POLITECNICO DI MILANO

Computer Science and Engineering

Implementation and Test Deliverable

CodeKataBattle

Authors:

Manuela Marenghi
Luca Cattani
Tommaso Fellegara

Professor:

Matteo Giovanni Rossi

Table of Contents

1. Introduction

1.1 Purpose

In this document are listed the implemented requirements, used languages and frameworks and how the testing phase has been carried out.

1.2 Scope

This document's aim is to define and report about the software implementation and test procedures defined for all the released software. It verifies that the CKB platform satisfies its functional requirements, to this end, we define a set of tests covering the different uses. The main goal of these tests is to ensure the correct behavior of the system in multiple scenarios.

1.3 Document structure

1. **Introduction:** the purpose of this document
2. **Implemented Requirements:** the list of functional requirements that are actually implemented in the software
3. **Adopted Development Frameworks:** adopted libraries, frameworks and languages used to make the CKB platform possible
4. **Structure of the source code:** explains how the source code is structured
5. **Testing:** provides an explanation of how the application has been tested
6. **Installation instructions:** explains how to install and deploy the application
7. **Effort spent:** keeps track of the time spent to complete this document

1.4 Definitions, acronyms, abbreviations

1.4.1 Definitions

- **Code kata** - the set of textual description, test cases and build automation scripts that form a coding problem users on the platform have to solve.
- **Code battle** - the grouping of code kata and battle settings, described by an educator, that constitute a coding challenge on the platform. Note that code battles are also simply referred to as "battles" in this document.
- **Tournament** - a collection of code battles created by one or more educators.
- **Users** - everyone using the platform, that is, students, educators and everyone who is browsing the platform and is not logged in yet.

1.4.2 Acronyms

- **CKB** - CodeKataBattle
- **API** - Application Programming Interface

1.4.3. Abbreviations

- **[Ri]** - i-th requirement

2. Implemented Requirements

Below are listed the requirements implemented in the software.
[R1] The system allows users to sign up
Implemented

[R2] The system allows users to sign in
Implemented

[R3] The system allows users to browse other users profiles
Implemented

[R4] The system allows users to browse the list of tournaments
Implemented

[R5] The system allows users to browse tournament rankings
Implemented

[R6] The system allows users to browse battle rankings
Implemented

[R7] The system allows educators to create tournaments
Implemented

[R8] The system allows educators to specify a tournament registration deadline
Implemented

[R9] The system allows students to join tournaments
Implemented

[R10] The system allows educators to define gamification badges, consisting in a title and one or more rules that must be fulfilled for a student to obtain the badge
Not implemented because not required by the assignment

[R11] The system requires educators to define a way to assign scores to students submission in an automated way
Implemented

[R12] The system allows educators to decide whether they have to assign personal scores to students solutions during the consolidation phase
Implemented

[R13] The system allows educators to assign a personal score during the consolidation phase if they decided to allow it when creating the battle
Implemented

[R14] The system allows the creator of a battle to terminate the consolidation phase after having evaluated all of the groups sources (if they decided to do so when creating the battle), effectively terminating the battle

Implemented

[R15] The system notifies all students subscribed to the platform whenever a new tournament is created

Implemented

[R16] The system notifies students when a battle is created if they are registered to that battle’s tournament

Implemented

[R17] The system notifies students when the battle’s final rankings become available

Implemented

[R18] The system notifies students when the final tournament ranking become available

Implemented

[R19] The system provides all students subscribed to a battle with that battles’s code kata by notifying them with a link to that code kata’s GitHub repository when a battle’s registration deadline expires if they are subscribed

Implemented

[R20] The system allows educators to create coding battles for a specific tournament if they either have been given permission from the tournament creator to do so or they created that tournament

Implemented

[R21] The system allows educators to grant permission to create new coding battles for a tournament they have created to other educators

Implemented

[R22] The system allows students to create a group for each battle

Implemented

[R23] The system allows students to invite other students to their group for a battle

Implemented

[R24] The system allows students to join a group for a battle

Implemented

[R25] The system allows students to compete in a coding battle if, when the registration deadline expires, they are part of a team composed by a number of students within the boundaries defined by that battle’s creator

Implemented

[R26] The system allows educators to specify battle deadlines when creating a new battle

Implemented

[R27] The system allows educators to specify boundaries for the number of students in each group when creating a new battle

Implemented

[R28] The system allows students and educators to see evolving rankings before a code battle has reached its submission deadline

Implemented

[R29] The system allows educators to upload code katas when creating a new battle by providing a textual description, a set of test cases and build automation scripts

Implemented

[R30] The system provides an API to allow users to submit their solution to a code battle, triggering the system to run automated tests to analyze the students code

