thws
Technische Hochschule
Würzburg-Schweinfurt

Backend Systems
Portfolio 03
Prof. Dr. Peter Braun

Document History

It is likely that due to students' questions, this document needs to be modified or extended over the next weeks. I will document the changes here.

Version	Date	Changes
1	09.10.2025	Initial version of the document

1 Introduction

The third portfolio exam in this module is the software development project. It is a group project, and you have already chosen the topic for the project and received approval from the lecturer.

You must demonstrate in this project that you can develop a small backend system using a specific software architecture. The solution must include an API and use a database to manage data.

2 Technical Requirements

1. The source code must be written in Java or Kotlin. Use proper package names to structure your code. Use easy-to-understand class and method names and generally try to produce readable code. It **should not** be necessary to add source code comments. Writing documentation of the project is optional – it will not be graded.

2. The backend **must** employ the Hexagonal software architecture. Feel free to contact the lecturer to get approval for the correct implementation of the hexagonal architecture.

3. **Please, read this very carefully:** Work in an incremental way, i.e., start to implement the hexagonal architecture with only one use case and then double-check with the lecturer to get approval of the correct implementation. Don't contact the lecturer for approval after you have implemented everything.

4. You **must** implement at least three components: the domain component (i.e., the core or business logic), the API component, and the persistence component. Depending on your project, there might be more components that should also fit into the hexagonal architecture. It is extremely important to communicate via interfaces (in the sense of Java) between the components and not reuse classes in different components.

5. The backend must provide an API, and you **must** apply the REST paradigm. The API **must** implement filtering (i.e., it must be possible to search for data using query parameters) and paging (i.e., the API only provides a subset of the results, and the API user can ask for more). You **must** implement a caching strategy.

6. You **must** implement the hypermedia principle. The backend replies with URLs to indicate possible next actions. It is not important whether you place the hyperlinks in the response header or body. It is crucial to find a suitable solution for creating and deleting relationships between entities.

7. Depending on your project, authentication and authorization might be necessary.

8. You **must** use Quarkus as a framework with RESTEasy as a provider of JAX-RS. In particular, the Spring framework is **not** allowed, because it is too difficult to implement REST correctly.

9. The backend **must** have a persistence component, and you **must** use the Java Persistence Framework. You can decide what persistence provider you use, e.g., Hibernate.

10. The system **must** include all CRUD operations and at least a 1-to-n relationship between database entities.

11. You **must** implement test cases, i.e., unit tests of the business logic, persistence component, and integration tests on the API level. Unit tests should verify individual components of the system, particularly the business logic.

12. In contrast, integration tests should ensure the correct functioning of the entire application, including database operations and API responses.

13. You **must** use Maven or Gradle as your build tool and adhere to the conventions for folder naming.

14. You **must** provide your software system so we can start a Docker container to test it. The best solution would be to integrate Docker into the Maven build process (this will be shown in one of the next units and is already implemented in some demo repositories – use `mvn verify` to see how it works). If you don't do this, we need a README file that explains the steps to start the Docker container manually and execute the integration tests. We will not install software just to run your code (e.g., a new database system).

3 Deliverables

This assignment's deliverables are

1. a **Git repository** (on THWS Gitlab, or any other Git platform) that contains your solution. The Git repository must contain the source code of a backend, including test cases. The repository's root directory should contain a README file explaining how to start the system and execute all test cases.

2. a **short screencast video** of about 5 minutes (this can be added to the repository or stored in any location where we can access it, e.g., in the THWS cloud, on YouTube as a private video, etc.). In the video, you **must** explain some technical aspects of your implementation (e.g., the components and how they communicate), and **should** demonstrate that the test cases are green. The video **must** be recorded by yourself and show the screen of your computer. Your voice must be audible. It is not necessary to show your face. Not all team members need to speak up in the video. The video must be provided in the MP4 format. The video should have a minimum screen resolution of 720p; 1080p would be preferable. But not more. It should not exceed 20 MB. Use tools like `ffmpeg` to compress it. It is not necessary to use slides in addition to the

screencast. There is no title page, agenda page, or summary page. No intro or outro video, no music. But it would be nice if you said "Hi" at the beginning :-) The video must be playable using VLC or QuickTime Player. Please review the video quality before submitting it.

You must submit a plain text document (i.e., no Word or PDF) to Moodle before the deadline containing information on where to find these two deliverables. The text file must follow a pattern: the first text line contains the repository URL, and the second text line contains the URL to the video. The third line contains a comma-separated list of the immatriculation numbers of all students who worked on this project. Example:

```
https://git.fiw.thws.de/backend-systems/units/Unit01
https://www.youtube.com/watch?v=CLi6YIWXkQI
5123007, 5123008, 5223001
```

Providing the document in this structure helps me to process all data automatically.

4 Some Information about the Evaluation

You can get 36 points for this assignment. When we grade your solution, we will execute the following steps:

- We clone the Git repository to our computer. We will verify that there were no Git commits after the deadline. Otherwise, we will stop evaluating your submission.
- We read the instructions in the README file on how to start the integration tests. We will stop evaluating your solution if there is no `README` file and your system does not start on `mvn verify`.
- We start the integration tests with `mvn verify` (if you use Maven; or in the way you described it in the `README` file). If the system works correctly and all tests pass, we proceed.
- Next, we will watch your video. We are interested in learning about your implementation of the hexagonal architecture.
- We read your software system's source code and check the requirements mentioned in Sec. 2.

If any criteria are not completely fulfilled, we will reduce the points.