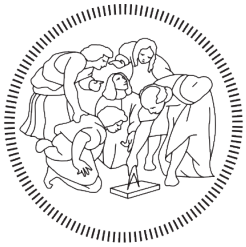
Software Engineering 2 - Prof. Di Nitto Elisabetta
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

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.
- Section 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

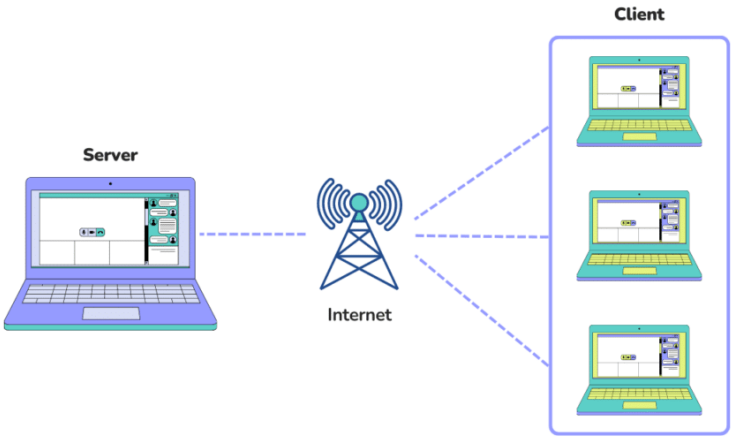


Figure 1: Client-Server architecture

Il sistema è un'applicazione distribuita che segue il noto paradigma client-server. Adotta la tecnica del thin client per facilitare il supporto di diverse piattaforme client. Questo approccio offre vantaggi significativi: primo, riduce i requisiti di hardware e software sui dispositivi client, poiché l'elaborazione principale è gestita dal server. Questo significa che anche i client con capacità di elaborazione limitata possono accedere efficacemente al sistema. Secondo, semplifica notevolmente la manutenzione e l'aggiornamento del software, in quanto le modifiche sono centralizzate sul server e non devono essere implementate su ogni dispositivo client individualmente. Infine, migliora la sicurezza, poiché i dati sensibili possono essere memorizzati e gestiti in modo più sicuro sul server, riducendo il rischio di perdite di dati o violazioni della sicurezza sui dispositivi client. Il nostro sistema utilizza come client le applicazioni web.

Tier architecture

The architecture of our system is organised into four tiers:

1. Thin Client: Il Thin Client è il punto di contatto con l'utente, come ad esempio un un browser web.
2. Web Server: Il Web Server funge da intermediario tra il client e l'Application Server. È responsabile della ricezione delle richieste HTTP/HTTPS dal Thin Client e dell'inoltro delle richieste elaborate all'Application Server.
3. Application Server: L'application Server è il cuore del sistema, dove la logica di business e le funzionalità dell'applicazione sono elaborate. Questo tier elabora le richieste provenienti dal Web Server, eseguendo operazioni complesse, gestendo la logica di business, le transazioni e l'interazione con il Database Server. L'Application Server può ospitare una varietà di servizi web, microservizi, e API che espongono la funzionalità dell'applicazione al Web Server e, di conseguenza, al Thin Client.
4. Database Server: Il Database Server gestisce la persistenza, il recupero e la gestione dei dati.

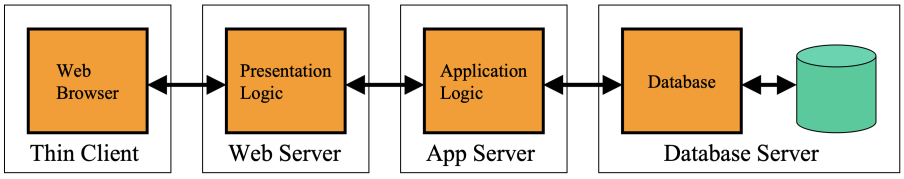


Figure 2: 4 Tier architecture

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.

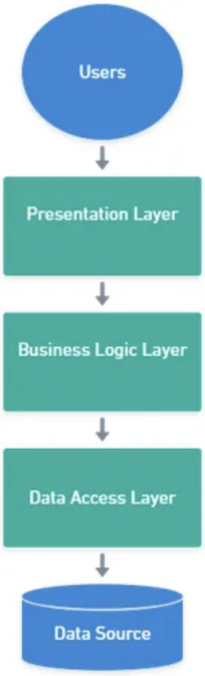


Figure 3: 3 Layers architecture

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.

