Software Engineering 2 - Prof. Di Nitto Elisabetta
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

CodeKataBattle

ITD
Implementation and Testing Document

December 21, 2023

Marco Cerino 244780
Mattia De Bartolomeis 245022

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose and Scope

This section, "Purpose and Scope," is dedicated to outlining the contents and objectives of our document on the implementation and testing of the CodeKata-Battle project. The document is designed to elucidate the architecture of the code, the development frameworks utilized, the steps required for installation, and details regarding our testing procedures. Our aim is to provide a clear and comprehensive guide that explains the methodologies adopted in the development and testing phases, ensuring a thorough understanding of the project's technical foundation and the effectiveness of our testing strategies.

1.2 Definitions, Acronyms, Abbrevations

1.2.1 Acronyms

DBMS	Database Management System
REST	Representational State Transfer
API	Application Programming Interface
OS	Operating System
UI	User Interface
RASD	Requirement Analysis and Specification Document
DD	Design Document

1.3 Reference Documents

The following documents and resources have been referenced in the preparation of this document:

- RASD
- DD
- Project Implementation and Testing Assignment

1.4 Document Structure

The structure of this document is organized into the following parts:

- **Implemented Requirements and Functions** This demonstrates the features that have been actually incorporated into the software and ex-

plains their alignment with the requirements outlined in the RASD document.

- **Structure of the code** This describes how the source code is organized for both the front-end and back-end parts of the application.
- **Design Choices and Adopted Frameworks:** It enumerates the programming languages, frameworks, and tools adopted for this project, alongside the design choices made in accordance with the solutions proposed in the Design Document.
- **Testing Strategies:** This section details the testing procedures carried out and examines the primary test scenarios evaluated.

2 Implemented Req and Functions

In this section, we examine the functionalities of the application mapped to all the requirements associated and analyzing which of them have been implemented and which have not.

2.1 User Registration

- [**R1**] The system must allow users to register on the platform as educator or student. **Implemented.**

Frontend
The frontend implements the registration logic through a React component called Registration. This component uses useState to manage the state of the registration form, including fields for first name, last name, email, password, password confirmation and user type (with 'educator' as the default value). The component presents a form that allows users to enter their data and select the desired account type. When the form is submitted, a client-side validation is performed, in particular to check that the password is at least 6 characters long and that the passwords entered match. If the checks are successful, a POST request is sent to the backend using Axios, addressed to the appropriate endpoint (/api/auth/signup/educator or /api/auth/signup/student) depending on the type of user. If successful, a confirmation message is displayed and the user is redirected to the login page after a short interval. In the event of an error, e.g. if the email is already in use, an appropriate error message is displayed.

Backend
The backend handles user registration via two specific endpoints in the AuthenticationController: /signup/educator for educators and /signup/student for students. When a registration request is received, the SignupService validates the data entered, including checking the length of the password and the validity of the email, and checks that the email is not already associated with another account. If the data is valid, the password is encrypted using PasswordEncoder and the new user is saved in the database. Subsequently, a ResponseEntity object containing the data of the registered user is returned. In the event of validation errors or if the email is already in use, an exception is thrown and handled appropriately to return an error response to the client.

2.2 User Login

- [**R2**] The system must allow users to login. **Implemented.**

Frontend
The React component `Login` manages the login process. It uses `useState` to keep track of the form data and the login error state. Upon form submission, a POST request is sent to the correct endpoint (`/api/auth/login/educator` or `/api/auth/login/student`) based on the selected user type. If the login is

successful, the received JWT token is saved in `localStorage`, and the user is redirected to the correct page. In case of an error, an appropriate error message is displayed.

Backend
The authentication controller in the backend handles login through two endpoints: `/login/educator` and `/login/student`. It receives the user's credentials via a POST request, verifies these credentials, and if correct, generates a JWT token using `JwtUtils`. The token is then returned to the client. If the credentials are not valid, it returns a 401 error.

2.3 Functions about Tournaments

- [**R3**] The system must enable educators who have created a tournament to grant permissions to other educator to create battles within that specific tournament. **Implemented.**
- [**R4**] The system must allow the educator to see which tournaments they have permission for. **Implemented.(In my tournaments section he sees only the tournaments for which he has permission.**
- [**R5**] The system must allow educator to insert information about a tournament. **Implemented.**
- [**R8**] The system must allow students to subscribe to tournament on the CKB platform before a specific deadline. **Implemented.**
- [**R17**] The system must allow user to see the list of available tournaments. **Implemented.**
- [**R33**] The system must allow educators with permission on the tournament to close it. **Implemented.**
- [**R35**] The system must allow educators with the necessary permissions to delete a tournament in which no battles are currently ongoing. **Implemented.**

Frontend

- **CreateTournament Component:** This component allows educators to input tournament details and specify authorized educators. Upon submission, it posts the data to the backend and navigates the user to the list of their tournaments.
- **Tournaments Component:** Displays a list of all available tournaments. Educators and students can view detailed information about each tournament by clicking on it, which navigates them to a detailed view.
- **TournamentDetails Component:** Offers a detailed view of a tournament, including its description, registration deadline, and battles. It also allows students to join the tournament and educators to manage it.

