Laravel Exercises

# 1. Routes (10 Exercises)

1. Create a route that returns a simple 'Hello, Laravel!' message.

2. Define a route that accepts a parameter (e.g., user/{id}) and returns 'User ID: {id}'.

3. Create a route with multiple optional parameters (e.g., profile/{name?}/{age?}).

4. Define a route that redirects from /old-route to /new-route.

5. Group routes under a common prefix, like /admin, and create routes for dashboard, users, and settings.

6. Use a middleware in a route group to protect a specific set of routes (e.g., /admin).

7. Define a route that accepts a query string (e.g., /search?query=Laravel) and returns 'Search results for: {query}'.

8. Create a route that returns JSON data for an array of products.

9. Set a route name (e.g., route('profile')) and use it in a view.

10. Create a fallback route that displays '404 - Page not found'.

# 2. Controller (10 Exercises)

1. Create a ProductController and define a method index() to return a list of products.

2. Create a PostController with a method show($id) to return details of a post based on the ID.

3. Define a resource controller for Category and generate all resourceful routes.

4. Use route-model binding to fetch a user from the database in a UserController.

5. Pass data from a controller method to a Blade view and display it.

6. Create a FormController to handle GET and POST requests for a contact form.

7. Use dependency injection to inject a service class into a controller.

8. Return a JSON response with a status code from a ReportController.

9. Use php artisan make:controller --invokable to create a single-action controller for a homepage.

10. Create a controller to handle file uploads and return the uploaded file's path.

# 3. Routes, Controllers, and Redirects (10 Exercises)

1. Create a route and controller to handle form submissions, validate input, and redirect back with a success message.

2. Set up a route that redirects from /login to /dashboard if the user is authenticated (use a middleware).

3. Define a route and controller for a dynamic dropdown menu that fetches categories and products.

4. Create a ReportController to generate a PDF file and redirect to the download link.

5. Add a route to handle failed validation and redirect back with error messages.

6. Build a route and controller to process a payment and redirect the user to a success page upon completion.

7. Redirect from a named route (e.g., profile.update) to another named route (e.g., profile.show).

8. Pass data through a session from one route to another via a controller.

9. Use a route closure to redirect a user based on their role (admin vs. user).

10. Build a custom route and controller for processing AJAX requests and returning JSON data.

# 4. Blade Templates (10 Exercises)

1. Create a master layout with @yield sections for content and a @section for the title.

2. Create a header.blade.php and footer.blade.php and include them in your layout using @include.

3. Use Blade directives like @if, @foreach, and @isset to display user data conditionally.

4. Implement a Blade component for a reusable card UI (e.g., @component('components.card')).

5. Pass data to a view and loop through it to display a list of products in a table.

6. Use Blade's @csrf directive to protect a form from cross-site request forgery.

7. Create a Blade template for a user profile and display dynamic user data.

8. Implement a navbar using Blade components and highlight the active menu using @if.

9. Use @push and @stack to inject scripts and styles in specific sections of a layout.

10. Create a pagination component using Blade that dynamically displays page links.

# 5. Database, Eloquent, and Seeding (10 Exercises)

1. Create a migration for a products table with fields like name, price, and stock.

2. Use a factory to generate 50 dummy products and seed them into the database.

3. Set up a one-to-many relationship between Category and Product and fetch all products under a specific category.

4. Define a User and Profile one-to-one relationship and fetch the profile for a given user.

5. Implement a many-to-many relationship between User and Role and attach roles to a user.

6. Use Eloquent scopes to filter products by price range.

7. Use soft deletes to archive a record and restore it later.

8. Create a pivot table for orders and products and query all products in a specific order.

9. Perform complex queries using Eloquent Query Builder (e.g., get top 5 most expensive products).

10. Implement a database transaction to handle multi-step data entry.

# 6. Collect, Request Data, and Validation (10 Exercises)

1. Validate a form submission for creating a new product (e.g., name required, price numeric).

2. Use the old() helper to repopulate form inputs after validation errors.

3. Collect and store uploaded files and validate the file type.

4. Use the Request class to retrieve query parameters from a URL.

5. Create a custom validation rule to check if a product's name is unique.

6. Use the collect() helper to work with a dataset and manipulate it.

7. Handle JSON request data for an API endpoint and validate the input.

8. Use Request::merge() to modify input data before validation.

9. Validate nested arrays in a request (e.g., validate an array of product attributes).

10. Set up a form request class for validation and reuse it in multiple controllers.

# 7. Authentication and Authorization (10 Exercises)

1. Set up Laravel Breeze for user authentication (login, registration, reset password).

2. Protect specific routes using middleware like auth and guest.

3. Allow only admins to access a route using middleware.

4. Use policies to define permissions for accessing and modifying a resource (e.g., posts).

5. Use gates to authorize specific actions in a controller.

6. Implement role-based access control (RBAC) for users and roles.

7. Add social login functionality using Laravel Socialite.

8. Allow only verified email users to access certain parts of the application.

9. Create a custom middleware to restrict access based on user age.

10. Implement two-factor authentication for enhanced security.

# 8. Final Project

Build a User Management System integrating everything:

Frontend: Use Blade templates for a clean UI.

Backend: Manage user authentication and role-based authorization.

Database: Set up tables for users, roles, and permissions with seed data.

Features:

User registration/login.

Admin dashboard for managing users and roles.

CRUD operations for posts with role-based permissions.