Final Refactorings

Before we move on to deployment, let's clean up the structure of our project.

We've already refactored our payment code into a dedicated payments package. In this exercise, you'll do the same for other parts of the app.

You've already seen how to organize code by feature, extract services, and clean up controllers—this is your chance to apply all of that and make your codebase production-ready.

The refactoring techniques used here are repetitive—you already know how to do them. So to save your time (and mine), I didn't record separate lessons for each refactoring.

Instead, I've provided a link to a commit for each step, so you can compare your work with mine.

Step 1: Refactor the Users Feature

- Create a `users` package.
- Move everything related to managing users (profiles, roles, addresses, etc) into this package.
- Move the classes in the `validations` package to the `users` package, and delete the `validations` package.
- Build the project. If the build fails, try deleting the target folder and rebuilding.

> This can happen because after moving `UserMapper` to the new package, MapStruct may generate a second implementation of the interface. If both implementations exist, Spring won't know which one to use and the application will fail to start.

Compare your code with this commit.

Step 2: Refactor the Auth Feature

- Create an `auth` package.
- Move all classes responsible for handling login, token generation, etc into this package.
- Move the classes in the `config` package to the `auth` package, and delete the `config` package.

Compare your code with this commit.

Step 3: Refactor the Products Feature

- Create a `products` package.
- Move product-related services, controllers, DTOs, and logic here.

Compare your code with this commit.

Step 4: Refactor the Carts Feature

- Create a `carts` package.
- Move all classes for managing shopping carts into this package.
- Earlier in the course, we decided not to use `ProductDto` inside `CartItemDto`. Instead, we introduced a separate class—`CartProductDto`—to represent products within the context of a cart.
- Now that all cart-related classes are in their own package, we can safely rename `CartProductDto` to `ProductDto` without conflicting with the `ProductDto` in the `products` package.

Compare your code with this commit.

Step 5: Refactor the Orders Feature

- Create an `orders` package.
- Move all classes for managing orders into this package.
- You can now rename `OrderProductDto` to `ProductDto`, since it's scoped within the `orders` package and won't conflict with other versions of `ProductDto`.
- After moving the related classes into their feature-specific packages, the `exceptions`, `mappers`, `repositories`, and `services` packages will be empty. You can safely delete them to further clean up the project structure.

Compare your code with this commit.

Step 6: Refactor the AuthController

- The logic in `AuthController` has grown and is doing more than it should. Now is a good time to extract that logic into the `AuthService` to keep the controller focused on handling HTTP requests.

Compare your code with this commit.

Step 7: Refactor the UserController

- The logic in `UserController` has grown and is doing more than it should. It's now a good time to extract this logic into a dedicated service to keep the controller focused on handling HTTP requests.
- Earlier in the course, we created a class called `UserService`. This class implements `UserDetailsService`, which is used internally by Spring during the authentication process. That makes it part of the infrastructure layer, not the application layer.
- We shouldn't move the controller logic into this class, because it serves a different purpose. Our services should encapsulate application logic—things like registering

users, updating profiles, and changing passwords. These are separate responsibilities.
- To fix this, rename the existing `UserService` to `UserServiceImpl`.
- Then, create a new `UserService` class, and move all application logic from the controller into this new class.

Compare your code with this commit.

Step 8: Final Cleanup

In this last step, we'll clean up a few leftover pieces from earlier in the course. These were useful for learning, but they're no longer needed in the final version of the app.

- Remove the `Message` entity and `MessageController`. We created these early in the course to practice building APIs. Now that our app has more meaningful features, we no longer need them.
- Move `AdminController` to the `admin` package.
- Move shared code to a `common` package. Some classes aren't tied to a specific feature. These should go in a common package:
  - `GlobalExceptionHandler`
  - `HomeController`
  - `ErrorDto`
  - `LoggingFilter`
- Delete the old layer-based packages:
  - `controllers`
  - `dtos`
  - `entities`
  - `filters`

Compare your code with this commit.

Final Package Structure

After all the refactorings, this is how our project is now organized—clean, modular, and easier to navigate: