A.Y. 2023/2024

POLITECNICO
MILANO 1863

Politecnico di Milano

ITD: Implementation Document

Davide Grazzani   Alessandro Masini

Professor
Matteo Giovanni Rossi

Version 1.0
February 6, 2024

Contents

1 Introduction

The code can be found in the official project repository on GitHub at the link: `https://github.com/d-graz/GrazzaniMasini`.

1.1 Purpose

The aim of this document is to provide a comprehensive technical overview of the system outlined in the RASD and DD document. We wil focus on the software architecture, emphasizing the interaction among system components. Furthermore, it will address the implementation, testing, and integration phases. While the document primarily targets programmers with its technical language, stakeholders are encouraged to read it as well, in order to gain an insights into the application's development's characteristics.

1.2 Definitions, acronyms, abbreviations

Acronyms

- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **ITD**: Implementation Document
- **API**: Application Programming Interface
- **DBMS**: Database Management System
- **UML**: Unified Modeling Language
- **UI**: User Interface
- **HTTPS**: HyperText Transfer Protocol Security
- **CSRF**: Cross Site Request Forgery
- **HTML**: HyperText Markup Language
- **CSS**: Cascade Style Sheet
- **JS**: JavaScript

- **JSX**: JavaScript XML
- **MVC**: Model View Controller
- **JSON**: JavaScript Object Notation
- **URL**: Uniform Resource Locator
- **ACID**: Atomicity-Consistency-Isolation-Durability
- **SQL**: Structured Query Language
- **IDE**: Integrated Development Environment

1.3 Revision history

- Version 1.0: first release.

1.4 References

- MariaDB: `https://mariadb.org/`
- Spring Boot: `https://spring.io/projects/spring-boot`
- React: `https://react.dev/`
- Axios: `https://axios-http.com/`

2 Development

2.1 Implemented Functionalities

We implemented the following functionalities:

- Creating a tournament
- Invite educators to a tournament
- Subscribe to a tournament
- Create a battle
- Automatically create a GitHub repository with the requested CodeKata
- Enroll to a battle
- Create a group
- Invite students to a group
- Automatically pull and run test cases on new push by each user
- Get the results of a battle (test cases[1])
- Get the results of a tournament (test cases[2])

2.2 Functionalities not implemented

As far as what concerns the functionalities that were not implemented the manual review of the code performed by the educators, due to time constrains, was not implemented. The static analysis of the code, since not required, was also not implemented. The implementation of the score function is partial: only the test cases are considered in order to make up for the user/group score in tournaments/battles. Notifications through email and Calendar integration were not implemented.

[1]See 2.2 section for more details.
[2]See 2.2 section for more details.

2.3 Implemented requirements

Here follows the list of requirements taken from the RASD. Implemented ones are checked with a checkmark.

R1. The system must allow the registration of new users using their e-mail ✓

R2. The system must notify a user about his/her successful registration

R3. The system must allow registered users to log-in using their e-mail ✓

R4. The system must allow logged-in users to use the application ✓

R5. The system must respect the GDPR ✓

R6. The system must allow a user to set a nickname for his/her profile $\sim$

R7. The system must allow a user to set a phone number for his/her profile

R8. The system must allow an educator to create a tournament if another tournament with the same name doesn't already exist $\sim$

R9. The system must reject an educator request to create a tournament if another tournament with the same name already exists $\sim$

R10. The system must allow the owner of a tournament to invite other educators to manage it ✓

R11. The system must notify an educator when he/she is invited to manage a tournament ✓

R12. The system must allow an educator to accept the invitation of another educator to manage a tournament ✓

R13. The system must allow an educator to set a registration deadline when creating a tournament ✓

R14. The system must allow to set a minimum number of students for each group when creating a new battle ✓

R15. The system must allow to set the maximum number of students for each group when creating a new battle ✓

R16. The system must allow to set a registration deadline when creating a new battle ✓

R17. The system must allow to set a final submission deadline when creating a new battle ✓

R18. The system must allow an educator to choose while creating a new battle which aspects of the quality level of the sources will be automatically evaluated

R19. The system must allow an educator to choose while creating a new battle whether a manual evaluation will be needed after the automated ones

R20. The system must be able to receive a new code kata from an educator when he/she is creating a battle ✓

R21. The system must allow the educators managing a tournament to create new battles in the context of such tournament ✓

R22. The system must notify all users of the platform when a new tournament is created ✓

R23. The system must notify all students subscribed to that tournament when a new battle of such tournament starts ✓

R24. The system must allow users to subscribe to a tournament ✓

R25. The system must allow students to create groups in the context of a battle ✓

R26. The system must allow a student to invite another to his/her own group, if such student is enrolled into the corresponding battle ✓

R27. The system must reject a student request to invite another student to his/her own group, if such student is not enrolled into the corresponding battle ✓

