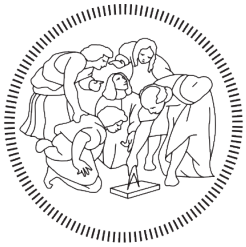
Software Engineering 2 - Prof. Di Nitto Elisabetta
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

CodeKataBattle

DD
Design Document

December 21, 2023

Marco Cerino 244780
Mattia De Bartolomeis 245022



POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

The primary objective of this document is to offer a comprehensive and detailed overview of the proposed software, with a special emphasis on its architectural framework, system modules, and their respective interfaces. Additionally, the document will present a dynamic view of the software's key functionalities, illustrated through elaborate interaction diagrams that depict the communication flow among various components. The document will also encompass critical aspects of the implementation, testing, and integration phases, ensuring a holistic understanding of the software development process.

1.2 Scope

The main human actors in this system are students and educators. Educators use this platform to challenge students by creating competitions where groups of students compete against each other, demonstrating and improving their skills. The challenges consist of a programming exercise in a chosen language (such as Java or Python). Students must follow a "test-first" 1approach, writing code to pass provided tests. There is also a non-human actor that plays a crucial role in the platform : Github. GitHub plays a central role in the CodeKataBattle (CKB) for hosting battles and facilitating collaboration among students. It enables automated evaluations through GitHub Actions, tracking teams' progress and updating battle scores in real-time. Here's how the system works: a teacher creates a "battle" following specific steps. They upload the problem description and the project to CKB, set the minimum and maximum number of students per group, define deadlines for registration and project submission, and set evaluation criteria. Once enrolled in a "battle," students form teams and start working on the project. The platform integrates GitHub, a source code management service, to facilitate collaboration. Whenever students upload a new version of their code, the platform automatically calculates the team's score, considering aspects such as the number of passed tests, timeliness of submissions, and code quality. The automated assessment also includes static code analysis to evaluate aspects like security, reliability, and maintainability. Teachers can assign personal scores based on students' performance. At the end of each "battle," the platform updates scores and displays the updated ranking. Students and teachers can monitor progress during the challenge. After the final submission deadline, a consolidation phase takes place, which may involve manual assessment by teachers. Once this phase is complete, all involved students are notified of the final "battle" ranking. Scores obtained in each "battle" contribute to the overall tournament score for each student. These scores are used to create an overall tournament ranking, showcasing students' performance compared to their peers.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Student - An individual enrolled in an educational institution or self-learner who uses the CKB platform to enhance their software development skills.
- Educator - A teacher or education professional who uses the CKB platform to create and manage code kata battles and tournaments for students.
- Educator with permission - Educator who has the authority to create battles within a tournament , and also has the ability to close that same tournament.
- Tournament - A series of code kata battles, created and organized by an educator, where teams of students compete to achieve the highest score in each battle and in the overall tournament.
- Battle - A programming exercise that consists of a textual description and a software project with build automation scripts and a set of test cases to be passed.
- Project - The solution proposed by the students for the exercise during the battle.
- Battle Score - A natural number between 0 and 100 assigned to each team in a battle, based on mandatory factors evaluated automatically and optional factors evaluated manually by educators
- Battle ranking - A ranking that lists students in order of performance. This ranking is determined by the battle score obtained by each student in the specific battle.
- Tournament Ranking - A ranking that lists students in order of performance within a single tournament on the CKB platform. This ranking is determined by summing the scores obtained by each student in all the code kata battles that make up the tournament they participated in

1.3.2 Acronyms

- CKB - Code Kada Battle.
- RASD - Requirement Analysis and Specification Document.
- UI - User interface.
- UML - Unified Modelling Language.

1.3.3 Abbreviations

- WP - World Phenomena.
- SP - Shared Phenomena.
- G - Goal
- R- Requirement

1.4 Revision History

1.5 Reference Documents

The creation of this document is based on :

- Slides of Software Engineering 2 course
- The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Elisabetta Di Nitto at the Politecnico di Milano, A.Y 2023/2024;

1.6 Document Structure

- Chapter 1: Introduction. his chapter offers a comprehensive overview of the project, highlighting the system's scope and purpose. It also includes details about this document itself.
- Chapter 2: Architectural Design. Aimed at the development team, this chapter provides an in-depth look at the system's architecture. It starts by discussing the adopted paradigm and how the system is organized into various layers. Then, it presents a high-level overview of the system and its modules.
- Chapter 3: User Interface Design. This chapter is intended for the graphical designers and contains various prototypes of the application, along with diagrams that illustrate the application's logical flow.
- Chapter 4: Requirements Traceability.This section links the RASD (Requirements Analysis and System Design) document with the DD (Design Document), offering a complete mapping of the requirements and goals outlined in the RASD to the logical modules presented in this document.
- Chapter 5: The final section is directed towards the development team and describes the procedures for implementing, testing, and integrating the software components. It includes detailed descriptions of the main functionalities, as well as a comprehensive report on how to implement and test them.

2 Architectural Design

2.1 Overview

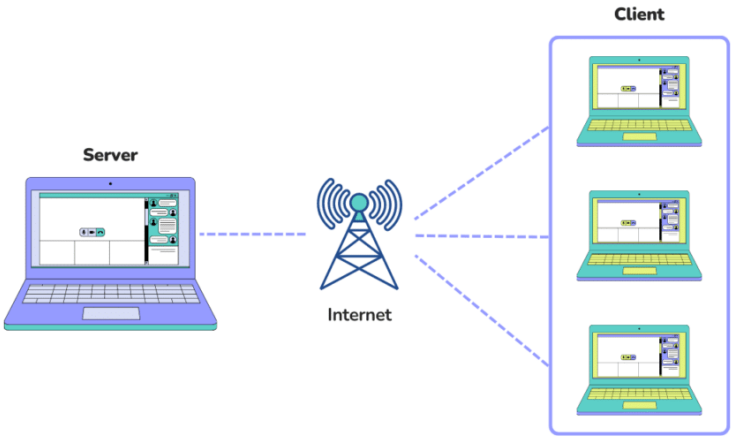


Figure 1: Client-Server architecture

The system is a distributed application that follows the well-known client-server paradigm. It adopts the thin client technique to facilitate the support of different client platforms. This approach offers significant advantages: first, it reduces hardware and software requirements on client devices, as the main processing is handled by the server. This means that even clients with limited processing capabilities can access the system effectively. Second, it greatly simplifies maintenance and software updating, as changes are centralized on the server and do not have to be implemented on each client device individually. Finally, it enhances security, as sensitive data can be stored and managed more securely on the server, reducing the risk of data loss or security breaches on client devices. Our system uses web applications as clients. In addition our system also embraces the microservices approach to efficiently handle specific application functionalities. Microservices are a software architecture in which different parts of the application are divided into independent services, each performing a specific function. This modular division allows for greater scalability and maintainability of the system.

Tier architecture

The architecture of our system is organised into four tiers:

1. Thin Client: The Thin Client is the user's point of contact, such as a web browser.
2. Web Server: The Web Server acts as an intermediary between the client and the Application Server. It is responsible for receiving HTTP/HTTPS requests from the Thin Client and forwarding the processed requests to the Application Server.
3. Application Server: The Application Server is the heart of the system, where the business logic and application functionalities are processed. This tier processes requests from the Web Server, performing complex operations, managing business logic, transactions, and interaction with the Database Server. The Application Server can host a variety of web services, microservices, and APIs that expose the application's functionality to the Web Server and, consequently, to the Thin Client.
4. Database Server: The Database Server handles the persistence, retrieval, and management of data.

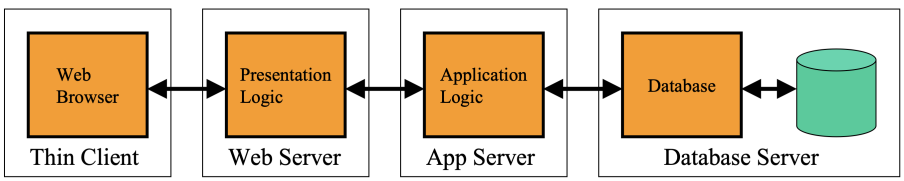


Figure 2: 4 Tier architecture

Layers architecture
An application consists of layers representing the different levels and types of abstraction of the software. The layers help to slice the application into more manageable units and support multiple implementations. The logical software architecture of our system consists of three layers:

1. Presentation layer: manages the presentation logic and the user interaction.
2. Application layer: manages the business logic and functions that the system must provide.
3. Data layer: manages the storage and retrieval of data.

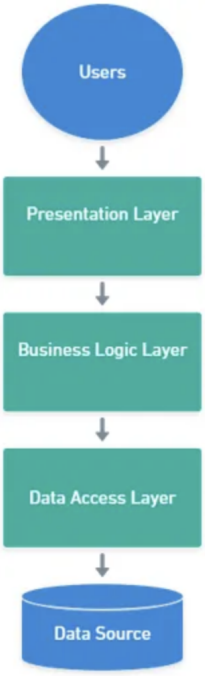


Figure 3: 3 Layers architecture

The first two tiers are used for the presentation layer, while the 3rd tier is for the application layer and the 4th tier is for the data layer.
The system architecture requires the client to interact with web servers. These servers act as an intermediate layer, facilitating communication between the client and the application layer via well-defined APIs. The application layer, in turn, communicates with the data layer via Database Management System (DBMS) APIs.
To ensure ease of portability and scalability, the APIs on the application server are designed to be RESTful. Such APIs are simple, standardised and stateless. This design choice increases flexibility and allows for easier integration of client code. When it comes to interacting with the data layer, we use Object-Relational Mapping (ORM) techniques. This approach exploits the principles of object-oriented programming, allowing seamless access to a relational database.
As far as security is concerned, each physical layer of the system is isolated by a dedicated firewall. In addition, all communication between these layers is encrypted, adhering to the HTTPS standard for secure data transmission.

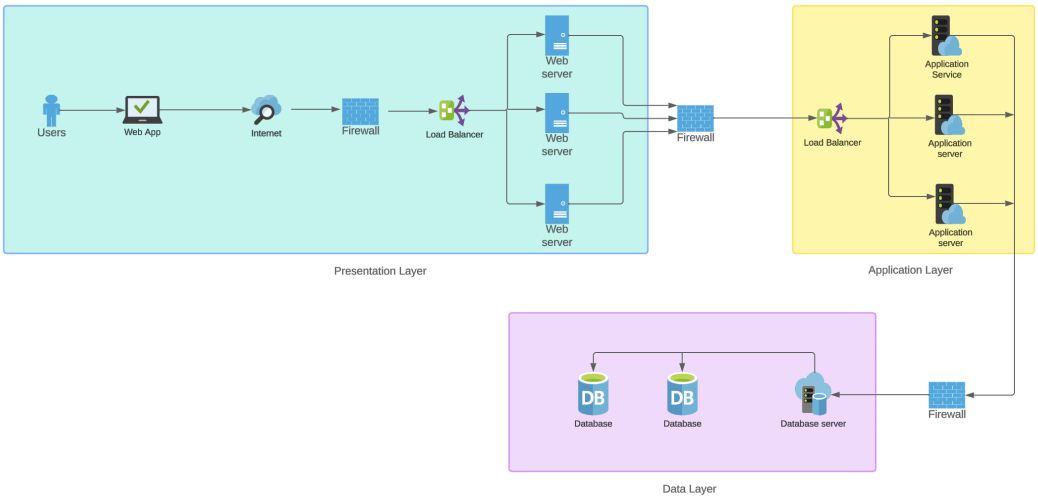


Figure 4: Architecture

Load Balancing In the architecture presented, there are two load balancers that play crucial roles in managing incoming traffic to the web and application servers.

1. First Load Balancer: Positioned after the firewall and before the web servers, this load balancer's task is to distribute incoming client requests to the web servers. When a request passes through the firewall, it is intercepted by the load balancer, which then determines which web server is most suitable to handle that request. The selection criteria are based on the current load of each server and the capacity to minimize response time. Thus, load balancing helps to prevent overloading a single web server, ensuring an equitable distribution of requests and optimizing resource utilization.
2. Second Load Balancer: Located between the web servers and the application servers, the second load balancer functions similarly to the first but at a different level of the architecture. After the web server has processed the initial request, it forwards it to the second load balancer. This load balancer then distributes the request to one of the available application servers, again based on the workload and the capacity to provide a rapid response. In this manner, the application layer is also protected from the risk of overload and can scale effectively in response to traffic variations.

In the following sections, each component of the system will be elaborated in more detail, providing a complete understanding of the entire architecture.

2.2 Component View

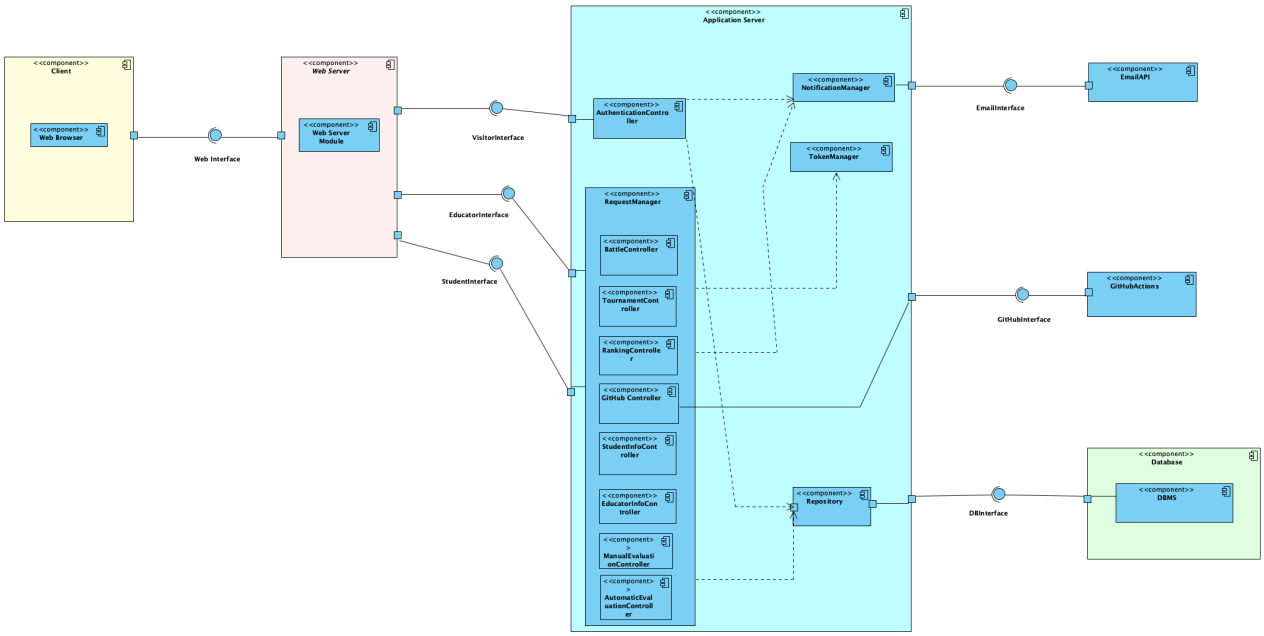


Figure 5: 3 Layers architecture

Short description of the most important components::

- Web Server: it handles managing HTTP requests from the client and forwarding them to the application server.
- Authentication Controller: it manages all requests from users who are not yet authenticated, giving them the opportunity to register (Signup Service) or log in (Login Service). To handle all requests, it utilizes the login interface and registration interface and issues authentication tokens that are useful for other services. The token is also used to identify whether the user is a student or educator.
- Request Manager: It verifies that each request comes from a user who has logged in using the token manager and then redirects each request to the appropriate controller.
- BattleController: It is responsible for managing all functionalities related to battles, such as creating new battles, creating a team, participating in new battles, etc.
- TournamentController: It is responsible for managing all functionalities related to tournaments, such as creating new tournaments, participating in new tournaments, etc.
- RankingController: It is responsible for managing ranking functionalities, such as displaying real-time scores for each team in each battle, displaying scores for each student in each tournament, etc.
- ManualEvaluationController: It allows only educator-type users to assign a score to each student's task for a specific battle where manual evaluation is enabled.
- AutomaticEvaluationManager: It is responsible for the automatic calculation of the project score for each team upon submission to GitHub by the teams participating in a specific battle. This controller is triggered by GitHub Actions.
- GitHub Controller: It is responsible for handling push events originating from GitHub and performs the necessary operations to calculate or update team scores in a repository-based competition, taking into account timeliness and functional scores.
- Token Manager: It manages the creation, validation, renewal, and revocation of authentication tokens.
- StudentInfoController: It is responsible to retrieve the information about the student.
- EducatorInfoController: It is responsible to retrieve the information about the educator.

- NotificationManager: It handles the management of notifications whenever requested. Notifications are sent via email through the Email Service.
- Repository: serves as a repository for data that constitutes the enduring state of the system, which, in our scenario, is preserved within a database. When the application initializes, this component ensures that the data is loaded for use. It encompasses a suite of essential methods for accessing this data and encapsulates the logic necessary for their manipulation. Essentially, when any system component requires database information for computational purposes, it entrusts the Repository component with the retrieval task. The Repository component is the sole interface with the database management system (DBMS), acting as the exclusive conduit for data interaction. Additionally, once the Repository component retrieves data from the database, this data is organized into data structures defined by the Model component for further processing and usage within the system.

