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

Exercise 0 - 16./20.10.2025

Disclaimer: The exercise is intended to be done manually - without Chat GPT. If you have any questions, feel free to work in a group or ask tutors or me for help. We will ensure at the end of the practical exercise class that you are able to solve the tasks.

What you will need for „active participation“ will always be highlighted in class.

Practical Exercises

1) Install an **IDE** of your choice (e.g. Eclipse or Jetbrains IDEA - Community edition is free, Enterprise Edition also free for student usage - non commercial) / on the lab machines it is already installed!

2) Use the **Spring initializr** https://start.spring.io/ to set up a basic MVC application including a frontend, backend and preparing for DB usage choosing:

- **Maven** as the build tool
- **Java JDK 21** (needs to be installed separately, e.g. Open JDK Amazon Coretto 21, https://docs.aws.amazon.com/corretto/latest/corretto-21-ug/downloads-list.html) - on lab machines, Java SDK 21 or 25 should be installed already - needs to be checked. If in doubt, 25 would also be allowed.
- **Dependencies** for **Spring Web**, **Thymeleaf**, Spring Data JPA*, H2*, Spring Data JDBC*, MySQL Driver* (* optional: can be added later. needed for future exercises)

3) Please run the hello world application. You should be able to reach **localhost:8080** in your browser and see some „whitelabel error page“ that is served by your application.

Hint: From the console / terminal you need to run **./mvnw spring-boot:run**

If you try using the IDE, ensure you did open the project from the folder and the build is performed using the right JDK v21 (IDEA: Settings - Build - BuildTools - Maven - Importing).

You can then press the green play button in your IDE in the Application class hosting the main() method or in the top bar.

4) **Initialize a private GitLab repo „distributed-applications-25-26“** and invite ***@cathrin.moeller*** as a “**Reporter**“. You will need this in the following weeks for the submissions. It also helps as we will continue working on the same application for multiple weeks. You can add members via Manage => Members => Invite members.

Commit the empty application to your GitLab repo running „***git init***“ in the root folder of your app. „***git add ***“ will add all code. „***git commit -m "0-Create application“***“ will add a commit with a proper message. Follow the instructions in GitLab to push to your repo.

Hint: In case of problems, there is a „force push“ option in branch rules, which can be activated temporarily.

5) Please add a **GET Controller endpoint** for the URL / and return a String containing „Hello World“. For this you need to create a new Java class file.

See lecture or visit docu: https://spring.io/guides/gs/spring-boot

Now re-running the stopped application, you should no longer see this error page.

What annotations from the springframework did you use? What is the „magic“ behind them?

Push the code after a new commit.

—— end of mandatory exercises for this week

If you have your GitLab ready, invited me and pushed everything so far, and run out of time, you can continue working on 6) next week

6) We are going to enhance the application to build a simple Webshop.

You will learn about the Model View Controller Architecture pattern later in the lecture. It helps separating the code.

We need with a **Model** class (Product.java) describing a product with the minimum attributes: id, name, price, size, color. In a hardcoded static list, 5 example products should be available. This file is pure Java, no Spring annotations needed. Please create public getter and setter methods for private attributes.

We also need a **Controller** endpoint to get the full list of products. This can be a new class (ProductController.java). You can add a path mapping to your GET request, serving all products e.g. via ***/products***.

The **View** in the browser can be the default: a simple JSON representation of the products. We will improve it in later exercise classes. If you just return List<Product>, your products will be mapped to JSON automatically.

What happens if some attributes of the product model class are private and have no public getter method? Try changing that in Product.java and loading the endpoint for the list of products again after re-running the application.

If you are done, please commit and push the code.

Theory reflection

1) What is the purpose / benefit of the **Spring** framework?
2) What are annotations? Which ones did you use today?
3) What imports from the Spring frameworks did you use?

4) What is included in the pom.xml file?

Bonus practical exercises

only if you are done with the above in time and need sth extra

7) Create utility methods to get distinct products by id and an endpoint for the product detail page (JSON is enough, no HTML for now needed)

You will then e.g. be able to browse the URL ***/product/1*** to see product with id number 1.

Hint: check the slides and extract the product list to be available for the whole controller class.

8) Extend the attributes of the products and create an endpoint / filtered view for only black products, only green products, only T-Shirts, ...
(JSON is enough, no HTML for now needed)