R28. The system must notify a student when he/she is invited to a group ✓

R29. The system must allow a student to join an existing group if invited ✓

R30. The system must allow groups to join a newly created battle if they respect that battle's constraints ✓

R31. The system must allow all students enrolled into a battle, to access its code kata ✓

R32. The system must allow a group to submit his/their solution ✓

R33. The system must update a group battle score after each new solution is delivered ✓

R34. The system must show to each group his/their rank in the current battle ✓

R35. The system must show to each group his/their current battle score ✓

R36. The system must notify all students subscribed to a tournament when the consolidation phase of a battle of such tournament has ended

R37. The system must be able to evaluate functional aspects of a group's delivery ✓

R38. The system must be able to evaluate the timeliness of a group's delivery

R39. The system must be able to evaluate the quality level of the sources of a group's delivery

R40. The system must show the results of its automated analysis to the educators managing the tournament $\sim$

R41. The system must allow an educator to see the solution provided by each group

R42. The system must allow an educator to manually mark a solution in the consolidation phase, if this was set up in the initial configuration of the battle

R43. The system must update the tournament score of each student participating into that tournament after the consolidation phase has ended ✓

R44. The system must allow a student to see his/her rank in a certain tournament ✓

R45. The system must allow a student to see his/her tournament score ✓

R46. The system must allow all users to see the list of ongoing tournaments ✓

R47. The system must allow all users to see the current tournament score leaderboard for each ongoing tournament ✓

R48. The system must allow an educator to close a battle before its expected conclusion ✓

R49. The system must allow an educator to set up a message explaining his/her motivations when closing a battle ahead of time

R50. The system must notify all students subscribed to a tournament when a battle in such tournament is closed ahead of time ✓

R51. The system must show the educator's motivation message when notifying students about the early closure of a battle ✓

R52. The system must allow an educator to close a tournament, if there is not an ongoing battle in such tournament ✓

R53. The system must reject an educator request to close a tournament, if there is an ongoing battle in such tournament ✓

R54. The system must notify all students subscribed to a tournament when such tournament is closed ✓

2.4 Design Choices

We created a UI, using React, that is easy to use and intuitive. We chose a simple and clean design to make the user experience as smooth as possible. The UI is focused for desktop use (and during test this was the only platform used) but it's also flexible enough to be used on mobile devices.

2.5 Adopted Development Frameworks

For the choice of the frameworks we took into account the easy of integration in respect to other frameworks, the ease of use alongside the popularly, documentation and support of such frameworks. On the front-end side the choice was React, a popular and widely used framework for building user interfaces alongside Axios for handling the API calls. For the back-end, instead, we chose Spring Boot, a popular framework for building Java-based applications, as well as MyBatis, a popular framework to map database tuples into Java object and perform queries directly from our Java code

MVC

In the MVC pattern, the View manages user requests and provides responses, the Model oversees data access and manipulation logic, and the Controller serves as a mediator between the Model and View to handle user requests and oversee data flow.

2.5.1 Programming languages

The programming languages used in the project are Java and JavaScript. We chose JavaScript for the frontend because of it's ease in creating interactive and engaging user interfaces. Meanwhile, Java was chosen for the back-end due to its robustness and scalability, as well as the fact that we were already familiar with the language. Regarding the chosen frameworks instead, on the front-end side React has been chosen in order to create a modern and responsive UI, while on the back-end side we opted for Spring Boot paired with MyBatis.

2.5.2 React Framework

Utilizing React for web development presents a multitude of advantages:

- **Fast Development**: React allows for the creation of reusable components, which can be used to build complex user interfaces at a faster rate. Moreover, React uses JSX, a syntax extension that allows for the mixing of HTML and JavaScript. This makes the code more readable and easier to write. It also uses one-way data binding, which means that the data flows in one direction, from the parent component to the child component which makes the code more predictable and easier to debug.

- **Performance**: React uses a virtual DOM, which is a lightweight copy of the actual DOM. This allows for faster updates and better performance.

- **Community and support**: React has a large and active community, which means that there are plenty of resources, third party libraries integration and support available.

2.5.3 Spring Boot Framework

The Spring Boot framework was chosen for the back-end development due to the following reasons:

- **Rapid Development**: Spring Boot facilitates rapid application development by offering a streamlined setup and configuration process, allowing developers to quickly bootstrap projects and focus on business logic rather than infrastructure.

- **Embeddable Servers**: It provides embedded servers allowing applications to be packaged as standalone JAR files without requiring external server deployment, simplifying deployment and scaling.

- **Robust Ecosystem**: Spring Boot leverages the broader Spring ecosystem, providing access to a vast array of libraries, tools, and community support, making it easy to integrate with other Spring projects and third-party libraries.
- **Security**: Leveraging the broader Spring ecosystem, it provides access to a vast array of libraries, tools, and community support, making it easy to integrate with other Spring projects and third-party libraries.
- **Community and support**: Spring Boot benefits from a large and active community of developers, offering extensive documentation, tutorials, and forums for troubleshooting and knowledge sharing, making it easier for developers to get started and resolve issues efficiently.