Backend
The backend functionalities are primarily handled through the `TournamentController`, which interacts with various services to perform operations related to tournaments.

- **Creating Tournaments:** The `/create` endpoint is responsible for receiving tournament details and the list of authorized educator emails. It creates the tournament in the system and notifies all registered students about the new tournament via email.
- **Joining a Tournament:** The `/join` endpoint allows students to subscribe to a tournament before a specific deadline, checking the tournament's open status.
- **Viewing Tournaments:** Endpoints like `/all`, `/educator`, and `/student` provide filtered lists of tournaments based on the user's role, enabling both students and educators to view available or associated tournaments.

2.4 Functions about Battles

- [**R6**] The system must allow educator to insert information about a battle for the tournament in which he has the permissions. **Implemented.**
- [**R10**] The system must allow students to subscribe in the battle. **Implemented.**
- [**R18**] The system must allow user to see the list of available battles. **Implemented.**
- [**R22**] The system must automatically create a GitHub repository containing the code kata upon the expiration of the battle's registration deadline. **Implemented.**
- [**R24**]The system must automatically pull and analyze the latest sources from student repositories upon each commit. **Implemented in part.**
- [**R36**] The system must allow educators with necessary permissions to delete a battle that is not currently ongoing. **Implemented.**
- [**R37**] The system automatically deletes a battle when no teams have subscribed to it by the registration deadline. **NOT IMPLEMENTED**

Frontend

- **BattleDetails Component**: Enables users to view battle details, subscribe, form teams, and for educators to manage the battle, including deletion options.
- **CreationBattle Component**: Allows educators to input new battle details and submit them to the backend, supporting battle creation.

Backend
The system includes several key components for battle management and integration with GitHub:

1. **BattleController**: It provides endpoints for creating, viewing, updating, and deleting battles.
2. **GitHubController**: It manages GitHub webhooks to automatically update battle rankings based on commits made to team repositories, based on the timeliensss parameter.

2.5 Functions about Rankings and Evaluation

- [**R19**] The system must integrate with GitHub for repository management and automate the process of code submission and evaluation. **Implemented.**
- [**R20**] The system must allow educators who have set manually evaluation to see the submitted projects. **Implemented.**
- [**R21**] The system must allow educators who have set manually evaluation to upload a manually score for each project. **Implemented.**
- [**R25**] The system must support automated test execution and static analysis of submitted code. **Not Implemented(not requested for ITD).**
- [**R26**] The system must calculate battle scores based on functional aspects, timeliness, and the quality level of sources. [**Only timeliness calculation was implemented, (static analysis was not required in the implementation)**
- [**R27**]The system must update battle scores in real-time as students push new commits. **Implemented.**
- [**R28**] The system must consolidate and finalize battle scores after the submission deadline and provide a final battle rank. **Implemented.**
- [**R30**] The system must calculate and update tournament score at the end of each battle. **Implemented.**
- [**R31**] The system must create and update the tournament rankings. **Implemented.**
- [**R32**] The system must create the battle rankings when a new battle is created. **Implemented.**
- [**R34**] The system must ensure that the final score of each battle for a team falls within the range of 0 to 100. **Not implemented.**

Frontend In the frontend these are the components with the task of managing the classes:

- **BattleDetails Component**: Provides details on battles, including current rankings.
- **RankingTable Component**: Displays rankings within tournaments, dy-

namically updating to reflect real-time changes in participant scores.

Backend

- **RankingController**: Manages retrieval and calculation of tournament and battle rankings, offering endpoints for:
  - Obtaining tournament rankings (`/tournaments`).
  - Retrieving battle rankings (`/battles`).
  - Calculating tournament and battle rankings upon completion (`/calculateTournamentRanking` and `/calculateBattleRanking`).
- **GitHubController**: Integrates with GitHub to automate code submission evaluation, updating battle rankings based on new commits. Utilizes webhooks (`/webhook`) to process push events and adjust rankings accordingly.

2.6 Functions about Notifications

- [**R7**] The system must notify via mail subscribed students to the platform upon the creation of new tournaments.**Implemented.**
- [**R9**] The system must notify via mail all students subscribed in the tournament whenever a new battle is created within that tournament.**Not implemented.**
- [**R12**] The system must notify via mail all students who are invited to form a team**Implemented.**
- [**R13**] The system must allow students to accept an invitation via email to join in a team.**Implemented.**
- [**R14**] The system must allow student to decline the invitation via email to join in a team.**Implemented.**
- [**R15**] The system must notify via mail the team's creator when an invited student declines his invitation.**Not implemented.**
- [**R16**] The system must notify via mail all members of a team when a new student joins their team.**Not implemented.**
- [**R23**] The system must notify via email the link to the repository containing the code kata to all students who are members of a team registered for the battle.**Implemented.**
- [**R29**] The system must notify via mail all students involved in a battle when final battle rank becomes available.**Implemented.**

Frontend

The frontend does not handle the sending of emails directly, but offers interfaces through which users can trigger events that lead to the sending of notifications. For example, the creation of a new tournament or an invitation to join a team are actions in the frontend that result in calls to the backend controllers, which in turn invoke MailService to send the emails.

