Distributed Applications WS2025/26 (Prof. Dr. Cathrin Möller)

Exercise 5 - 20./24.11.2025

Do not hesitate to ask questions or if you want feedback for your coding style.

Please commit and push to GitLab regularly and ensure @*cathrin.moeller* and my two tutors @pinky-jitendra.tinani and @rahul.patil are invited as a reporter.

Exercise 3 and 4 are due this week (until Sunday November 23rd 23:59)!

Learning goals for today

Today we want to continue coding on our webshop application with Spring Boot.

We want to get familiar with some helpers provided by the JPA package. Using this package we can use database functionality. To avoid complexity with a local running database, we can use the in memory H2 database which is the simplest relational database to connect with. Without changing code, just by changing configuration, it could be replaced by MySQL or other relational DBs. By changing configuration, we can also persist the content of the H2 database in a local file, so it does not get deleted restarting our application.

Using a View - Controller - Service - Repository layer pattern will encapsulate all the database interactions in the Repository. Therefore, a replacement of the whole repository code can easily be done, without having to touch anything at the view or controller code.

Practical Exercises

1) We want to use (in memory) database functionalities. Ensure in you pom.xml the spring-boot-starter-data-jpa package is installed together with a dependency to a h2 database. Change your Product class and add the **@Entity** annotation and the **@Id** annotation from jakarta.persistence. You might need to add a default constructor as well. Feel free to change the id attribute to be a **@GeneratedValue**. If you use that, you cannot set the id manually anymore (which is an advantage as you can skip counting as well).

2) We can use **Repositories** as a convenience mechanism to get access to our database as many helper functions are already provided. Feel free to create a folder for repositories. Create an **interface ProductRepository** which extends the JPARepository<Product, Long> to access the database for Products by Id (=Long). We will use this later from the product service class.

3) Create a class **LoadProductDatabase.java** annotated with **@Configuration** and a **@Bean CommandLineRunner** that initializes your products into the database. Use repository.save() and enrich the hardcoded products for the catalog to have at least 10 different products available. Refactor your code and move all the hardcoded products from your ProductService to be instantiated here. Log each entry to the command line using the Logger and a LoggerFactory from org.slf4j. See lecture slides if you get stuck.

Hint: you can keep the inventory setting hardcoded. One approach would be to call the repository.findAll, iterate over all products with a for-loop and set a fixed hardcoded / random stock value. In a real life application, you would have an inventory database as well that has a column for the reference id of the product to connect to the product table.

4) Refactor now the **ProductService** to have a **dependency** to the **ProductRepository** and call the repository to get the products. Refactor all the methods and use the repository with findAll() / findById() / save() / deleteById()… There should not be any hardcoded list of products remaining and all products should be updated and fetched using the repository. If you switched to generated ids, make sure you refactor the code. Restart your app and make sure your pages to view products, create products, … are still functional.

5) A **H2 database** stores data in memory by default. This means, any added products get lost stopping and restarting the server. Go to your application.properties file and add these lines to store data to a **local file**:

