# ITD

Federico Valentino, Nicola Zarbo

January 2023

# Contents

# Chapter 1

# Introduction

## 1.1 Purpose

CodeKataBattle (CKB) is a new platform that helps students improve their software development skills by training with peers on code katas . Educators use the platform to challenge students by creating code kata battles in which teams of students can compete against each other, thus proving (and improving) their skills.
A code kata battle is essentially a programming exercise in a programming language of choice (e.g., Java, Python). The exercise includes a brief textual description and a software project with build automation scripts (e.g., a Gradle project in case of Java sources) that contains a set of test cases that the program must pass, but without the program implementation. Students are asked to complete the project with their code. In particular, groups of students participating in a battle are expected to follow a test-first approach and develop a solution that passes the required tests. Groups deliver their solution to the platform (by the end of the battle). At the end of the battle, the platform assigns scores to groups to create a competition rank.

## 1.2 Scope

This document aims to describe how the implementation and integration testing took place. This is the last step in the CKB platform development cycle. Testing means to check that the application works as expected and described in the DD document.

## 1.3 Definitions, Acronyms, Abbreviations

| Acronym | Definition |
|---|---|
| CKB | CodeKataBattle |
| CK | Code Kata |
| UI | User Interface |
| SOA | Service Oriented Architecture |
| API | Application Programming Interface |
| HTML | HyperText Markup Language |
| CSS | Cascading Style Sheet |
| JS | JavaScript |
| DBMS | DataBase Management System |

Table 1.1: Acronyms used in the document

## 1.4 Revision History

## 1.5 Reference Documents

- RASD Document for CKB
- DD Document for CKB

## 1.6 Document Structure

The document is structured as follows:

- Chapter 1: Introduction to the document and its scope
- Chapter 2: Report on the implemented functions, modules in the application and mapping to requirements and goals
- Chapter 3: Contains the adopted development frameworks
- Chapter 4: Explains the structure of the source code
- Chapter 5: Contains all the information on how we performed the testing
- Chapter 6: Contains all the system requirements and installation instructions for the application

# Chapter 2

# Requirements and functions implemented

## 2.1 Implemented Functions

With respect to the RASD and DD documents, we decided to implement the following functions:

- Educator implemented functions:
  - createTournament();
  - viewTournaments();
  - viewTournament();
  - createBattle();
  - viewBattle();
  - closeTournament();
  - addCollaborator();
  - manualEval();
  - viewLeaderboard();
- Student implemented functions:
  - viewTournaments();
  - viewTournament();
  - viewBattle();
  - joinBattle();
  - subscribeToTournament();
  - viewLeaderboard();

## 2.2 Implemented Modules

We decided to implement the following modules:

- Web Server
- API Gateway
- Message Broker
- User Service
- Leaderboard Service
- Tournament Service
- Battle Service
- ManualEval Service
- Notification Service(It isn't connected to any email service but outputs what it would've sent in an email)
- Application and Source DBMS

The GitHub handler and software evaluation modules haven't been implemented since they are not required for showing the basic functionalities of the system. Next follows a table representing which requirements have been satisfied.

| **ID** | **Description** | |
|---|---|---|
| R1 | The platform allows a signed in educator to create tournaments | ✓ |
| R2 | Educators can create battles in the context of a specific tournament they are involved in (Either by creation or by invitation) | ✓ |
| R2.1 | The platform allows an educator that created a tournament, to invite other educator and to grant them permission to create battles in the tournament context | ✓ |
| R2.2 | The platform allows an educator to upload the codekata (description and software project, including test cases and build automation scripts) when creating a battle | ✓ |
| R2.3 | The platform allows an educator to set subscribtion and submission deadlines when creating a battle | ✓ |
| R2.4 | The platform allows an educator to set minimum and maximum group size when creating a battle | ✓ |
| R2.5 | The platform allows an educator creating a battle to include a manual evaluation stage | |
| R3 | The platform allows students to subscribe to a tournament | ✓ |
| R4 | The platform allows a student subscribed to a tournament to join a battle in that tournament context | ✓ |

| | | |
|---|---|---|
| R4.1 | The platform allows students to create a group by inviting other students when joinin a battle | ✓ |
| R5 | The platform allows Students to login | ✓ |
| R6 | The platform allows Educators to login | ✓ |
| R7 | If manual evaluation is required during consolidation stage the platform allows an educator to go through sources and add a score to the group score | ✓ |
| R8 | The platform allows all users to view student ranks from previous and current tournaments | ✓ |
| R8.1 | For every tournament the platform maintains a ranking for students based on the sum of their battle scores | ✓ |
| R9 | The platform pulls group sources from Github when it receives a notification within the submission deadline | |
| R9.1 | The platform analyzes, runs testcases and scores the source of a group solution for a codekata battle and updates the group score accordingly | |
| R10 | The platform allows all user to sign in | ✓ |
| R11 | The platform allows educators to close tournaments | ✓ |
| R12 | When a battle deadline expires the platform sets the battle status to consolidation stage | ✓ |
| R13 | The platform allows all users who are involved in a battle to look at group ranks for that battle | ✓ |
| R13.1 | For every battle the platform maintains a ranking of the groups based on their battle score | ✓ |
| R14 | The platform will notify signed students of a newly created tournament | ✓ |
| R15 | The platform shall notify students who are subscribed into a tournament that a new battle is available in that tournament context | ✓ |
| R16 | The plaftorm shall notify students who are participating in a battle that the final rank for that battle is available | ✓ |
| R17 | The plaftorm shall notify students who are subscribed in a tournament that the final rank for that tournament is available | ✓ |
| R18 | The platform shall create a new repository with Github for every new battle in any tournament after the submission deadline expires | |
| R19 | The platform shall send the battle repository link to students who joined that battle after the submission deadline expires | ✓ |

With respect to the RASD document then these are the following goals that have been satisfied:

| **ID** | **Description** | |
|---|---|---|
| G1 | An Educator can manage a tournament. | ✓ |
| G2 | An educator can create battles inside of a tournament in which he is involved. | ✓ |
| G3 | Students can participate in tournaments created by an educator | ✓ |
| G4 | Students can participate and compete in battles created by an educator, alone or in groups. | |
| G5 | Students are scored based on their performance in battles. | |
| G6 | The platform allows students and educators to compare the performance of students. | ✓ |

# Chapter 3

# Adopted Development Frameworks

As said in our DD document, the application is a micro-service architecture, with a REST interface and some client scripting. In the following sections you will find a list of adopted frameworks and technologies in order to accomplish our requirements.

## 3.1 Programming Languages

For sake of standards and application speed we decided to use the Java Programming Language, which is one of the most used languages in web and distributed applications. Of course, there are some pros and cons about using this type of language:

- Pros:
  - Speed: of course, since it is a compiled language, Java permits to have very good performances on these elaborations;
  - Standard: as shown in figure 1, Java is the De Facto standard in enterprise web development and represents a very good solution for portable applications;
  - Stability: Java is a mature language that has immensely evolved over the years. Hence it's more stable and predictable.
  - Object-Oriented: The object-oriented nature of Java allows developers to create modular programs and write reusable codes. This saves lots of efforts and time, improving the productivity of the development process.
  - Well-documented

- Cons:
  - High verbosity: with respect to other programming languages (such as Python), Java contains a more verbose and less-readable syntax.
  - High memory consumption: since Java Programs run on top of Java Virtual Machine, it consumes more memory.

For the client-side scripting, we decided to adopt JavaScript for the web app. JavaScript is a text-based programming language used both on the client- side and server-side that allows to make web pages interactive. We decided to develop our web pages from scratch by using HTML and CSS. This choice gave us way more freedom in the customization of the web pages without having to rely on third party services.

## 3.2 Java Frameworks

### 3.2.1 Spring

In order to accomplish the goal of having a micro-service event-based architecture, we decided to adopt the Spring development framework. Spring is an open source framework, used for RESTful Java application development. It's built on top of the Java Enterprise Edition (JEE) and represents an efficient and modern alternative to the classic Enterprise Java Bean (EJB) model. Of course, using a framework means working on a solid base, which is well tested and documented. In fact, Spring contains a proper paradigm in order to build micro-services and managing events on its API.

## 3.3 Others

### 3.3.1 Java Server Pages and Java Servlets

To build the WebServer that will host the web pages we decided to use JSPs and Java Servlets because of their ease of use. Java servlets and Java server pages (JSPs) are Java programs that run on a Java application server and extend the capabilities of the Web server. Java servlets are Java classes that are designed to respond to HTTP requests in the context of a Web application. You can look at JSPs as an extension of HTML that gives you the ability to seamlessly embed snippets of Java code within HTML pages. These bits of Java code generate dynamic content, which is embedded within the other HTML/XML content. A JSP is translated into a Java servlet and executed on the server. JSP statements embedded in the JSP become part of the servlet generated from the JSP. The resulting servlet is executed on the server.

# Chapter 4

# Source Code Structure

## 4.1 Microservices and APIGateway

Microservices and the APIGateway are packaged through the use of the Gradle framework.

### 4.1.1 Packages

**Package it.polimi.se2.codekata**

- **DBMS:** this package contains all the Data Base Management System classes and relational database objects for the CKB application.
- **GeneralStuff:** it contains classes and enums that don't belong in particular to any other package.
- **Microservices:** it contains all the microservices classes for the CKB platform.
- **Topics:** it contains all the events the message broker will process.
- **WebUtil:** this package contains the code for the API Gateway

## 4.2 WebServer

The WebServer has been packaged through the use of the Gradle framework.

