

Politecnico di Milano
Computer Science and Engineering
Software Engineering II

Design Document - CodeKataBlade

José Alejandro Sarmiento

January 7, 2024

v1.0

Contents

1 INTRODUCTION	2
1.1 Purpose	2
1.2 Scope	2
1.3 Definitions, Acronyms, Abbreviations	2
1.3.1 Definitions	2
1.3.2 Acronyms	3
1.3.3 Abbreviations	3
1.4 Revision history	4
1.5 Reference Documents	4
1.6 Document Structure	4
2 ARCHITECTURAL DESIGN	5
2.1 Overview: High-level components and their interaction	5
2.1.1 Interaction	6
2.1.2 Advantages of Three-Tier Architecture	6
2.2 Component view	7
2.2.1 Presentation Layer	7
2.2.2 Application Layer	8
2.2.3 Data Layer	9
2.3 Deployment view	10
2.4 Runtime view	11
2.5 Component interfaces	19
2.6 Selected architectural styles and patterns	22
2.7 Other design decisions	23
3 USER INTERFACE DESIGN	24
3.1 Mockups	24
3.2 Flowcharts	31
4 REQUIREMENTS TRACEABILITY	31
5 IMPLEMENTATION, INTEGRATION AND TEST PLAN	31
6 EFFORT SPENT	31
7 REFERENCES	31

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to provide a detailed description of the architecture of the system to be, CodeKataBlade, and to show how the requirements presented in the RASD document are met by the architecture. The document also contains the implementation, integration and test plan for the system.

1.2 Scope

CodeKataBlade is a software system designed to facilitate and enhance the practice of code katas, which are small coding exercises aimed at improving programming skills. The system provides a platform where educators can create tournaments and battles, and students can register, form teams, and submit their solutions to coding problems.

CodeKataBlade allows educators to create tournaments, which serve as spaces for organizing multiple battles. Educators can invite other educators to participate in tournaments and perform manual evaluations on the submissions of battles they own. The system also provides a leaderboard that is updated in real-time, displaying the rankings of participants based on their performance in the battles.

Students can register to tournaments and battles, either individually or as part of a team. They can submit their solutions to coding problems within the specified deadlines. The system automatically evaluates the submissions using build automation scripts and provides feedback to the students. A set of test cases is used to assess the correctness and efficiency of the solutions.

CodeKataBlade integrates with GitHub, a popular code hosting platform, allowing students to fork repositories and work on their solutions using version control. GitHub Actions, a CI/CD tool, is utilized to automate the evaluation process and provide continuous integration and delivery capabilities.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Tournament:** A tournament is a space where educators can create battles and students can register to them. It has a registration deadline and a leaderboard that is updated every time a battle ends.
- **Battle:** A battle is a space where students can register to and submit their solutions to a problem. It has a registration deadline, a final submission deadline, a leaderboard that is updated every time a new evaluation is performed and a set of test cases that will be used to evaluate the submissions.

- **GitHub:** GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.
- **GitHub Actions:** GitHub Actions is a CI/CD tool that allows you to automate your software development workflows in the same place you store code and collaborate on pull requests and issues.
- **Educator:** An educator, in the context of the system, is a user of the platform that can create tournaments and battles, invite other educators to tournaments, perform manual evaluations on the submissions of a battle they own and consolidate the results of a battle.
- **Student:** A student, in the context of the system, is a user of the platform that can register to tournaments and battles, create teams and submit their solutions to a battle.
- **Build Automation Scripts:** Build automation scripts are scripts that are run automatically by the system every time a commit is pushed to the main branch of the forked repository of a battle. They are used to evaluate the submissions of the students.
- **Test Case:** A test case is a set of conditions under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do.
- **Timeliness:** Timeliness is the quality of doing something or producing something at the right time. In the context of the system, it refers to the time in which a student submits their solution to a battle with respect to the start of the battle and the final submission deadline.

1.3.2 Acronyms

- **CKB:** CodeKataBattle
- **S2B:** System to Be
- **TDD:** Test-Driven Development
- **CI/CD:** Continuous Integration/Continuous Delivery
- **UI:** User Interface
- **API:** Application Programming Interface
- **UML:** Unified Modeling Language

1.3.3 Abbreviations

- **Gn:** Goal number n
- **Wn:** World Phenomena number n
- **SPn:** Shared Phenomena number n

- **D_n:** Domain Assumption number n
- **R_n:** Requirement number
- **U_{Cn}:** Use Case number n

1.4 Revision history

1.5 Reference Documents

The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2023/2024.

1.6 Document Structure

- **1. Introduction:** This section provides an overview of the entire document. It describes the purpose and scope of the system, the definitions, acronyms and abbreviations used in the document, the revision history and the reference documents.
- **2. Architectural Design:** This section describes the architecture of the system. It provides an overview of the high-level components and their interaction, the component view, the deployment view, the runtime view, the component interfaces, the selected architectural styles and patterns and other design decisions.
- **3. User Interface Design:** This section provides a mockup of the user interface of the system.
- **4. Requirements Traceability:** This section describes how the requirements defined in the RASD map to the design elements defined in this document.
- **5. Implementation, Integration and Test Plan:** This section describes the order in which the components of the system will be implemented, integrated and tested.
- **6. Effort Spent:** This section describes the amount of time spent by each group member to redact this document.
- **7. References:** This section provides a list of the reference documents used to redact this document.

2 ARCHITECTURAL DESIGN

2.1 Overview: High-level components and their interaction

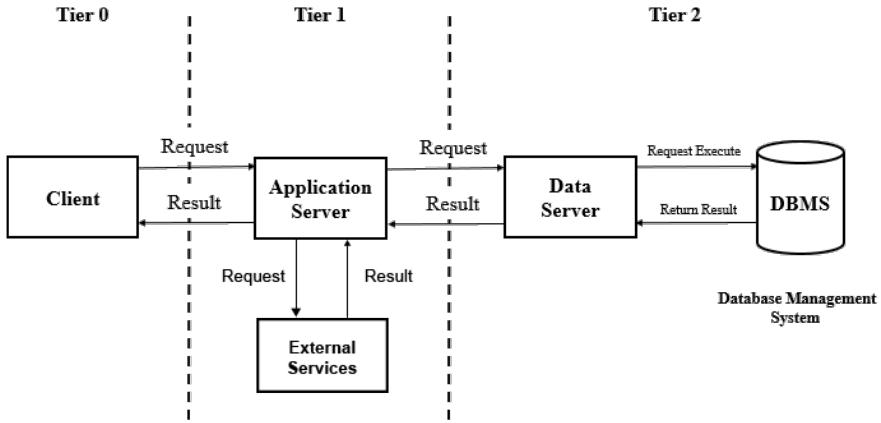


Figure 1: Three-tier Architecture

The chosen architecture for the system is a three-tier architecture, a widely adopted model for developing scalable and maintainable web applications. This architectural style divides the application into three interconnected layers: presentation, application, and data, with the addition of the external services like GitHub and email services. The rationale behind selecting this architecture is to achieve separation of concerns, modularity, and scalability.

Presentation Layer

The presentation layer constitutes the user interfaces for both students and educators, encompassing web or mobile interfaces. It is responsible for user interactions, displaying data, and triggering actions within the system.