2.3 Deployment View

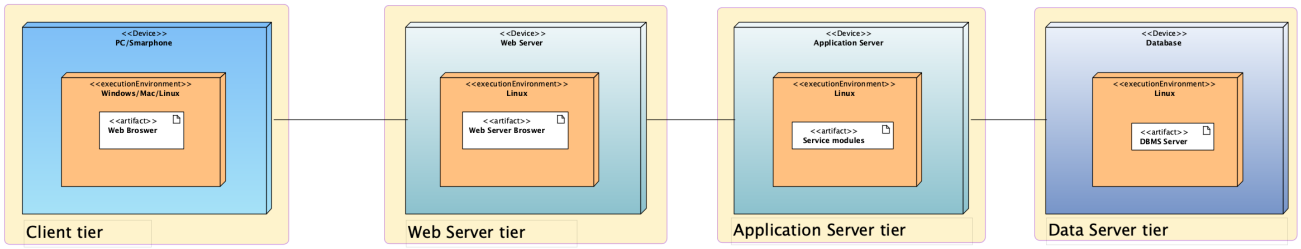


Figure 6: Architecture

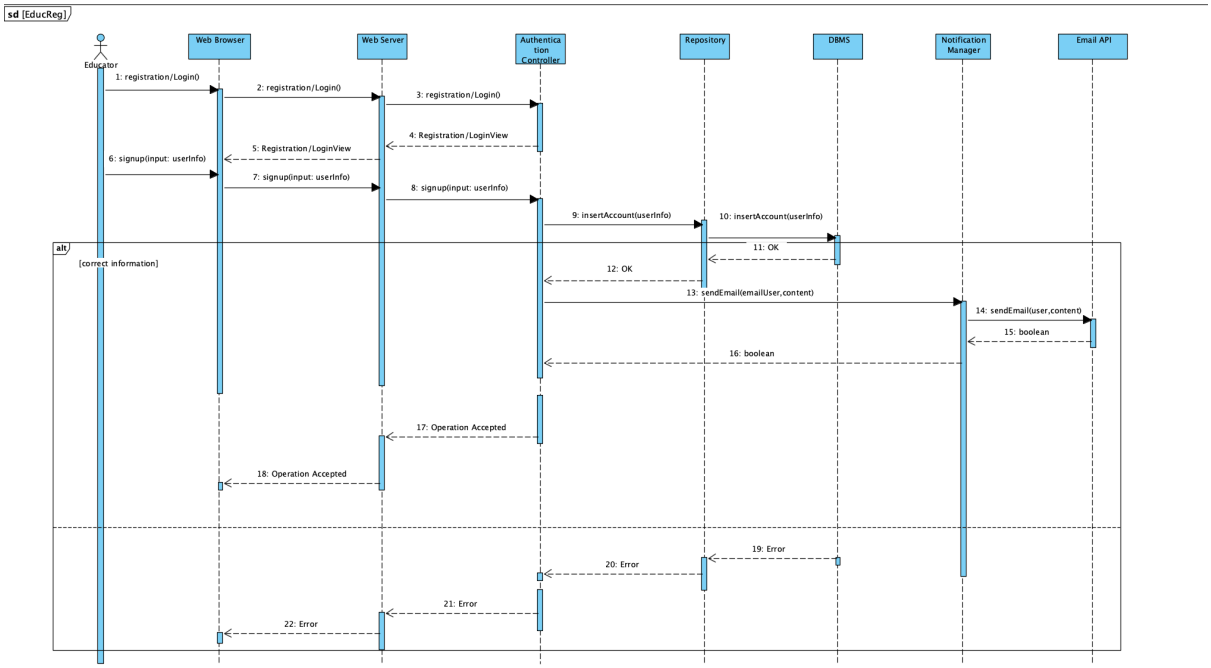
The diagram presented shows the organisation of the system, highlighting the components required for its operation. Each device illustrated runs a specific operating system that supports the execution of the software components. To expand the system, additional copies of these devices can be added. The tiers are four:

- **Client tier**: This layer includes devices such as PCs and smartphones using operating systems like Windows, Mac, or Linux. The main software component at this level is the web browser, which allows users to interact with the system.
- **Web server tier**: This layer consists of web servers running a Linux operating system. The software component represented here is Apache, which handles processing client requests and serving web pages.
- **Application server tier**: This layer includes application servers, also on Linux, running service modules. These modules manage the business logic of the application and usually communicate between the web server and the database server.
- **Data server tier**: The final layer is composed of a database running on Linux, with a DBMS (Database Management System) Server as the software component. This manages the storage and retrieval of data.

2.4 Runtime views

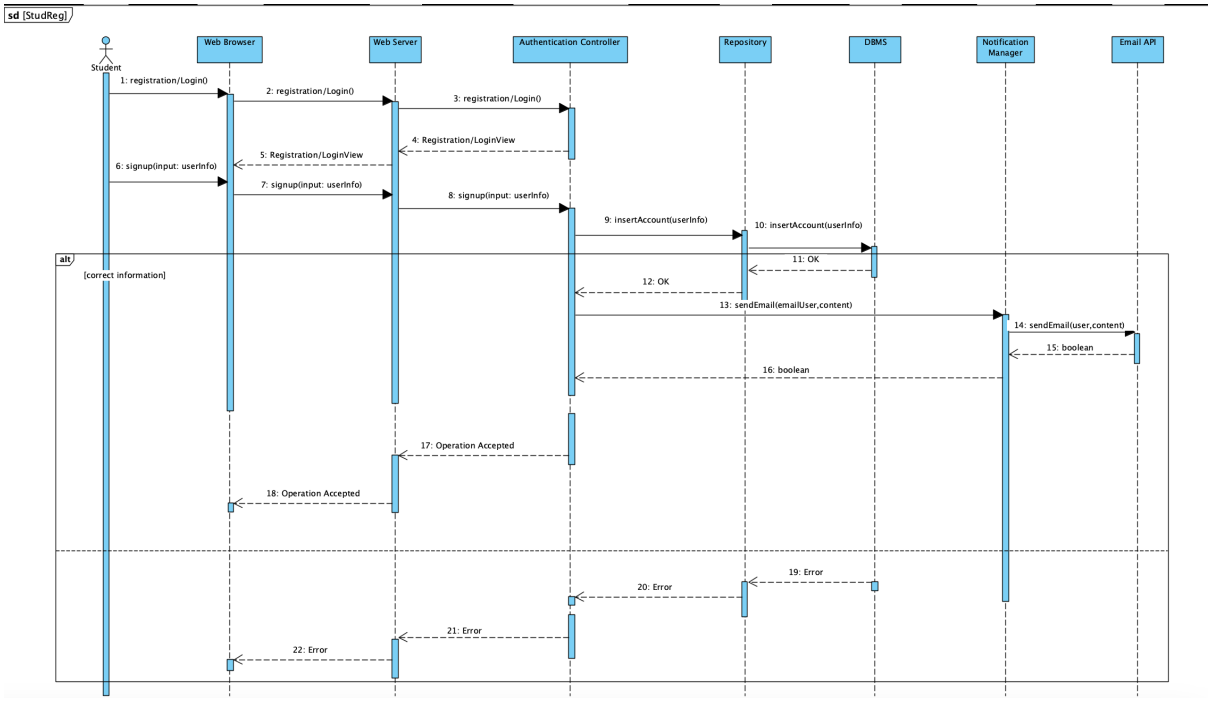
This chapter details the dynamic perspectives linked to the use cases outlined in the RASD for CKB. It illustrates how different components interact to facilitate the functions provided by CKB, offering a comprehensive look at its operational architecture.

2.4.1 Educator Registration



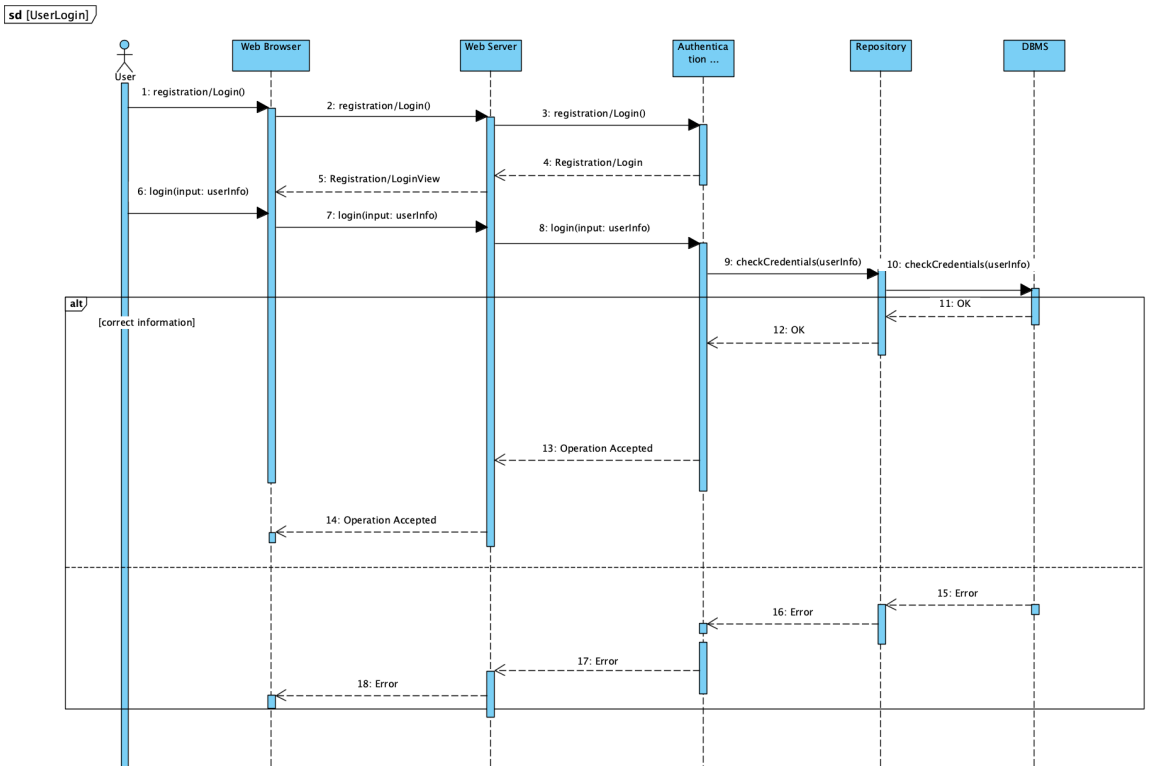
The sequence diagram depicted shows the registration flow of an educator. The educator starts the process from the homepage by clicking on the 'Registration/Login' section. The system offers the options of registration or login. Upon choosing registration, the educator is presented with a form to complete and a box to tick if he/she is an educator. After entering the data, the educator confirms registration. The system verifies the data and invites the educator to check his/her mailbox to confirm the registration via a link sent by the platform. If the data is incorrect, the DBMS sends an error message that propagates to the Web Browser. On the backend, components such as the Web Server, the Authentication Controller, and others manage the registration process, from creating the account to entering the data in the archive, to sending an email for verification. Once the educator confirms registration via email, his or her data is entered into the database, successfully completing the registration process.

2.4.2 Student Registration



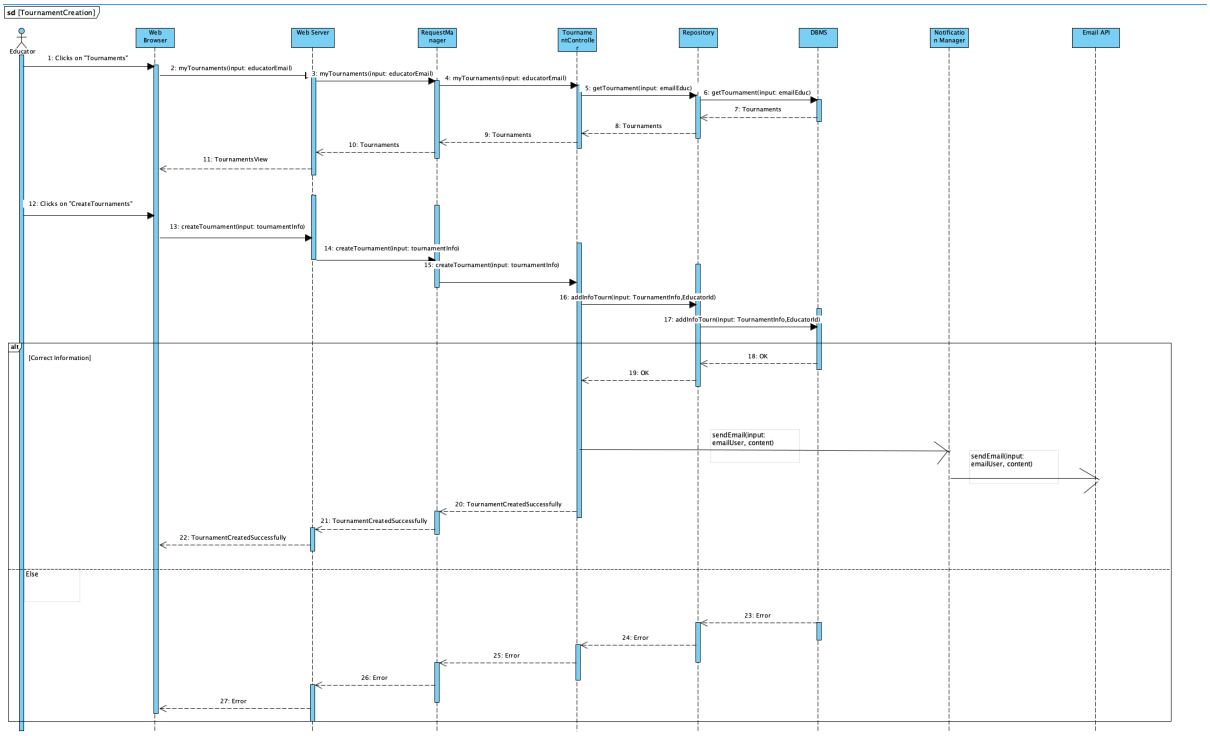
In the sequence diagram displayed, a student initiates registration via the homepage, completes a form with his or her personal data and confirms registration. The Web Browser interacts with the Web Server, which in turn involves the Authentication Controller. This component co-ordinate the registration operation, with the Authentication Controller sending the data to the Repository layer, which forwards it to the Database Management System (DBMS) for actual entry of the information into persistent storage. The process continues with the generation of a confirmation email, which relies on the Notification Manager to send the message to the user. The flow ends when the student, by verifying and accepting the registration via the link received by email, activates the final entry of his or her data into the database. If, on the other hand, the data entered are found to be inadequate, an error message is sent.

2.4.3 User Login



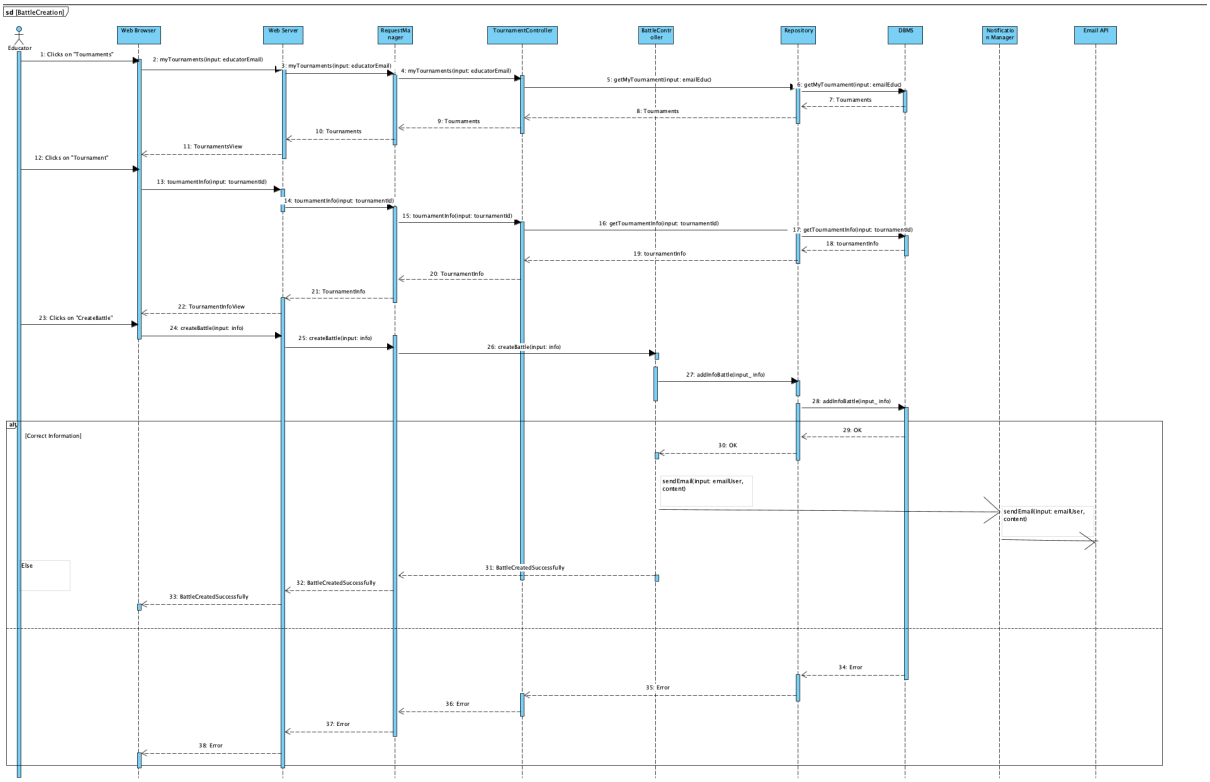
The sequence diagram shown illustrates the login process of a user. The user starts by clicking on the "Registration/Login" section of the homepage, after which the system presents registration or login options. By choosing 'Login', the user is prompted to enter their email and password. Once the credentials have been entered and the "Confirm" button clicked, the system via the Web Server passes the information to the Authentication Controller. The latter component makes a call to the Repository, which queries the DBMS to verify the user's credentials. If the information is correct, the process ends with the transaction being accepted and the system grants the user access to their profile dashboard. Otherwise, an error is reported and the user cannot log in.