Backend
The main service for managing notifications in the backend is MailService, which uses JavaMailSender to send e-mails to users. This service is used by various controllers to send specific notifications based on events occurring in the platform.
MailService handles the sending of all email notifications to users. For each type of notification, a SimpleMailMessage with a specific subject and text is composed and then sent to the user's email address.

3 Structure of the code

3.1 Backend project structure

The backend of the project was developed using Visual Studio Code as the Integrated Development Environment (IDE). However, any IDE that supports Spring can be used for development. The primary directory of the project is the 'codebattle' folder, which encapsulates the fundamental components of a Spring Boot application. Within this folder, you will find the following essential elements:

1. **pom.xml**: This file serves as the project's configuration and contains all the dependencies required for the project to function properly.
2. **src/main**: Inside this folder, you will discover the primary source code of the application, which is an integral and crucial part of the overall project.
3. **src/test** : This folder contains application tests, written using test frameworks such as JUnit
4. **src/main/resources**: This directory contains additional resources utilized by the application. In particular, this folder houses the 'application.properties' file, which defines various configuration settings for the application, including database connection parameters, security credentials, email server settings, file upload limits, and JSON serialization options.

Specifically within the main folder, we can find:

1. **java.Scheduler** : This folder contains classes responsible for implementing schedules or planned tasks within the application. Specifically, it houses the class responsible for scheduling actions to occur at important events, such as the registration or submission deadlines for battles, and so on.
2. **java.com.codekatabattle.codebattle.Controller**: This folder contains the classes that handle HTTP requests and define the application's entry points.
3. **java.com.codekatabattle.codebattle.Model** : This directory contains the classes that represent the entities or data models of the application. They are annotated with '@Entity' using JPA for persistence.
4. **java.com.codekatabattle.codebattle.Repository** : This folder contains the classes responsible for accessing persistent data, via JPA
5. **java.com.codekatabattle.codebattle.Service** : This folder contains the classes that implement the business logic of the application. They are used by controllers to perform specific operations.
6. **java.com.codekatabattle.codebattle.DTO** : This folder contains DTO (Data Transfer Object) classes, which are used to transfer data between different application components.

7. **java.com.codekatabattle.codebattle.CodebattleApplication** : This file is the main entry point for the Spring Boot application. When the application is started, the main method is executed, which starts the entire Spring Boot application process.
8. **java.com.codekatabattle.codebattle.JwtTokenFilter** : This file contains the class, which is responsible for handling JSON Web Tokens (JWTs) used for authentication and authorization within the application.
9. **java.com.codekatabattle.codebattle.SecurityConfig** : This file contains the class that configures security settings for the application, including password encoding, authorization rules, and JWT token filtering.

We will now delve into some of these repositories in more detail

3.1.1 Model

Models define the structure of the data and the relationships between the different entities in the system. Here is a brief description of each model in the backend of the web app:

- AuthorisedEducator: represents the educators authorised to manage a tournament. It uses a compound key that includes both the tournament and the educator.
- AutomatedEvaluation: stores automated project evaluations, with functional, timeliness and total scores
- Battle: defines a battle within a tournament, including details such as the battle ID, associated tournament, student limits, description, and deadlines for registration and delivery of projects.
- BattleRanking: keeps track of team scores in battles, associating each team with a specific battle and recording its score.
- Educator: represents an educator, with fields for email, password, first and last name.
- ManualEvaluation: corresponds to manual evaluations assigned by evaluators to projects, linked to a specific automated evaluation and includes a personal score assigned by the educator.
- Student: represents a student, with fields for email, password, first name and last name.
- StudentTournament: associates students with the tournaments in which they participate, using a compound key that includes both the student and the tournament.
- Team: defines a team participating in a battle, with a unique name, invitation code and association to a specific battle.
- TeamParticipant: represents student participation in teams, associating each student with a specific team.

- TournamentRanking: tracks students' scores in tournaments, associating each student with a specific tournament and recording the score obtained.

For each entity in the application, a repository is associated. These repositories act as intermediaries between the application service layer and the database, providing an abstraction that facilitates CRUD (Create, Read, Update, Delete) operations for entities, without the need to write SQL or database-specific code.