Application Layer

The application layer serves as the business logic layer, managing core functionalities such as user authentication, tournament and battle processes, submissions, and evaluations. It facilitates bidirectional communication with both the presentation layer and the data layer.

Data Layer

The data layer, comprising the database and external services, handles the storage and retrieval of persistent data. It communicates with the application layer to respond to data requests, ensuring efficient data management.

2.1.1 Interaction

The interaction between the components is as follows:

Presentation Layer and Application Layer

The presentation layer communicates with the application layer through well-defined APIs. This interaction handles user inputs and actions, ensuring a smooth user experience.

Application Layer and Data Layer

The application layer interacts with the data layer to fetch or store information in the database. This interaction ensures that the platform has access to the necessary data for its operation.

Application Layer and External Components

External services, such as the GitHub API for code repositories and an email service for notifications, are integrated into the application layer. This interaction enhances the platform's functionality by incorporating external features.

2.1.2 Advantages of Three-Tier Architecture

- **Loose Coupling:** Separation of layers promotes loose coupling, enhancing modularity and ease of maintenance.
- **Scalability:** Each layer can be independently scaled based on specific requirements, allowing for better performance and resource utilization.

2.2 Component view

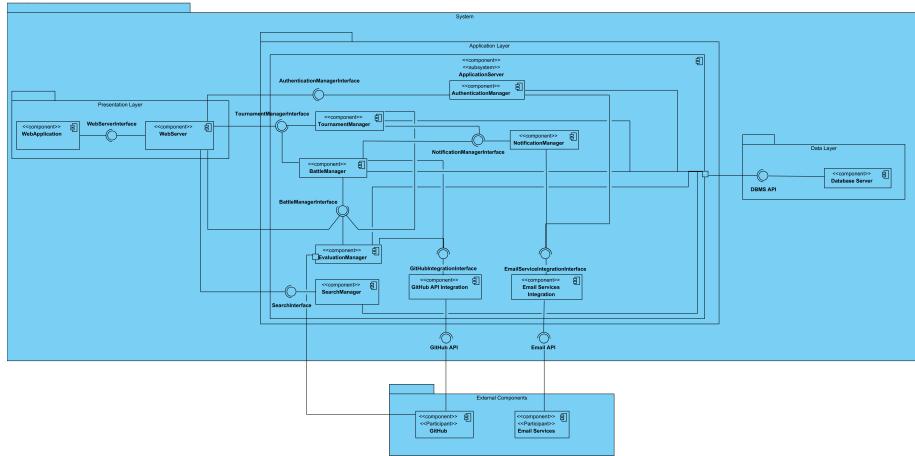


Figure 2: Component Diagram

2.2.1 Presentation Layer

Web Application:

- Represents the web application that will be used by educators and students.
- Allows educators to create tournaments and battles, invite other educators to tournaments, and perform manual evaluations on the submissions of battles they own.
- Allows students to register to tournaments and battles, create teams, and submit their solutions to battles.
- Provides a leaderboard that is updated in real-time, displaying the rankings of participants based on their performance in the battles.
- Provides a search functionality that allows users to search for tournaments and battles.
- Interfaces with the Web Server for communication with the Application Layer.

Web Server:

- Represents the component responsible for hosting the web application.
- Handles requests and responses from all users, including educators and students.

- Invokes APIs for communication with the Application Layer.

2.2.2 Application Layer

AuthenticationManager:

- Manages user authentication for educators and students.
- Generates and validates authentication tokens.
- Interfaces with email services for account confirmation and recovery.
- Interfaces with the Data Layer for user authentication.

TournamentManager:

- Manages the creation, moderation, and deletion of tournaments.
- Handles tournament-related processes and rules.
- Interfaces with the NotificationManager for tournament-related notifications.
- Interfaces with the Data Layer for tournament data.

BattleManager:

- Manages the creation, moderation, and deletion of battles within tournaments.
- Handles battle-related processes such as registrations, its deadlines, its GitHub repository creation, and its leaderboard.
- Interfaces with the TournamentManager for tournament-related processes such as tournament leadboard updating.
- Interfaces with the NotificationManager for battle-related notifications.
- Interfaces with GitHub API Integration for repository creation.
- Interfaces with the Data Layer for battle data.

EvaluationManager:

- Handles the evaluation of code submissions based on set criteria.
- Utilizes build automation scripts and test cases for evaluation.
- Interfaces with GitHub API Integration for repository pulling.
- Interfaces with the Data Layer for storage of evaluation results.
- Has a port through which it receives messages from GitHub which are called during GitHub Actions on submissions and evaluations.

NotificationManager:

- Manages the sending of notifications to users.
- Interfaces with Email Services Integration for sending notifications via email.

SearchManager:

- Manages the search functionality of the platform.
- Allows users to search for tournaments and battles.
- Interfaces with the Data Layer for tournament and battle data.

External Components Integration

GitHub API Integration:

- Interacts with the EvaluationManager to notify of new submissions.
- Communicates with the external GitHub API for repository creation and pulling.

Email Services Integration:

- Integrates with Email Services for sending notifications.
- Sends notifications for tournament invitations, submission updates, etc.
- Communicates with the external Email Services API for sending emails.

2.2.3 Data Layer

Database:

- Manages the storage and retrieval of persistent data.
- Stores information related to tournaments, battles, users, submissions, etc.

2.3 Deployment view

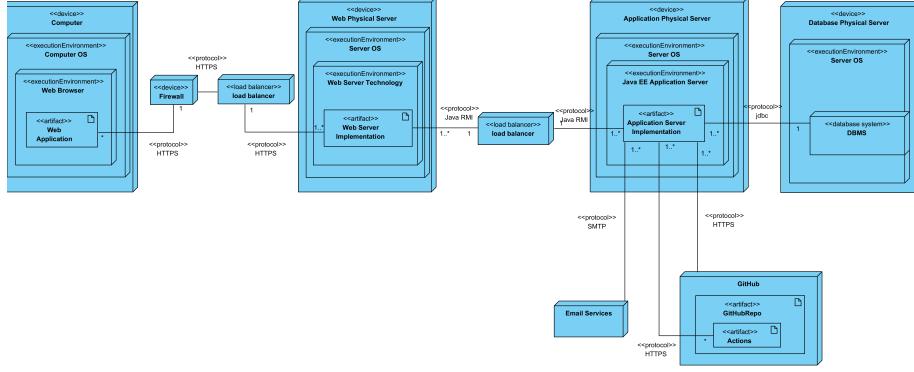


Figure 3: Deployment Diagram

On the deployment diagram, we can see the different components of the system and how they are deployed on the different nodes. In this case, the shown deployment assumes the application is developed on Java. The web application runs on the users' PC which communicates with the web server, which runs on a Web Server Technology which could be Apache Tomcat, Nginx or other. This two elements represent the presentation layer. The web server is connected to the application server, which runs on a Java EE Application Server. This element represents the application layer. The application server is connected to the database and to the external components. The database is managed by a Database Management System such as MySQL, PostgreSQL or other.

2.4 Runtime view

In the following diagrams we can see the sequence diagrams of the main functionalities of the system. These are more fleshed out versions of the use case diagrams presented in the RASD document. It is important to note that the calls to the database server are just not depicted as they would in a real scenario, since these are normally done via queries and not via custom method calls. It was represented via method calls so that it is easier to understand the meaning calls to the database server and the parameters used.

UC1 - Login

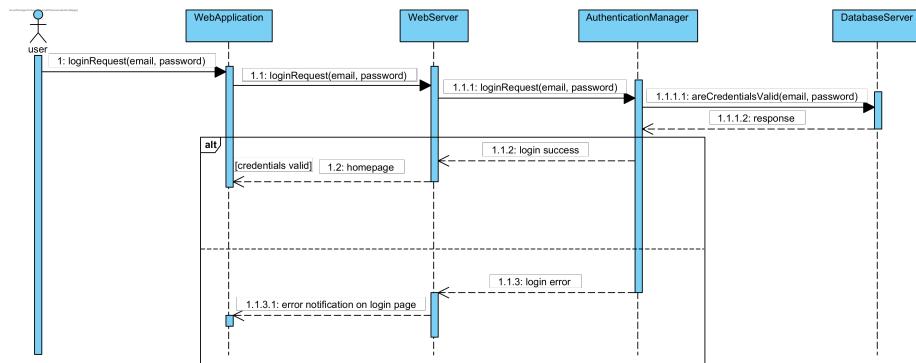


Figure 4: Use Case 1

UC2 - Retrieve Password

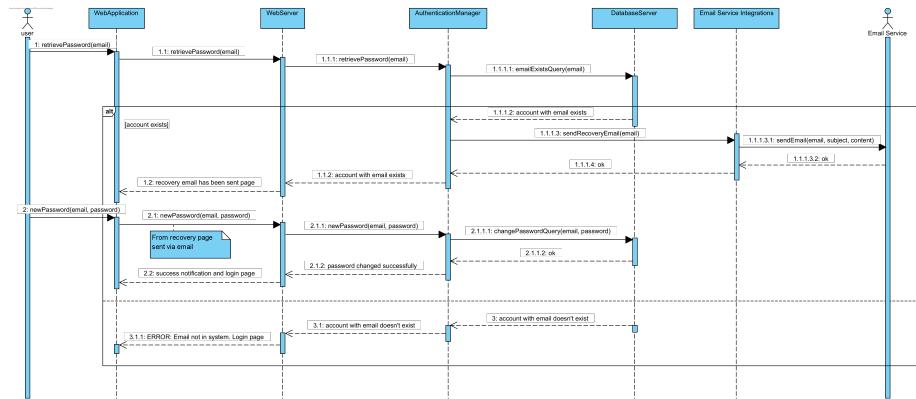


Figure 5: Use Case 2

UC3 - Register

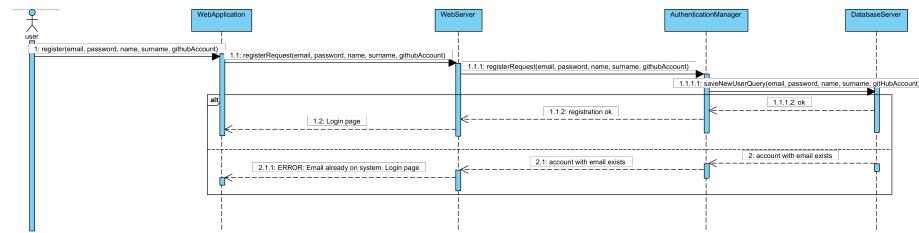


Figure 6: Use Case 3

UC4 - Educator Creates Tournament

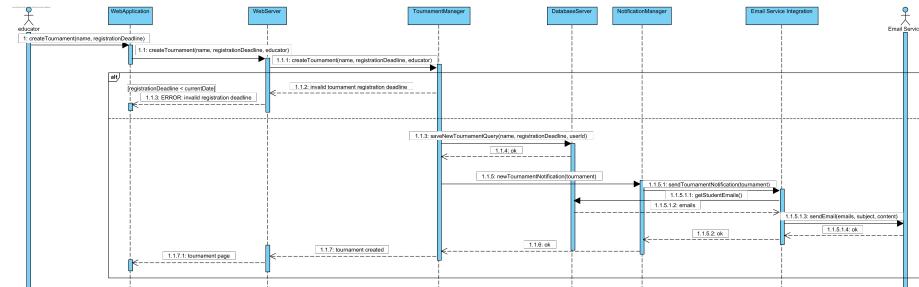


Figure 7: Use Case 4

UC5 - Educator invites other educator to tournament

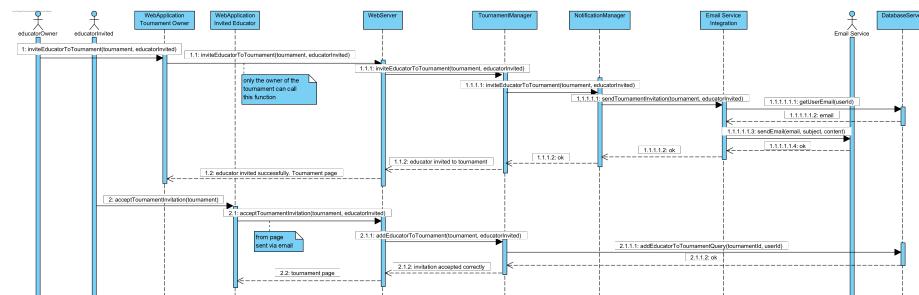


Figure 8: Use Case 5

UC6 - Educator creates battle

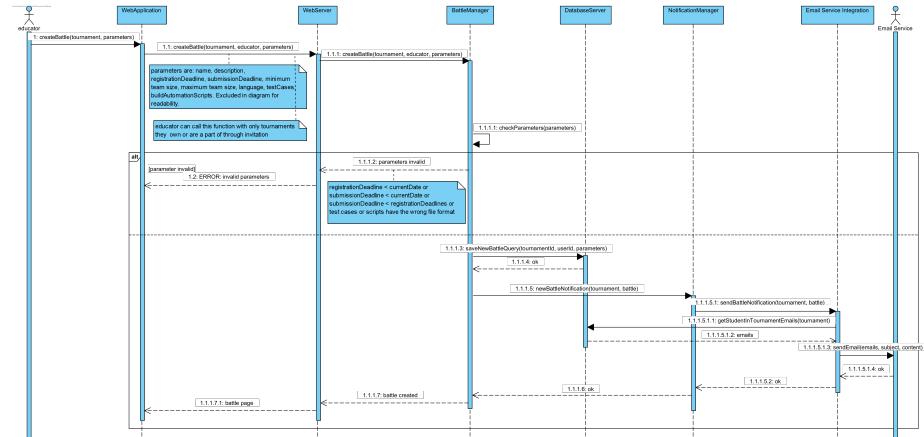


Figure 9: Use Case 6

UC7 - Tournament Registration

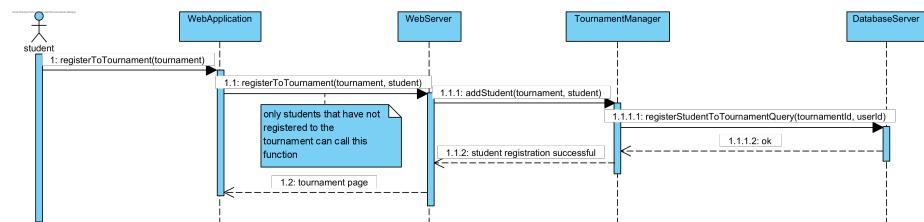


Figure 10: Use Case 7

UC8 - Tournament Unregistration

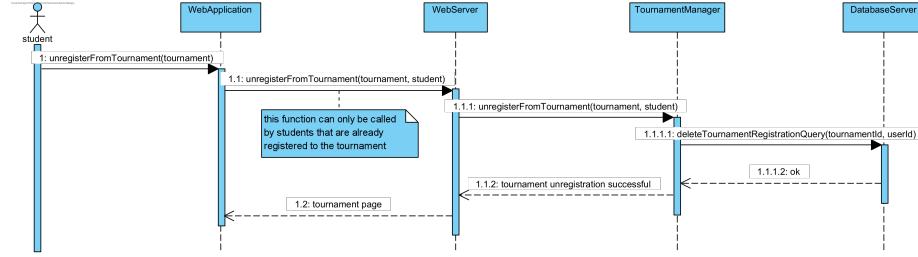


Figure 11: Use Case 8

UC9 - Battle Registration

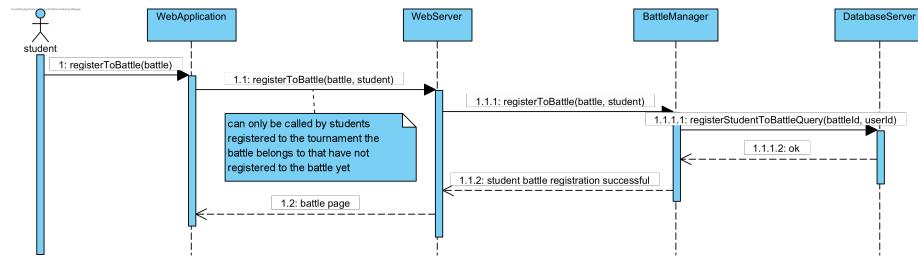


Figure 12: Use Case 9

UC10 - Student joins battle via invite

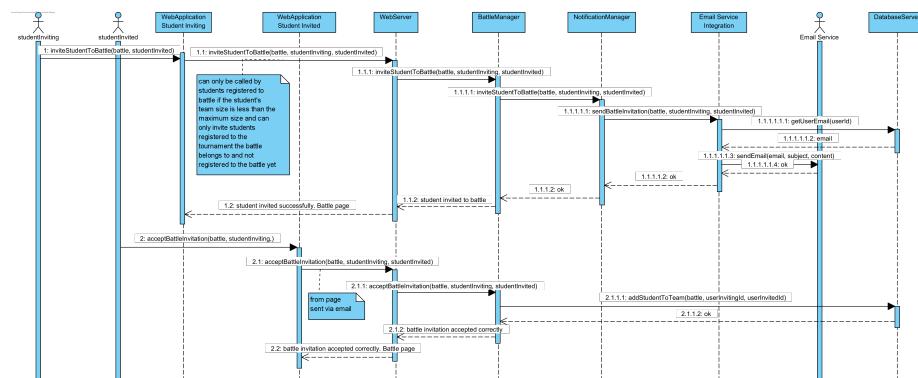


Figure 13: Use Case 10

UC11 - Battle Unregistration

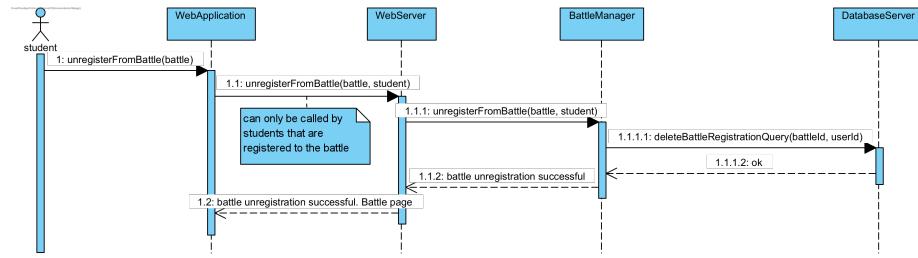


Figure 14: Use Case 11

UC12 - Students makes submission

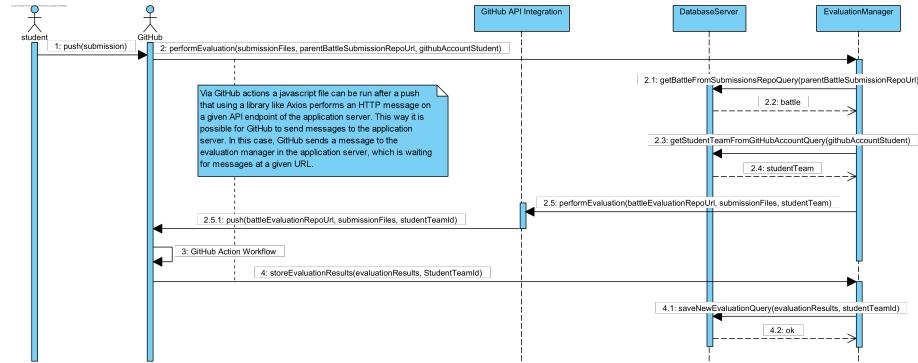


Figure 15: Use Case 12

UC13 - Educator Performs Manual Evaluations and Closes Battle

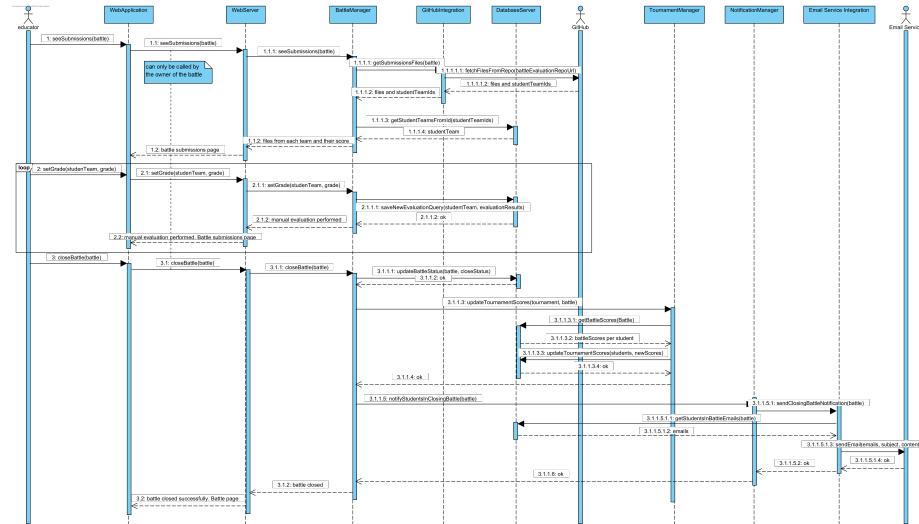


Figure 16: Use Case 13

UC14 - Educator Closes Tournament

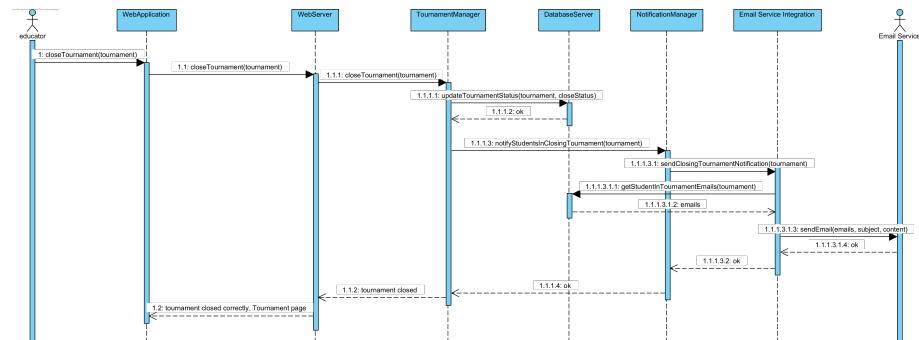


Figure 17: Use Case 14

UC15 - Find Tournament via search

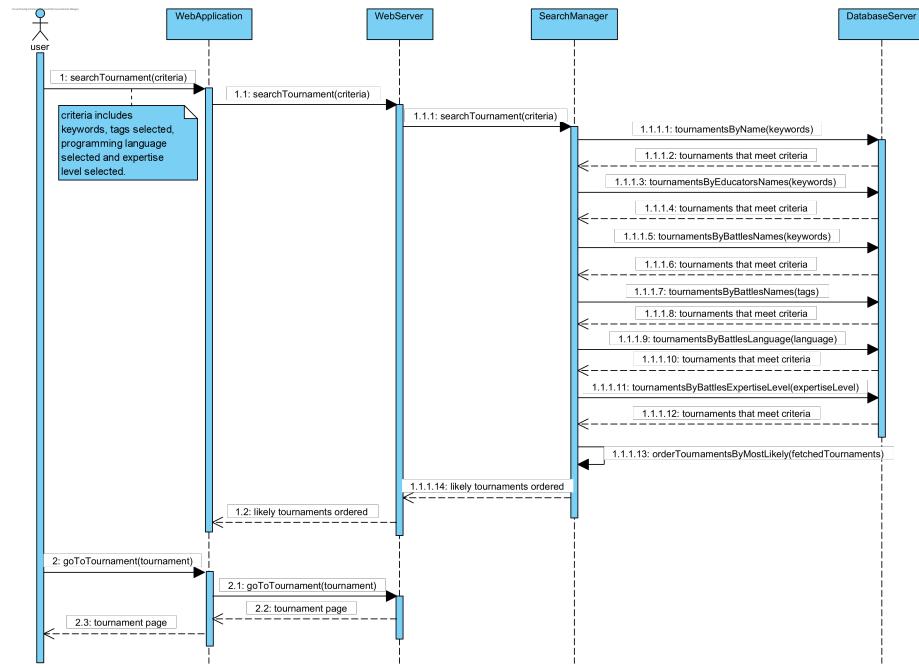


Figure 18: Use Case 15

UC16 - Find Tournament on main page

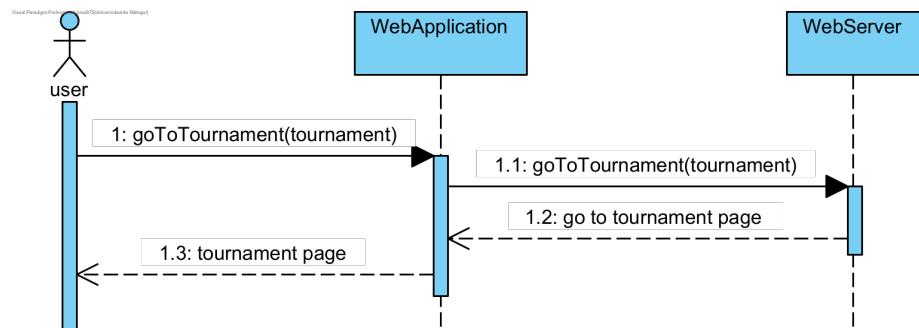


Figure 19: Use Case 16