2.4.4 Tournament Creation



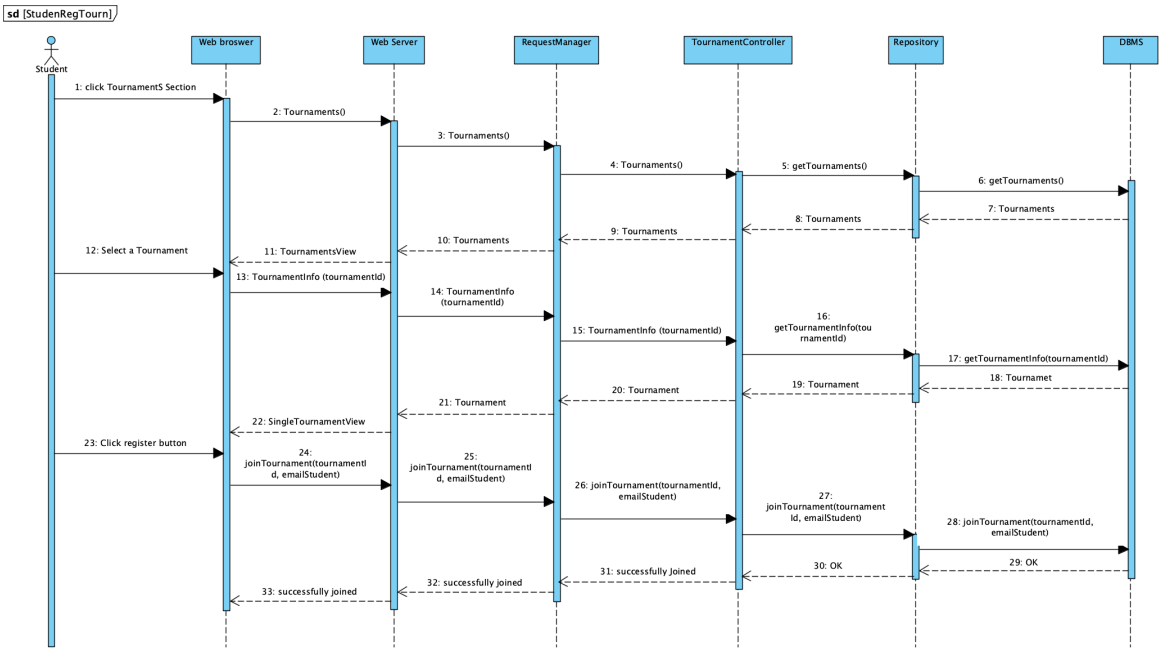
The sequence diagram shows the process the educator follows to create a tournament on the platform. After selecting the option to create a tournament, the educator enters the required data via the Web Browser. This information is sent to the Web Server, which interfaces with the Request Manager. The Request Manager co-ordinates the data flow, sending the information to the Tournament Manager.
The Tournament Manager is responsible for the business logic related to the creation of tournaments. It communicates with the Repository, which represents the logical structure of the data, and queries the DBMS to verify the existence and validity of the email addresses provided. Once the verification is complete, the Repository updates the DBMS with the new tournament information.
Once the creation of the tournament is confirmed, the Notification Manager comes into play, invoking the Email API to notify students registered on the platform. The process concludes by saving the tournament information in the DBMS and notifying interested users, showing positive feedback through the educator user interface.

2.4.5 Battle Creation



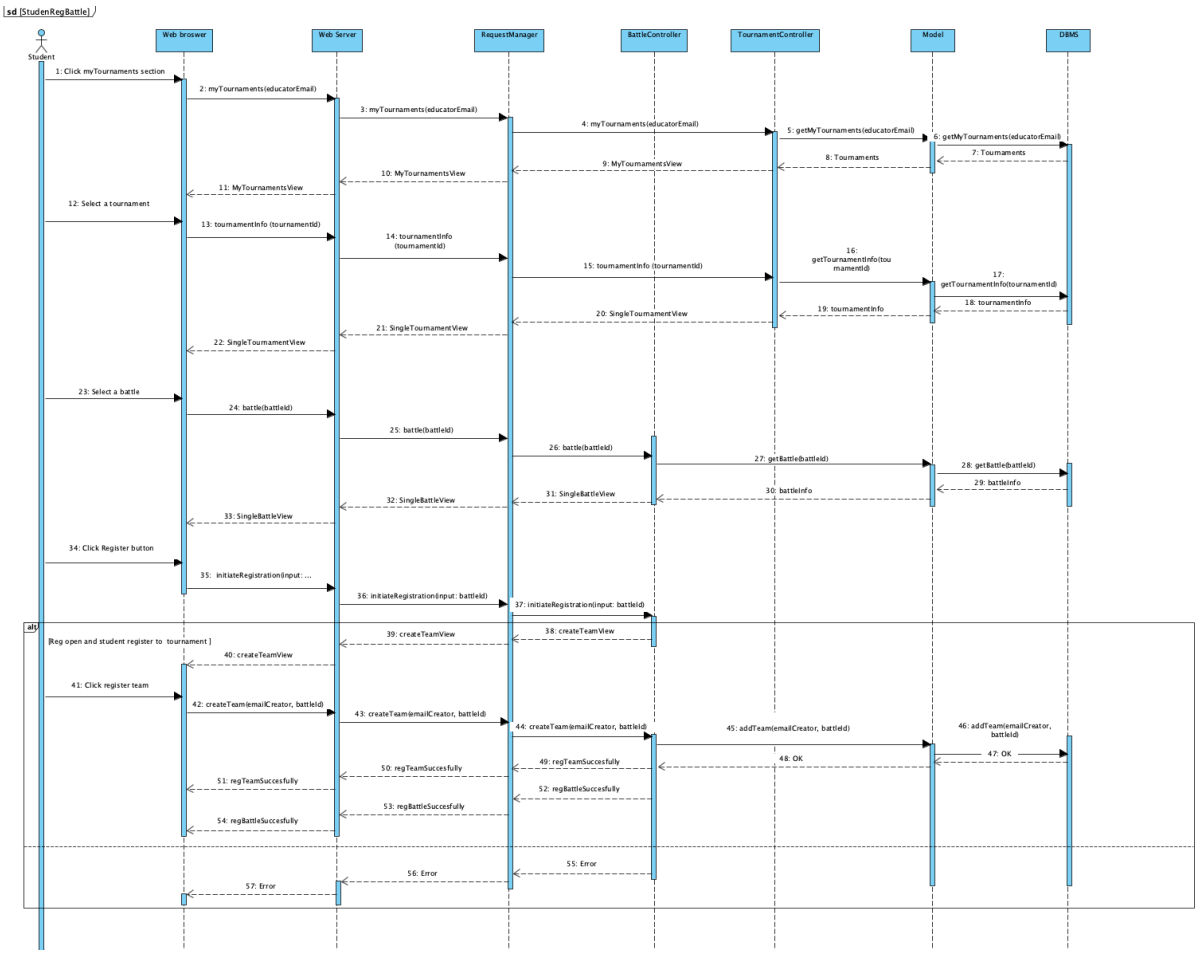
In the sequence diagram, the educator interacts with the system to create a battle within a tournament. This interaction begins in the Web Browser and is sent to the Web Server, which acts as a conduit for all subsequent requests and responses. The flow continues with the Request Manager which processes the request to create the battle, routing it to the Tournament Controller. The latter has the task of managing the tournament-specific data and interfaces with the Model to manipulate the logical structure of the data. The Repository communicates with the DBMS to verify the uniqueness of the battle name and, if valid, to store the new information in the database. Once the DBMS confirms the addition of the data with an 'OK', the Notification Manager is activated. The Notification Manager coordinates with the Email API to send notifications to students registered for the tournament, informing them of the newly created battle.

2.4.6 Student registers for the Tournament



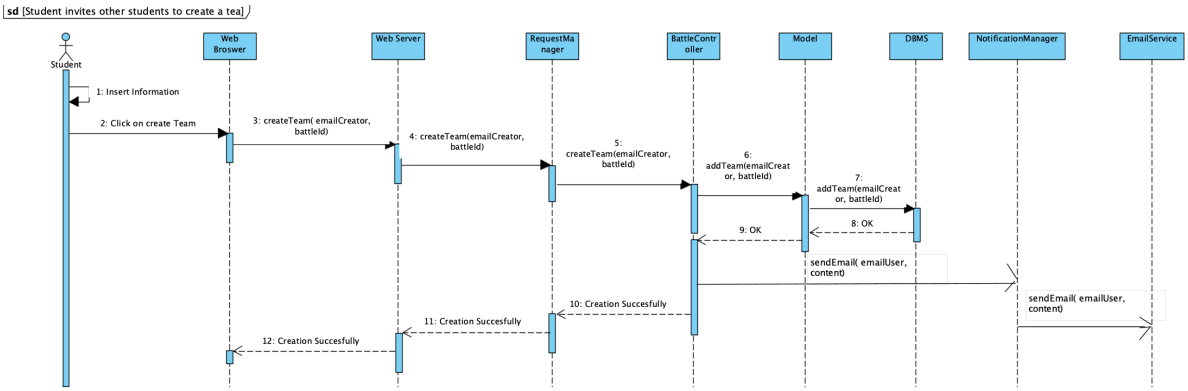
The sequence diagram describes the process of registering a student for a tournament through the system. The student selects the "Tournaments" section from the dashboard, and the system displays a list of available tournaments. After browsing through the options, the student chooses a tournament of interest and the system presents the specific page of the selected tournament. Once on the tournament page, the student initiates registration by clicking the "Register" button. This action triggers a series of events between the system components: the Web Browser transmits the request to the Web Server, which then passes the information to the Request Manager. The Request Manager coordinates the request with the Tournament Controller, which in turn requests the Model to retrieve and provide tournament information from the Database Management System (DBMS). After displaying the tournament information, when the student confirms the registration, the Tournament Controller invokes the "joinTournament" function, which again interacts with the Repository and the DBMS to register the student's participation in the tournament. The DBMS confirms the operation with an "OK", indicating that the registration has been successfully completed and that the student's data has been correctly updated in the database. The process ends with the Web Browser confirming that the student has been successfully registered, and the student receives a message confirming their participation in the tournament.

2.4.7 Student registers for the Battle



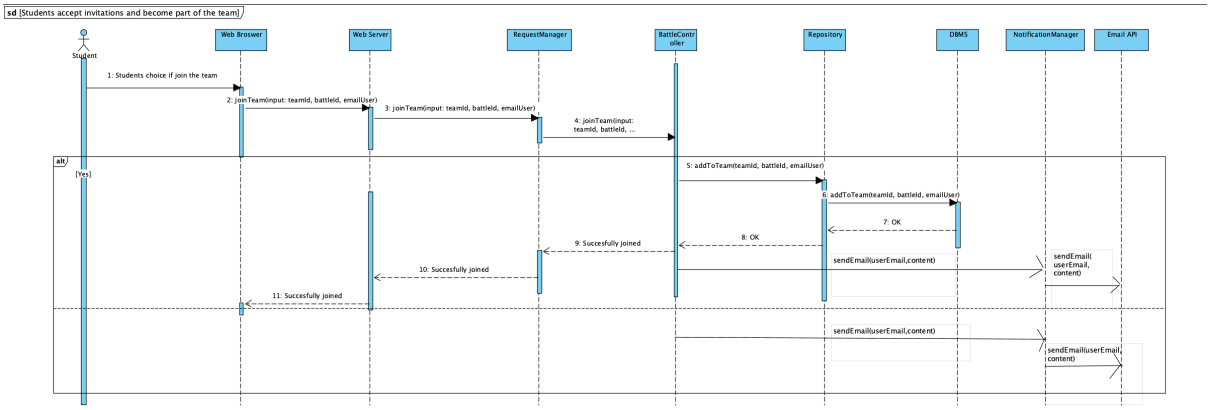
The sequence diagram describes the process of registering a student for a battle within a tournament. After selecting the desired tournament and battle from the Web Browser, the Request Manager directs the request to the Battle Controller for operations relating to the specific battle. The Battle Controller, in turn, queries the Repository to obtain the details of the battle. The Repository acts as an abstraction of the tournament data and facilitates interaction with the Database Management System (DBMS). This allows the student to view the details of the specific battle and press the register button. At this point, the student displays createTeamView in which they register their team by entering the necessary information. Once the DBMS has confirmed the addition of the data with an 'OK', the Battle Controller receives this confirmation and forwards the success information to the Request Manager. The Request Manager then informs the Web Server that the registration has been successfully completed. Finally, the Web Server communicates with the Web Browser, allowing the system to display a confirmation message to the student confirming the team's registration for the selected battle. If the registration for the battle is not open, or the student is not registered for the tournament of which the battle is a part, the student receives an error message.

2.4.8 Student invites other students to create a Team



The sequence diagram shows the flow of events for the formation of a team by a student within the platform. After selecting "Register" for a specific battle, the student enters the team name and selects the students to form the team.
The flow begins with the student entering the required information into the Web Browser. This information includes the team name and email addresses of the invited students. After entering this information, the student proceeds by clicking the "Create Team" button. The Web Browser sends the request to the Web Server, which passes the details to the Request Manager. The Request Manager directs the request to the Battle Controller, which handles the business logic related to the formation of teams in battles. The Battle Controller invokes the Repository to add the team to the system. The Repository then interacts with the Database Management System (DBMS) to register the new team, associating the team name, battle identifier and email addresses of the invited students with the corresponding record.
Once the DBMS confirms the addition of the team with an "OK" response, indicating that the team has been successfully created in the database, the Notification Manager is activated. The Notification Manager coordinates with the Email API to send an email notification to all invited students, informing them of their inclusion in the team and providing relevant details.
The system then confirms the creation of the team to the student via the Web Browser, thus completing the process.

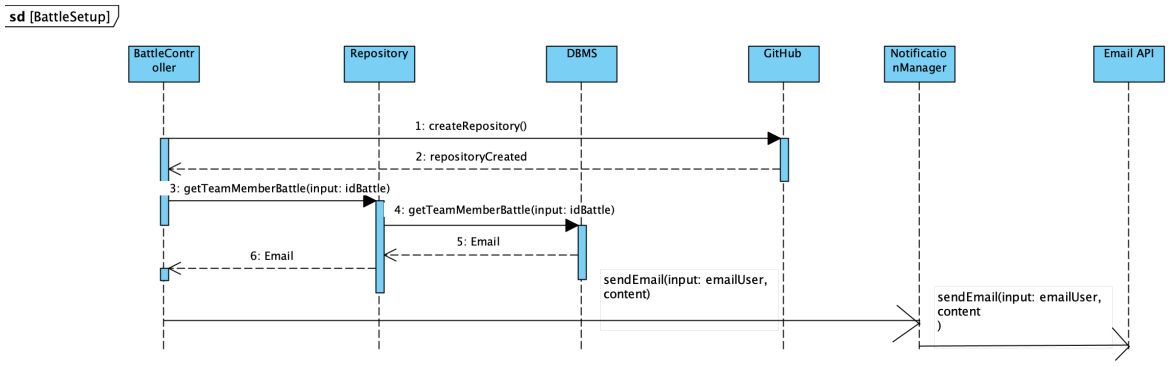
2.4.9 Student accept invitations and become part of the Team



The sequence diagram starts with the students receiving an email containing a link. When students click on the link, the Web Browser processes the request and directs them to the homepage of the platform (CKB). Once the system displays the confirmation message, students can choose to accept the invitation

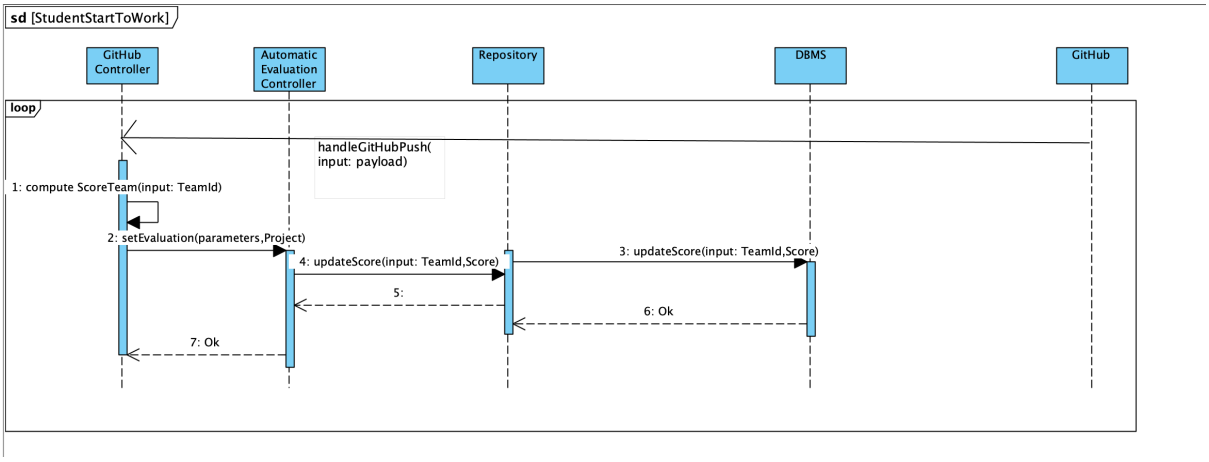
by clicking 'Yes'. At this point, the Web Server receives an indication of acceptance of the invitation and passes this information to the Request Manager. The Request Manager acts as coordinator of the data flow, ensuring that the request is correctly addressed. It then transfers the team membership request to the Battle Controller. The Battle Controller invokes the Repository to make the change in the system, adding the user to the specific team. The Repository interfaces with the DBMS to update the team record with the new members. The DBMS performs the update operation and, once the transaction is confirmed with an 'OK', the repository notifies the Battle Controller that the addition has been successfully completed. After the Battle Controller has received the confirmation, the Notification Manager is activated to handle the notification of users. The Notification Manager makes use of the Email API to send an email notification to all team members, confirming their team membership. If the student declines the invitation, only the team creator is notified.