### 4.2.1 Packages

**Package it.polimi.se2.codekata.webserver**

- **Server:** it contains all the classes for the servlets

### 4.2.2 Others

All the source code for the web pages is inside the WebServer/src/webapp folder.

# Chapter 5

# Testing

In this section we will briefly describe how we tested the application, following the general guidelines given in the Design Document. We decided to write test cases only the Microservices and API gateway. Instead the web app was tested by manually searching for possible errors/bugs/glitches.

## 5.1 Microservices and API gateway

We wrote the test cases using the JUnit 5 suite and by following the suggested Spring testing approach. We did unit testing for all the microservices and database handlers (DBs do not really exists, but there are stubs handling them). Next we followed by some integration testing.

### 5.1.1 Integration Tests

All the tests described are considered integration tests because every component needed to interact one way or another with other components in order to achieve success.

- **Battle Service:** We tested the methods regarding the battle interaction, addBattle and joinBattle. In the test for addBattle we also tested the collaboratoration option for a tournament. The joinBattle method instead has been tested by adding different groups for a battle and by making sure the addition satisfied the group constraints.
- **Leaderboard Service:** We tested the getLeaderboardTournament() and getLeaderboardBattle() methods. Both tests regarded the assertion that the leaderboards were ordered correctly.
- **ManualEval Service:** We tested the addManualScore() method by adding a random battle to the DB, making a team join that and using the method to add a score.

- **Tournament Service:** We tested all the method which modify entries in the DB.
- **User Service:** We tested the login() and register() functions in order to assert the correct return of users IDs.
- **API Gateway:** We tested every method to see if the returning JSON string was in the correct format through the use of REGEX strings.

The above tests, which were in total 26 resulted in a 100% success rate. Thanks to the obtained result it is therefore possible to state that the back-end is sturdy and built on solid source code.

### 5.1.2 Unit Tests

Furthermore, we made some tests on the proper functioning of the DBMS. On a total of 11 tests the DBMS was successful in all of them.

# Chapter 6

# Installation Instructions

### 6.0.1 Requirements

- Java SE JDK 21(OracleJDK, OpenJDK)
- Apache Tomcat Server v9.0

## 6.1 Microservices and API gateway

### 6.1.1 Installation

- Download the application JAR package from the Release page on our Github repo
- Open your terminal and navigate to the folder hosting the JAR
- Launch from your terminal with the following command:

```
java -jar CodeKataMicroservices.jar
```

### 6.1.2 Build from source

- Start by cloning the GitHub repo
- Open a terminal and navigate to the CodeKataBattleFolder
- Start the build process by typing
  - For Linux:

    ```
    ./gradlew bootJar
    ```

  - For Windows:

    ```
    gradlew.bat bootJar
    ```

- Next navigate to the build/libs folder where you will find the compiled JAR file.

## 6.2 Web Server

### 6.2.1 Installation

- Download the .war file for the web application from the Release page on our Github repo
- Download Apache Tomcat Server v9.0
- Follow the installation process for Tomcat
- Install the webapp on the Tomcat Server
  - With tomcat gui: Run the tomcat server on localhost and from the server homepage select "Manager App". On this page use the deploy form to install the war file.
  - With file explorer : inside the tomcat folder find "/webapps", from here copy the .war file.
- Set enviroment variables:
  - For Windows: in "apache-tomcat-9.xxx/bin/" create file "setenv.bat". Inside copy "set APIGATEWAY_URL=http://<URL_TO_SERVICES_WITH_PORT>.
  - For Linux: in "apache-tomcat-9.xxx/bin/" create file "setenv.sh" write "EXPORT APIGATEWAY_URL="http://<URL_TO_SERVICES_WITH_PORT>".
  - (the quotation marks are necessary for linux)
- Run the server with the files startup.bat or startup.sh depending on your OS.
- Connect to http://localhost:8080/CKBWebServer

### 6.2.2 Build WAR file from source

- Start by cloning the GitHub repo
- Open a terminal and navigate to the WebServer Folder
- Start the build process by typing
  - For Linux:

    ```
    ./gradlew war
    ```

  - For Windows:

    ```
    gradlew.bat war
    ```

- Next navigate to the build/libs folder where you will find the compiled WAR file.
- Follow the installation instruction to start the Apache Tomcat server

# Chapter 7

# Effort spent

| Student | Time for Implementation |
|---|---|
| Valentino | 60 |
| Zarbo | 60 |

# Chapter 8

# References

## 8.1 Used tools

- GitHub for project versioning
- Overleaf as LaTeX editor
- IntelliJ Idea as Java IDE
- Spring Boot for Microservices and Events