UC17 - Find Battle on main page or on tournament page

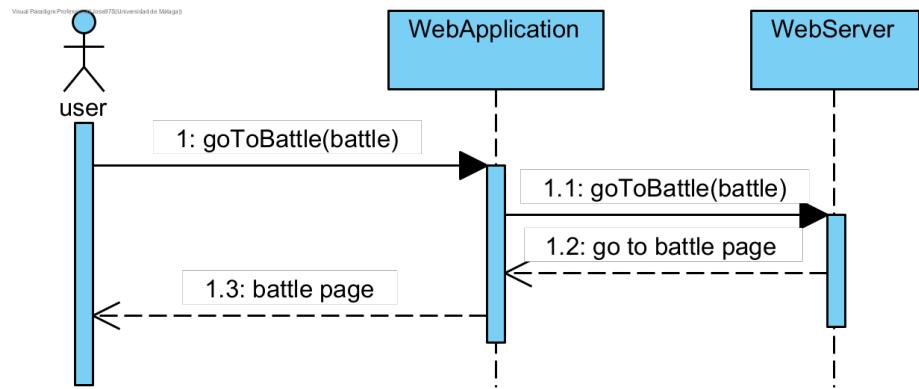


Figure 20: Use Case 17 and 18

2.5 Component interfaces

WebServer:

- void **loginRequest**(string **email**, string **password**)
- void **registerRequest**(string **email**, string **password**, string **name**, string **surname**, string **gitHubAccount**)
- void **retrievePassword**(string **email**)
- void **newPassword**(string **email**, string **password**)
- void **createTournament**(string **name**, Date **registrationDeadline**, Educator **educator**)
- void **inviteEducatorToTournament**(Tournament **tournament**, Educator **educatorInvited**)
- void **acceptTournamentInvitation**(Tournament **tournament**, Educator **educatorInvited**)
- void **declineTournamentInvitation**(Tournament **tournament**, Educator **educatorInvited**)
- void **createBattle**(Tournament **tournament**, Educator **educator**, string **name**, string **description**, Date **registrationDeadline**, Date **finalSubmissionDeadline**, int **maxNumberOfParticipants**, int **minNumberOfParticipants**, string **language**, File[] **testCases**, File[] **buildAutomationScripts**)
- void **registerToTournament**(Tournament **tournament**, Student **student**)
- void **unregisterFromTournament**(Tournament **tournament**, Student **student**)
- void **registerToBattle**(Battle **battle**, Student **student**)
- void **inviteStudentToBattle**(Battle **battle**, Student **studentInviting**, Student **studentInvited**)
- void **acceptBattleInvitation**(Battle **battle**, Student **studentInviting**, Student **studentInvited**)