2.4.10 Battle Setup



The sequence diagram illustrates the process of setting up the environment for a coding battle after the registration deadline. Once registration is closed, the platform proceeds with the creation of a GitHub repository to contain the code kata, i.e. the project on which the students will work. This is managed by the Battle Controller, which invokes the CreateRepository() function on the GitHub component. Once the repository has been created, the BattleController retrieves all the emails of the participants in the battle and via the NotificationManager sends an email to all students. This email contains the link to the GitHub repository and instructions for setting up an automated workflow using GitHub Actions. The Notification Manager relies on the Email API to send these details to the students.

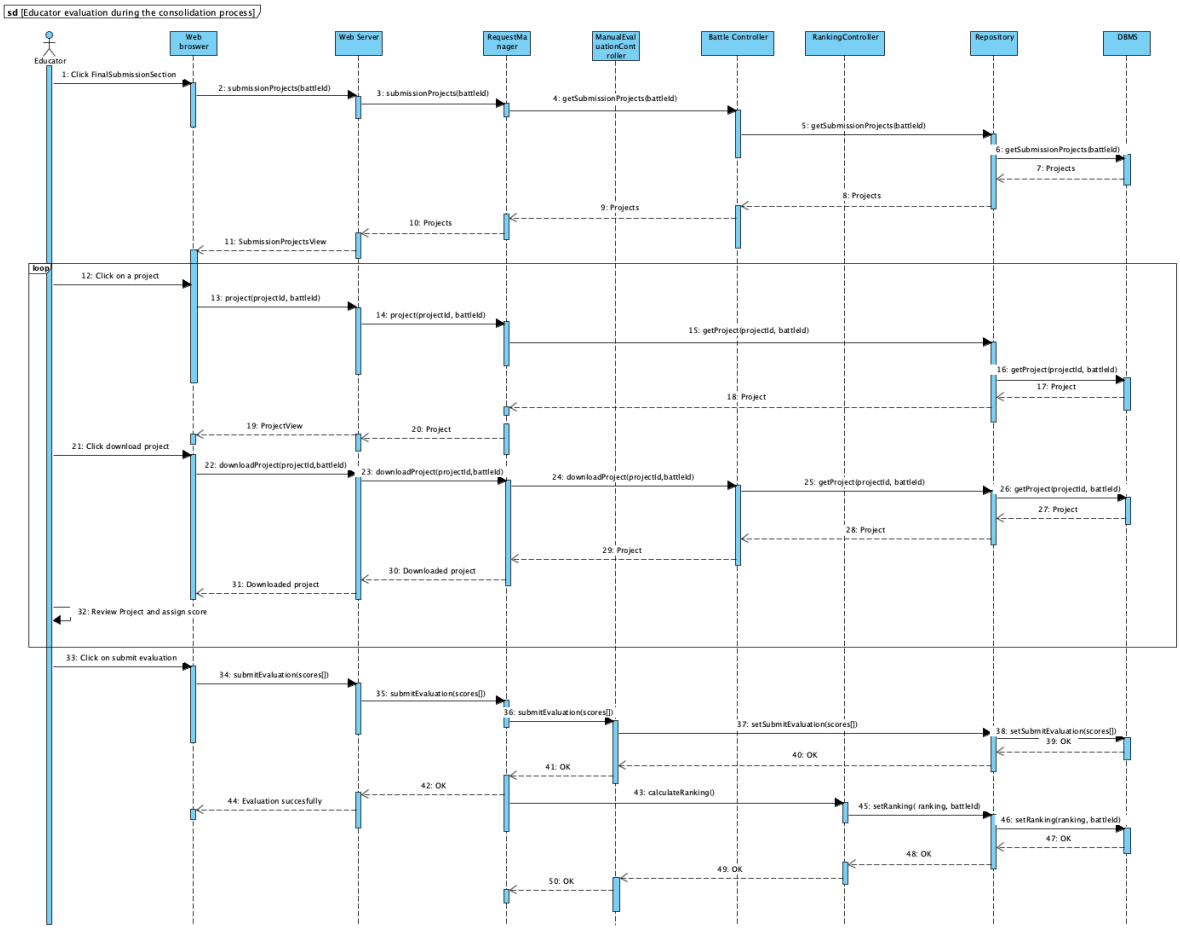
2.4.11 Student start to work



The sequence diagram describes the operations performed once the battle environment has been successfully configured and the students begin work. The students start working on the code within the

GitHub repository dedicated to the project. As they develop their work, they commit to GitHub for each significant update they want to save and share. This is a continuous and iterative process, as indicated by the 'loop' element in the diagram. When a new commit is made, the GitHub component, which was configured during the battle setup phase, sends a notification to the CKB platform. These notifications are a signal that there are new changes to be evaluated. Upon receiving notification of a new commit, the CKB system, through the GitHub Controller, compute the score and , through the Automatic Evaluation Controller, send the request to Repository to updates the DBMS with the new score. Finally, when the submission deadline is reached, the system stops monitoring further code pushes. This moment marks the end of the battle.

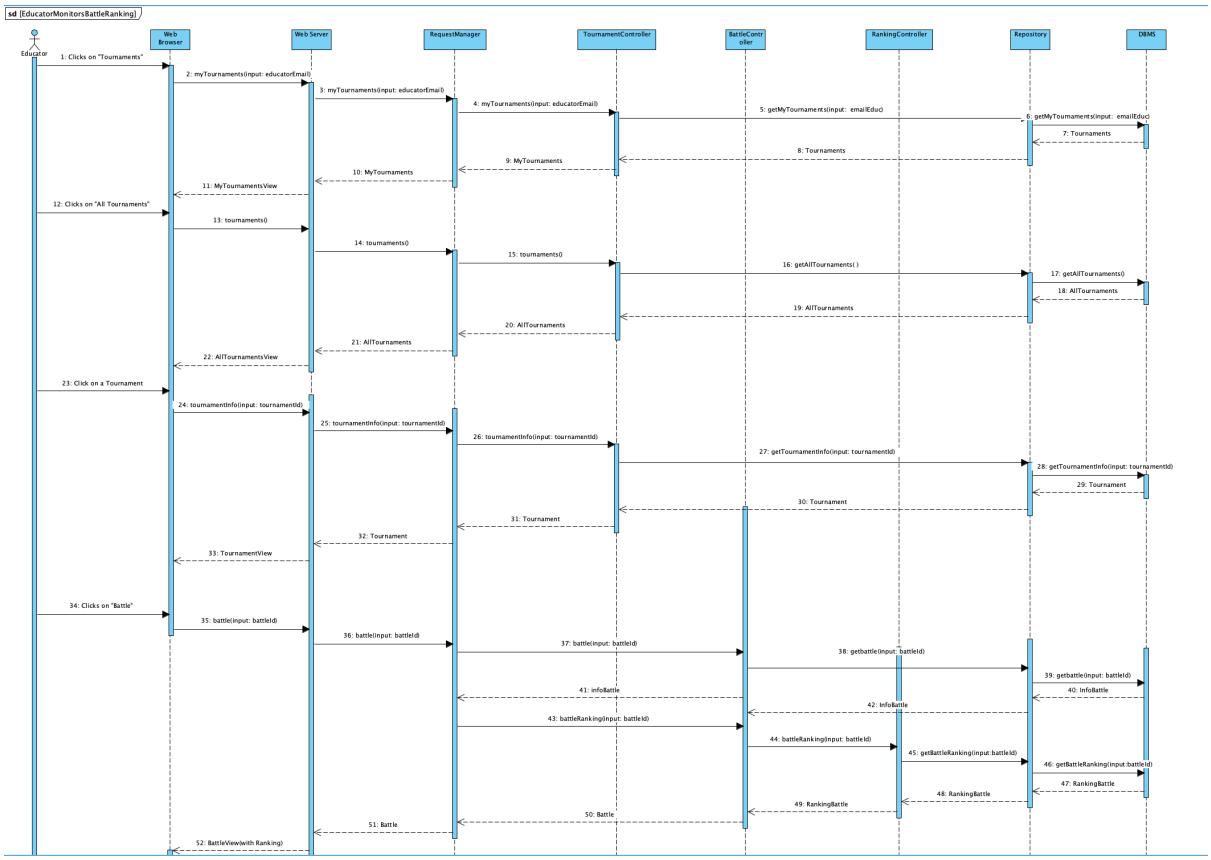
2.4.12 Educator evaluation during the consolidation process



The sequence diagram reflects the evaluation process an educator follows after the deadline for project submissions has ended in a battle. The educator accesses the platform through his or her browser and selects the 'Final Submission' section to view the students' final submissions. This action is recorded by the Web Server, which communicates with the Request Manager, which in turn interacts with the Manual Evaluation Controller. The Manual Evaluation Controller requests the Repository to retrieve projects from the Database Management System. These projects are then presented to the educator, who can click on each one to examine the details. For each project, the educator has the option of downloading the corresponding files, examining them and assigning a score based on his or her evaluation. When the educator enters all the evaluations (loops), they are sent by the educator via the web browser, to the Web Server, then to the Request Manager and finally to the Manual Evaluation Controller, which updates the database with the new scores via the Model. Once the evaluation of all projects is complete, the Ranking Controller comes into play. This component processes the final scores, combining the educator's manual evaluations with those generated through automated processes. The final results are saved in the database and, through the Notification Manager, an email notification is sent to all participating

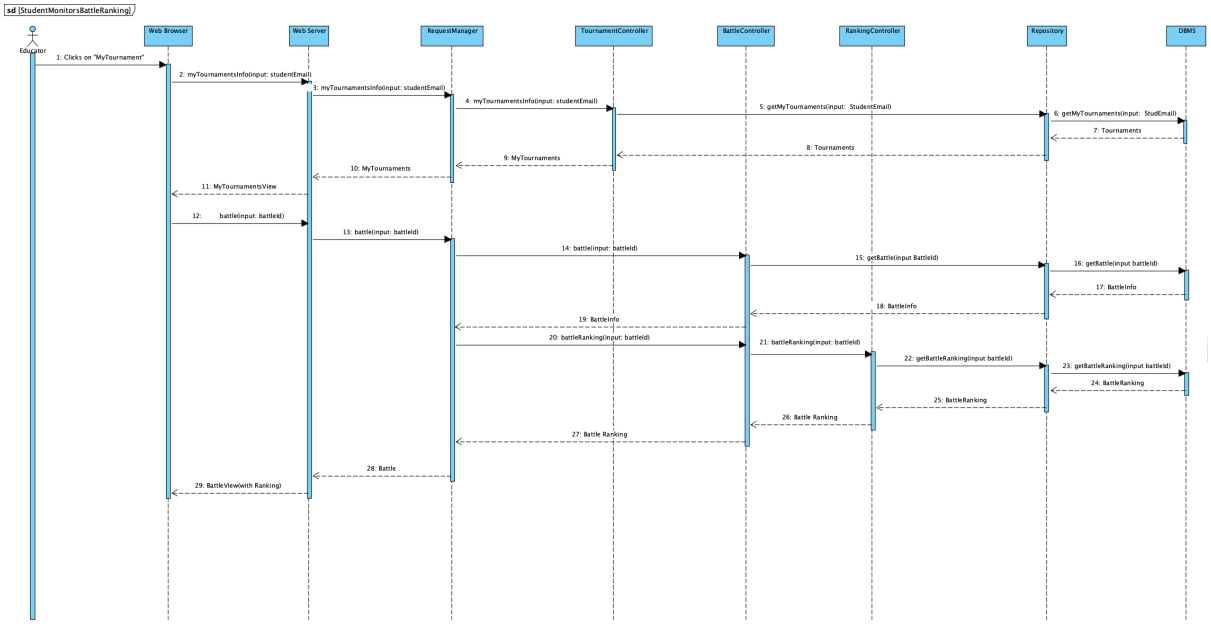
students, providing them with ranking information that is now available.

2.4.13 Educator monitors battle ranking



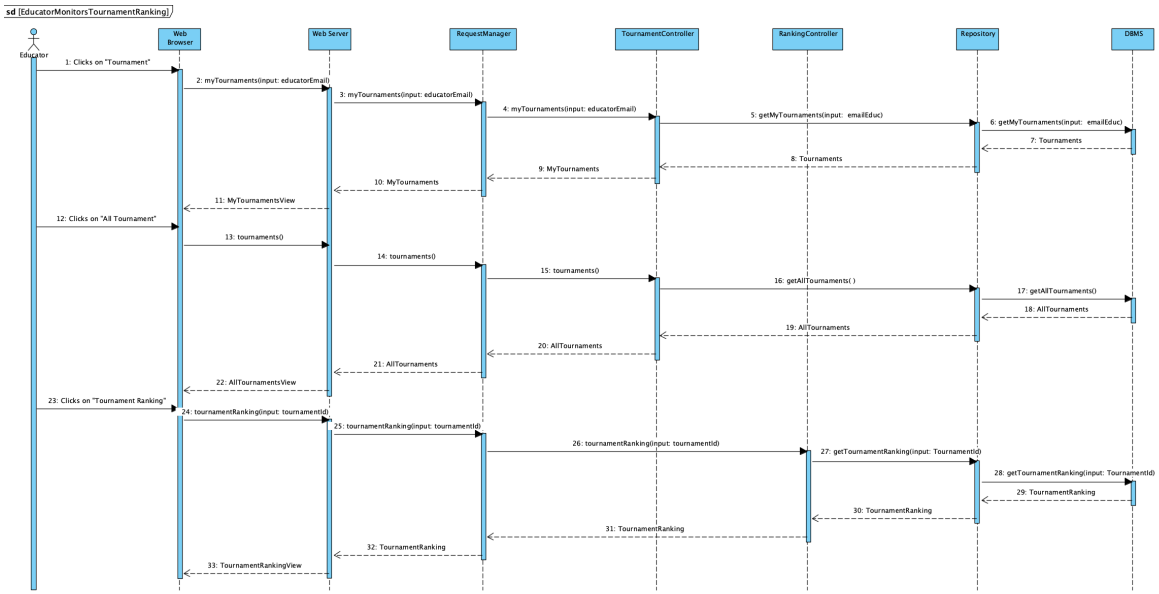
In the sequence diagram, when the educator navigates through the platform to monitor the rankings, he initiates a series of interactions between various system components. After selecting the tournament section, his web browser sends a request to the web server, which is the entry point for all incoming requests. The web server passes the request to the Request Manager, which is routed to the Tournament Controller. This controller specialises in handling tournament-related operations. When the educator chooses a tournament and then a specific battle to display the rankings, the Tournament Controller and the Battle Controller requests the Repository to retrieve the relevant information from the Database Management System (DBMS). The DBMS performs the necessary operations to extract the required data and returns it to the Repository. The Repository then passes this information to the Ranking Controller, which is responsible for managing the rankings based on the criteria defined for the tournament. The ranking controller retrieves the rankings and, via the other components, presents them to the educator.

2.4.14 Student monitors battle ranking



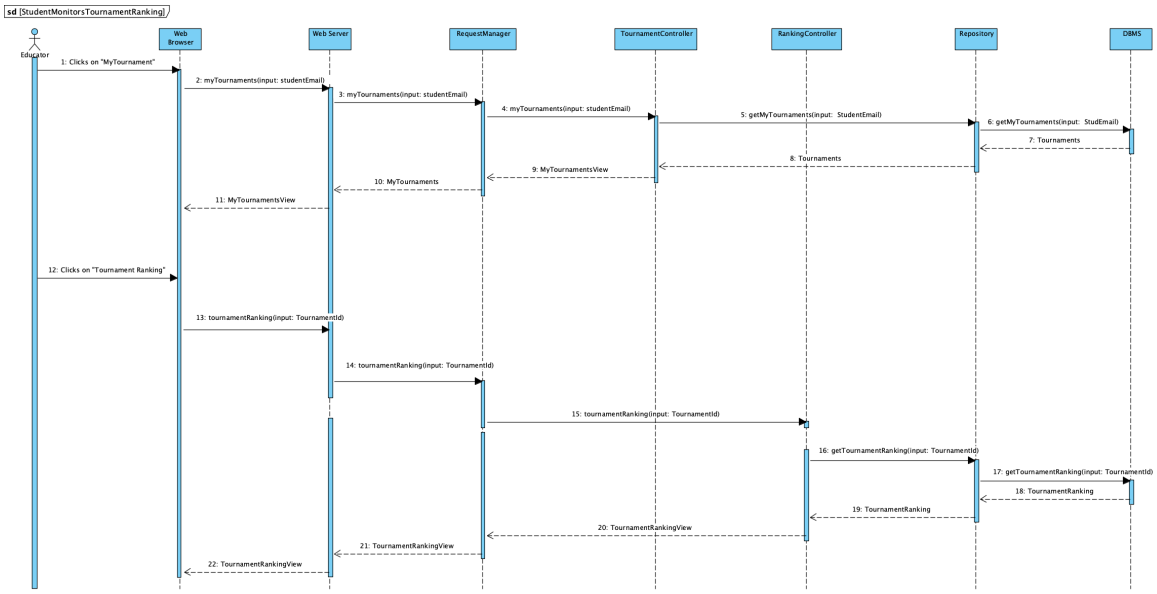
The sequence diagram illustrates how a student can access and view the ranking of a battle within a tournament via an online platform. Starting on the homepage, the student selects the "Tournaments" section, where the web server receives and forwards the request to the Request Manager. This, in turn, directs the request to the Tournament Controller. The Tournament Controller is responsible for managing the tournament information. When the student selects a tournament, the Tournament Controller requests the Repository to retrieve all information about the selected tournament from the DBMS, including the list of associated battles. The DBMS responds with the requested data, which the Repository passes to the Tournament Controller, who then presents it to the student via the platform. Next, the student selects a specific battle to view the ranking. At this point, the Battle Controller goes into action, retrieving the detailed information of the individual battle, again through the Repository and the DBMS. Once the battle information has been retrieved,through the request controller, the Ranking Controller processes the data to generate a rankings display for the student to view. Eventually, the system presents the battle rankings to the student.