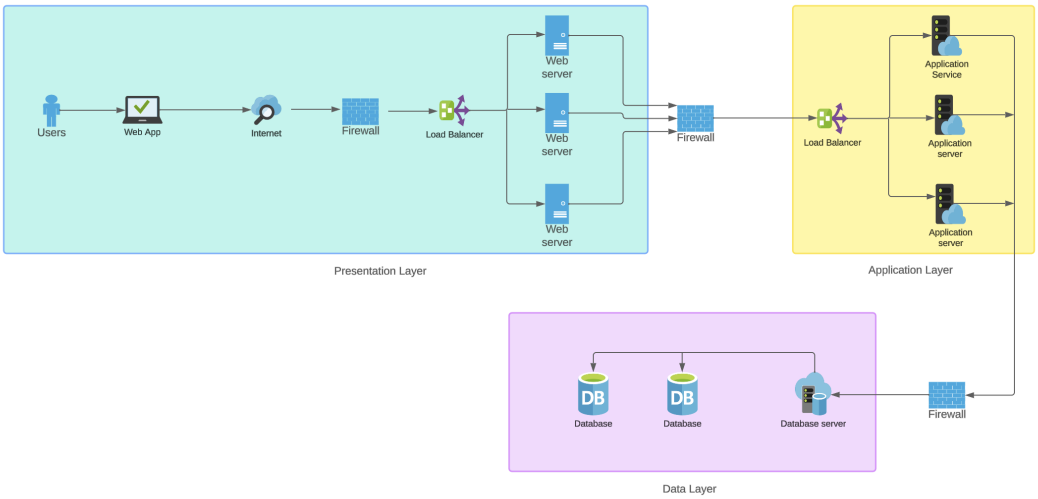


Figure 4: Architecture

Load Balancing

Nella architettura presentata ci sono due load balancer che svolgono ruoli cruciali nel gestire il traffico in ingresso verso i server web e applicativi.

1. Primo Load Balancer: Situato dopo il firewall e prima dei server web, questo load balancer ha il compito di distribuire le richieste dei client in arrivo ai server web. Quando una richiesta passa attraverso il firewall, viene intercettata dal load balancer, che poi determina quale server web è il più idoneo per gestire quella richiesta. Il criterio di scelta è basato sul carico corrente di ciascun server e sulla capacità di minimizzare il tempo di risposta. In questo modo, il bilanciamento del carico aiuta a prevenire il sovraccarico di un singolo server web, garantendo una distribuzione equa delle richieste e ottimizzando l'utilizzo delle risorse.
2. Secondo Load Balancer: Situato tra i server web e i server applicativi, il secondo load balancer agisce in modo simile al primo, ma a un livello diverso dell'architettura. Dopo che il server web ha elaborato la richiesta iniziale la inoltra al secondo load balancer. Quest'ultimo poi distribuisce la richiesta a uno dei server applicativi disponibili, ancora una volta basandosi sul carico di lavoro e sulla capacità di fornire una risposta rapida. In questo modo, anche il livello applicativo è protetto dal rischio di sovraccarico e può scalare efficacemente in risposta a variazioni del traffico.

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

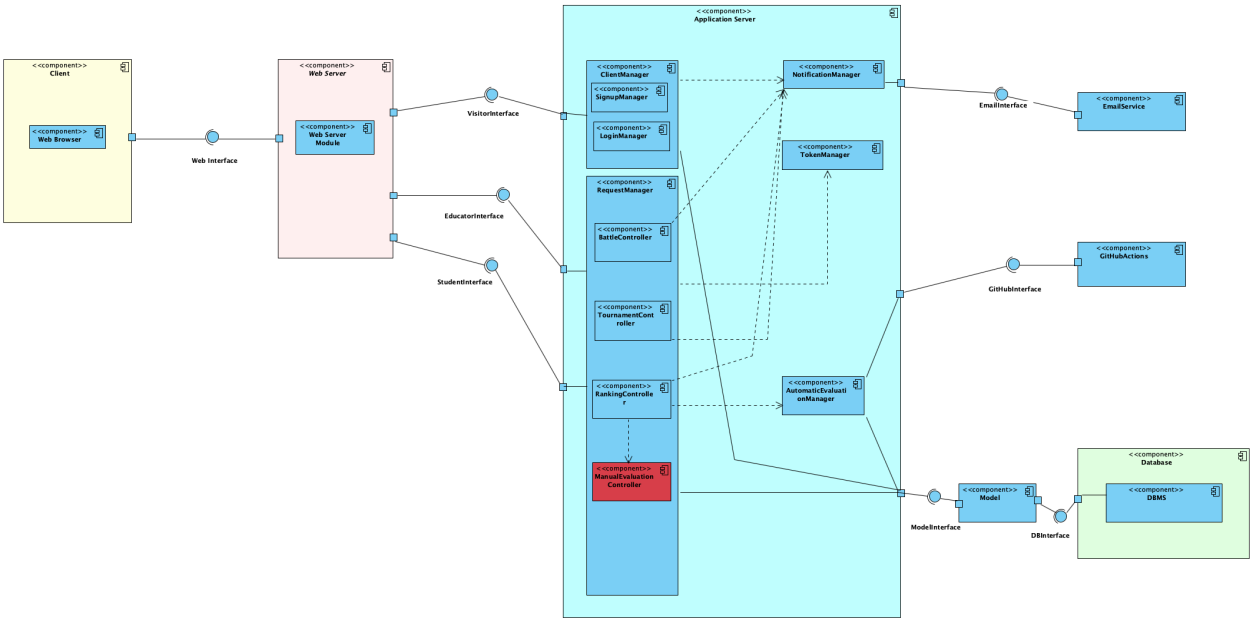


Figure 5: 3 Layers architecture

Breve descrizione dei componenti più importanti:

- Web Server: it handles managing HTTP requests from the client and forwarding them to the application server.
- Client Manager: it manages all requests from users who are not yet authenticated, giving them the opportunity to register (Signup Manager) or log in (Login Manager). 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.
- Token Manager: It manages the creation, validation, renewal, and revocation of authentication tokens.
- NotificationManager: It handles the management of notifications whenever requested. Notifications are sent via email through the Email Service.
- Model: serves as the repository for data that constitute the enduring state of the system, which, in our scenario, is preserved within a database. When the application initiates, this component ensures the data are loaded for use. It encompasses a suite of methods essential 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 Model component with the retrieval task. The Model not only fetches the requisite data but also organizes them into specialized data structures, thus priming them for imminent computational tasks. It is the sole component that interfaces with the database management system (DBMS), acting as the exclusive conduit for data interaction.

2.3 Deployment View

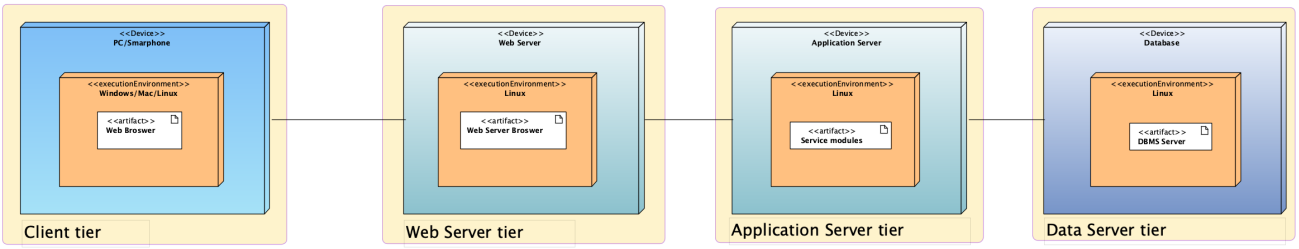


Figure 6: Architecture

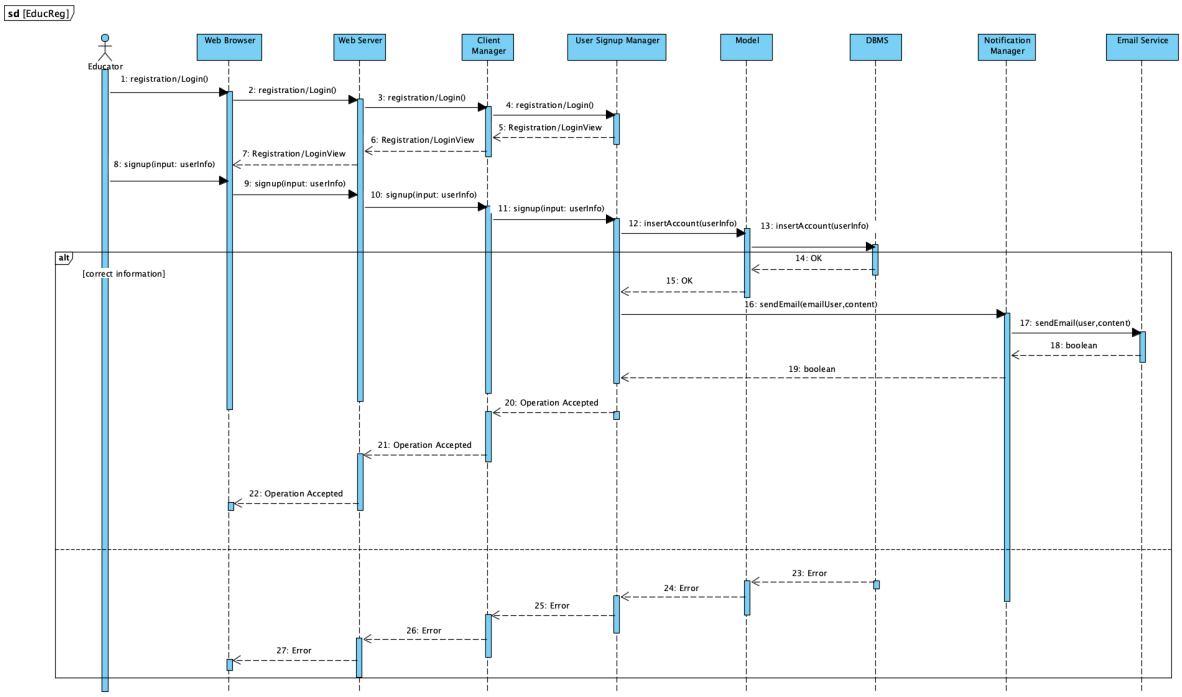
The diagram presented shows the organisation of the system, highlighting the components required for its operation. Each device illustrated runs a specific operating system that supports the execution of the software components. To expand the system, additional copies of these devices can be added. The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

- **Client tier**: Questo strato include dispositivi come PC e smartphone che utilizzano sistemi operativi come Windows, Mac o Linux. Il componente software principale in questo livello è il browser web, che permette agli utenti di interagire con il sistema.
- **Web server tier**:Questo livello è costituito da server web che eseguono un sistema operativo Linux. Il componente software rappresentato è Apache che si occupa di elaborare le richieste dei client e servire pagine web.
- **Application server tier**: Questo strato include server di applicazioni, anch'essi su Linux, che eseguono moduli di servizio. Questi moduli gestiscono la logica di business dell'applicazione e di solito comunicano tra il web server e il database server.
- **Data server tier** : L'ultimo livello è composto da un database che gira su Linux, con un DBMS (Database Management System) Server come componente software. Questo gestisce lo storage e il recupero dei dati.

2.4 Runtime views

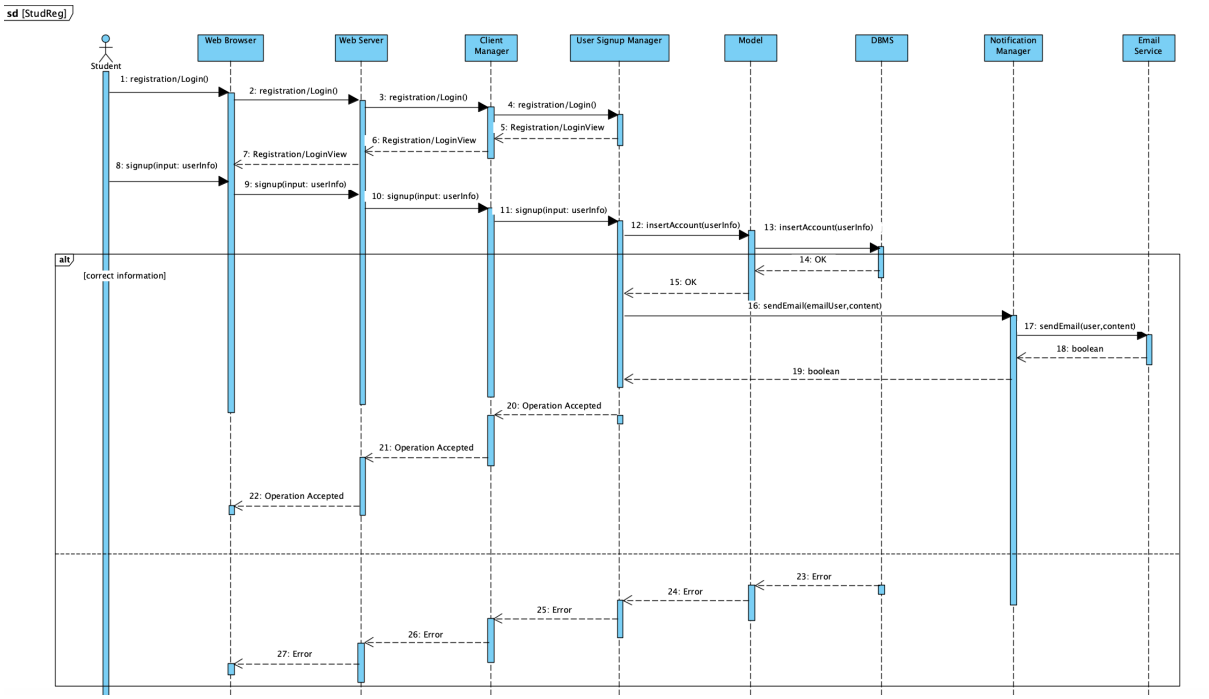
This chapter details the dynamic perspectives linked to the use cases outlined in the RASD for CKB. It illustrates how different components interact to facilitate the functions provided by CKB, offering a comprehensive look at its operational architecture.

2.4.1 Educator Registration



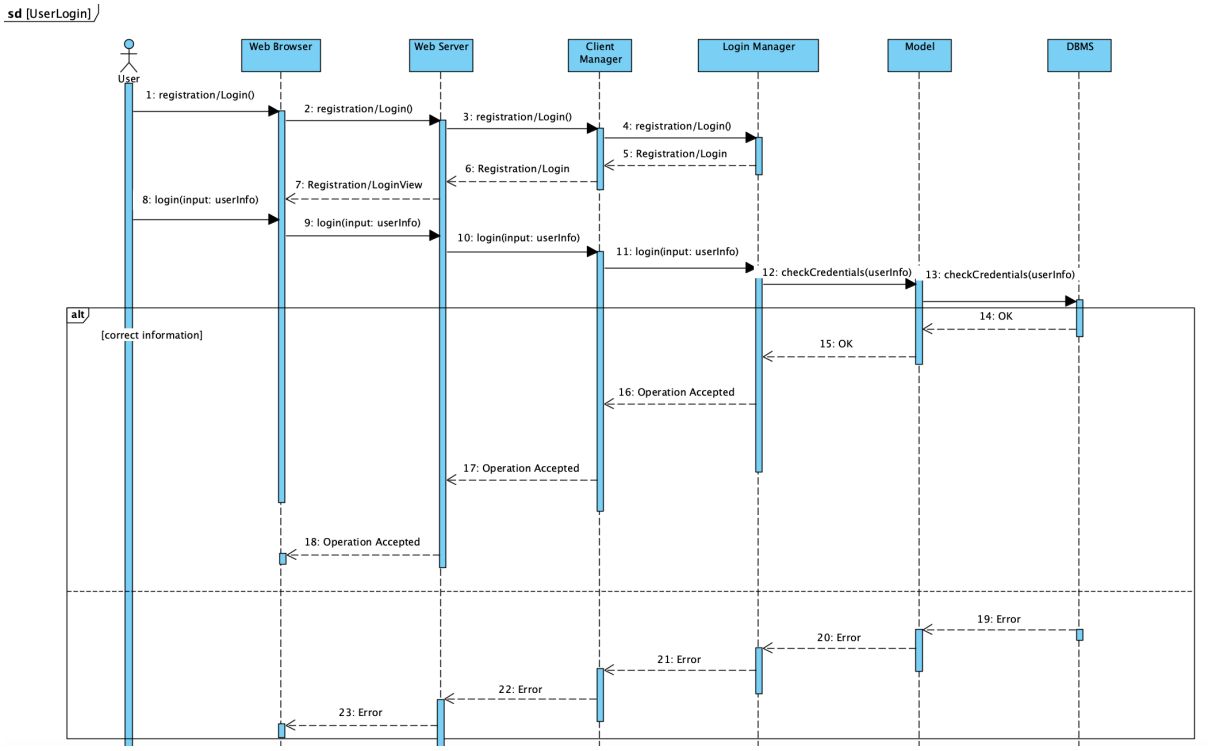
The sequence diagram depicted shows the registration flow of an educator. The educator starts the process from the homepage by clicking on the 'Registration/Login' section. The system offers the options of registration or login. Upon choosing registration, the educator is presented with a form to complete 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 Client Manager, 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



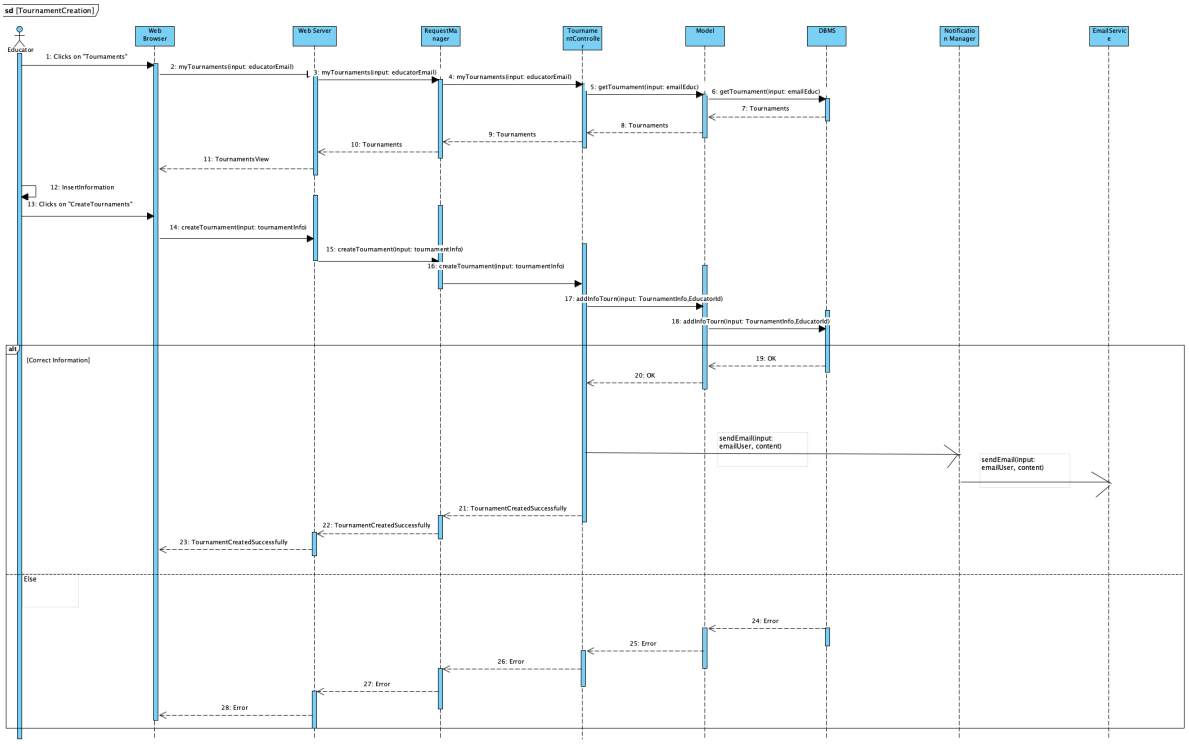
In the sequence diagram displayed, a student initiates registration via the homepage, completes a form with his or her personal data and confirms registration. The Web Browser interacts with the Web Server, which in turn involves the Client Manager and the User Signup Manager to manage the sequence of events. These components co-ordinate the registration operation, with the User Signup Manager sending the data to the Model 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 and Client Manager passes the information to the User Signup Manager. The latter component makes a call to the Model, 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