- void **unregisterFromBattle**(Battle **battle**, Student **student**)
- void **seeSubmissions**(Battle **battle**)
- void **setGrade**(StudentTeam **studentTeam**, float **grade**)
- void **closeBattle**(Battle **battle**)
- void **closeTournament**(Tournament **tournament**)

- void **searchTournament**(string **keyword**, string[] **tags**, string **language**, string **expertiseLevel**)
- void **goToTournament**(Tournament **tournament**)
- void **goToBattle**(Battle **battle**)

AuthenticationManager:

- void **loginRequest**(string **email**, string **password**)
- void **registerRequest**(string **email**, string **password**, string **name**, string **surname**, string **gitHubAccount**)
- void **retrievePassword**(string **email**)
- void **newPassword**(string **email**, string **password**)

TournamentManager:

- void **createTournament**(string **name**, Date **registrationDeadline**, Educator **educator**)
- void **inviteEducatorToTournament**(Tournament **tournament**, Educator **educatorInvited**)
- void **addEducatorToTournament**(Tournament **tournament**, Educator **educatorInvited**)
- void **addStudent**(Tournament **tournament**, Student **student**)
- void **unregisterFromTournament**(Tournament **tournament**, Student **student**)
- void **updateTournamentScores**(Tournament **tournament**, Battle **battle**)
- void **closeTournament**(Tournament **tournament**)

In addition to these methods, the TournamentManager must also periodically check if the registration deadline of a tournament has passed, and if so, close the tournament registrations.