2.4.15 Educator monitors tournament ranking



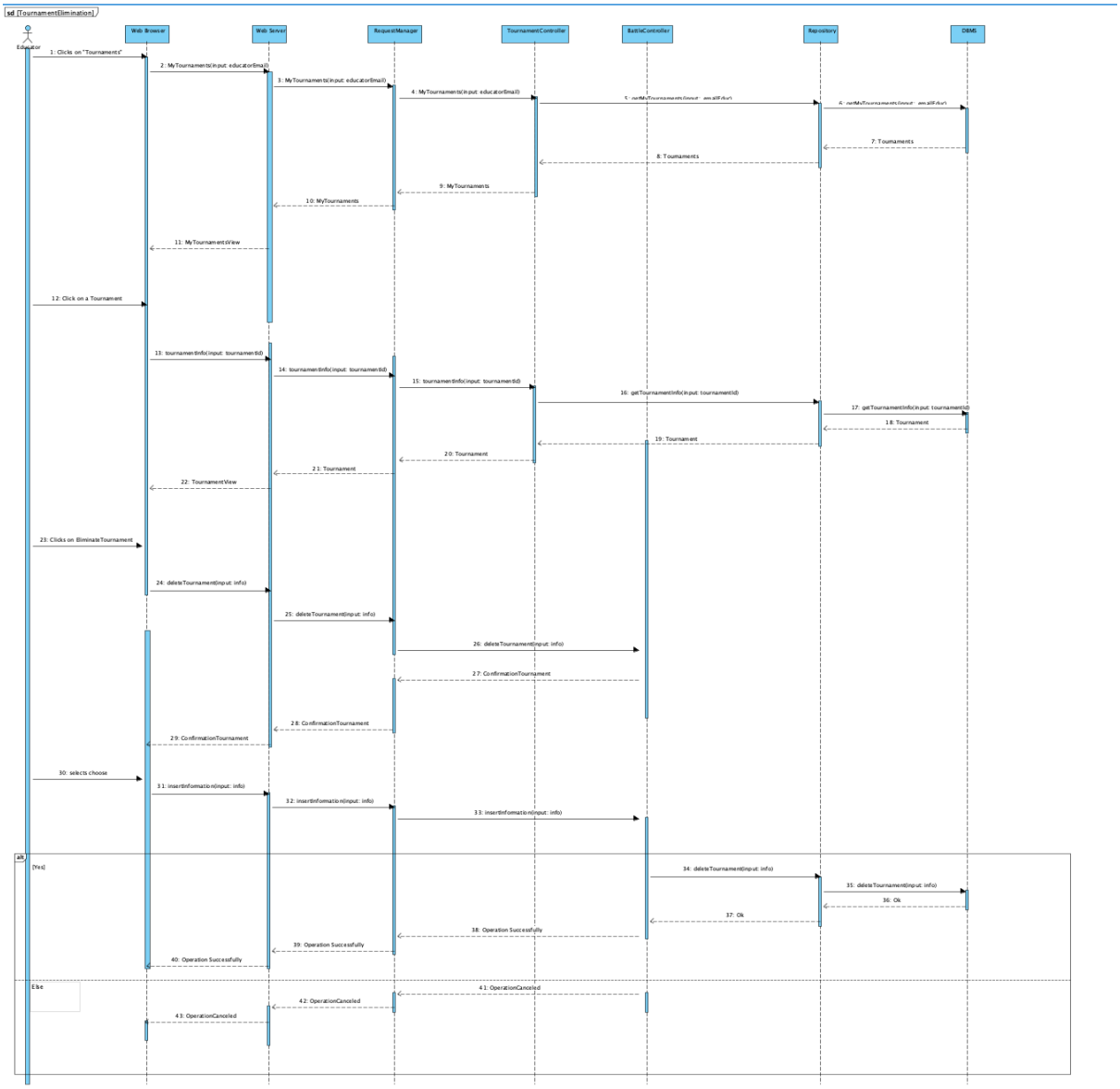
In the sequence diagram, an educator follows a series of steps on the online platform to view the ranking of a tournament. Starting from the homepage, the educator accesses the 'Tournaments' section, which shows him/her a control dashboard. This dashboard provides an overview of the tournaments the educator has created or for which he/she has permission to organise battles. After clicking on 'All Tournaments', the system displays a page listing all tournaments available on the platform, including those to which the educator cannot make changes. The educator then selects a specific tournament from the list. In response, the system displays a new dashboard detailing the battles associated with that tournament, also providing the option to view the tournament leaderboard. When the educator clicks on the "Tournament Rankings" button, the system processes a request to view the rankings for the selected tournament. This request is sent from the web browser to the web server, then passed through the Request Manager, which directs the request to the Ranking Controller. The Ranking Controller, in turn, interacts with the Repository to query the Database Management System (DBMS) for ranking data. The Repository retrieves this information and passes it back to the Ranking Controller. The latter passes the ranking view to the RequestManager which passes it to the Web Server until it reaches the Web Browser where the educator displays the tournament ranking.

2.4.16 Student monitors tournament ranking



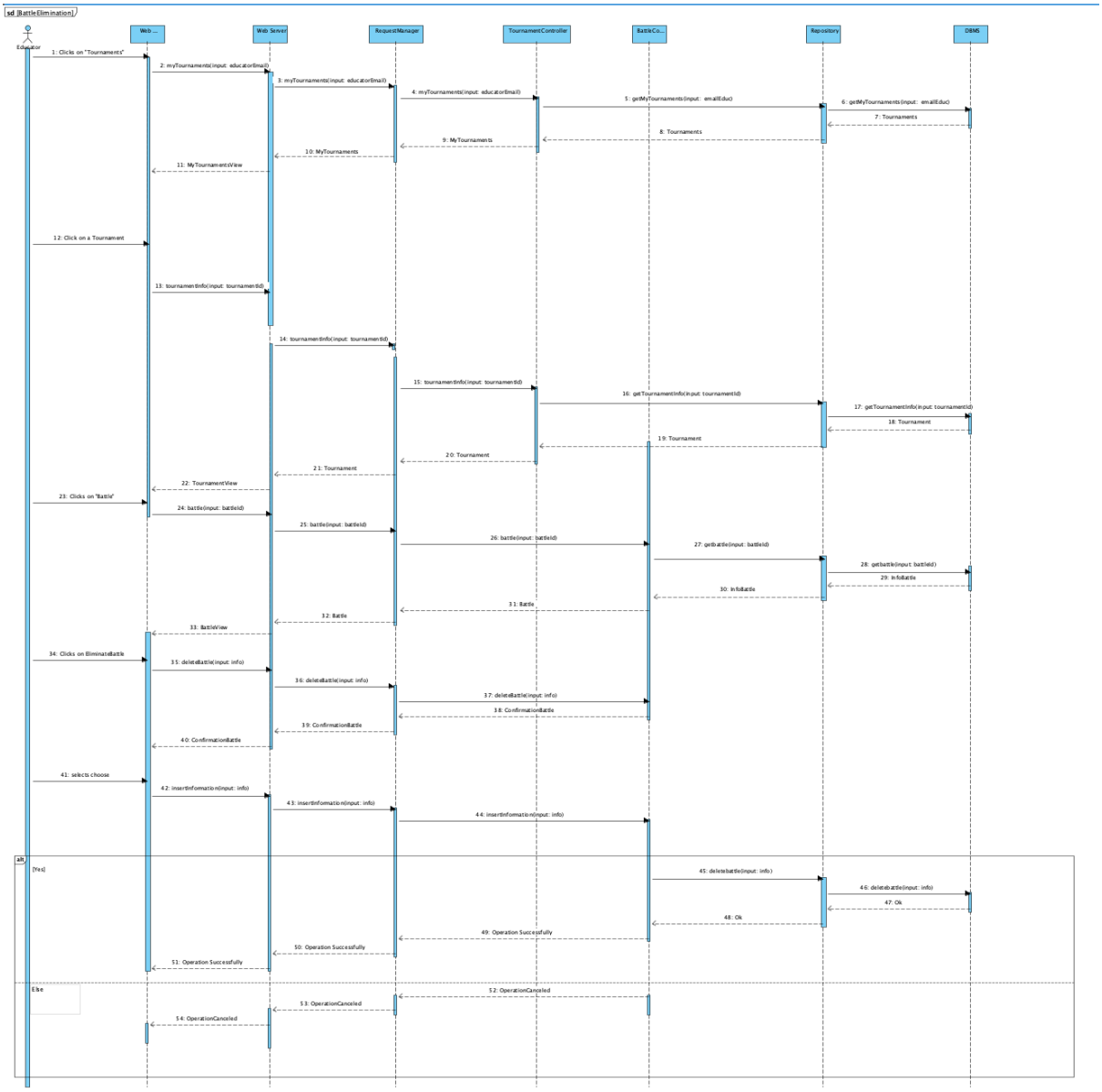
In the sequence diagram, a student follows a procedure to view the ranking of a tournament within CKB. It all starts on the homepage, where the student clicks on the "Tournaments" section, triggering a series of requests through the platform. The first request is sent from the student's web browser to the web server, which then forwards it to the Request Manager. The Request Manager, acting as the request distributor, sends the appropriate request to the Tournament Controller. The Tournament Controller is the module responsible for retrieving the list of tournaments the student is a member of or has allowed to view, interacting with the Repository to obtain this data from the Database Management System. The available tournaments are then presented to the student in a control view. When the student selects a specific tournament, the system displays a page with the battles for that tournament and provides a button to view the tournament standings. By clicking on the "Tournament Ranking" button, the system, through the same backend interactions, requests the Ranking Controller to retrieve the rankings for the selected tournament. The Ranking Controller, using the Repository and interacting with the DBMS, obtains the current rankings. This information is then passed back through the system and displayed to the student.

2.4.17 Tournament Closure



In this scenario, the educator takes a series of actions on the platform to close a tournament. After selecting the 'Tournaments' section from the homepage, a control dashboard is displayed through his web browser. This dashboard is the result of a series of interactions between the browser, the web server, and the Request Manager, which coordinates incoming requests. The Request Manager forwards the tournament display request to the Tournament Controller. The Tournament Controller then interacts with the Repository to extract the tournament data from the Database Management System (DBMS). When the educator chooses to close a tournament by clicking on the "Close Tournament" button, the system displays a confirmation message to ensure that the intention is deliberate and not an accidental action. If the educator confirms the closure, the system performs a check via the Tournament Controller to see if there are any battles still open in the tournament. If there are no open battles, the Tournament Controller sends a command to the Repository to change the tournament status from 'Open' to 'Closed' in the DBMS. The Repository updates the DBMS with the new status and, once the DBMS confirms that the operation has been successfully performed, the system informs the educator that the closing of the tournament has been successfully completed. This feedback is provided directly through the user interface in the educator's dashboard. 24

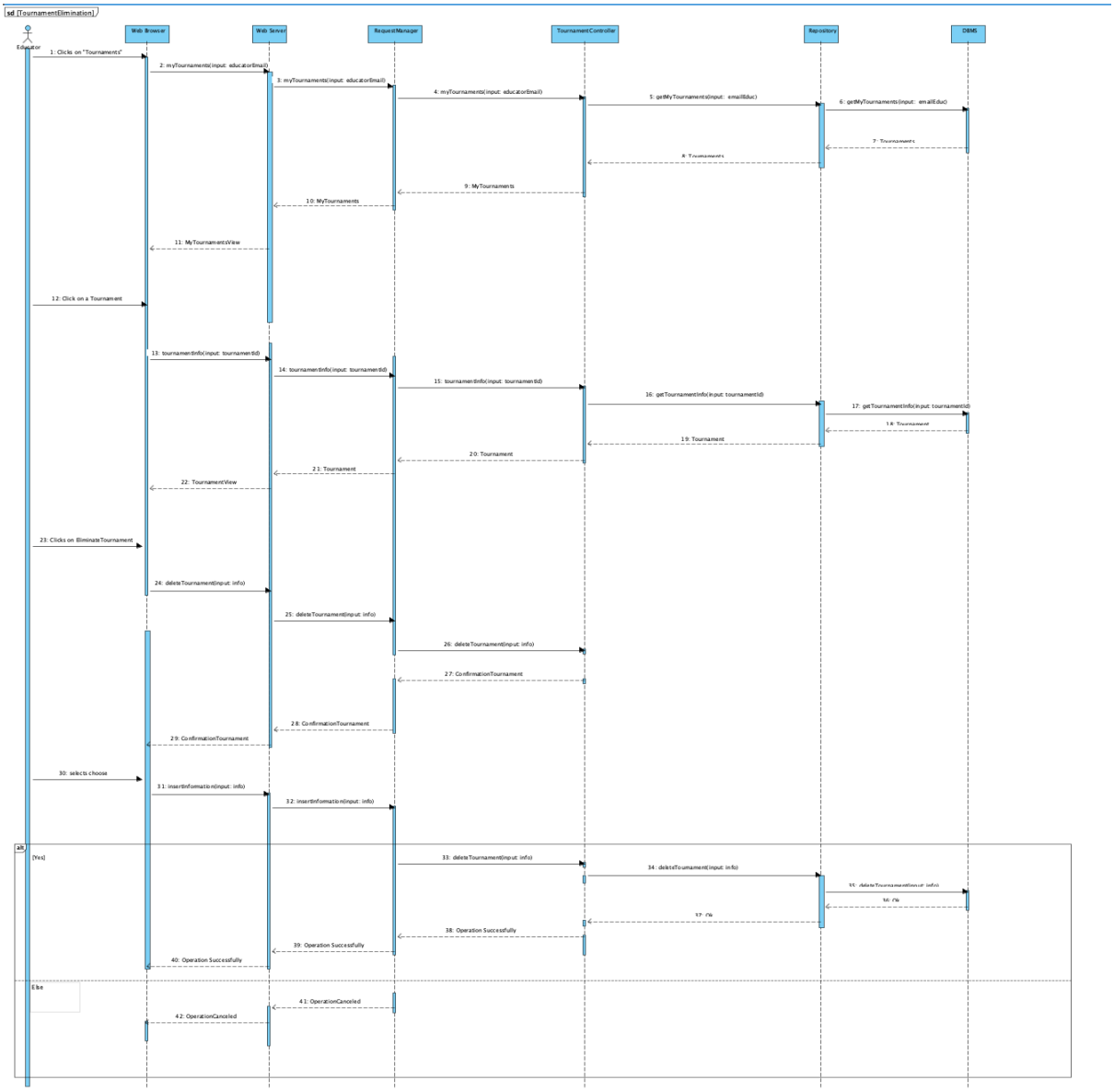
2.4.18 Battle elimination



In the sequence diagram, the educator begins the process of eliminating a battle within a tournament. Once the "Tournaments" section has been clicked, the web server processes the request and via the Request Manager directs it to the Tournament Controller, who retrieves the list of available tournaments from the interaction with the Repository and the Database Management System (DBMS). By selecting a specific tournament, the educator accesses the tournament dashboard where the various battles are listed. The system, again through the Tournament Controller, shows the detailed page of the selected tournament and presents the option to "Delete the battle". The latter allows the educator to proceed with the elimination. When the educator chooses to delete a battle, the system presents a confirmation message to ensure that they wish to proceed. This confirmation is processed once again by the web server and the Request Manager, which send the request to the Battle Controller. After the educator's confirmation, the Battle Controller checks the battle status. If the battle can be deleted, the Battle Controller proceeds with the deletion request to the DBMS. The DBMS performs the deletion and sends a confirmation response to the Repository, which in turn informs the Battle Controller of the deletion. Ultimately, the educator receives visual feedback via the web browser displaying the message "Elimination Completed Successfully," indicating that the battle has been removed from the database and that the tournament has been updated to reflect this change. If the educator clicks no, the operation

is canceled.

2.4.19 Tournament Elimination



In the sequence diagram, the educator uses the online platform to cancel an existing tournament. Start from the homepage, where you select the "Tournaments" section and are presented with a control dashboard listing all the tournaments available to you. This view is the result of a series of requests managed by the web server, which interacts with the Request Manager which forwards the request to the TournamentController which retrieves the data from the DBMS via the Repository. Once a specific tournament is chosen, the system displays a detailed page for that tournament via the Tournament Controller, which acts as an intermediary between the educator's request and the tournament information held in the system. The dashboard also includes a "Delete Tournament" button, which the educator can select to begin the cancellation process. When the educator clicks "Delete Tournament", a warning message appears. By confirming your choice, the system triggers a check through the Tournament Controller to ensure that there are no battles still ongoing within the tournament. If there are no obstacles, the system proceeds with the elimination of the tournament. This step involves the Repository, which interacts with the Database Management System (DBMS) to update the tournament status. The DBMS performs the tournament cancellation and reports the completed operation to the Repository. Finally, the system

confirms the elimination of the tournament to the educator, displaying a confirmation message on the web browser. This signals that all changes have been successfully made in the database and that the tournament has been permanently removed from the platform.