Implemented

[R31] The system allow educators to create new variables to use during the definition of the rules for gamification badges, using a pseudo-language

Not implemented because not required by the assignment

3 Adopted Development Frameworks

3.1 Programming languages

3.1.1 Java

Java is a versatile, object-oriented programming language favored for its ability to run on any platform with the Java Virtual Machine (JVM).
Its advantages include its simplicity, extensive built-in libraries for various tasks, and strong support for multithreading, making it ideal for developing scalable applications.
However, its verbosity and boilerplate code can lead to slower development times, and its memory consumption can be higher compared to languages like C or C++. Additionally, while Java's garbage collection simplifies memory management, it can sometimes result in unpredictable pauses, impacting real-time applications.
In this project java was used to implement all api endpoints and, more in general, the bulk of the backend.

3.1.2 Javascript

JavaScript is a multi-paradigm language, supporting both object-oriented and functional programming styles and is also supported by all major web browsers, this makes it flexible and adaptable for various types of projects. JavaScript also has a large developer community and an intuitive syntax, meaning that there are plenty of resources, tutorials, and forums available for developers to seek help when needed. However while JavaScript is supported by all major browsers, there may be differences in how the code is interpreted and executed by different browsers and, since it’s executed on the client side, it can be vulnerable to security threats.
Moreover, it’s lack of native support for multithreading, paired with the fact that it’s not a compiled language, can limit performance.

3.1.3 Bash

Bash, is a powerful and widely-used scripting language primarily designed for command-line interaction in Unix-like operating systems. It excels at automating repetitive tasks and managing system processes, making it an ideal choice for various project scenarios.
Its versatility lies in its ability to handle file manipulation, process execution, and system administration tasks efficiently. Integrating Bash scripts into a project enhances productivity by automating routine operations, streamlining workflows, and ensuring consistency. Additionally, Bash scripts can be easily adapted and extended, promoting scalability and maintainability in project development.
However, bash scripting can be less forgiving in terms of error handling compared to some other languages, has limited support for complex data structures and ss scripts grow in complexity, maintaining readability can become challenging very fast. Additionally, writing insecure code in Bash scripts can lead to vulnerabilities, especially if the scripts involve user input or interact with external systems
In our case, bash scripting was used to both develop internal tools used to enhance productivity and implement parts of the application that handle file operations.

3.2 Libraries, plugins and frameworks

- **Docker**: is a platform that enables developers to package applications and their dependencies into lightweight, portable containers. It is used to simulate the microservices system in local
- **Maven**: is a build automation tool used primarily for Java projects
- **Spring Boot**: is a framework built on top of the Spring framework, designed to simplify the development of Java applications by providing a convention-over-configuration approach.
- **JUnit**: testing framework for java projects
- **Mock Server**: a library providing the implementation of mock servers, used to mock the behavior of other servers during testing
- **Lombok**: library that helps reduce boilerplate code in Java project
- **Eureka Server:** is a part of the Netflix Eureka framework, which is used for service registration and discovery in a microservices environment
- **Mysql Connector**: a JDBC driver
- **Github Api**: a library that defines an object oriented representation of the GitHub API

4. Structure of the source code

4.1 Server side

The source code of the application can be found in the *code* directory, here multiple directories can be found, containing the implementation of the application, the server side is split in a total of six microservices:

- Account manager
- Mail service
- Github manager
- Tournament manager
- Battle manager
- Solution evaluation service

In addition to the above services, we've implemented an api gateway and a discovery server to provide more scalability and increase decoupling among services present, these can be found in the api-gateway and discovery-server directories respectively.

The code of each of the microservices is organized in the following packages:

- **DTO** (Data transfer object) which is itself split in dto.in and dto.out: this package includes POJOs representing the format of the input data that is to be received and sent respectively.
- **Controller**: this package contains the controllers of the microservice which actually implement the api endpoints provided by the application
- **Service**: this package contains the classes that runs the business logic of the microservice

Moreover if a service has to connect to a database, following packages are also present:

- **Model**: this package contains the entities of the microservices if they use the database or if they receive an entity through the Internet
- **Repository**: this package contains the repository interfaces of spring boot through them it is possible to perform CRUD actions on the database

If any service needs to include configuration files or classes:

- **Config**: this package contains any necessary configuration files

In addition, for services that make use of bash scripts, the scripts are located in a *scripts* directory under the *main* directory of the interested services.