- void **monitorDeadlines()**

BattleManager:

- void **createBattle**(Tournament **tournament**, Educator **educator**, string **name**, string **description**, Date **registrationDeadline**, Date **finalSubmissionDeadline**, int **maxNumberOfParticipants**, int **minNumberOfParticipants**, string **language**, File[] **testCases**, File[] **buildAutomationScripts**)
- void **registerToBattle**(Battle **battle**, Student **student**)

- void **inviteStudentToBattle**(Battle **battle**, Student **studentInviting**, Student **studentInvited**)
- void **acceptBattleInvitation**(Battle **battle**, Student **studentInviting**, Student **studentInvited**)
- void **unregisterFromBattle**(Battle **battle**, Student **student**)
- Map<StudentTeam, File[]> **seeSubmissions**(Battle **battle**)
- void **setGrade**(StudentTeam **studentTeam**, float **grade**)
- void **closeBattle**(Battle **battle**)

In addition to these methods, the BattleManager must also periodically check if the registration deadline and submission deadline of a battle has passed, and if so, go to the next phase of the battle. In the case the registrationDeadline of a Battle has passed, the BattleManager besides closing the registrations, must also create the GitHub repositories for submission and evaluation, uploading to these repositories the test cases and build automation scripts. More specifically, in the submission repository, the BattleManager must setup the GitHub Actions workflow to communicate to the EvaluationManager when a new submission is made. On the other hand, in the evaluation repository, the BattleManager must setup the GitHub Actions workflow, test cases and build automation scripts to perform evaluations on uploaded files and communicate to the EvaluationManager the results of these. Besides this, it must also invite the students that registered to the battle to the submission repository.

- void **monitorDeadlines**()

NotificationManager:

- void **newTournamentNotification**(Tournament **tournament**)
- void **inviteEducatorToTournament**(Tournament **tournament**, Educator **educatorInvited**)
- void **newBattleNotification**(Tournament **tournament**, Battle **battle**)
- void **inviteStudentToBattle**(Battle **battle**, Student **studentInviting**, Student **studentInvited**)
- void **notifyStudentsInClosingBattle**(Battle **battle**)
- void **notifyStudentsInClosingTournament**(Tournament **tournament**)

EvaluationManager:

- void **performEvaluation**(File[] **submissionFiles**, string **parentBattleSubmissionRepoUrl**, string **gitHubAccountStudent**)

- void **storeEvaluationResults**(float **evaluationResult**, int **StudentTeamId**)

SearchManager:

- void **searchTournament**(string **keyword**, string[] **tags**, string **language**, string **expertiseLevel**)

GitHub API Integration:

- void **performEvaluation**(String **battleEvaluationRepoUrl**, File[] **submissionFiles**, StudentTeam **studentTeam**)
- Map<StudentId, File[]> **getSubmissionFiles**(Battle **battle**)

Email Service Integration:

- void **sendRecoveryEmail**(string **email**)
- void **sendTournamentNotification**(Tournament **tournament**)
- void **sendTournamentInvitation**(Tournament **tournament**, Educator **educatorInvited**)
- void **sendBattleNotification**(Tournament **tournament**, Battle **battle**)
- void **sendBattleInvitation**(Battle **battle**, Student **studentInviting**, Student **studentInvited**)
- void **sendClosingBattleNotification**(Battle **battle**)
- void **sendClosingTournamentNotification**(Tournament **tournament**)

2.6 Selected architectural styles and patterns

3-tier Architecture: As explained in the overview, the chosen architecture for the system is a three-tier architecture, a widely adopted model for developing scalable and maintainable web applications. This architectural style divides the application into three interconnected layers: presentation, application, and data, with the addition of the external services like GitHub and email services.

Model-View-Controller: The Model-View-Controller (MVC) is a software architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects. In the case of this system, it is used to separate the presentation layer from the application layer.

RESTful API: REST stands for Representational State Transfer. It is a software architectural style that defines a set of constraints to be used for creating web services. Web services that conform to the REST architectural style, called RESTful web services, provide interoperability between computer systems on the Internet. RESTful web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations. In the case of this system, it is used to define the communication between the presentation layer and the application layer.

2.7 Other design decisions

Entity Relationship Diagram: The structure defined for the database is as follows:

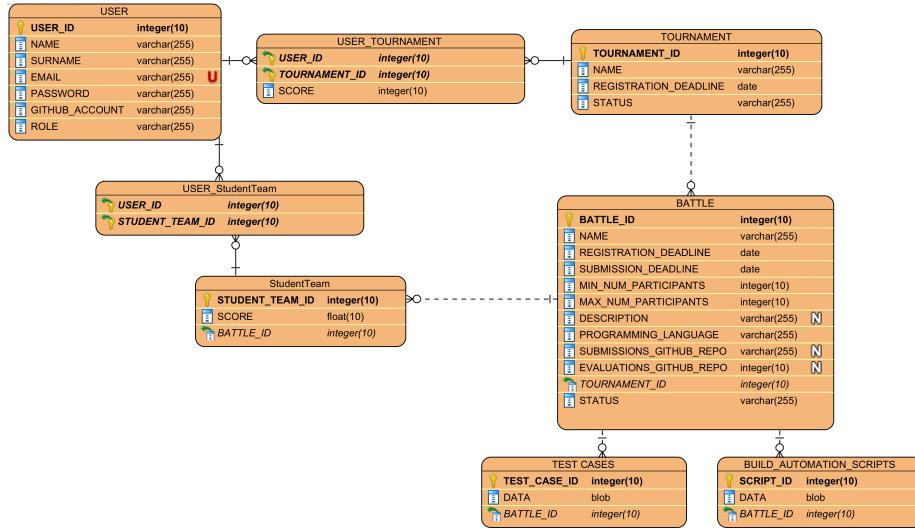


Figure 21: Entity Relationship Diagram

Also, in order to prevent a single point of failure in the database, it is replicated in a master-slave configuration.