2.5 Component interfaces

- **Authentication Controller**: This component offer, through interface VisitorInterface this functions:
  1. registration/login()
  2. login(input: email,password)
  3. signup(input: email,password)
- **BattleManagementController** : This component offer, through interface StudentInterface this functions:
  - battle(input: battleId)
  - createTeam(input: emailCreator, battleId,name)
  - joinTeam(input: teamId, battleId, emailUser)
  - Confirm invitation(input: code,studentEmail)
- **BattleManagementController**: This component offer, through interface EducatorInterface this functions:
  1. battle(input: battleId)
  2. getSubmissionProjects(input: battleId)
  3. createBattle(input: Battle)
  4. deleteBattle(input: battleId)
  5. updateBattle(input: battleId, emailEducator)
  6. getSubmissionProjects(input: battleId)
- **TournamentController**: This component offer, through interface StudentInterface this functions:
  1. myTournaments(input:studentEmail)
  2. tournaments()
  3. tournamentInfo(input: tournamentId)
  4. joinTournament(input: tournamentId, emailStudent)
- **TournamentController**: This component offer, through interface EducatorInterface this functions:
  1. myTournaments(input: educatorEmail)
  2. tournaments()
  3. tournamentInfo(input: tournamentId)
  4. createTournament(input: tournamentInfo)
  5. deleteTournament(input: tournamentId, emailEducator)
  6. closureTournament(input: tournamentId, emailEducator)
  7. downloadProject(input: projectId)
- **RankingController**: This component offer, through interfaces StudentInterface and EducatorInterface this functions:
  1. battleRanking(input: battleId)
  2. tournamentRanking(input: tournamentId)
  3. calculateRanking()
- **ManualEvaluationController**: This component offer, through interfaces EducatorInterface this functions:
  1. submitEvaluation(scores[])

- **AutomaticEvaluationController**: This component offer these functions:
  1. setEvaluation(input: parameters, Project)
- **GitHubController**: This component offer these functions:
  1. handleGitHubPush(input: payload)
- **StudentInfoController**: This component offer these functions:
  1. getStudentInfo(input: email)
  2. getAllStudents()
- **EducatorInfoController**: This component offer these functions:
  1. getEducatorInfo(input: email)
  2. getAllEducators()
- **NotificationManager**:This component offer these functions:
  1. sendEmail(input: emailUser, content)
- **Repository**:
  1. addInfoTournament(input: tournamentInfo, emailEducator)
  2. getMyTournaments(input: emailEducator)
  3. getTournamentInfo(input: tournamentId)
  4. addInfoBattle(input: info)
  5. getTournaments()
  6. getTeamMemberBattle(input: idBattle)
  7. joinTournament(input: tournamentId,emailStudent)
  8. getBattle(input: battleId)
  9. addToTeam(input:teamId, battleId, emailUser)
  10. updateScore(input: teamId,score)
  11. getSubmissionProjects(input: battleId)
  12. getProject(input: projectId, battleId)
  13. setSubmitEvaluation(input: scores[])
  14. insertRanking(input:ranking, battleId)
  15. insertAccount(input: userInfo)
  16. getBattleRanking(input: battleId)
  17. getTournamentRanking(input: tournamentId)
  18. closeTournament(input: tournamentId, emailEducator)
  19. getAllTournaments()
  20. deleteBattle(input: battleId, emailEducator, tournamentId)
  21. deleteTournament(input: tournamentId, emailEducator)
  22. checkCredentials(input: userInfo)
- **WebServer**:
  1. registration/login()
  2. login(input: email,password)
  3. signup(input: email,password)
  4. battle(input: battleId)
  5. createTeam(input: emailCreator, battleId,name)
  6. joinTeam(input: teamId, battleId, emailUser)
  7. Confirm invitation(input: code,studentEmail)
  8. getSubmissionProjects(input: battleId)

9. createBattle(input: Battle)
10. deleteBattle(input: battleId)
11. updateBattle(input: battleId, emailEducator)
12. getSubmissionProjects(input: battleId)
13. myTournaments(input:studentEmail)
14. tournaments()
15. tournamentInfo(input: tournamentId)
16. joinTournament(input: tournamentId, emailStudent)
17. createTournament(input: tournamentInfo)
18. deleteTournament(input: tournamentId, emailEducator)
19. closureTournament(input: tournamentId, emailEducator)
20. downloadProject(input: projectId)
21. battleRanking(input: battleId)
22. tournamentRanking(input: tournamentId)
23. calculateRanking()
24. submitEvaluation(scores[])
25. setEvaluation(input: parameters, Project)
26. handleGitHubPush(input: payload)
27. getStudentInfo(input: email)
28. getAllStudents()
29. getEducatorInfo(input: email)
30. getAllEducators()
31. sendEmail(input: emailUser, content)

2.6 Architectural Styles and Patterns

2.6.1 Four-Tier System Architecture

This system has a 4-tier architecture. The benefits of this choice are:

- **Scalability**: With components logically divided, it's easier to scale the system. If, for example, the load on the web server increases, you can add more web servers without modifying the other tiers.
- **Flexibility**: Each tier can be modified, updated, or replaced independently of the others.

2.6.2 RESTful Architecture

The choice of a RESTful architecture for communication with the application server offers several advantages:

- **Simplicity and Standardization**: REST uses standard HTTP methods, which are understood by developers and supported by virtually all web infrastructure.
- **Scalability**: The stateless nature of the server leads to simplified upkeep, allowing for the addition or migration of a server without altering any data.
- **Independence**: The client and the server are independent; the client doesn't need to know anything about the business logic, and the server doesn't need to know anything about the UI. This separation allows for the client and the server to evolve independently.

2.6.3 Model View Controller

Model-View-Controller (MVC) architecture was chosen for the development of your web application. It includes:

- **Model**: This part manages the data and the rules of the application, offering methods to manipulate these data.
- **View**: It is responsible for the visual presentation of the data. It can exist in multiple forms, allowing different representations of the information according to the user's needs.
- **Controller**: Functions as a mediator between Model and View. It responds to events, such as user interaction with the interface (for example, pressing a button), processing data in the Model that will then be reflected in the View.

2.7 Other design decisions

In this part, we discuss various design choices implemented in the system to ensure its optimal functioning.

2.7.1 Availability

We have added load balancing and replication to make the system more reliable and to keep it running smoothly. This means if one part has a problem, the system can still work because it has backups and shares the work. This helps to make sure the system can handle data safely and stay available for use without interruption.

3 User Interface Desgin

In this section we will present mockups of our application.