2.5.4 MyBatis framework

The MyBatis framework was chosen due to its:

- **Simplicity and Control:** MyBatis offers a straightforward approach to database interactions, allowing us to write SQL queries directly in our Java code, this simplicity grants us more control over our database operations, as well as making it easier and faster for us to fix possible mistakes that we might do while extracting our information
- **Flexibility and Customization:** With MyBatis, we can easily map complex SQL queries to our Java objects, therefore providing flexibility in the handling of various data structures. Its customizable mapping configurations enable us to tailor database access to our specific application requirements, accommodating complex data models and evolving needs with ease.

2.6 API Integration

As for external API calls, only the GitHub API was used to:

1. Create a repository for the CodeKata of a battle.
2. Get notified as soon as the push of a user occurs.
3. Pull his/her repository and evaluate the provided solution

Due to time constrains it was not practical to implement the Google Calendar API and the Email API.

2.7 DataBase

In order to manage our data, a dockerized version of MariaDB was used. MariaDB is an open-source relational database management system known for its robust performance, scalability, and reliability. It provides powerful features such as ACID compliance, transactions, and SQL support, making it suitable for a wide range of applications, and in particular our application. Moreover, Docker enables easy deployment and management of MariaDB instances, facilitating seamless integration into existing infrastructure and providing a more secure environment for data storage and management.

2.8 Hosting

We chose to host our front-end and back-end on a private server. This server is running Fedora Server Edition version 38 and provides a secure environment for hosting our application. The application is accessible at `https://codekatabattle.dgraz.ddns.net/` with a sample of a database. The application is reachable through Nginx Proxy Manager, which provides secure access to the application itself.

3 Source Code

3.1 Backend Structure

The back-end structure is built as such:

- **src/main/java/it/polimi/se2/grazzanimasini/ckb** folder contains:
  - **Controllers:** which are the clases that are handling the interaction with the front end, by forcing the incoming requests to provide specific informations and in specific shapes and then passing them to the manangers
  - **Managers:** these are the classes which are handling the business logic of the application, by receiving data and requests form the controllers, performing the required elaboration and requesting data to the database classes
  - **Database:** these are the classes which are interacting directly with the database, extracting tuples from it and then mapping them into java objects
  - **Model:** model classes are a bit everywhere in these folders/categories of classes and are the ones reopresenting the different kind of entities (classses) of our application, often corresponding to real world entities
- **src/main/resources** folder contains:
  - **it/polimi/se2/grazzanimasini/ckb/database**: this folder contains all the XML files used to map specific function calls to specific SQL queries on the database, as long as specific mapper to handle the construction of the Java objects starting from the corresponding database tuples
  - **/tournamentPictures:** this folder contains the picture of each tournament that is stored in the database
  - **/ck_to_be_uploaded:** in this folder are stored the CodeKata provided by each educator before it is finally uploaded to its ad hoc created repository
  - **/repositoriesForEvaluation:** in this folder are stored the cloned repositories of each user that has performed a new push and has signaled it to the CodeKata application, while the test cases are run onto it and the score is calculated
  - *application.properties*: this files contains all the main properties used by Spring and the backend in general to work, like specific url, email or token

3.2 Frontend Structure

Here is represented the structure of the web app. The main focus is on the *src* folder, which contains the all the source code used by React to generate our front-end.

- **CallsHandling**: contains all the necessary functions to make correct calls to the back-end.
- **Configuration**: contains the configuration of the app such as the routes and some custom colors.
- **CustomComponents**: contains all the custom components used in the various interfaces.
- **Interfaces**: contains all the interfaces of the application, which represents the main pages of the application.
- **App.js**: is the main file of the app, linking all the components together and ensuring the correct routing. It also includes a secure routing mechanism that prevents the user from accessing pages that require authentication.
- **package.json**: contains all the dependencies of the app needed by React to run the app.

4 Testing

While developing the application, we noticed a strong codependency between front-end and back-end components. We decide to follow a unit testing approach by isolating the front-end and back-end components and testing them separately; to solve the aforementioned problem we created some stubs to simulate the behavior of both respective-ends, while testing the other.