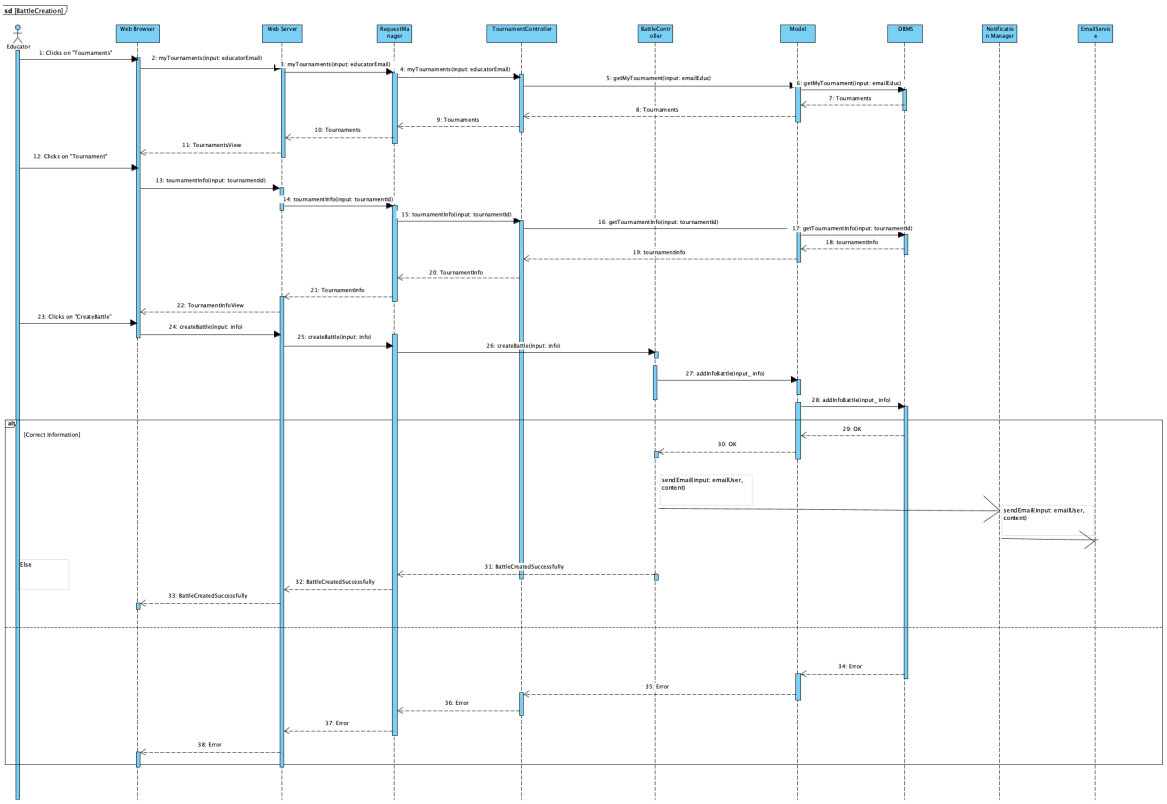


The sequence diagram shows the process the educator follows to create a tournament on the platform. After selecting the option to create a tournament, the educator enters the required data via the Web Browser. This information is sent to the Web Server, which interfaces with the Request Manager. The 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 Model, 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 Model updates the DBMS with the new tournament information.

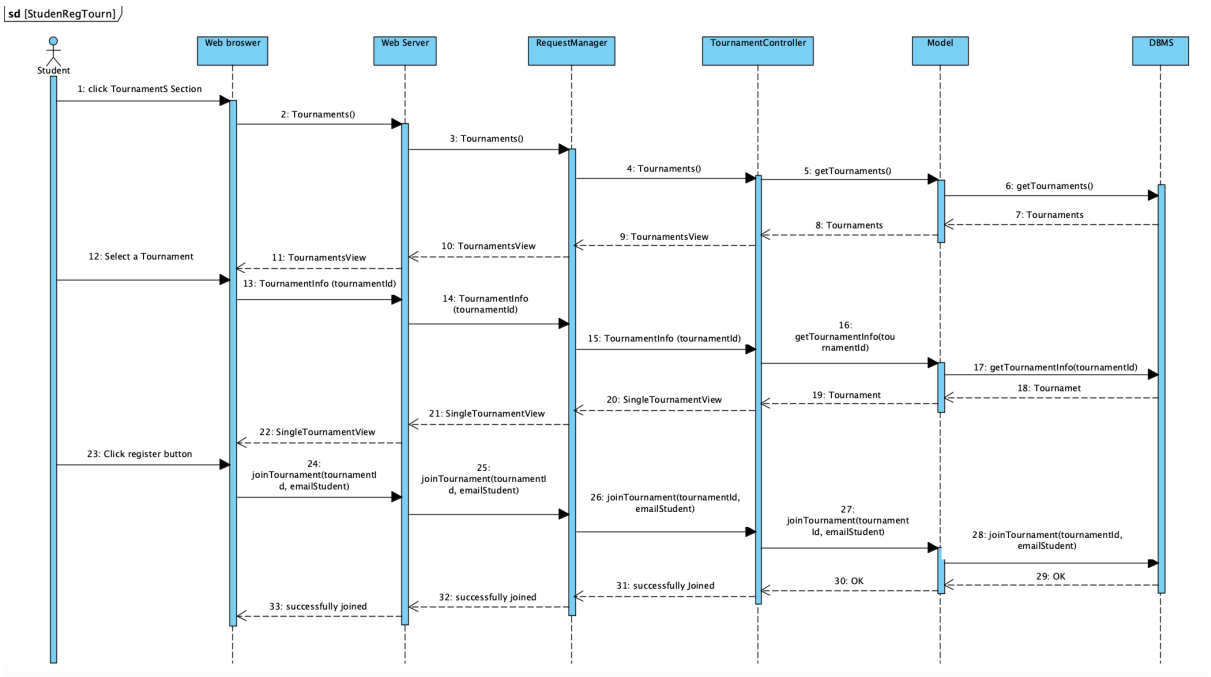
Once the creation of the tournament is confirmed, the Notification Manager comes into play, invoking the Email service to notify students registered on the platform. The process concludes by saving the tournament information in the DBMS and notifying interested users, showing positive feedback through the educator user interface.