3.1.2 Controller

In our web app, controllers act as a bridge between the frontend UI and the backend services, handling user requests and returning appropriate responses. Here is a brief description of each controller in the backend of the web app:

- AuthenticationController: Manages user authentication, including login and signup for students and educators. Uses JWT (JSON Web Tokens) for secure management of user sessions
- BattleController: It handles the logic related to battles, including creation, obtaining information, checking status, and user enrolment. It also supports the deletion of battles and provides access to the details of associated projects.
- EducatorInfoController: Provides access to information on educators, including personal details, tournaments created and associated battles. It is the access point for obtaining aggregated data on educators.
- StudentInfoController: Similar to the EducatorInfoController, this controller provides information about the students, the tournaments they have participated in, and the battles they are involved in.
- EvaluationController: Manages the manual evaluation of projects by educators, allowing them to assign scores based on customised criteria
- GitHubController: Integrates the web app with GitHub for repository management and automatic evaluation of projects based on commits. Manages GitHub webhooks to update battle scores in real time.
- RankingController: It calculates and provides rankings for both tournaments and individual battles, based on the scores obtained by teams and students.
- TournamentController: It manages the logic of tournaments, including the creation, registration of students, and closing of tournaments. It also provides functionality for deleting tournaments and checking student registration.

3.1.3 Service

Services provide business logic and access to data. Here is an overview of each service. Here is a brief description of each service in the backend of the web app:

- AutomatedEvaluationService: It manages the creation or updating of automated evaluations for projects, combining functional and timeliness scores to determine a total score.
- BattleService: It offers functionality for managing battles, including creating, updating, deleting and scheduling tasks related to battle deadlines. It also provides access to the projects associated with each battle.
- EducatorService: Provides access to information on educators, enabling retrieval of educator details based on email
- EvaluationService: It manages manual project evaluations, allowing educators to assign personal scores that complement the automated evaluations.
- GitHubService: It handles interaction with GitHub repositories, such as the creation of new repositories for battles.
- JwtUtils: It offers functionality for the generation and management of JWT tokens, used for the secure authentication of users.
- LoginService: It manages the login process for educators and students, verifying credentials and returning a JWT token for the authenticated user.
- MailService: Sends e-mails to users for various reasons, such as notifications regarding tournaments, battles, or team invitations.
- ProjectService: It manages the creation and updating of projects, associating them with teams and saving evaluation results.
- RankingService: It calculates and manages rankings for both battles and tournaments, based on the scores obtained by participants.
- SignupService: It manages the registration process for educators and students, including data validation and password encoding.
- StudentService: It offers access to student information and manages student registration for tournaments.
- TeamService: It provides functionalities for managing teams, including creating new teams, adding members to teams and verifying team membership.
- TournamentService : It manages the logic related to tournaments, including the creation, registration, closure and deletion of tournaments, as well as the management of educator authorisations and the assignment of students to tournaments.

3.1.4 DTO

Several DTOs were also implemented in the development of the web app, which we will not go into individually. DTOs (Data Transfer Objects) are simple objects used to transfer data between the layers of an application or between different systems. DTOs make it possible to transfer data in an optimised way

between the client and the server or between different system components.

DTOs are particularly useful in scenarios where:

- Data encapsulation: You want to transmit only a specific part of an entity's data rather than the entire entity. For example, you might want to send a user's details without including their password or other sensitive information.
- Customising the data sent: In some situations, the data required by the client does not exactly match the data modelled by the entities. DTOs make it possible to create a customised data structure to meet these specific needs.

3.2 Frontend project structure

The backend of the project was developed using Visual Studio Code as the Integrated Development Environment (IDE). However, any IDE that supports React and CSS can be used for development. The primary directory of the project is the 'my-app' folder which encapsulates the fundamental components of a Spring Boot application. Within this folder, you will find the following essential element:

- **src** : This is the main source code folder of the React application. It contains all the files and folders necessary for the creation and operation of the application. It contains the following essential elements:
  1. **component**: This folder contains JS files which manage the frontend's operating logic, sending requests to the server and receiving data from it
  2. **style**: This folder contains CSS style sheets used for the presentation of the application.
  3. **image**: his folder is used to contain static resources such as images, icons, or any other files that are to be imported into the application.
  4. **App.js**: This file represents the main component of the application. It contains the basic structure of the application and often includes the main router or page rendering log
  5. **ProtectedRoute.js**: The 'ProtectedRoute' file checks whether a JWT token is present in the browser's localStorage. If the token is absent, the component redirects the user to the access page. Otherwise, the component allows the user to access the protected content.This allows only authenticated users to view the specific content

4 Design Choices and Adopted Framework

In building our web app, we have adopted a set of technologies and frameworks to ensure efficiency, reliability and desired functionality. In this section, we will examine the key technologies we used in the development of our web app. We will explain the advantages and disadvantages of each technology choice and provide an in-depth view of the reasons why we selected them for our project.

4.1 Backend technologies

The following technologies were adopted for the development of the backend:

4.1.1 Programming language

Java was chosen as the programming language for the backend due to its widespread usage among our team and its various advantages, which include:

- *Security*: Security is crucial when running online educational platforms, and Java is known for its security features.
- *Stability and reliability*: offers robust stability and reliability, which is important when running programming competitions.

Di seguito invece alcuni svantaggi del suo utilizzo:

- *Complexity*: Java can be more complex than some languages, with a more verbose syntax.
- *Performance*: In some situations, Java may have lower performance than other lighter languages.