3.1 Mockup for Visitor User

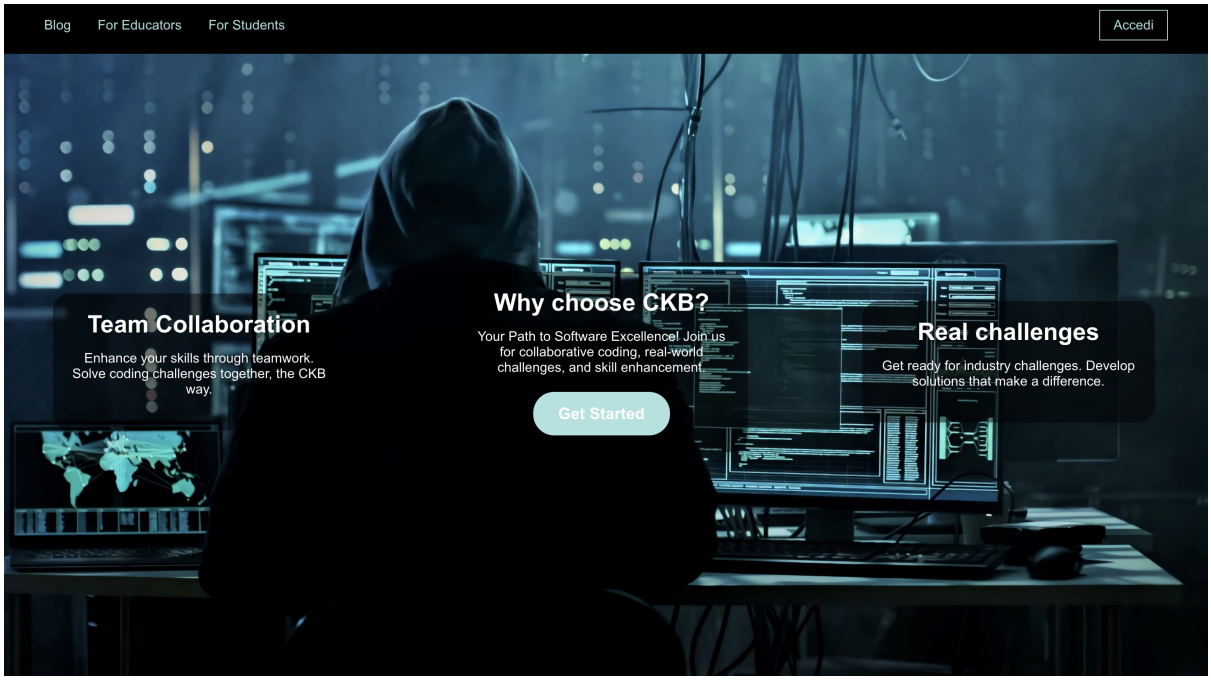


Figure 7: Mockup homepage

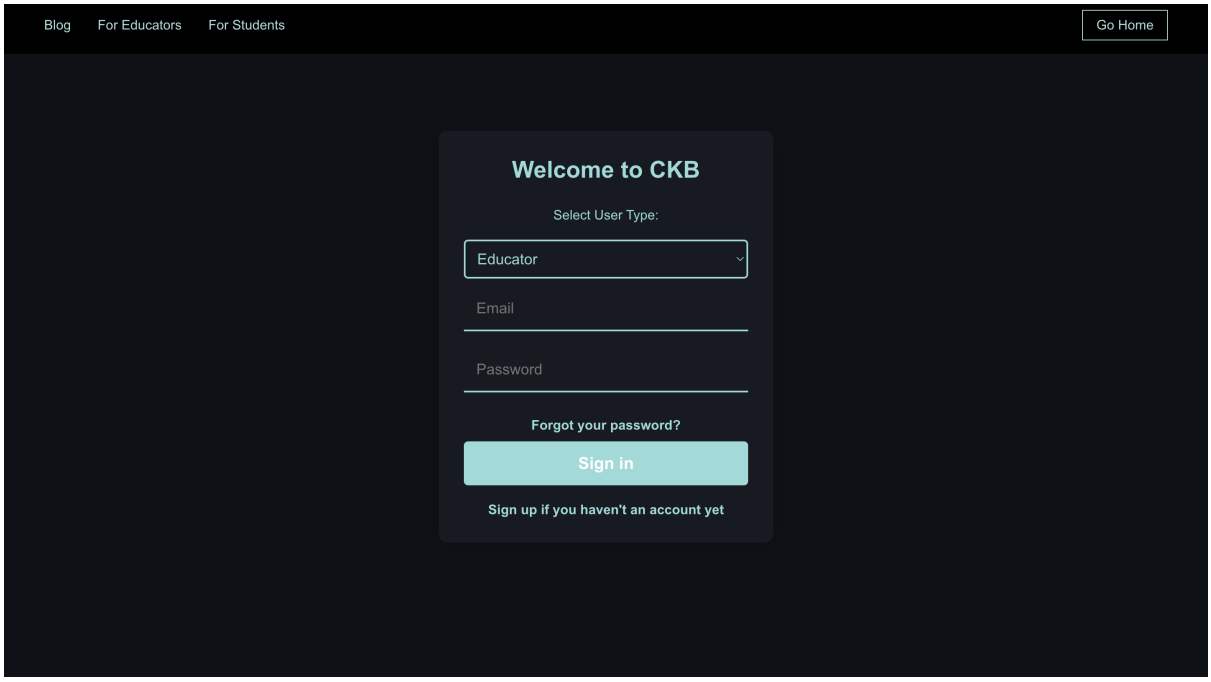


Figure 8: Mockup login page

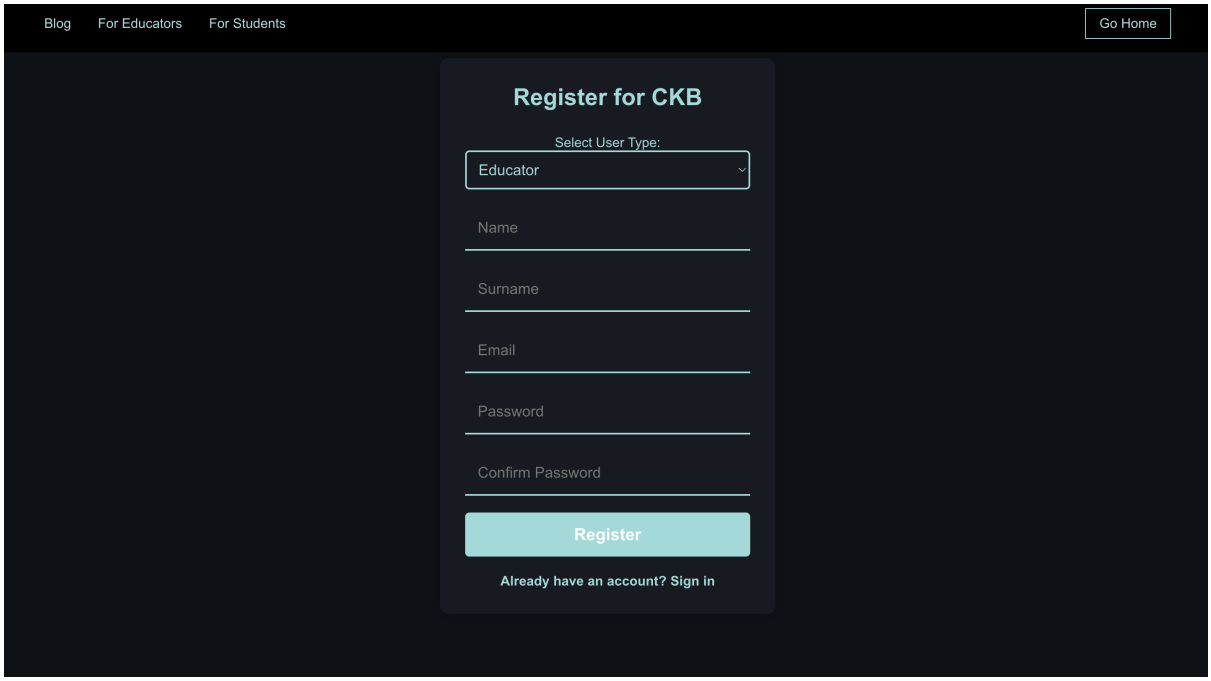


Figure 9: Mockup registration page

3.2 Mockup for Student User

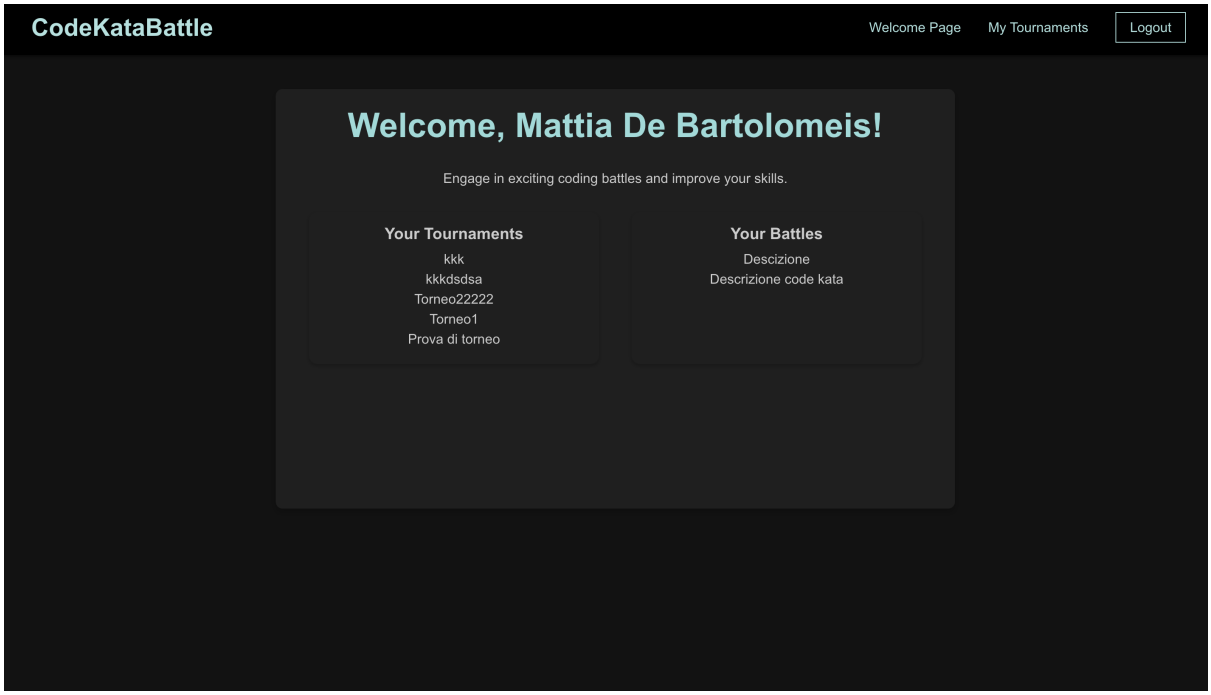


Figure 10: Mockup welcome page

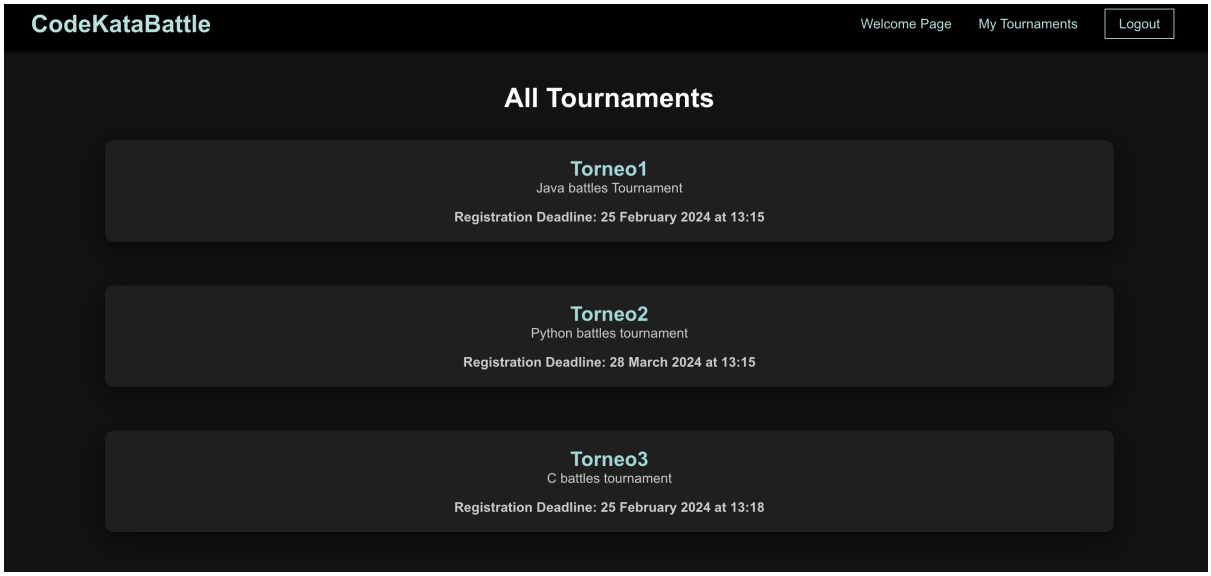


Figure 11: Mockup MyTournaments page

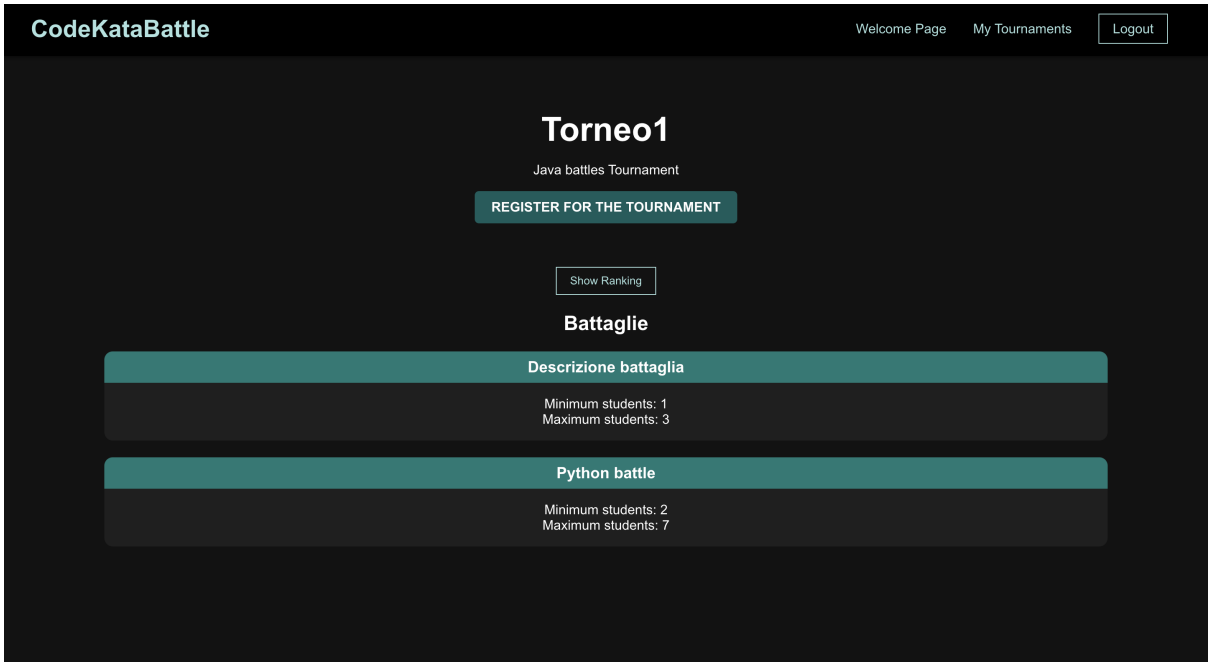


Figure 12: Mockup TournamentDetails page

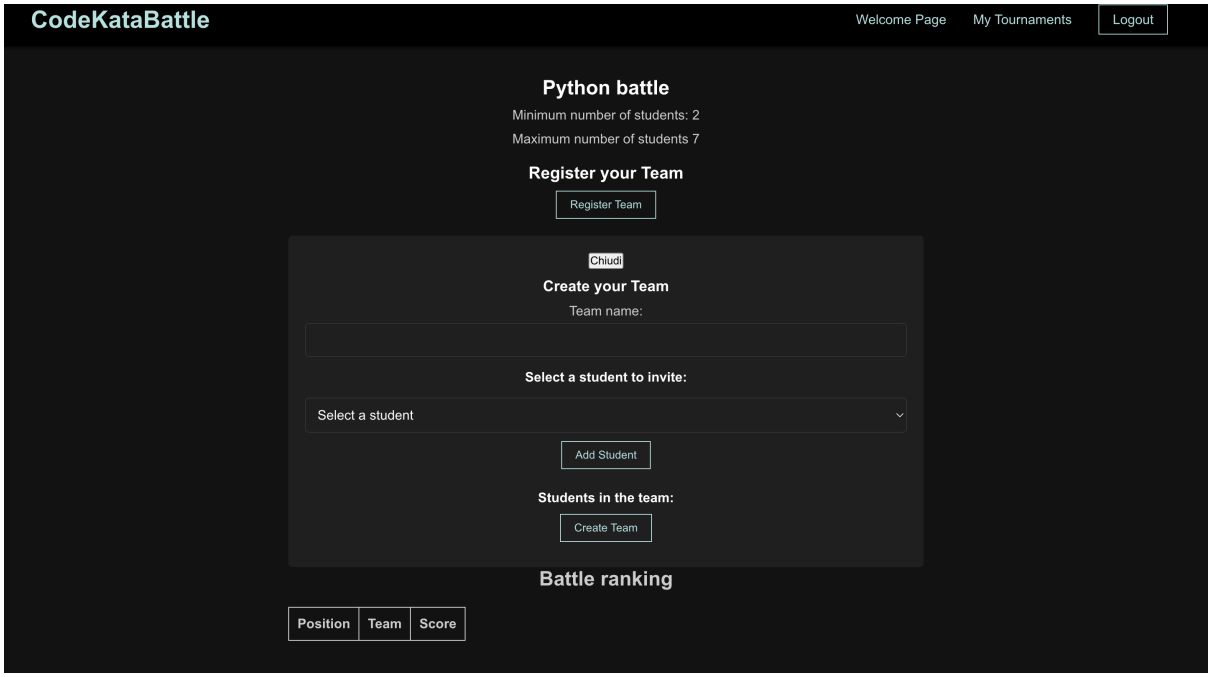


Figure 13: Mockup BattleDetails page

3.3 Mockup for Educator User

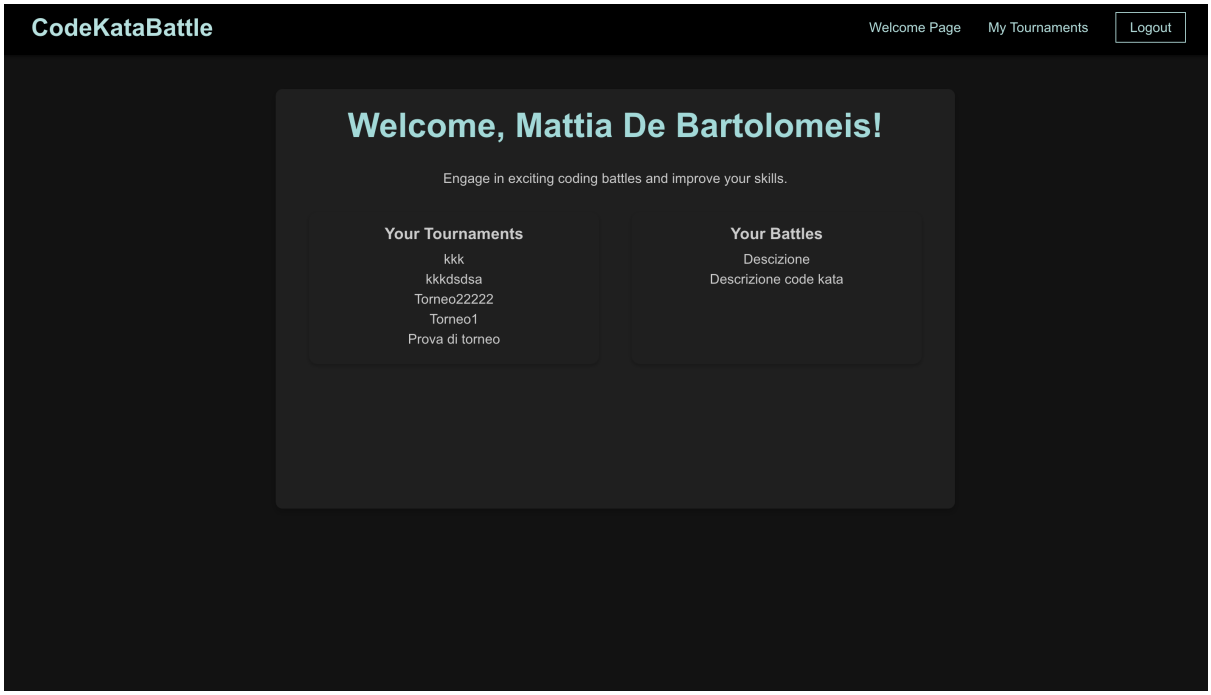


Figure 14: Mockup Welcome page

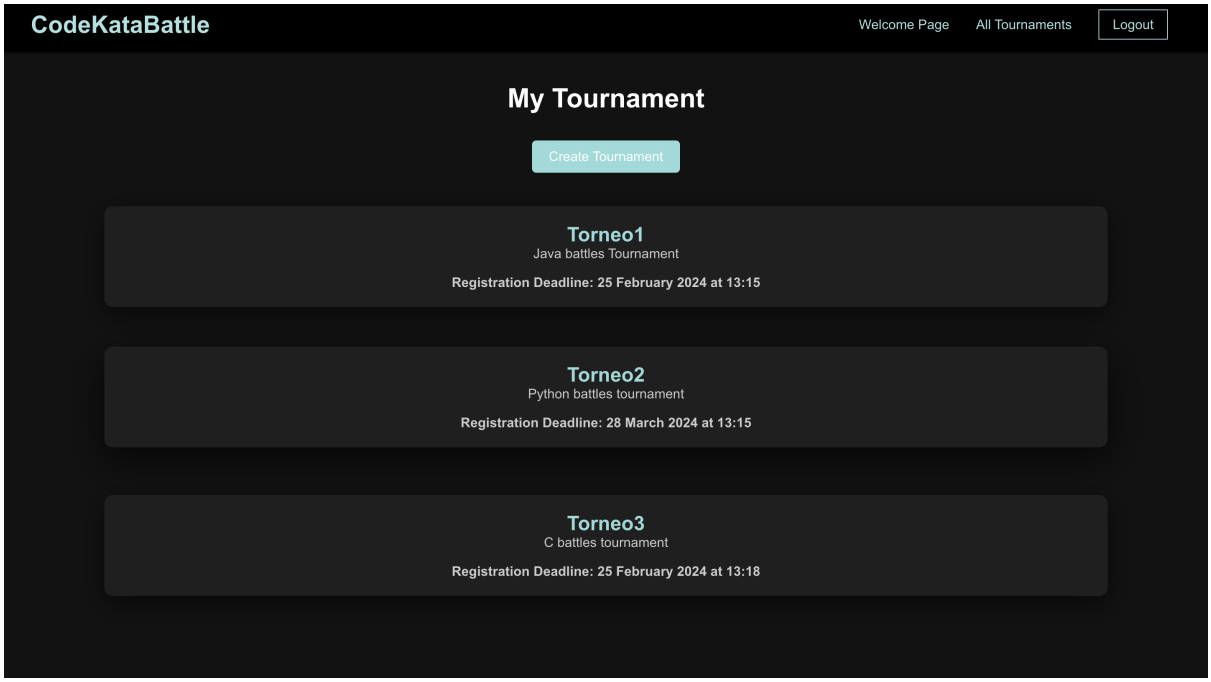


Figure 15: Mockup MyTournaments page

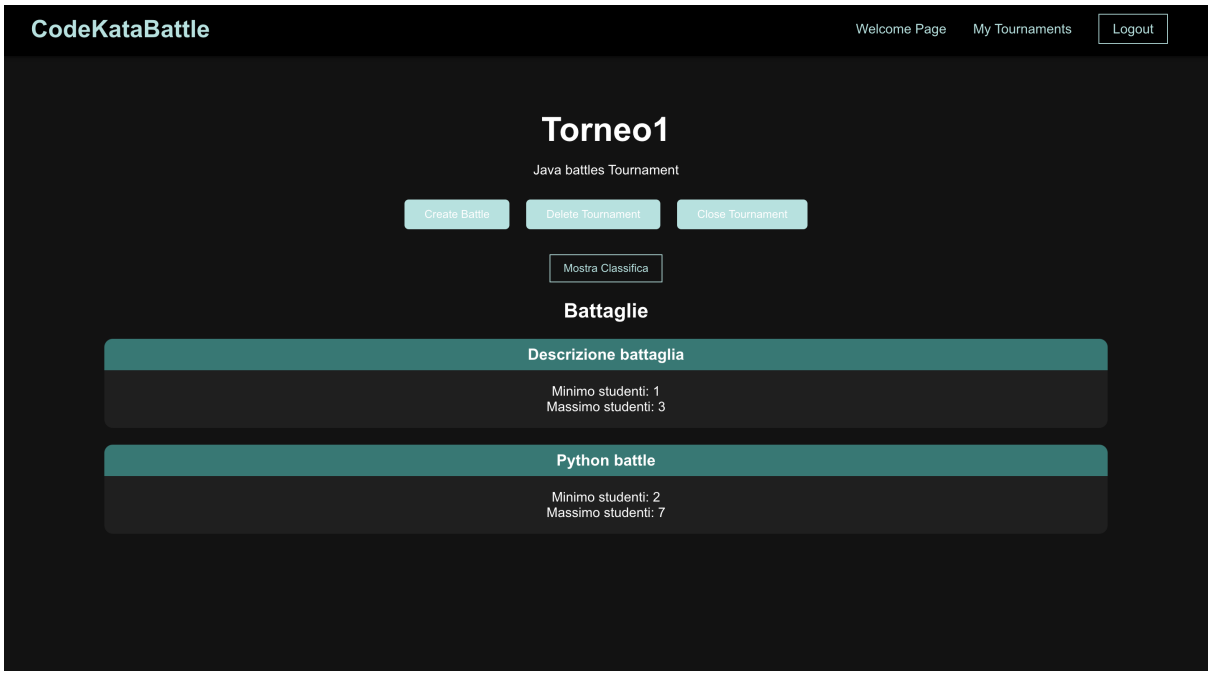


Figure 16: Mockup TournamentDetails page

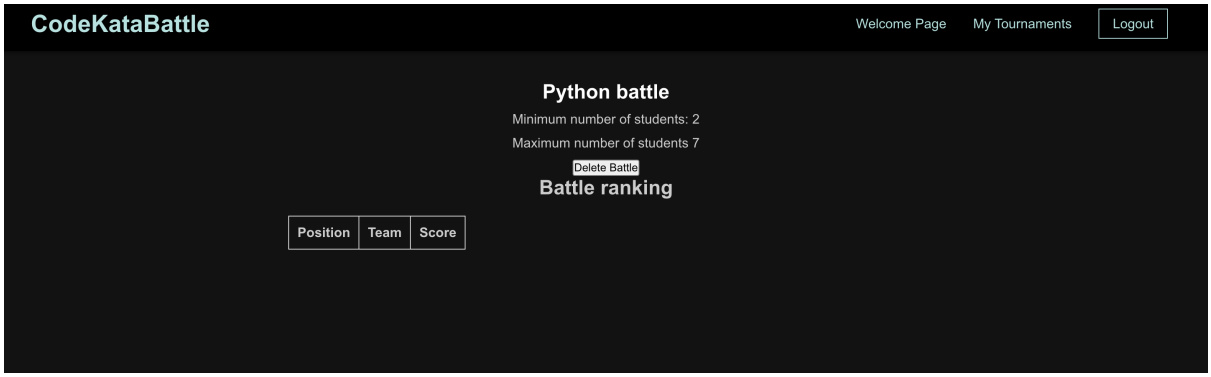


Figure 17: Mockup BattleDetails page

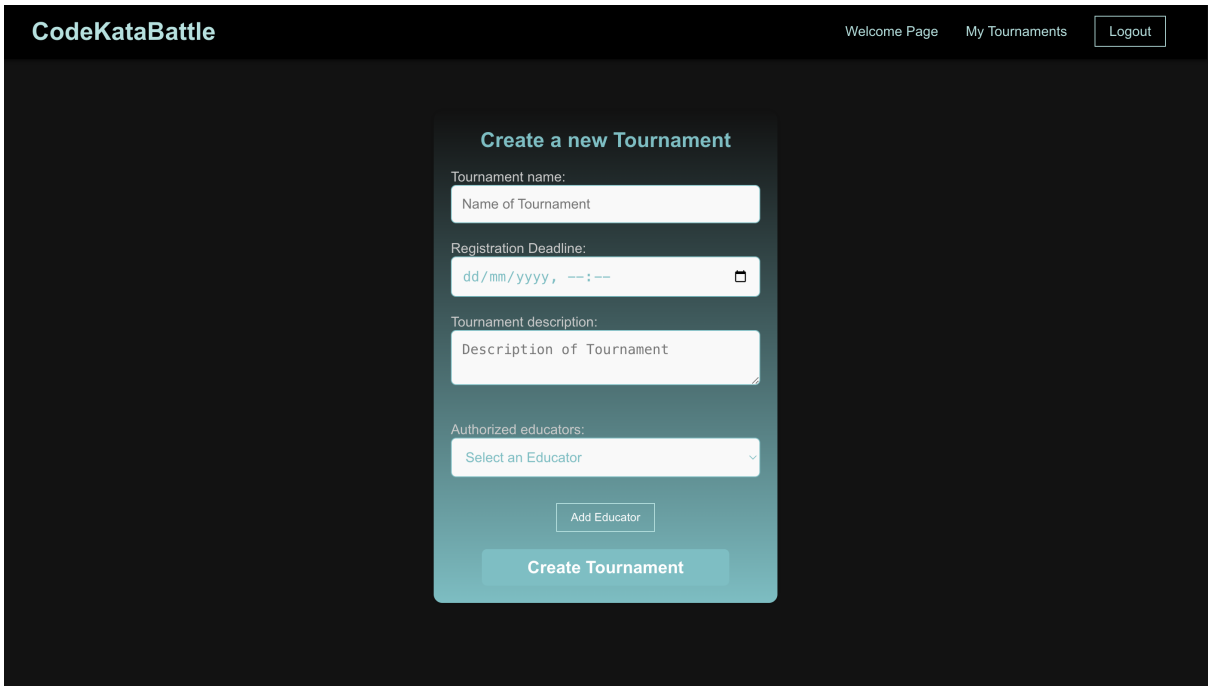


Figure 18: Mockup TournamentCreation page

4 Requirement Traceability

This section of the document outlines how each requirement specified in the RASD is associated with element of the component diagram.