2.4.5 Battle Creation



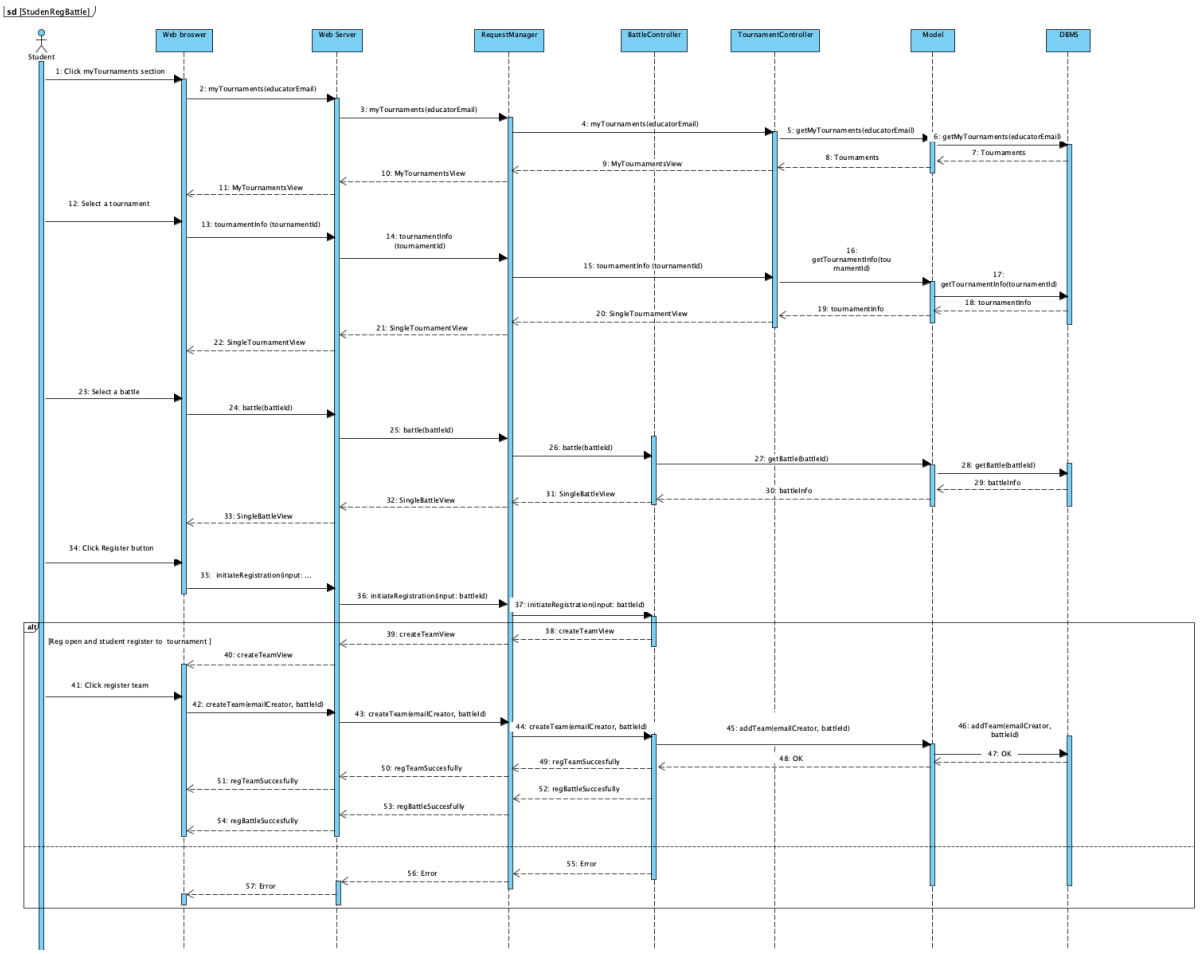
In the sequence diagram, the educator interacts with the system to create a battle within a tournament. This interaction begins in the Web Browser and is sent to the Web Server, which acts as a conduit for all subsequent requests and responses. The flow continues with the Request Manager which processes 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 Model 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 service to send notifications to students registered for the tournament, informing them of the newly created battle.

2.4.6 Student registers for the Tournament



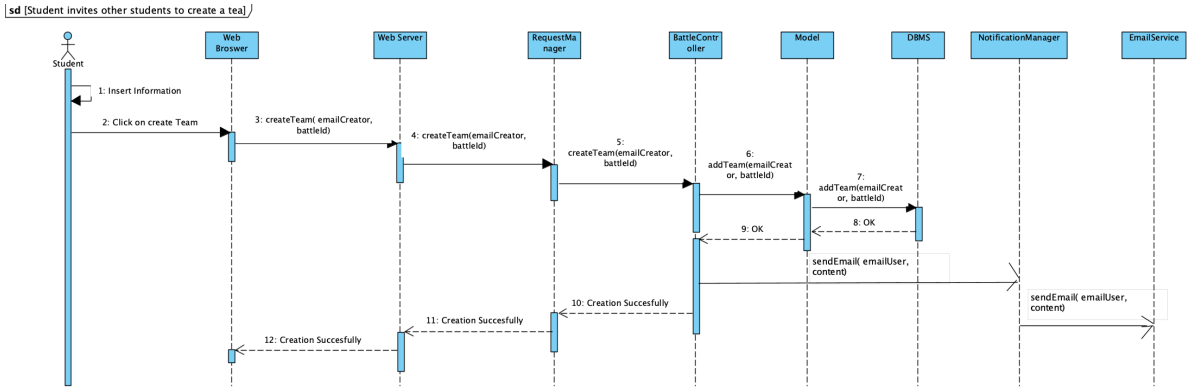
The sequence diagram describes the process of registering a student for a tournament through the system. The student selects the "Tournaments" section from the dashboard, and the system displays a list of available tournaments. After browsing through the options, the student chooses a tournament of interest 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 Model 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 Model to obtain the details of the battle. The Model 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