4.1 Unit Testing

Backend Testing
For the testing of the backend, fake calls have been made using the Postman application, more specifically we tested:

- The interfaces **getter** methods, to get informations in particular shapes or conditions, depending on our requirements, also testing if calls lacking the token or with an outdated one would be answered
- The **insertion** of new data through the insert methods, often requiring an object as a body
- The **removal** of items such as notifications, which are from then on considered as removed by the user, while not actually being removed from the database, for possible analysis in the future
- The **creation of a repository** for each inserted battle, by testing its creation with zip created starting from different kind of files and folder
- The **automatic evaluation** of new push made by the user, by trying to push different kind of jar executables along with other files to our experimental user_repository, thesting if the automatic workflow that the user should perform was actually triggering a response from our CodeKataBattle application

Frontend Testing
The testing of the frontend components has been made by creating specific stubs which after a fixed amount of time would answer to the components request for data with a fixed set of information, like they were coming from our back-end.
More specifically we tested:

- Tested the correct rendering of core components (such as the AppBar...)
- Tested tournament interface
- Tested battle interface
- Tested the correct navigation between the various interfaces of the application

4.2 System Testing

During system testing the application was tested as a whole, focusing on the interaction between the front-end and back-end components. In particular, we tested the correct integration of front-end and back-end communication, as this part was not included during unit testing.
More specifically we tested:

Sign up and login

- The correct registration of a user after the sign up operation
- The correct sign in of a user which already has an account

Tournament creation:

- The correct upload of the tournament's picture
- The correct creation of a new tournament
- If all users of the platform were actually notified about such creation
- If another educator could be invited to manage such tournament
- If the invited educator actually received such notification, and could accept it
- The correct visualization of tournaments in the main application page
- If users could actually subscribe to such tournament

Battle creation:

- The correct upload of the Code Kata for a specific battle
- The correct creation of the battle
- If the creation of the corresponding repository was handled correctly
- If the newly created repository actually stored the files uploaded by the educator
- If all users subscribed to the same tournament were actually notified about it
- If the list of battles were correctly visualized inside a tournament's page
- If students could actually enroll in such battle

Groups:

- The correct creation of a new group
- If another student could be invited into such group
- If the invited student did actually receive the corresponding notification, and if he/she could accept it
- If the new group could actually partecipate into a battle

Delivery evaluation:

- If a new push performed by a member of the group with the correct workflow did actually trigger our application
- If our application could actually clone the user's repository
- The correct evaluation through test cases provided in the CodeKata of the student's executable

Tournament closure:

- If a tournament could be closed correctly
- If when the tournament was closed the subscribed users where correctly notified about the event
- If a closed tournament was no more visible or accessible by users

Battle early closure:

- If a battle could be closed correctly
- If the user enrolled into such battle where correctly notified when this event occured
- If such battle was from then on, no more visible by users

5 Installation

Since both the front-end and back-end are hosted on a private server, the installation process is not required. However, an offline installation process is provided below. From here on we assume that the repository has been cloned (using the command below) and the user is in the root directory of the project.

```
git clone https://github.com/d-graz/GrazzaniMasini

cd GrazzaniMasini/
```

5.0.1 Dbms installation

For this application we are going to need to install MariaDB as our DBMS, to do that we need to install from the official website the dbms at a version $\geq$ 11.4, following the default instructions while doing the installation, after that it will probably be in C:/Program Files/MariaDB 11.4 (or other version), so we go into the bin folder of MariaDB (C:/Program Files/MariaDB 11.4/bin):

```
#launch mariadb
mariadb -u root -p

#after this we will be asked for a password,
#is the one that we have set up during the installation
#which by default is "password"

#once inside let's create our database
create database code_kata_battle;

#and exit
exit

#Now we are going to import our premade dump
#of the tables of the database which is
#on our repository at Implementation/ckb_back_end/database
mariadb -u root -p code_kata_battle < /path/where/our/dump/is

#Now we are goint to insert our password again,
#and we have finished this step
```

5.1 Backend Installation

Also the back-end installation is easy and fast, it only requires: OpenSDK 17 or superior, Maven $\geq$ 3.9.5 and your favourite command line text editor (vim in the example) installed on the system

```
#Navigate to the resources directory
cd Implementation/ckb_back_end/Code_Kata_Battle/src/main/resources

#Edit the application.properties file, changing the propery
#"resourcePath" to the absolute path corresponding to your own
#resources folder (the folder this file is in):
vim application.properties
#resourcesPath = /your/new/path

#Return to the back-end folder:
cd Implementation/ckb_back_end/Code_Kata_Battle

#give to the maven file execution permissions
chmod +x ./mvnw

#Run Maven, which will launch spring
mvnw spring-boot:run
```

5.2 Frontend Installation

Front-end installation is straightforward as it only requires `NPM` ≥ 21.6 to be installed on the system. Once NPM is installed, the user should navigate to the front-end directory `Implementation/ckb_front_end` and execute `npm install` to fetch all the necessary dependeces. To run the application now just `npm start`.

```
#Navigate to the front-end directory
cd Implementation/ckb_front_end

#Install all the dependeces
npm install

#Run the application
npm start
```

6 Effort Spent

Student	Time for implementation
Davide Grazzani	94
Alessandro Masini	106