Framework (Spring Boot)
We have utilized Spring in conjunction with Maven, resulting in the following advantages:

- *Extensive Community Support*: Spring Boot benefits from a large developer community, with abundant documentation and online resources available.
- *Dependency Management*: Integration with Maven simplifies dependency management and the inclusion of external libraries.

4.1.2 Project Dependencies

Our CodeKataBattle (CKB) platform relies on several external dependencies to enhance its functionality and provide essential features. Here is an overview of the key dependencies used in our project:

- **Spring Boot Starter Data JPA:** This dependency enables the use of Spring Data JPA for efficient data access and manipulation.

- **Spring Boot Starter Web:** We utilize this dependency to set up Spring Web, which is essential for building RESTful APIs and web components.
- **Spring Boot Starter Test:** For testing purposes, we depend on this Spring Boot starter that includes various testing libraries.
- **PostgreSQL:** We use the PostgreSQL database as our runtime data store, providing robust and reliable data management.
- **Spring Boot Starter Security:** This dependency allows us to implement security features, ensuring the protection of our platform's data and functionality.
- **GitHub API:** We have integrated the GitHub API to automate the creation of repositories when registration deadlines for battles expire. This integration streamlines the workflow for our platform.
- **JJWT (Java JWT):** This library assists in handling JSON Web Tokens (JWT) for secure authentication and authorization in our platform.
- **Spring Boot Starter Quartz:** This dependency plays a role in scheduling tasks and jobs within our platform. For instance, it is used to automate the start of repository creation on the GitHub platform at the exact moment when the registration deadline for a battle expires.
- **Spring Boot Starter Mail:** This dependency equips our platform with the essential components required for sending email notifications to our platform's users. It ensures that users are promptly informed about significant events and updates. Whether it's notifying users of newly created tournaments, inviting them to join teams, or delivering timely notifications on battle scores, the Spring Boot Starter Mail simplifies the process of keeping our user community well-connected and engaged.
- **Jackson Datatype JSR310:** We use this Jackson datatype to handle date and time serialization in a standardized manner.

4.2 Frontend Technologies

The frontend of our application was developed using a combination of React and CSS, which provided a responsive and interactive user interface. Here's an overview of the technologies and methodologies adopted for the frontend development:

4.2.1 Programming Language

JavaScript, with **React** as the core library, was selected for building the user interface due to React's component-based architecture and the following advantages:

- *Declarative UI*: React makes it straightforward to create interactive UIs. Designing simple views for each state in your application, and React will

efficiently update and render just the right components when your data changes.

- *Component-Based*: Building encapsulated components that manage their own state, then compose them to make complex UIs.
- *Learn Once, Write Anywhere*: React can also render on the server using Node and power mobile apps using React Native.

Despite its benefits, challenges include:

- *Learning Curve*: Understanding React's lifecycle methods, JSX, and state management can be complex for beginners.

4.2.2 Styling

CSS was used for styling the application, enhanced by pre-processor frameworks and methodologies such as:

- *Modularity and Reusability*: CSS methodologies like BEM (Block, Element, Modifier) were employed to enhance CSS modularity and reusability.
- *Responsive Design*: Media queries and flexbox were utilized to create a responsive design that adapts to various screen sizes and devices.

4.2.3 Frontend Dependencies

Our application leverages several key libraries and frameworks to enhance its functionality and user experience. Here is a brief overview of the primary dependencies:

- **Axios**: Utilized for making HTTP requests to our backend services, Axios simplifies promise-based HTTP client usage.
- **React Router**: Manages navigation and routing in our SPA (Single Page Application), enabling dynamic user experiences without page reloads.

4.3 Database choices

To ensure the secure handling of the substantial volume of data managed by the CKB system, we evaluated multiple Database Management Systems (DBMSs). Given our decision to employ a relational database model, we sought a DBMS capable of fulfilling our requirements. Our considerations included MySQL, Microsoft SQL Server, and PostgreSQL. Ultimately, we selected PostgreSQL for several compelling reasons. Notably, it stands out as one of the only open-source solutions alongside MySQL, aligning with our preference for open-source technologies. Furthermore, our prior experience with PostgreSQL in various projects has endowed us with a familiarity that we anticipate will facilitate its integration and use in our current endeavor.
Although we took advantage of Spring's ability to automatically manage database creation, it was still essential for us to undertake a thorough database design at

the beginning of the project. This planning process allowed us to clearly define the entities and relationships that make up our data model. This initial planning phase greatly facilitated the mapping of entities in the Spring context, thereby optimising the integration and efficiency of our system. The detailed planning ensured that, despite the automation, the database structure exactly reflected our needs and requirements, contributing to the efficiency and scalability of the implemented solution.