- **R1**:The system must allow users to register on the platform as educator or student.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - Authentication Controller
    - NotificationManager
  - *Email API*
  - *Repository*
  - *DBMS*
- **R2**: The system must allow users to login.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - Authentication Controller
  - *Repository*
  - *DBMS*
- **R3**: The system must enable educators who have created a tournament to grant permissions to other educator to create battles within that specific tournament.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Repository*
  - *DBMS*
- **R4**: The system must allow the educator to see which tournaments they have permission for.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Repository*
  - *DBMS*
- **R5**: The system must allow educator to insert information about a tournament.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Repository*
  - *DBMS*

- **R6**: The system must allow educator to insert information about a battle for the tournament in which he has the permissions.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
      - BattleController
  - *Repository*
  - *DBMS*
- **R7**: The system must notify via mail subscribed students to the platform upon the creation of new tournaments
  - *Application server:*
    - *NotificationManager*
  - *Email API*
- **R8**: The system must allow students to subscribe to tournament on the CKB platform before a specific deadline.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Repository*
  - *DBMS*
- **R9**: The system must notify via mail all students subscribed in the tournament whenever a new battle is created within that tournament
  - *Application server:*
    - *NotificationManager*
  - *Email API*
- **R10**: The system must allow students to subscribe in the battle
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - BattleController
  - *Repository*
  - *DBMS*
- **R11**: The system should enable students to invite other students who do not yet have a team to create a team.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - BattleController
  - *Repository*

  - *DBMS*
- **R12**: The system must notify via mail all students who are invited to form a team.
  - *Application server:*
    - *NotificationManager*
  - *Email API*
- **R13**: The system must allow students to accept an invitation via email to join in a team.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - BattleController
  - *Repository*
  - *DBMS*
- **R14**: The system must allow student to decline the invitation via email to join in a team.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - BattleController
  - *Repository*
  - *DBMS*
- **R15**: The system must notify via mail the team's creator when an invited student declines his invitation.
  - *Application server:*
    - *NotificationManager*
  - *Email API*
- **R16**: The system must notify via mail all members of a team when a new student joins their team.
  - *Application server:*
    - *NotificationManager*
  - *Email API*
- **R17**: The system must allow user to see the list of available tournaments.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Repository*
  - *DBMS*
- **R18**: The system must allow user to see the list of available battles.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController

  - *Repository*
  - *DBMS*
- **R19**: The system must integrate with GitHub for repository management and automate the process of code submission and evaluation
  - *Application server*
    - RequestManager
      - BattleController
      - GitHub Controller
    - AutomaticEvaluationManager
  - *Repository*
  - *DBMS*
  - *GitHub API*
- **R20**: The system must allow educators who have set manually evaluation to see the submitted projects
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager
  - *Repository*
  - *DBMS*
- **R21**:The system must allow educators who have set manually evaluation to upload a manually score for each project.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - ManualEvaluationController
  - *Repository*
  - *DBMS*
- **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
  - *Application server:*
    - *NotificationManager*
  - *Email API*
- **R24**: The system must automatically pull and analyze the latest sources from student repositories upon each commit.
  - *Application server*
    - RequestManager
      - BattleController
      - GitHub Controller
      - AutomaticEvaluationManager
  - *Repository*
  - *DBMS*
  - *GitHub API*
- **R25**:The system must support automated test execution and static analysis of submitted code.

  - *Application server:*
    - RequestManager
      - AutomaticEvaluationManager
  - *GitHub*
- **R26**: The system must calculate battle scores based on functional aspects, timeliness, and quality level of sources.
  - *WebServer*
  - *Application server:*
    - RequestManager
      - AutomaticEvaluationManager
  - *GitHubActions*
- **R27**: The system must update battle scores in real-time as students push new commits.
  - *Application server:*
    - RequestManager
      - AutomaticEvaluationManager
      - GitHubController
  - *GitHub API*
  - *Repository*
  - *DBMS*
- **R28**: The system must consolidate and finalize battle scores after the submission deadline and provide a final battle ranking.
  - *Application server:*
    - *RequestManager*
      - RankingController
      - ManualEvaluationController
      - AutomaticEvaluationManager
  - *Repository*
  - *DBMS*
- **R29**:The system must notify via mail all students involved in a battle when final battle rank becomes available.
  - *Application server:*
    - *NotificationManager*
  - *Email API*
- **R30**: The system must calculate and update tournament score at the end of each battle.
  - *Application server:*
    - *RequestManager*
      - RankingController
  - *Repository*
  - *DBMS*
- **R31**: The system must create and update the tournament ranking.
  - *Application server:*
    - *RequestManager*
      - RankingController
  - *Repository*

  - *DBMS*
- **R32**: The system must create the battle ranking when a new battle is created.
  - *Application server:*
    - *RequestManager*
      - RankingController
      - BattleController
  - *Repository*
  - *DBMS*
- **R33**: The system must allow educators with permission on the tournament to close it
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Repository*
  - *DBMS*
- **R34**: The system must ensure that the final score of each battle for a team falls within the range of 0 to 100.
  - *Application server:*
    - *RequestManager*
      - ManualEvaluationManager
    - AutomaticEvaluationController
  - *DBMS*
- **R35**: The system must allow educators with the necessary permissions to delete a tournament in which no battles are currently ongoing.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Repository*
  - *DBMS*
- **R36**: The system must allow educators with necessary permissions to delete a battle that is not currently ongoing.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - BattleController
  - *Repository*
  - *DBMS*
- **R37**: The system automatically deletes a battle when no teams have subscribed to it by the registration deadline.
  - *Application server:*
    - RequestManager:

        - BattleController
  - *Repository*
  - *DBMS*

5 Implementation, Integration, and Test Plan

5.1 Overview

This chapter discusses the implementation of the system, the integration of components, and the test plan. The goal of testing is to identify and correct most bugs before the application is released. It is emphasized that implementation and integration are processes that go hand in hand; often the order of integration follows that of implementation. During the definition of the implementation strategy, the integration test plan is also considered.

5.2 Implementation Plan

The implementation and testing plan for our system will follow a combined bottom-up and thread strategy, in order to take advantage of both approaches.
Bottom-Up Strategy The bottom-up strategy focuses on the incremental integration of the system components, starting from the most basic parts and moving towards more complex functionalities. This method offers several advantages:

1. Ease in Tracking Bugs: Since integration occurs progressively, it is easier to identify and resolve bugs in the basic modules before they affect broader components.
2. Incremental Testing: Each component can be tested as it is integrated, allowing for the validation of each part of the system in successive phases.
3. Flexibility and Adaptability: This approach allows for greater flexibility, as changes can be made more easily during the early stages of development.

Thread Strategy The thread strategy, in our context, implies the identification and implementation of the system functionalities through different execution threads. The advantages include:

1. Assessable Intermediate Deliverables: Through the use of threads, intermediate deliverables can be produced that can be assessed by stakeholders, facilitating the validation of the realized system.
2. Parallelism in Development: Different functionalities can be developed in parallel by different teams, increasing efficiency and reducing development times.

5.2.1 Features Identification

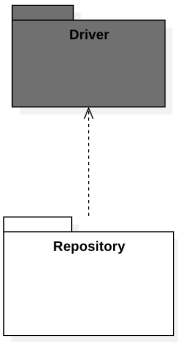
In the "Features Identification" phase, we list and analyze the key features of the system. This process is fundamental to understand the order of implementation of components, as some functionalities depend on others previously implemented.
[F1] Sign in and Sign Up These are the basic functionalities that allow users to access and use the platform. They must be implemented first, as they are the entry point for students and educators and are prerequisites for all other functionalities.
[F2] Tournament Management
After establishing user access, the next step is to allow educators to create and manage tournaments. This functionality is fundamental because it structures the entire context in which the programming battles take place. Without tournament management, it would not be possible to organize and coordinate the programming challenges.
[F3] Battle Management Once the tournaments are set up, it is necessary to implement the management of individual battles. This includes uploading code katas, defining teams, and configuring deadlines. Battles are the central element of student interaction on the platform and depend on the prior configuration of tournaments.
[F4] Evaluation
With battles underway, it is essential to have an evaluation system that analyzes the solutions submitted by students. This process includes verifying solutions against test cases and manual evaluation by educators. The evaluation functionality must follow battle management, as it is essential for determining team scores.
[F5] Ranking Management
After evaluating battles, it is necessary to manage rankings of both individual battles and overall tournaments. This allows students and educators to see performance, creating a competitive context. This functionality depends on the evaluation of battles and the management of tournaments and battles.
Notifications Management

This feature manages the notifications for the user. Unlike other functionalities, notification management will be developed and refined continuously throughout the entire development process. They do not follow a fixed order but are integrated and adapted based on the progress of other functionalities.

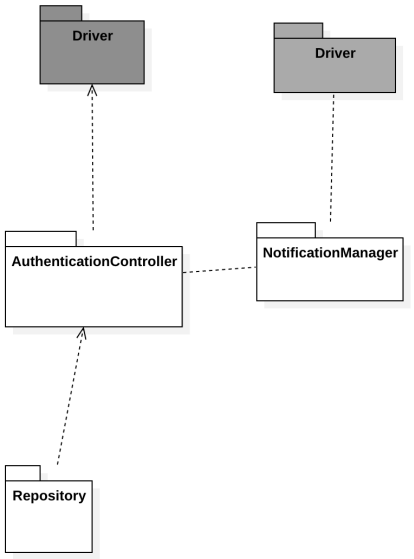
5.3 Component Integration and Testing

In the next section, we will show for each phase which components are implemented, how the integration of these components takes place, and how they are tested, supported by an integration diagram specific to each phase.

The first step is to implement the Repository, which is the component that implements all methods that allow access to the Database and perform queries and updates on it. Since we are following a bottom up approach, we will use drivers to test the integration of the components, as it can be seen from the following diagrams:

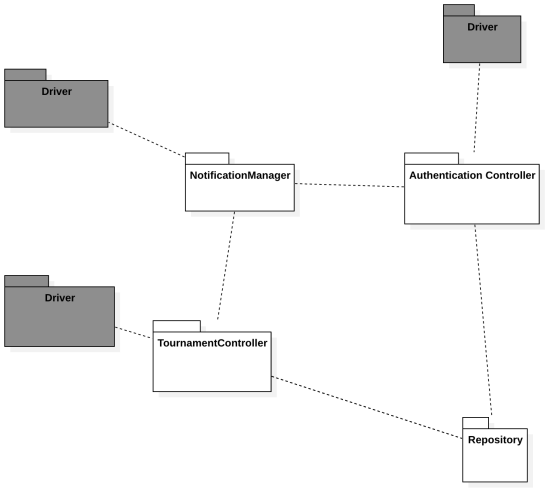


[F1] Sign in and sign up developing
Subsequently, we focus on building out the authentication mechanism first and part of the User notification component because the SignUpManager uses it to send email to users.

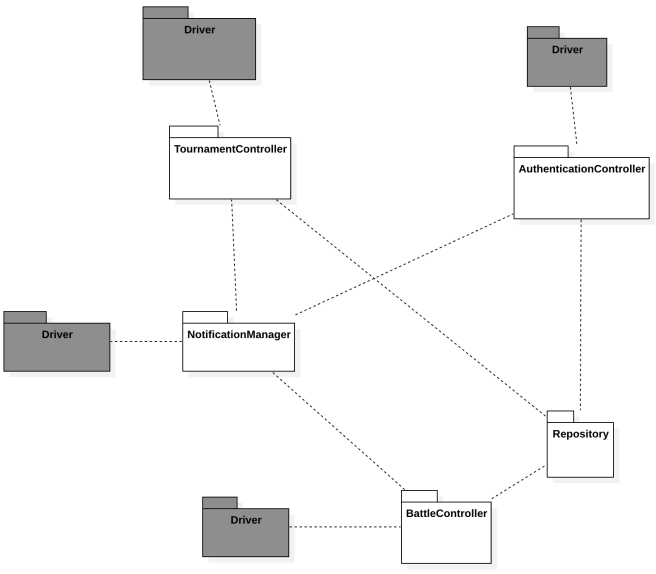


[F2] Tournament management
Following the authentication setup, we turn our attention to the tournament management aspect, since

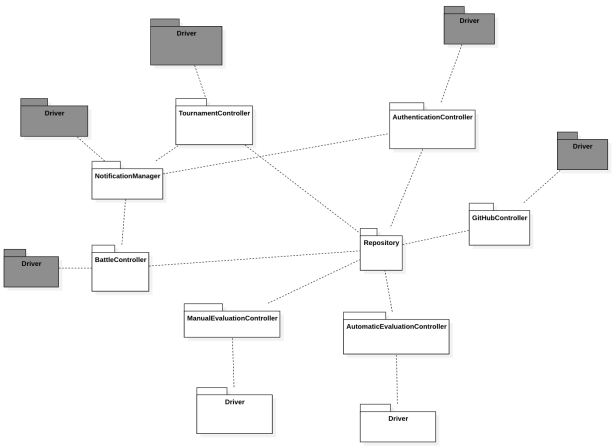
it serves as a foundation for subsequent functionalities that hinge on its proper operation. Alongside this, a segment of the Notification Manager is also implemented. This integration is vital as it plays a key role in tournament management, particularly in notifying subscribed users about tournament creation.



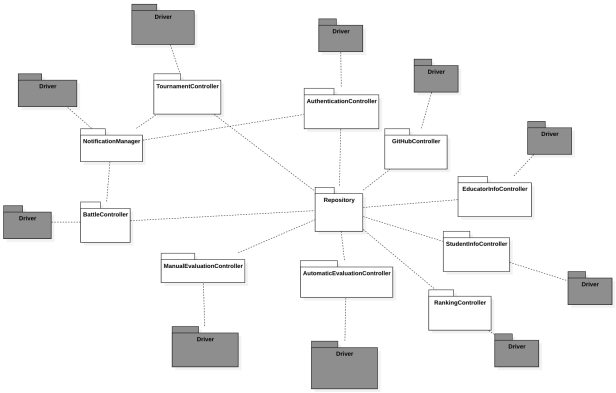
[F3] Battle management
Our next focus is on developing the battle management system.



[F4] Evaluation
After the battle management component is fully operational, we advance to the development of the evaluation system. This component encompasses both automated and manual assessment methods, designed to evaluate the outcomes of the battles. The placement of the evaluation system in the development sequence is strategic, as it relies heavily on data generated from completed battles. Alongside this, a segment of the Notification Manager is also implemented. This integration is vital as it plays an important role in battle management.



[F5] Ranking management
Following the evaluation system, our attention shifts to the development of the ranking component. This feature is essential as it compiles and displays the results of the battles and tournaments, providing a clear and concise representation of participants' standings.



5.4 System Testing

After integrating the components into the CKB system, we proceed with a series of tests. The goal is to ensure that the functionalities meet expectations and that the application behaves appropriately under different workloads. We outline various testing areas, each focused on specific needs:

- **Functional Testing**: To verify that each single function of the platform, from notifications to score updates, operates correctly according to the specifications.
- **Performance Testing**: Evaluates the speed, responsiveness, and stability of the application under various workloads. It's important to ensure that the application's performance is up to the challenges, especially when the platform calculates competition scores in real-time, a central element of the user experience on CKB.
- **Load Testing**: Verifies the app's ability to handle a high volume of users or transactions simultaneously.
- **Stress Testing**: To push the system beyond its usual operational limits and observe its ability to maintain the integrity and security of operations even under extreme conditions.

All the tests in general are fundamental for studying the behavior of the system.

5.5 Acceptance Testing

Following the tests described above, we will proceed with user acceptance testing. This step is crucial: even if we have verified the compliance with functional and non-functional requirements, it remains essential to obtain the final approval from the client.

Acceptance testing is conducted using a black-box approach, involving users in interactions with the system that reflect real-world use. This method allows confirming whether the software meets the original specifications desired by the client. It is a definitive check that validates the product's full correspondence to the expectations and needs for which it was commissioned.

6 Time Spent

The time tables written below represent just an approximation of the time spent for the writing and discussions the team had for each specific chapter of this document. It is important to note that both team members worked together for the entire duration of each task.

Task	**Hours**
Chapter 1	20-20
Chapter 2	30-30
Chapter 3	30-30
Chapter 4	25-25
Chapter 5	25-25

7 References

- Testing models and component architecture made with: StarUML
- Sequence diagram models made with: StarUML
- High level architectures are made using www.draw.io
- Software Engineering 2 course material - Politecnico di Milano 2022-2023