3 USER INTERFACE DESIGN

3.1 Mockups

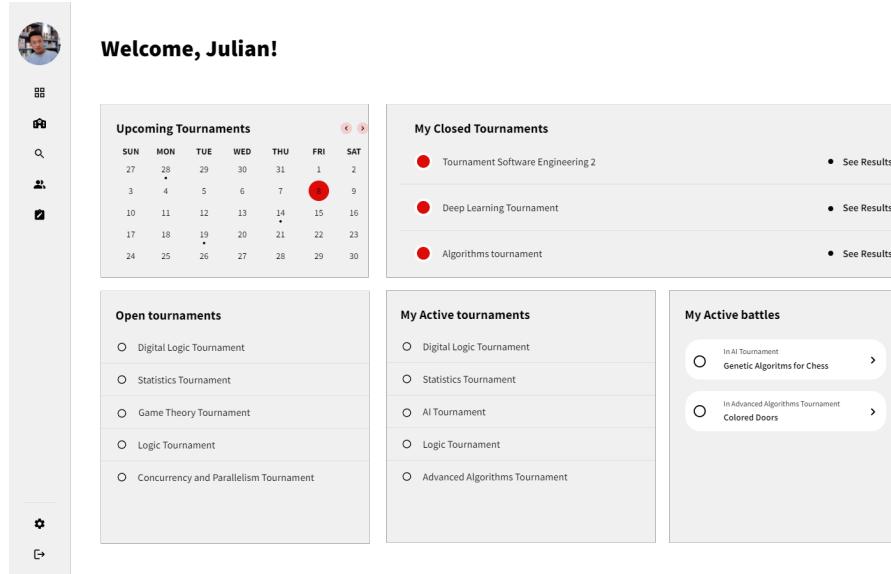


Figure 22: Homepage

Homepage for all users: On this screen, both students and educators can see the tournaments and battles they are a part of, the closed tournaments in which they participated and open tournaments they can register to or be invited to respectively. From this screen, they can travel to the tournament or battle page by clicking on the respective card. If there are more tournaments or battles than the ones that can be shown, the user can scroll down on each of the tabs to see more. There is also a calendar that shows the important dates of both the upcoming tournaments and battles and the ones that are currently happening and the user is a part of.

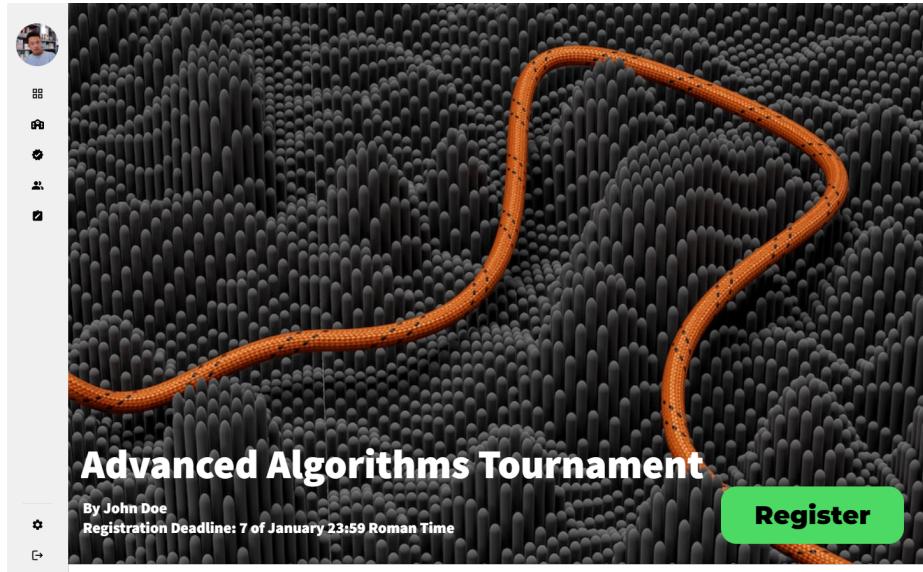


Figure 23: Tournament Page for Students before Registration Deadline expires

Tournament Page for Students before Registration Deadline Expires:
On this screen, students can see the name and registration deadline of the tournament. If the student is not registered, they can register to the battle. If the student is already registered, they can unregister from the battle.

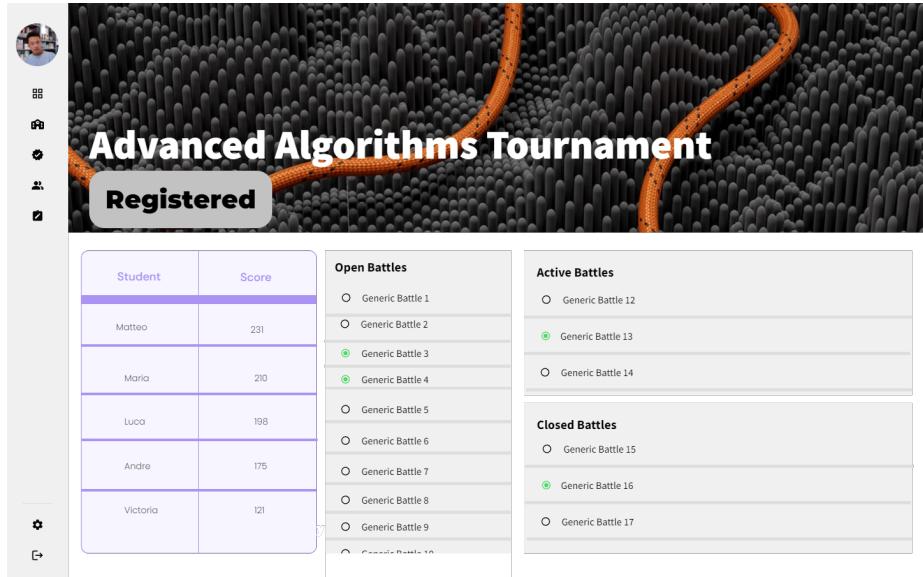


Figure 24: Active Tournament Page for Students

Active Tournament Page for Student: On this screen, students can see the leaderboard, open, active and closed battles of that tournament. If the student seeing the page is registered to the tournament it will be shown with a non-clickable indicator. If the student is registered to a battle, it will be shown with a green indicator. Students can go to any of the tournaments by clicking on the respective name.

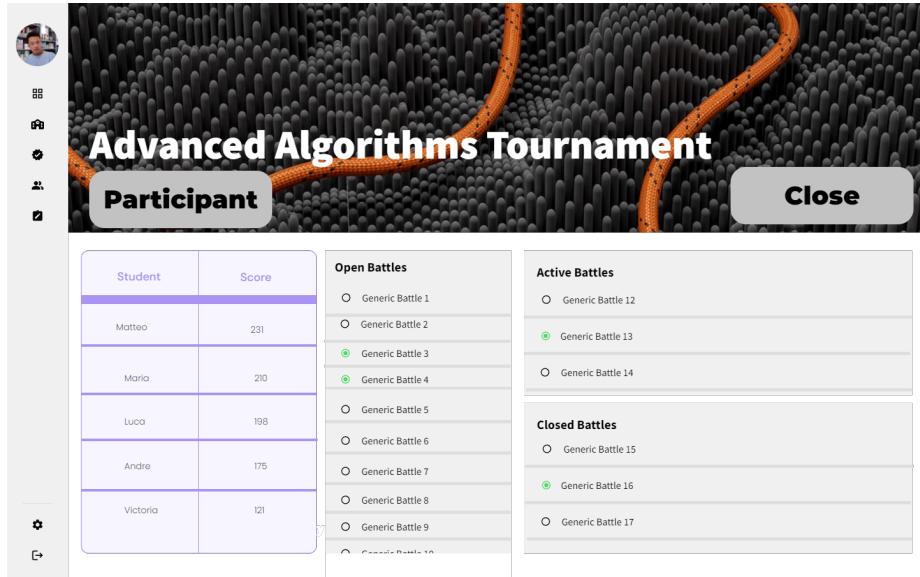


Figure 25: Active Tournament Page for Educator

Active Tournament Page for Educator: On this screen, educators can see the leaderboard, open, active and closed battles of that tournament. If the educator is the owner or is a participant of the tournament, this will be shown with a non-clickable indicator. If the educator is the owner of a battle, a close tournament button will be shown. This button is unavailable if there are open or active battles in the tournament. If a educator is the owner of a battle, it will be shown with a green indicator. Educators can go to any of the battles in the tournament by clicking on the respective name.