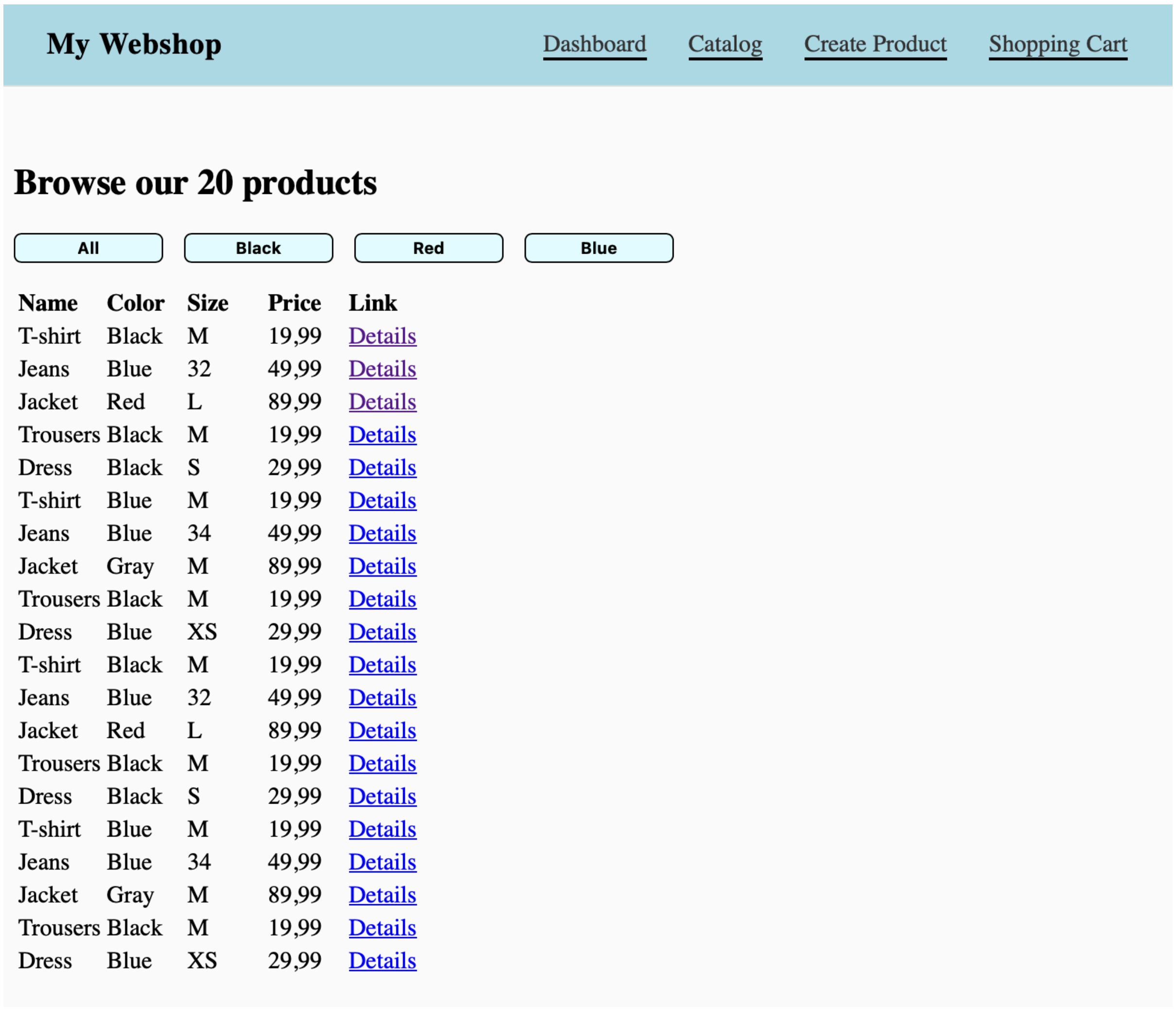
```
# H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2

# Datasource
spring.datasource.url=jdbc:h2:file:./spring-boot-h2-db
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.hibernate.ddl-auto=update
```

If you restart the app, your **LoadProductDatabase.java** would continue adding more and more products to your database. Add an if-else condition that stops adding products to the database if the size of stored products exceeds a fixed value (e.g. 20/50/100 products).

6) Refactor or create methods in your ProductService to retrieve products by color. Connect the ProductService to the ProductRepository. There should not be any more hardcoded products inside the ProductService. You can create multiple **@NamedQuery** in your Product class and create utility methods in your repository like „fetchAllBlackProducts“()

