Implementation and Testing Document

CodeKataBattles

Karl Monrad Kieler {karlmonrad.kieler@mail.polimi.it}
Aske Schytt Meineche {askeschytt.meineche@mail.polimi.it}
Leonie Dragun {leonie.dragun@mail.polimi.it}

09 February 2024

CodeKataBattles running at:
`https://codekatabattles.streamlit.app/`

Source code at:
`https://github.com/Askenm/SoftwareEngineering/tree/Alt_navigation`

Contents

1 Implemented Functionalities

motivations for including them and excluding others if applicable

2 Adopted Development Framework

2.1 Development Framework

Streamlit

2.1.1 Streamlit

2.1.1.1 Advantages

It has many existing components that are easy to use, which aids quick implementation of functionalities. many existing components, integrations for front-end development

2.1.1.2 Disadvantages

Efficiency scales down with the complexity of the app. Navigation of the app with Role Based Access patterns is not supported. Log-in/ sign-up for functionalities with role-based configuration are not supported.

2.2 Adopted Programming languages

2.2.1 Python

Python is the language used to implement the actual app, and handles both the frontend, backend and notification service.

2.2.2 PSQL

PSQL is used for all the communication with the database server, such as entering new subscriptions, submissions or groups into the DB, as well as retrieving all data that is displayed on the app.

2.3 Adopted Middleware

2.3.1 GitHub API

While the GitHub API could allow for automation of many of the functionalities related to battle-creation and submissions, certain parts of the API, is very focused on work on personal computers. This means that building generalizable scripts with dynamic user credentials can be quite troublesome. We have built a GithubManagementService that handles automaic creation of battles, but unfortunately we did not succeed in making it completely generalizable. Therefore, we instead opted for a solution where a link to an existing repository is pasted into the battle creation page. While, this is a cumbersome workaround, the Educator would no matter what, need to enter a text editor in order to code up the tests. Therefore, we don't see it as an unacceptable solution.

2.3.2 Frontend Server - Streamlit Community Cloud

2.3.2.1 Advantages

It is effortless to deploy and set up.

2.3.2.2 Disadvantages

The machines used to deploy the app are not the most powerful, therefore leading to a slow app

2.4 Posgres DB in Azure

The Postgres Database that runs the service is hosted on Microsoft Azure, allowing us to scale quite effortlessly.

2.5 Backendserver - Ubuntu

An Ubuntu Server is running the main backend functionalities, such as checking for new Badge awardees, new submissions and sending notifications.

3 Source Code Structure

4 Testing

4.1 Unit Testing

In the development of our application, unit testing has been performed to ensure the reliability and functionality of each component. Unit tests have been implemented for all major classes defined in the backend, covering various methods and functionalities. The following sections outline the unit tests conducted for each class:

1. **Student Class**
   Unit tests have been created to verify the functionality of methods within the Student class, including but not limited to:

   - Retrieval of student's profile information from the database
   - Retrieval of student's association to tournaments and battles, such as subscriptions and registrations respectively

   Mocking frameworks and dependency injection techniques have been employed to isolate the class's behavior from external dependencies during testing.

2. **Educator Class**
   Comprehensive unit tests have been implemented to validate the behavior of methods in the Educator class, encompassing functionalities such as:

   - Retrieval of educator's profile information from the database
   - Retrieval of student's submissions to battles
   - Assignment of scores to manual submissions and their addition to the database

3. **Tournament Class** Unit tests have been developed to ensure the correctness of functionalities provided by the Tournament class, including:

   - Creation of new tournaments
   - Addition of new subscriptions to tournaments
   - Retrieval of tournament information
   - End tournaments

4. **Badge Class** The Badge class is thoroughly tested to validate its functionalities, which involve:

   - Creation of badges in the database

- Awarding of badges
- Retrieval of badge information from the database

5. **Authentication Info Class** Unit tests have been devised to verify the correctness of methods within the AuthenticationInfo class, which handles authentication-related operations such as:
   - User registration - creation of new users
   - User login - Retrieval of user credentials
6. **Battle Class** Extensive unit testing has been conducted to validate the behavior of the Battle class, which manages battle-related operations, including:
   - Creation of battles in the database
   - Retrieval of battle information
   - Retrieval of registered student groups and their members from the database
   - Retrieval of group submissions from the database

4.2 System Testing

System testing was conducted to evaluate the functionality and usability of the application from end-to-end, focusing on various user roles and system workflows. The testing process encompassed both educator and student user scenarios, as well as system workflows to ensure the application operates as expected in real-world usage scenarios. Below are the details of the system testing performed:

4.2.1 Educator User Test Cases

1. Create an Educator User
   - This test case validates the ability of the system to allow educators to create user accounts.
   - Steps include providing necessary information such as username, email, and password, and verifying successful creation of the user account.
2. Create a Tournament
   - This test case assesses the functionality for educators to create tournaments within the system.
   - Steps involve specifying tournament details such as name, description, and subscription deadline, and confirming successful creation of the tournament in the database.
3. Create a Battle
   - The objective of this test case is to verify the capability of educators to create battles only within tournaments that they themselves have created
   - Steps include specifiying battle details, like name, description, the according GitHub Repository, registration deadline, and final submission deadline, and confirming the successful creation of the battle in the database.
4. Create a Badge
   - This test case evaluates the system's ability to allow educators to create badges to be awarded to students, for tournaments they themselves have created.

- Steps involve specifying badge details such as name, description, and criteria for earning the badge (badge logic), and confirming successful creation of the badge in the database.

5. End a Tournament
   - This test case ensures that educators can conclude tournaments they have created.
   - Steps include marking the tournament as finished through the "End Tournament" button functionality and verifying that appropriate actions are taken, such as updating the tournament entry in the database with an end-date.
6. Manually score a submission
   - This test case assesses the functionality for educators to manually score submissions to battles that allow manual scoring.
   - Steps assigning scores, confirming successful recording of scores in the system, and verifying that appropriate actions are taken, such as the notification of the student whose submission has been scored via email.

4.2.2 Student User Test Cases

1. Create a Student User
   - This test case evaluates the system's ability to allow students to create user accounts.
   - Steps include providing necessary information such as username, email, and password, and verifying successful creation of the user account.
2. Subscribe to a Tournament
   - The objective of this test case is to validate the capability of the system to let students subscribe to tournaments.
   - Steps involve browsing available tournaments, selecting desired tournaments, confirming successful creation of a submission in the database, and verifying that the subscribed receives a confirmation email.
3. Register to a Battle with a Group
   - This test case assesses the functionality for students to register for battles as part of a group.
   - Steps include, selecting a battle, forming a group with other students subscribed to the tournament and not yet part of a group and confirming successful registration for the battle.
4. Browse other Student's profiles
5. This test assures the functionality for any user of the system to find and view other Student's profile pages
6. Steps include, utilisation of the search bar, viewing of profile pages, and confirming successful displaying of the appropriate user informations.

4.2.3 System Workflows

1. Awarding of a Badge
   - This system workflow tests the end-to-end process of awarding a badge to a student based on predefined criteria.

- Steps include fulfilling badge requirements, triggering the badge awarding process, and verifying that the badge is successfully awarded to the student, and the student is notified via email.

2. Correct displaying of front-end elements accordant with the logged in user

Each test case and system workflow was executed systematically, and the results were documented to identify any deviations from expected behavior, defects, or areas for improvement. By conducting thorough system testing, we aimed to ensure that the application meets the requirements and expectations of both educators and students while maintaining reliability and usability throughout the user experience.

5 Installation Instructions?

6 Efforts Spent