POLITECNICO DI MILANO

Software Engineering 2

CodeKataBattle - Improve your programming skills online

Acceptance Test

Document

Authors:

Nicolò Giallongo - 10764261

Giovanni Orciuolo - 10994077

Giuseppe Vitello - 10766482

Table of Contents

1 - Introduction..... 2
1.1 - Specifications of the analyzed project..... 2
1.2 - Revision History..... 2
1.3 - Document Structure..... 2
2 - Acceptance Test Report..... 3
2.1 - Installation Setup..... 3
2.2 - Test Cases..... 3
2.2.1 - Backend automated tests..... 3
2.2.2 - Frontend..... 4
2.3 - Additional Considerations..... 6
3 - Effort Spent..... 7
4 - References..... 7

1 - Introduction

1.1 - Specifications of the analyzed project

- Authors:
  - Michele Bersani (10707192)
  - Paolo Chiappini (10743051)
  - Andrea Fraschini (10725610)
- Link to repository
- Main reference documents considered:
  - IT Assignment Document AY 2023/24
  - RASD, DD and ITD for the reviewed project

1.2 - Revision History

Version	Date	Changelog
1.0	11/02/2024	First version of the document

1.3 - Document Structure

1. **Introduction**: contains identification regarding the project we have analyzed.
2. **Acceptance Test Report**:
   a. Installation setup: what we did to install the prototype as well as any problem we have faced in this phase, as well as any incoherence we have found when following the documentation accompanying the software).
   b. The acceptance test cases we have applied and the corresponding outcome.
   c. Any additional point raised on the quality of documentation and code.
3. **Effort Spent**: contains information about the number of hours each group member has worked on this document.
4. **References**: contains information about the resources used to redact this document.

2 - Acceptance Test Report

2.1 - Installation Setup

The following list of steps were taken to install the prototype:

- Clone the repository on our own machine;
- Analyze README.md and the ITD document to learn how to install and run the application;

For the backend:

- Launch the authentication microservice on Docker, using docker-compose;
    - **Problem**: Server container fails to launch due to the fact that the DB container has not finished setting up yet. A restart of the container is required, as it is not automatic. This problem might be solved by adding a health-check configuration to the MySQL service inside the docker-compose.yml configuration file.
- Setup main application properties using environment variables;
- Launch the main application through IntelliJ IDEA;

For the frontend:

- Install dependencies through the shell command **`npm run install`**;
- Check .env for coherence with our instance of the backend;
- Run the shell command **`npm run preview`**;
    - **Problem**: The command fails with an error stating that the project has not been built. We need to run **`npm run build`** to solve this issue, but it's not stated anywhere in the documentation accompanying the software.
    - **Problem**: **`npm run build`** errors out, so we used **`npm run dev`** to run the frontend.

2.2 - Test Cases

2.2.1 - Backend automated tests

Running automated tests both on the backend and on the authentication service was quick and easy, without any issues. All of the provided 71 tests passed (55 for CKB application + 16 for the Authentication microservice). The tests are complete and test the application layer as a whole.

CKB application coverage: >80%
Authentication Microservice coverage: >90%

2.2.2 - Frontend

The frontend does not offer automated tests. As such, we tested the system in a manual way, referencing the list of implemented features in the provided ITD document. Below, we list the tested features based on the implemented functional requirements, along with the result:

Code	Description	Result	Description
R1	The system allows the user to sign up.	**PASS {Ref: 2.3.3}**	OK, but errors are not handled properly.
R2	The system allows the registered user to log in.	**PASS {Ref: 2.3.3}**	OK, but errors are not handled properly.
R3	The system allows the student to subscribe to tournaments.	**PASS**	OK
R4	The system allows the student to enroll in a battle.	**PASS**	OK
R5	The system allows the student to invite another student to a battle, creating a group.	**PASS**	OK
R6	The system allows the educator to create a tournament.	**PASS**	OK
R8	The system allows the educator to create a battle within a tournament.	**PASS**	There is a problem related to scheduled tasks being executed immediately. Avoided the issue by commenting the offending code.
R9	The system allows the educator to grant permissions to another educator.	**PASS**	OK
R10	The system allows the educator to make a manual assessment of the solution provided by the groups if specified in the scoring configurations.	**N/A**	Not testable due to R12 unavoidable failure.

R11	The system must create the GitHub repository containing the code kata for the battle.	**PASS {Ref: 2.3.2}**	The system creates the repository but the routine to populate it with files fails. Avoided the issue by manually uploading files on GitHub.
R12	The system must run tests and give an evaluation of the provided solutions.	**FAIL {Ref: 2.3.2}**	The endpoint to submit solutions does not work (probably due to a JWT filter misconfiguration).
R13	The system allows the user to see the current rank evolving during the battle.	**N/A**	Not testable due to R12 unavoidable failure.
R14	The system must update the personal tournament score of each student, that is the sum of all battle scores received in that tournament, at the end of each battle.	**N/A**	Not testable due to R12 unavoidable failure.
R15	The system allows the educator who created the tournament to close it.	**PASS**	
R18	The system should automatically close battles after their submission deadline is reached.	**FAIL**	There is a problem related to scheduled tasks being executed immediately.
R19	The system allows users to search by battle and by tournament.	**PASS**	
R20	The system allows invited users to either accept or reject.	**PASS**	
R21	If configured, when the battle ends the system must notify the educators of the tournament that a manual evaluation is required	**FAIL**	There is a problem related to scheduled tasks being executed

	to consolidate scores.		immediately.

Test on requirement R8 passed with issues, tests on R18 and R21 failed due to the identical issue of scheduled tasks being executed immediately regardless of execution date.

2.3 - Additional Considerations

The following points were raised during the review of the project:

- {2.3.1} **[BACKEND] Missing standard API contract**: The developers gave us a Postman collection available for consumption privately, however this is not enough to guarantee an API contract (for example, API clients can't be generated automatically and, as such, there might be discrepancies between API versions). To resolve this issue, a standard like OpenAPI might be adopted. Spring Boot offers a way to automatically generate the contract in JSON format based on the provided controllers, so it would be easy to adopt for this project.
- {2.3.2} **[BACKEND] Missing proper failure handling for asynchronous routines**: Many features of the application are managed through a separate task executor (with a thread pool). These are managed by Spring itself and are implemented in a proper, correct way. The identified issue is that there is no logging in case of an error during an async procedure. For example, if the routine to upload project files fails for some reason, there are no errors logged in the console, and the GitHub repository remains empty.
- {2.3.3} **[BACKEND] Missing proper handling of errors coming from the Auth Microservice**: API calls through Unirest do not take into account an error response from the microservice (e.g. 404 or 500), only taking into account invocation exceptions. This causes subsequent JSON mapping exceptions which shadow the actual exception, requiring further debugging to be inspected.
- {2.3.4} **[FRONTEND] Missing possibility of logging out of the application**: at the moment, the only way of "logging out" is to log in as another user.
- {2.3.5} **[BACKEND] Tests rely too much on mocks to function**: mocks are useful when the developer does not want to test a particular external integration. However, in this project, there are too many important things which are mocked, rendering some tests incapable of determining the correct functioning of the application, which in turn gives errors in a real environment.

3 - Effort Spent

Group Member	Effort Spent (hours)
Nicolò Giallongo	8h
Giovanni Orciuolo	8h
Giuseppe Vitello	8h

4 - References

- Version Control System: GitHub (https://github.com)
- Document written with: Google Documents (https://docs.google.com)