5 Testing

5.1 Unit testing

Unit Testing in the backend of our Java application extensively uses the JUnit and Mockito frameworks to ensure the reliability and robustness of key components. These tools allow us to isolate and test specific software functionalities without the need to depend on external components or the overall system integration. The main focus of our unit tests has been directed towards critical application services, such as BattleService, AutomatedEvaluationService, SignupService, TeamService, and TournamentService, LoginService, ProjectService, RankingService which implement significant algorithms and manage fundamental business logic. For each of these services, we adopted a direct approach to testing, mocking the necessary dependencies, such as repositories and other services, to verify the correct behavior of the methods under test. This methodology has allowed us to ensure that each unit functions as expected under various conditions, including success cases, errors, and edge behaviors.The test cases performed are the following:

Schema	**Test Description**	**Desired Output**
BattleService:Battle	Validates that the battle method correctly returns a populated Battle when there is a matching battle for the provided ID.	The expected output is that the Battle is present and contains the expected Battle object. Test passed
BattleService:Save Battle	Tests the save logic of a new battle, including the next ID generation, the persistence of the Battle object, and the scheduling of jobs with Scheduler.	The expected result is that the Battle object is saved with an incremented ID and that the scheduling jobs have been correctly scheduled. Test passed
BattleService:Verify Battle Active	Checks if a battle for a given tournament is currently active based on the registration and submission deadlines.	The expected result is that the method returns false, indicating that the battle is not considered active if the registration deadline has passed. Test passed

BattleService:Update Battle	Assesses the ability to update the description of an existing battle and verifies that the update has been performed.	The expected result is that the Battle object is updated with the new description. Test passed
BattleService:Delete Battle	Verifies the service's ability to delete a battle given its identifier.	The expected result is that the battle with the specified ID has been deleted. Test passed
BattleService:Get Project	Ensures that the getProject method correctly retrieves a project given its ID and that the returned project matches the expected one.	The expected result is that the Project is present and contains the expected project. Test passed
Automated EvaluationService: Create New Evaluation	Verifies that a new automated evaluation is created if there is no existing evaluation for the specified project.	The expected output is that a new AutomatedEvaluation object is created with the correct project, functional score, timeliness score, and total score calculated as the average of the functional and timeliness scores. Test passed
Automated EvaluationService: Update Existing Evaluation	Verifies that an existing automated evaluation is correctly updated with new scores.	The expected result is that the existing AutomatedEvaluation object is updated with new functional and timeliness scores, and the total score is recalculated as the average of these two scores. Test passed
LoginService: Login Educator Success	Verifies successful login for an educator by checking the email and password match.	The expected output is that the educator's email matches the one provided and the authentication is successful.Test passed

LoginService: Login Educator Invalid Email	Verifies that an exception is thrown when attempting to login with an email that does not exist in the repository.	The expected result is that an IllegalArgumentException is thrown due to invalid email.Test passed
LoginService: Login Student Success	Verifies successful login for a student by checking the email and password match.	The expected output is that the student's email matches the one provided and the authentication is successful. Test passed
LoginService: Login Student Invalid Password	Verifies that an exception is thrown when a student attempts to login with an incorrect password.	The expected result is that an IllegalArgumentException is thrown due to invalid password.Test passed
ProjectService: Create New Project	Verifies that a new project is created if there is no existing project with the same GitHub repository URL.	The expected output is that a new Project object is created, the GitHub repository URL matches the one provided, and the codeKataTeam content is correct. Test passed
ProjectService: Update Existing Project	Verifies that an existing project is correctly updated with a new codeKataTeam if a project with the same GitHub repository URL is found.	The expected result is that the existing Project object is updated, the GitHub repository URL is correct, and the new codeKataTeam content matches the updated array. Additionally, the project repository's save method should have been called once. Test passed
RankingService: Get Battle Ranking	Verifies that the getBattleRanking method correctly retrieves the battle ranking for a given battle and tournament ID.	The expected output is that the actual rankings are not null, not empty, and equal to the expected rankings. Test passed

RankingService: Get Tournament Ranking	Verifies that the getTournamentRanking method correctly retrieves the ranking for a specific tournament based on its ID.	The expected result is that the actual rankings are not null, not empty, and equal to the expected rankings. Test passed
RankingService: Create Tournament Ranking	Verifies that the createTournamentRanking method calculates and saves the rankings of a tournament based on provided data from another method in the repository.	The expected output is that TournamentRanking is saved in the repository once for each ranking DTO. Test passed
RankingService: Calculate Battle Ranking	Verifies that the calculateBattleRanking method correctly calculates the total score for each team in a battle, combining scores from automated and manual evaluations, and saves the results in the repository.	The expected result is that BattleRanking is saved in the repository once for each team. Test passed
SignupService: Register Educator	Verifies that an educator is correctly registered and the password is encoded before saving.	The expected output is that the Educator object is not null, the password is encoded, and the Educator is saved in the repository. Test passed
SignupService: Register Student	Verifies that a student is correctly registered and the password is encoded before saving.	The expected result is that the Student object is not null, the password is encoded, and the Student is saved in the repository. Test passed

