Foundations of Distributed Systems
Portfolio Exam 3 (2024ss)

From 2024-05-09 to 2024-06-20

Prof. Dr. Peter Braun and Prof. Dr. Tobias Fertig

thws Technical University of Applied Sciences Würzburg-Schweinfurt

1 Introduction

The portfolio is the form of examination for the module Foundations of Distributed Systems. Students must complete several partial examinations over the semester, which are assessed individually. The final grade for this module comprises partial grades, and the partial exams are weighted according to their difficulty.

Five partial exams (i.e., assignments) will be issued this semester. A total of 90 points can be earned over all five exams. The final grade is then formed from the total points achieved per the usual grading scheme. It is not necessary, but it is recommended that all five assignments be worked on. An overview of the content and the due dates is given in Table 1.

Table 1: Overview of the assignments in the Summer Semester 2024

No	Ratio	Content	Publish	Duration	Due Date
1	15%	Multiple-Choice Quiz			25.04.24
2	20%	Design document	18.04.24	3	09.05.24
3	40%	Backend and Test-cases	09.05.24	6	20.06.24
4	15%	Multiple-Choice Quiz			27.06.24
5	10%	Retrospective Document	27.06.24	4	25.07.24

This document describes the third of five assignments. You can earn 36 out of 90 points for this task.

2 Organization of Portfolio Exam 3

This document contains the description of the third portfolio assignment. It is published on 2024-05-09 on Moodle. The assignment is due on 2024-06-20 at 6 pm. You can ask questions about the assignment during lecture hours. For urgent or very individual questions, you can also use the Mattermost channel or email.

This assignment's deliverables are

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

- a **short video** (this can be added to the repository or store the video anywhere you like and where we can access it, e.g., in the THWS cloud, on Youtube, etc.). In the video, you must explain some technical aspects of your implementation.

You must submit a text document to Moodle before the deadline containing information on where to find these two deliverables.

Each solution is reviewed and jointly evaluated by two examiners.

3 Application Domain

Our Department of Computer Science needs an IT system to manage partner universities. A partner university is a university in another country with which our department has agreed to exchange students. For a partner university, we need to keep the following information in the database; the list is not exhaustive and can be extended:

- Name of the university (e.g., Christ University)
- Country (e.g., India)
- Name of the Department with which we have the contract (e.g., Department of Information Technology)
- URL of the department's Website
- Name of a contact person (e.g., Dr. Ravi Kumar)
- Number of students we can send to the partner university (must be ≥ 0)
- Number of students we can accept from the partner university (must be ≥ 0)
- First day of the next spring semester (e.g. 25 January 2025)
- First day of the next autumn semester (e.g. 1 September 2024)

The system administrator will update the last two dates manually every year. You don't need to check for plausibility on these dates.

We want to save information about modules (only those offered at partner universities) our students can take during their semester abroad. The list is not exhaustive and can be extended:

- Module name (e.g. Introduction to Quantum Computing)
- Semester when this module is usually offered (e.g. 1 for spring, 2 for autumn)
- Number of credit points (must be a number > 0, can be only integer)

The relationship between partner universities and modules is one-to-many (and not many-to-many), i.e. for one university, we keep a list of many modules, but each module is only offered at exactly one university. Modules do not exist without a partner university.

4 Requirements of the Software

4.1 General Requirements

You must design a RESTful client/server system with CRUD, hypermedia, filtering, and paging. It is not necessary to implement authentication or caching. It is sufficient to store the data in an in-memory storage as shown in videos and provided as part of the template for this assignment. You can also use a (more) real database, e.g., H2 (Java-based, is already integrated as Maven dependency in the template project; can be used in memory or file-based), MySQL (and similar systems), or even NoSQL databases like CouchDB or MongoDB. The database is not the core of this assignment; you can choose any system you like or want to learn.

Your implementation must consist of the following modules:

- The **backend** with a REST API, the business logic, and the database layer.
- The **frontend** is meant to be a module (class) that uses the REST API of the backend. The frontend must be implemented RESTful, i.e., it knows the allowed next actions and manages all URLs. This class only knows the dispatcher URL as a hard-coded value and no other URLs. You don't implement a graphical user interface or any other form of user interaction.
- **Test cases** use the frontend (the class you implemented in the last point) and work against the backend, i.e., you implement integration tests. Test cases don't know about URLs; no URL should be hard-coded in any test case. You don't need to implement unit tests.