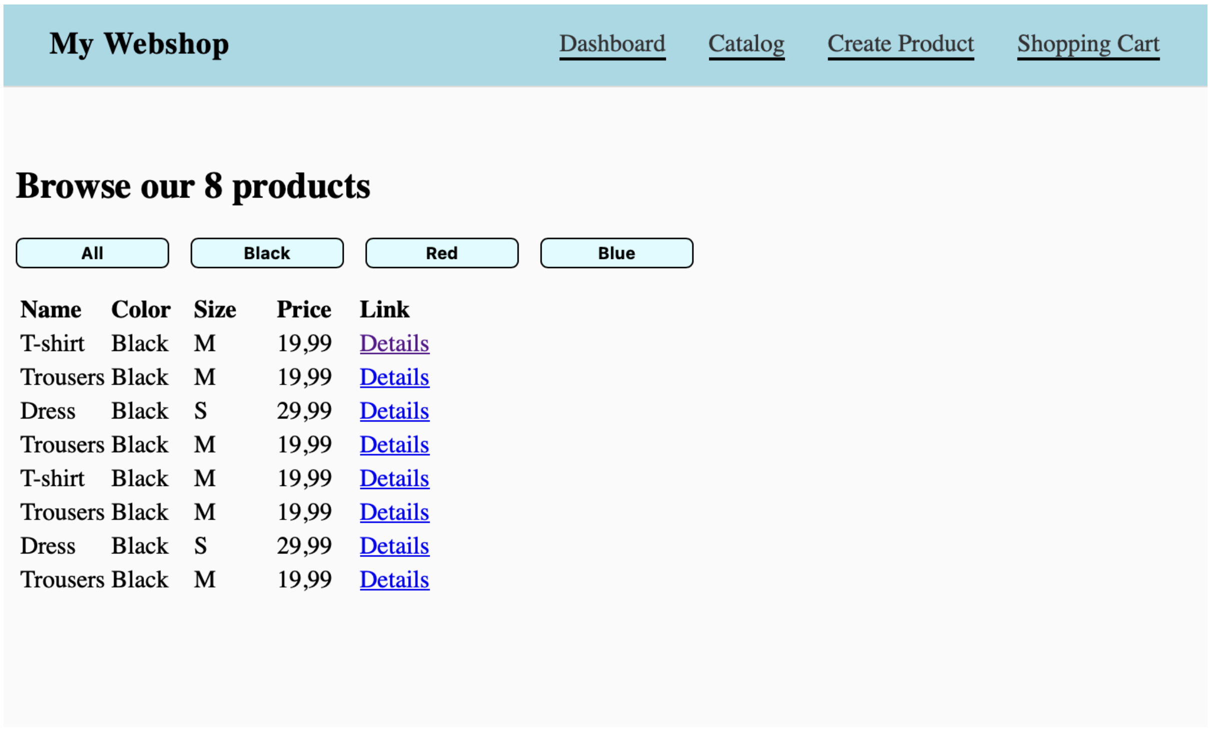
7) Enrich your catalog page to add color **„filter“ buttons** (wrapping <a href=„…“>) that triggers a GET request which finds only the black products, only red products,… Ensure the controller for the catalog asks the service (if you added a facade layer - indirectly via facade)

which calls a specific custom method of the repository to find only the products with that color. The result should look similar to this:



Browse our 20 products

Name	Color	Size	Price	Link
T-shirt	Black	M	19,99	Details
Jeans	Blue	32	49,99	Details
Jacket	Red	L	89,99	Details
Trousers	Black	M	19,99	Details
Dress	Black	S	29,99	Details
T-shirt	Blue	M	19,99	Details
Jeans	Blue	34	49,99	Details
Jacket	Gray	M	89,99	Details
Trousers	Black	M	19,99	Details
Dress	Blue	XS	29,99	Details
T-shirt	Black	M	19,99	Details
Jeans	Blue	32	49,99	Details
Jacket	Red	L	89,99	Details
Trousers	Black	M	19,99	Details
Dress	Black	S	29,99	Details
T-shirt	Blue	M	19,99	Details
Jeans	Blue	34	49,99	Details
Jacket	Gray	M	89,99	Details
Trousers	Black	M	19,99	Details
Dress	Blue	XS	29,99	Details



Browse our 8 products

Name	Color	Size	Price	Link
T-shirt	Black	M	19,99	Details
Trousers	Black	M	19,99	Details
Dress	Black	S	29,99	Details
Trousers	Black	M	19,99	Details
T-shirt	Black	M	19,99	Details
Trousers	Black	M	19,99	Details
Dress	Black	S	29,99	Details
Trousers	Black	M	19,99	Details