Finally in the root of the code directory, a total of four files can be found:

- **pom.xml** the parent pom of the project, providing common dependencies to all the submodules
- **docker-compose.yml** the docker compose file allowing for an easy way to run the whole backend of the application
- **docker-build.sh** a script that builds all docker images present in the project named as <first input argument>/<directory where the Dockerfile is located>
- **docker-build-push.sh** a script that builds and pushes all docker images to dockerHub, used by the repositories GitHub actions workflows

4.2 Client side

The client side can be accessed from the web-app directory where there is the structure of all pages. The homepage of the web-app is in the index.html file that is the start of the flow. Then all other html files are in the same directory representing other pages of the site, some accessible only from others.
There are also two other directories:

- ***utility*** that contains some supportive js files, with functions used in html files
- ***public*** that contains a directory for files containing css in order to decorate html elements and a directory for images

4.3 Used ports

- **3000** → web app (client)
- **8080** → api gateway
- **8081** → solution evaluation service
- **8082** → battle manager
- **8083** → github manager
- **8085** → mail service
- **8086** → account manager
- **8087** → tournament manager
- **8761** → discovery server

5. Testing

The testing process has been split in two main stages

5.1 Unit testing

When the software is under development, it undergoes unit testing to ensure that each functionality works as expected, particular focus is put on “trying to break the functionality” to also ensure that failures are handled gracefully.
During this initial phase, each component should be tested as a stand-alone unit, so interactions with external components should be simulated using mock servers and services that rely on interactions with databases should exploit a test database with fake data.
After this testing phase is successful, the newly developed sources are to be pushed to github where a github action will be triggered to re-execute unit tests and exclude any platform bias.

5.2 Integration testing

After a component has been thoroughly unit tested, more tests can be carried out without the simulation of external components, effectively simulating how the application will run in a production-like environment.
This phase can first be performed by locally testing the components by running them on a testing machine, then tests should be run on docker containers to simulate a production environment even more closely.
After these tests are locally carried out, the sources should be pushed to github where a workflow will perform all of the application’s unit tests (to ensure nothing broke in the process of testing), build all of the service docker images and run them with docker compose, then all integration tests present in the repository are performed, effectively performing end-to-end tests, simulating in the closest possible way how the application will run in production

6. Installation instructions

The first step required to run the application is to clone the repository, this can be done by following the instructions on GitHub.

6.1 Backend

The only requirement to run the application backend is to have docker compose V2 installed on your system.
The whole backend infrastructure can be easily and platform-independently installed and run by exploiting docker containers, in the *code* directory of the project, a *docker-compose.yml* can be used to start the application both with official and unofficial images:

- **Official images**
    - To run the official images it's sufficient to open a command line interface in the *code* directory and run the *docker compose up* command
    - After having run the command, docker will start pulling the official images of the application from DockerHub
    - After a few minutes, when all images have been downloaded, docker will run all images
    - As the application is starting, you can check which services have started by connecting to *localhost:8761* which is the address of the discovery server, once all the 6 services services and the api gateway are listed there, the backend is finally running
- **Unofficial images**
    - You can apply any modifications to the sources and build your own images by following the next few steps, which are supposed to be performed on a system that has a running version of docker and maven, all commands are to be run in the *code* directory
    - Build and package the application by running the *mvn clean install -DskipTests* command
    - If you can run bash scripts, run the *docker-build.sh* script present in the *code* directory by running *./docker-build.sh <your-test-tag>* this will build all docker of the project's images, and name them <your-test-tag>/<service-name>
    - The only thing left is to actually run the new images! To do that you can run the *ID=<your-test-tag> docker compose up* command in the *code* directory or modifying the docker compose file to user the images you've just built

6.2 Frontend

To use the application, after having run the docker compose file, you need to host an http server in the *web app* directory to be able to fetch the necessary files, the following steps are to be run in a linux distribution, or a WSL instance

- Run the *sudo apt install npm* command to download the Node Package Manager (if you do not have apt on your machine, substitute it with a package manager of your preference, e.g., yum, pacman, …)
- After having installed npm, you'll need to install the *http-server* command by running the *sudo npm install --global http-server* command

- Now you are all set to host an http-server on your machine, go in the *code/web-app* directory of the project and run the *http-server ./ -p 3000* command
- You can now finally search for localhost:3000 in your browser to use the frontend of the application

7. Effort spent

Tommaso Fellegara

Introduction	1
Implementation Requirements	1
Adopted Development Frameworks	1
Structure of the source code	0
Testing	1
Installation instructions	1

Manuela Marenghi

Introduction	1
Implementation Requirements	0
Adopted Development Frameworks	1
Structure of the source code	1
Testing	1
Installation instructions	1

Cattani Luca

Introduction	0
Implementation Requirements	1
Adopted Development Frameworks	1
Structure of the source code	1
Testing	1
Installation instructions	1