SignupService: Register Educator With Email Already In Use	Verifies that an exception is thrown when registering an educator with an email that is already in use.	The expected output is that an IllegalArgumentException is thrown, and no Educator is saved in the repository. Test passed
SignupService: Register Student With Email Already In Use	Verifies that an exception is thrown when registering a student with an email that is already in use.	The expected result is that an IllegalArgumentException is thrown, and no Student is saved in the repository. Test passed
TeamService: Create Team	Verifies that a team is saved in the database and checks that the save method of teamRepository is called with the correct team.	The expected output is that the team's name matches Team A and the teamRepository's save method is called once. Test passed
TeamService: Join Team	Verifies that a student can join a team and checks that the save method of teamParticipantRepository is called with the correct TeamParticipant.	The expected result is that the student's email and the team's ID match the provided values, and the teamParticipantRepository's save method is called once. Test passed
TeamService: Find By Codice Invito	Verifies that a team is retrieved through its invitation code and checks that the findByCodiceInvito method of teamRepository returns the correct team.	The expected output is that the codiceInvito matches the provided value and the teamRepository's findByCodiceInvito method is called once. Test passed

TeamService: Is Member Of Team	Verifies if a student is a member of a team through the invitation code and checks that the existsByTeamTeamIdAndStudentEmail method of teamParticipantRepository is used to determine team membership.	The expected result is that the student is confirmed as a member of the team, and the teamParticipantRepository's existsByTeamTeamIdAndStudentEmail method is called once. Test passed
TournamentService: Tournament Info	Verifies that the tournamentInfo method correctly returns a tournament when provided with a valid ID.	The expected output is that the tournament is present and matches the expected tournament. Test passed
TournamentService: Tournaments	Verifies that the tournaments method returns a list of all tournaments.	The expected result is that the size of the result list matches the size of the expected tournaments list. Test passed
TournamentService: Create Tournament	Verifies that a new tournament is correctly saved.	The expected output is that the tournament's name matches "New Tournament". Test passed
TournamentService: Join Tournament	Verifies that a student can register for a tournament and that the saveAndFlush method of studentTournamentRepository is called with the correct StudentTournament.	The expected result is that the result is not null, and the student's email and tournament's ID match the provided values. Test passed
TournamentService: Close Tournament (Tournament Not Found)	Verifies that attempting to close a tournament that does not exist returns false.	The expected output is that the result is false and the tournamentRepository's save method is never called. Test passed

TournamentService: Close Tournament (Unauthorized Educator)	Verifies that a tournament is not closed when the educator is not authorized and the tournament exists.	The expected result is that the result is false and the tournamentRepository's save method is never called. Test passed

Table 1: Unit Test

5.2 Integration testing

In the context of application development, integration tests play a key role in ensuring that the different components of a system interact correctly with each other. The main objective is to ensure that critical functionalities, such as user registration and login, the creation and updating of tournaments and battles, and the management of rankings, are executed without errors, guaranteeing security, efficiency and reliability.
Tests were carried out on the following controllers

- AuthenticationController
- BattleController
- TournamentController
- RankingController
- EducatorInfoController
- StudentInfoController

Test Endpoint and Description	**Description**	**Desired Outcome**
POST /loginEducator: Success	Verify successful login for an educator	200 OK with JWT; Test passed
POST /loginEducator: Failure	Verify login failure for an educator with incorrect credentials	401 Unauthorized; Test passed
POST /loginStudent: Success	Verify successful login for a student	200 OK with JWT; Test passed
POST /loginStudent: Failure	Verify login failure for a student with incorrect credentials	401 Unauthorized; Test passed

POST /registerEducator	Verify the registration of an educator	200 OK with educator details; Test passed
GET /battle/infoBattle	Retrieve information for a specific battle	200 OK with battle details; Test passed
PUT /battle	Update the information for a battle	200 OK; Test passed
POST /battle/delete	Delete a specific battle	200 OK; Test passed
GET /api/educators/email	Get information for a specific educator	200 OK with educator details, tournaments, and battles; Test passed
GET /api/educators/all	Get a list of all educators	200 OK with list of educators; Test passed
GET /rankings/tournaments	Get the rankings for a specific tournament	200 OK with list of rankings; Test passed
POST /rankings/calculateTournamentRanking	Calculate the ranking for a tournament	200 OK with calculation confirmation; Test passed
GET /rankings/battles	Get the rankings for a specific battle	200 OK with list of rankings; Test passed
POST /rankings/calculateBattleRanking	Calculate the ranking for a battle	200 OK with calculation confirmation; Test passed
GET /tournament/myTournament	Retrieve information for a specific tournament	200 OK with tournament details; Test passed
POST /tournament/create	Create a new tournament	200 OK with tournament details including name; Test passed

Table 2: Integration Testing Summary for the Educational Tournament Management System

Installation Procedure

This guide provides the steps to install and configure the backend and frontend of our web application. The procedure is applicable regardless of the operating system, provided all the necessary dependencies are met.

Cloning the Repository

1. Ensure you have the git client installed on your system.
2. Open a terminal and navigate to the directory where you wish to clone the repository.
3. Execute the following command:

```
git clone https://github.com/matamattia/DeBartolomeisCerino
```

The repository can also be cloned directly through GitHub's interface.

