migrations files in addition to model file.
- **Migration**
- Open `database/migrations/timestamp_create_category_news_table.php` file.
- Add the foreign keys `category_id` from Category table and `news_id` from News table.
- `$table->unsignedBigInteger('category_id');`
- `$table->foreign('category_id')->references('id')->on('categories')->onDelete('cascade');`
- `$table->unsignedBigInteger('news_id');`
- `$table->foreign('news_id')->references('id')->on('news')->onDelete('cascade');`
- **Model**
- Open `app/Models/CategoryNews.php` file.
- Add the foreign keys `category_id` from Category table and `news_id` from News table.
- `protected $fillable = [`
- `'category_id',`
- `'news_id'`
- `];`
- **Factory**
- Open `database/factories/CategoryNewsArticleFactory.php` file.
- Create fake values for fields `category_id` and `news_id`.
- `'category_id' => fake()->numberBetween(1,10),`
- `'news_id' => fake()->numberBetween(1,20)`
- **Seeder**
- Open `database/seeder/DatabaseSeeder.php` file.
- Populate database with 50 dummy records
- `CategoryNews::factory(50)->create();`
- Check that fake records of News and Category are created before CategoryNews recodes.
- Migrate changes to database schema.
- `php artisan migrate`
- [if migration failed due to an existing table, drop it from the database then re-migrate].
- Refresh migrations to empty the tables and reseed.
- `php artisan migrate:refresh`
- Run seeding
- `php artisan db:seed`
- Check database changes in MySQL (phpMyAdmin)

- **Category Module**
- **Model**
- Open **app/Models/Category.php** file.
- Create function to retrieve the associated records (plural).
- ```
public function news() {
 return $this->belongsToMany(News::class);
}
```
- **Resource**
- Open **app/Http/Resources/CategoryResource.php** file.
- Return only the category's **id**, **name** and the associated **news\_count**.
- ```
[
    "id" => $this->resource->id,
    "name" => $this->resource->name,
    "news_count" => $this->resource->news()->count()
];
```
- **News Module**
- **Model**
- Open **app/Models/News.php** file.
- Create function to retrieve the associated records (plural).
- ```
public function categories() {
 return $this->belongsToMany(Category::class);
}
```
- **Resource**
- Open **app/Http/Resources/NewsResource.php** file.
- Return only the article's **id**, **title**, **thumbnail** and the associated **album\_size**, **album**, **categories\_count** and **categories**.
- ```
[
    'id' => $this->resource->id,
    'title' => $this->resource->title,
    'thumbnail' => $this->resource->thumbnail,
    'album_size' => $this->resource->images()->count(),
    'album' => ImageResource::collection($this->images),
    'categories_count' => $this->resource->categories()->count(),
    'categories' => CategoryResource::collection($this->categories)
];
```

6- Image Upload

- Open **config/filesystems.php** file.
- Images uploaded will be saved locally into **storage/app/public/*** and accessible from the browser from **public/storage/***.
- Create a symbolic endpoint from **public/storage** to **storage/app/public**.
- ```
php artisan storage:link
```
- Image file will be publicly accessible on: **http://localhost:8000/storage/[image-path]**

## 7- News Module (Controller -> Route)

- **Controller**
- Create News controller.
- `php artisan make:controller NewsController --api`
- Open **`app/Http/Controllers/NewsController.php`** file.
- Add the code snippet in (Appendix-I)
- **Route**
- Create routes for News' CRUD operations.
- Open **`routes/api.php`** file.
- Add either:
  - `Route::resource('news', NewsController::class)`
  - Or
  - `Route::get('news', [NewsController::class, 'index'])->name('news.index');`

## 8- Image Module (Controller -> Route)

- **Controller**
- Create Image controller.
- `php artisan make:controller ImageController --api`
- Open **`app/Http/Controllers/ImageController.php`** file.
- Add the code snippet in (Appendix-II)
- **Route**
- Create routes for Image's CRUD operations.
- Open **`routes/api.php`** file.
- Add either:
  - `Route::resource('album', ImageController::class)`
  - Or
  - `Route::post('album', [ImageController::class, 'store'])->name('album.store');`

## 9- Testing API with Postman

- Apache server is started.
- MySQL server started.
- Migrate changes to database schema.
- `php artisan migrate`
- Test the server running
- `php artisan serve`
- Open Postman
- **`http://localhost:8000/api/news/*`**
- **`http://localhost:8000/api/album/*`**
- For create, use **Body -> form-data**
- For update, use **Body -> x-www-form-urlencoded**



## Appendix-I

### *NewsController.php*

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Category;
use App\Http\Resources\CategoryResource;
use App\Http\Resources\CategoryCollection;
use App\Models\News;
use App\Http\Resources\NewsResource;
use App\Http\Resources\NewsCollection;
use App\Models\Image;
use Illuminate\Support\Facades\Validator;
use Exception;

class NewsController extends Controller
{
 /**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
 public function index()
 {
 $articles = News::all();

 return response()->json([
 "status" => "success",
 "error" => false,
 "data" => new NewsCollection($articles),
],200);
 }

 public function public_index()
 {
 $articles = News::where('visible', true)->get();

 return response()->json([
 "status" => "success",
 "error" => false,
 "data" => new NewsCollection($articles),
],200);
 }
}
```