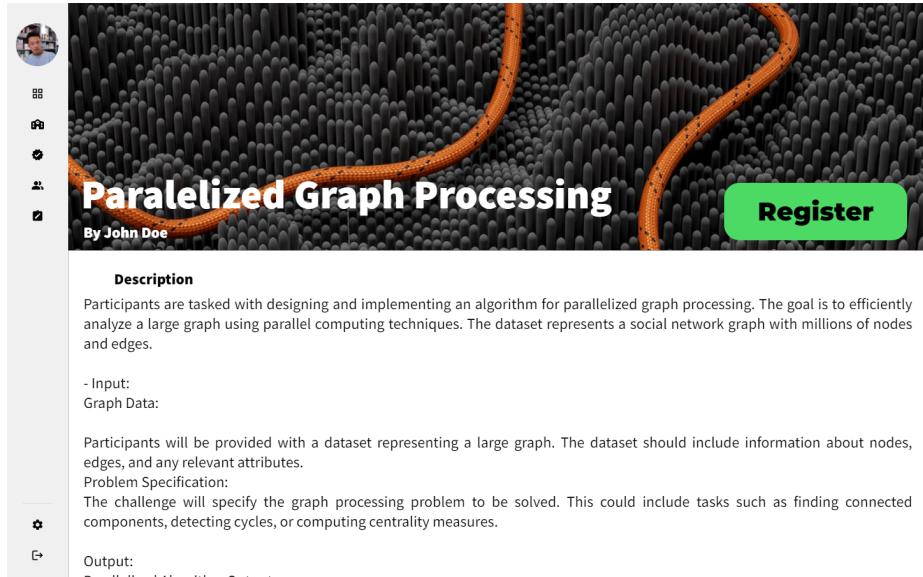


Figure 26: Open Battle Page

Open Battle Page: On this screen, users can see the details of the battle, such as the description, the language and other information. Additionally, if the user is a student and if the student is registered to the tournament the battle belongs to and is not registered to the battle, they can register it. If the student is already registered to the battle, they can unregister from the battle. Also, if the student is registered, they can invite other students to the battle.

The screenshot shows a web page for a battle titled "Parallelized Graph Processing" by "John Doe". The page has a dark background with a textured orange snake-like graphic. On the left, there's a sidebar with user icons and a registration status bubble saying "Registered". The main content area includes a "Description" section detailing the task of designing and implementing a parallelized graph processing algorithm, and a "Student" table showing scores for four participants: Louis (73), Rick (71), Christian (65), and Elena (57). The table has columns for "Student" and "Score".

Student	Score
Louis	73
Rick	71
Christian	65
Elena	57

Figure 27: Active Battle Page

Active or Closed Battle Page: On this screen, users can see the details of the battle, such as the description, the language and the leaderboard. Additionally, if the user is a student and registered to the battle, it displays this information.

The different tournament and battle pages are located on the same URL for the same tournament or battle. The differences shown on the UI are dependant of the status of the tournament or battle and the type of user.

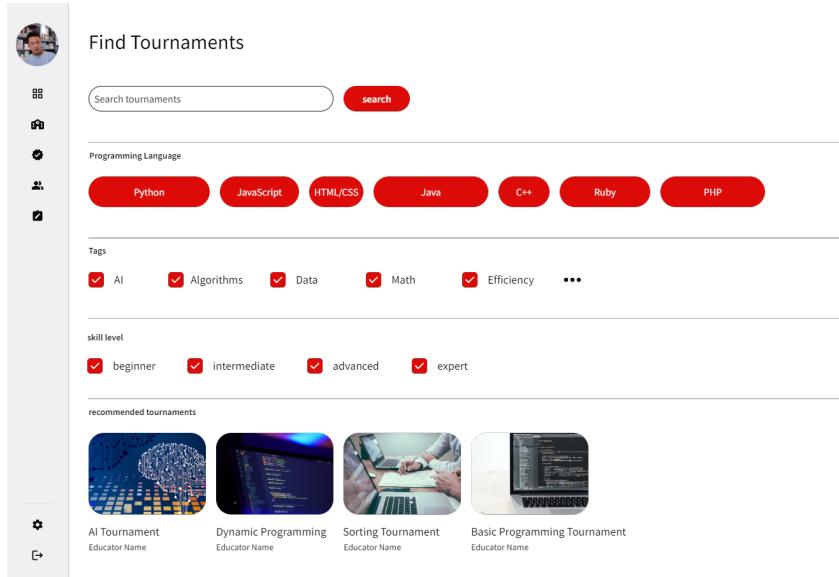


Figure 28: Search Environment

Search Environment: On this screen, users can look for tournaments according to keywords, language, tags and expertise level. The results are shown in a list, and the user can click on any of the tournaments to go to the tournament page.

Other screens like login, register, password recovery, etc. are not shown as they are standard and similar to other applications.

3.2 Flowcharts

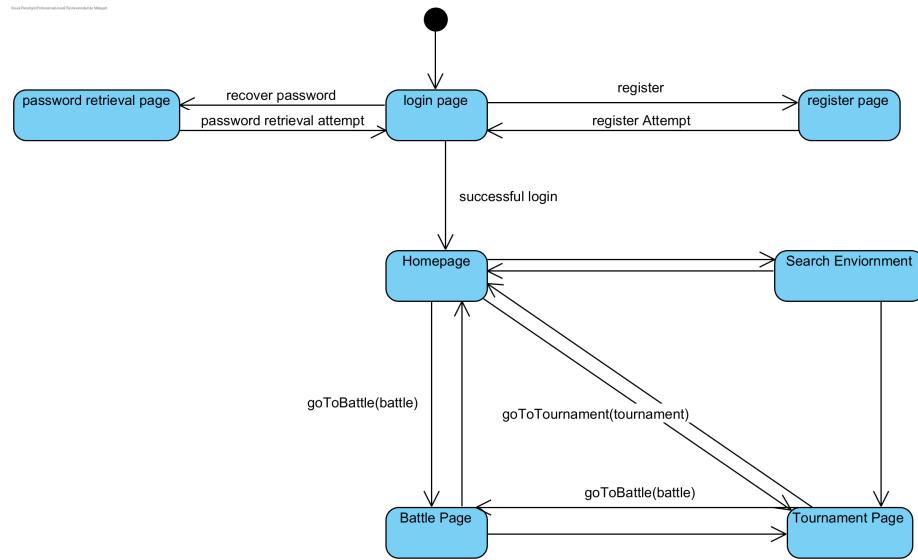


Figure 29: User Interfaces Flowchart

This diagram shows the flow of the user interfaces of the system on a normal execution. There are other possible flows, such as entering to specific pages for accepting invitations or new password creation, but these are only accessible via email link and are not shown here.

4 REQUIREMENTS TRACEABILITY

5 IMPLEMENTATION, INTEGRATION AND TEST PLAN

6 EFFORT SPENT

7 REFERENCES