Backend Configuration

To set up the backend, follow these comprehensive steps:

1. Ensure Java JDK is installed on your system. Spring requires Java to be installed as a prerequisite. You can download it from the official Oracle website or use OpenJDK.
2. Download and install Maven, which is necessary for dependency management and project building. Instructions can be found at `https://maven.apache.org/install.html`.
3. Install an Integrated Development Environment (IDE) that supports Spring applications. Popular choices include Spring Tool Suite (STS), IntelliJ IDEA, or Eclipse with the Spring Boot plugin.
4. Navigate to the backend directory within the cloned repository:

```
cd path/to/backend
```

5. Open the backend project in your chosen IDE. Most IDEs have an option to open or import existing Maven projects.
6. Once the project is loaded, run the main application class. This class is typically annotated with `@SpringBootApplication` and can be run directly from the IDE to start the Spring application.

Database Setup

To successfully run the backend, a PostgreSQL database is required. Follow these steps to set up your database and configure the application to connect to it:

1. Install PostgreSQL on your system if it is not already installed. You can download it from `https://www.postgresql.org/download/` and find installation instructions for your operating system.
2. Once PostgreSQL is installed, open the PostgreSQL command line tool, 'psql', and create a new database:

   ```
   CREATE DATABASE CodeKataBattle;
   ```

3. Create a user and grant it access to the database:

   ```
   CREATE USER post WITH ENCRYPTED PASSWORD 'gres';
   GRANT ALL PRIVILEGES ON DATABASE "CodeKataBattle" TO post;
   ```

4. Open your Spring application's 'application.properties' file located in the 'src/main/resources' directory of your backend project.
5. Configure the database connection by adding or updating the following lines:

   ```
   spring.datasource.url=jdbc:postgresql://localhost:5432/CodeKataBattle
   spring.datasource.username=post
   spring.datasource.password=gres
   spring.jpa.hibernate.ddl-auto=update
   spring.jpa.show-sql=true
   ```

   This configuration sets up the application to connect to the 'CodeKataBattle' database using the username 'post' and the password 'gres'. It also instructs Spring JPA to update the database schema automatically based on the entity classes and to log SQL statements.

Ensure that PostgreSQL is running on your system before starting the backend application. These settings will enable the application to connect to the database and operate correctly.

Frontend Configuration

Setting up the frontend requires Node.js and npm (Node Package Manager), which comes bundled with Node.js. Follow these steps to prepare and run the frontend part of your application:

1. Ensure Node.js is installed on your system. If not, download and install it from `https://nodejs.org/`. This will also install npm, which is required to manage the project's dependencies.
2. Navigate to the frontend directory within the cloned repository:

   ```
   cd path/to/frontend
   ```

3. Install the project dependencies by running the following command. This step reads the 'package.json' file and installs all the necessary packages defined there:

```
npm install
```

4. Once the installation is complete, you can start the frontend application by executing:

   ```
   npm start
   ```

   This command will compile the React application and usually opens a web browser window automatically displaying the app. If it doesn't, you can manually navigate to `http://localhost:3000` (or the port specified in the terminal) to view the application.

Make sure your backend service is running before starting the frontend to ensure the application can communicate with the backend correctly.

GitHub Integration

Our application integrates with GitHub for automatic repository creation upon the registration of a new battle. To enable this feature, a GitHub Personal Access Token (PAT) is required.

Using the Provided Token

For immediate setup, you can use the following token by setting it as an environment variable named 'GITHUB_TOKEN':

export GITHUB_TOKEN=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

This token allows the application to create repositories under the GitHub user 'codekatabattlese'.
Be sure to print out the value to check that it is set correctly with

echo $GITHUB_TOKEN

Enabling GitHub Integration with ngrok

To ensure GitHub can interact with our application running locally for automatic repository creation, it's essential to use ngrok. Ngrok exposes local servers behind NATs and firewalls to the public internet over secure tunnels.

Setting up ngrok

Follow these steps to set up ngrok and enable GitHub repository creation feature:

1. Download ngrok from `https://ngrok.com/download` and follow the installation instructions for your operating system.
2. Once installed, open a terminal and start ngrok to expose your application's port (for example, 8080 if that's where your application runs) to the internet:

```
./ngrok http 8080
```

3. ngrok will generate a public URL (e.g., `https://12345abcde.ngrok.io`) that forwards to your local development server.
4. Use this URL as the webhook endpoint in your GitHub yaml file, for github actions workflow configuration!.

Note: Keep ngrok running while you're using the GitHub integration feature. Every time ngrok is restarted, a new URL is generated, which requires updating the webhook endpoint in GitHub settings.

6 Time Spent

We did not monitor the exact hourly contributions of each team member; however, we approximate that the workload reached up to 80 hours per individual. The project's implementation commenced on January 4, 2024, and concluded on February 4, 2024, encompassing the completion of the ITD document.

Name	**Hours**
Mattia De Bartolomeis	90
Marco Cerino	90

Furthermore, it is important to emphasise that the entire team collaborated actively throughout the implementation and testing, distributing tasks and collaborating equally on every aspect of the project.

7 References

- Application Development: Visual Studio Code- React documentation - Spring documentation
- Github integration : GitHub API documentation