```

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
 $validator = Validator::make($request->all(), [
 "title" => "required|min:3|unique:news,title",
 'thumbnail' => 'required|image|mimes:jpg,png,jpeg,gif,svg|max:2048',
]);

 if($validator->fails()) {
 return response()->json([
 "status" => "fail",
 "error" => true,
 "validation_errors" => $validator->errors()
]);
 }

 $news_id = NULL;
 try {

 $image_path = $request->file('thumbnail')->store('thumbnails','public');
 $article = News::create([
 "title" => $request->title,
 "body" => $request->body,
 "thumbnail" => $image_path,
 "completed" => $request->completed,
 "visible" => $request->visible,
]);
 $news_id = $article->id;

 if($request->categories) {
 $categories = $request->categories;
 $article->categories()->attach($categories);
 }

 if($request->images) {
 $images = $request->images;
 Image::whereIn('id', $images)->update(['news_id' => $news_id]);
 Image::where('news_id', $news_id)->whereNotIn('id', $images)->delete();
 }

 return response()->json([

```

```

 "status" => "success",
 "error" => false,
 "message" => "Success! news article created.",
 "data" => new NewsResource($article),
], 201);
}
catch(Exception $exception) {

 if($news_id) {
 $article = News::find($news_id);
 $article->categories()->detach();
 Image::where('news_id', $id)->delete();
 $article->delete();
 }

 return response()->json([
 "status" => "fail",
 "error" => true,
 "message" => $exception->getMessage(),
], 404);
}
}

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
 $article = News::find($id);

 if($article) {
 return response()->json([
 "status" => "success",
 "error" => false,
 "data" => new NewsResource($article),
], 200);
 }

 return response()->json([
 "status" => "failed",
 "error" => true,
 "message" => "Failed! no news article found."
], 404);
}

```

```

public function public_show($id)
{
 $article = News::where('visible', true)->find($id);

 if($article) {
 return response()->json([
 "status" => "success",
 "error" => false,
 "data" => new NewsResource($article),
], 200);
 }

 return response()->json([
 "status" => "failed",
 "error" => true,
 "message" => "Failed! no news article found."
], 404);
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
 $article = News::find($id);

 if($article) {

 $validator = Validator::make($request->all(), [
 "title" => "required|min:3",
]);

 if($validator->fails()) {
 return response()->json([
 "status" => "fail",
 "error" => true,
 "validation_errors" => $validator->errors()
]);
 }

 $article['title'] = $request->title;

 // if has body

```

```

 if($request->body) {
 $article['body'] = $request->body;
 }

 // if has thumbnail image
 if($request->file('thumbnail')) {
 $image_path = $request->file('thumbnail')->store('thumbnails',
'public');
 $article['thumbnail'] = $image_path;
 }

 // if has visible
 if($request->visible) {
 $article['visible'] = $request->visible;
 }

 // if has completed
 if($request->completed) {
 $article['completed'] = $request->completed;
 }

 // if has categories
 if($request->categories) {
 $categories = $request->categories;
 $article->categories()->sync($categories);
 }

 // if has album images
 if($request->images) {
 $images = $request->images;
 Image::whereIn('id', $images)->update(['news_id' => $id]);
 Image::where('news_id', $id)->whereNotIn('id', $images)-
>delete();
 }

 $article->save();
 return response()->json([
 "status" => "success",
 "error" => false,
 "message" => "Success! news article updated.",
 "data" => new NewsResource($article)
], 201);
 }

 return response()->json([
 "status" => "failed",
 "error" => true,
 "message" => "Failed no news article found."
], 404);

```

```

}

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
 $article = News::find($id);

 if($article) {
 $article->categories()->detach();
 Image::where('news_id', $id)->delete();
 $article->delete();
 return response()->json([
 "status" => "success",
 "error" => false,
 "message" => "Success! news article deleted."
], 200);
 }

 return response()->json([
 "status" => "failed",
 "error" => true,
 "message" => "Failed no news article found."
], 404);
}
}

```

## Appendix-II

### *ImageController.php*

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Image;
use App\Http\Resources\ImageResource;
use App\Http\Resources\ImageCollection;
use Illuminate\Support\Facades\Validator;
use Exception;

class ImageController extends Controller
{
 /**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
 public function index()
 {
 //
 }

 /**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
 public function store(Request $request)
 {
 $validator = Validator::make($request->all(), [
 'image' => 'required|image|mimes:jpg,png,jpeg,gif,svg|max:2048',
]);

 if($validator->fails()) {
 return response()->json([
 "status" => "fail",
 "error" => true,
 "validation_errors" => $validator->errors()
]);
 }

 try {
```

```

 $image_path = $request->file('image')->store('images','public');
 $image = Image::create([
 "url" => $image_path
]);
 return response()->json([
 "status" => "success",
 "error" => false,
 "message" => "Success! image created.",
 "data" => $image,
], 201);
 } catch(Exception $exception) {
 return response()->json([
 "status" => "failed",
 "error" => $exception->getMessage()
], 404);
 }
}

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
 //
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
 //
}

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)

```



```
{
 $image = Image::find($id);

 if($image) {
 $image->delete();
 return response()->json([
 "status" => "success",
 "error" => false,
 "message" => "Success! image deleted."
], 200);
 }
 return response()->json([
 "status" => "failed",
 "error" => true,
 "message" => "Failed no image found."
], 404);
}
}
```

Dialog

×

Title

Body

☐ visible

☐ completed

Thumbnail

Browse

☒ Category 1

☐ Category 2

☒ Category 3

☒ Category 4

Submit