4.2 Functional Requirements

The backend must provide HTTP operations on URLs to:

- Create, read, update, and delete partner universities
- Search for partner universities (you decide on the query parameters and default values), including ordering (ascending and descending, you decide on the attributes used for ordering, but it must be at least possible to order by the university's name).
- Create, read, update, and delete modules
- It is not necessary to search for modules

The frontend must provide methods for using all backend functions, and the test cases must check all backend functions using the front end.

The most important technical requirement is, of course, the hypermedia principle. That means you must provide hyperlinks to drive the application state. The backend must provide:

- Self links **as an attribute** of the resource (and not in the HTTP response header)
- Links to sub-resources **as an attribute** of the resource (and not in the HTTP response header)
- All other links **must** be part of the response header (and not of the HTTP response body)

Clients must only know the URL of the dispatcher and must be informed by the backend about all possible next actions by URLs and relations types. The Hypermedia principle must be applied in particular when working with the collection of partner universities. The response must contain:

- Hyperlinks to the previous and next page of the collection (if available, e.g., when the last page is reached, there is no link to the next page)
- Hyperlinks to queries, i.e., the frontend receives URL query templates and must only replace the placeholder with values
- Hyperlinks to set or reverse the order (ascending, descending).

4.3 Other Technical Requirements

- Your implementation must be written in Java or Kotlin.
- The backend must be executable as a Docker container. The template project contains a Maven configuration file that uses Docker. Use this as an example of how to use Docker.
- You must work with Maven as a build system.
- Your implementation must run when starting `mvn verify` (let Maven execute the integration tests) on the command line (that's the way we will test your implementation).
- It is recommended to use the template provided for this assignment, which contains a **new version** of the Sutton framework. This new version now works with Java 21, Tomcat 10, and Jersey 3, and we simplified the implementation of `State` classes. Check the README file in the repository for more information. The template comes with the in-memory storage you already know from the videos, which is sufficient for this assignment. The repository also contains a demo application, which shows most aspects you need for this assignment – **but not all**! It should not be necessary to modify any classes from the Sutton framework except if you find bugs. Please let me know about problems.

- If you don't want to use the template with Sutton, you can use any framework you like, e.g. Jersey (without Sutton), Spring, RestEasy, Vert.x, Play, Micronaut, Ktor, or Quarkus. You should be sure you can fulfill all of the abovementioned requirements if you go for one of these frameworks.

5 Requirements of the Video

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.

In the video, you must demonstrate the following two aspects of the assignment:

- Change the URL of the backend, for example, the context path, and change the definition of the dispatcher URL in the frontend implementation. After that, execute all test cases. All test cases should be green, of course. By this, you demonstrate that frontend and backend are only loosely coupled in the sense of REST.

- Show and briefly explain the source code used to process a GET collection request on partner universities. You must explain which Hyperlinks the backend provides to inform clients about possible queries, ordering, and paging.

Here are some more recommendations for the video:

- Record a screencast and store the video in format MP4.

- The video should have at least a screen resolution of 720p; better would be 1080p. But not more.

- The video should not be longer than 5 minutes.

- The video should not be larger than 15 to 20 MB (just a guess). Use compression tools like `ffmpeg` (Linux, macOS) or Handbrake (all platforms).

- You must record your voice. You can speak English or German.

- It is not necessary to use slides in addition to the screencast. No title page, no agenda page, no summary page. No intro or outro video, no music. But it would be nice if you say "Hi" at the beginning :-)

- The video must be playable using VLC or Quicktime Player. Please check the quality of the video before submitting it.

6 Rubrics

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 check that there was no Git commit after the deadline (2024-06-20 at 6 pm German time). Otherwise, we will stop evaluating your solution.

- 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` (or in the way you described it in the `README` file). If the system works correctly and all tests are green: +8 points.

- Next, we watch your video. If you can demonstrate that changing the base URL of the backend does not break your system: +5 points.

- We check the model classes for partner universities and modules. We also check the CRUD operations on the primary and secondary levels. If you have implemented all requirements and the relationship between the two resources: +5 points

- We double-check that the hypermedia principle for the GET collection state for partner universities shown in the video is identical to the one in the source code.

- We check that you have implemented all necessary hypermedia links (paging, filtering, sorting). We will do this by reading your source code and executing some manual tests in Postman. If all is complete: +10 points.

- We read the source code of the test cases. If you have at least one test case for every endpoint: +8 points.

If any criteria is not completely fulfilled, we will reduce the number of points.

You should not need more than about 20 hours for this assignment.