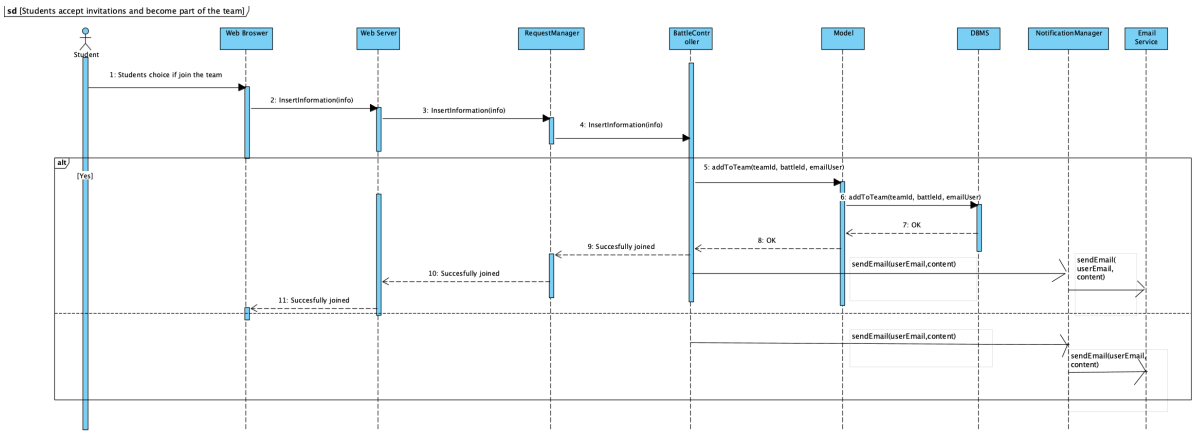
The sequence diagram shows the flow of events for the formation of a team by a student within the platform. After selecting "Register" for a specific battle, the student enters the team name and selects the students to form the team.

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 Model to add the team to the system. The Model 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 service 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.

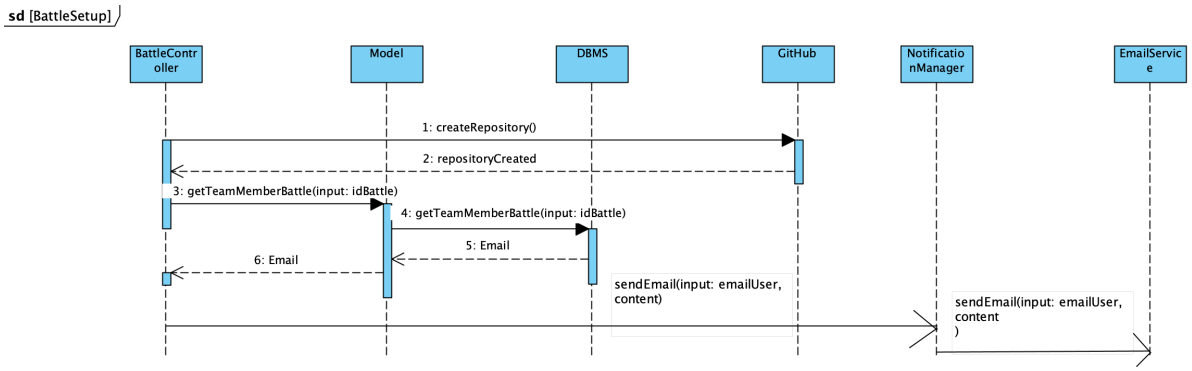
2.4.9 Student accept invitations and become part of the Team



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

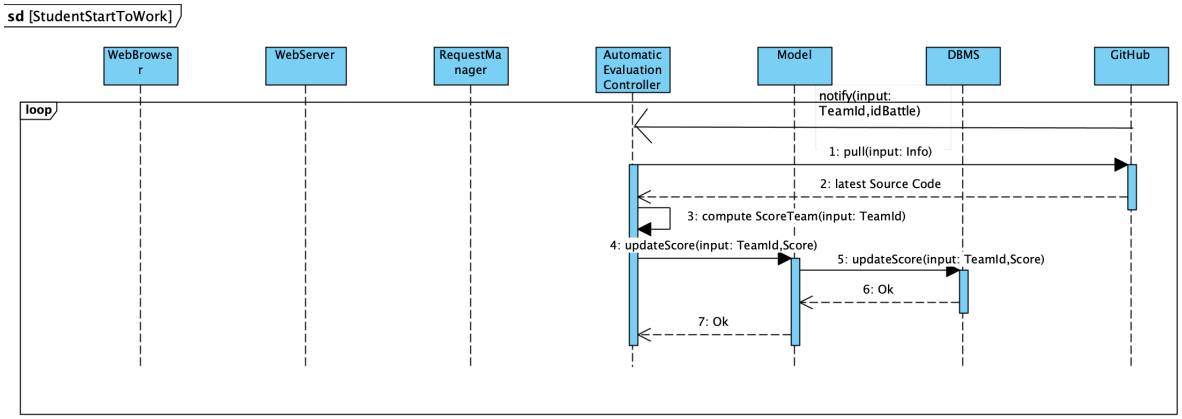
the invitation by clicking 'Yes'. At this point, the Web Server receives an indication of acceptance of the invitation and passes this information to the Request Manager. The Request Manager acts as coordinator of the data flow, ensuring that the request is correctly addressed. It then transfers the team membership request to the Battle Controller. The Battle Controller invokes the Model to make the change in the system, adding the user to the specific team. The Model 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 Model 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 service 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 service to send these details to the students.

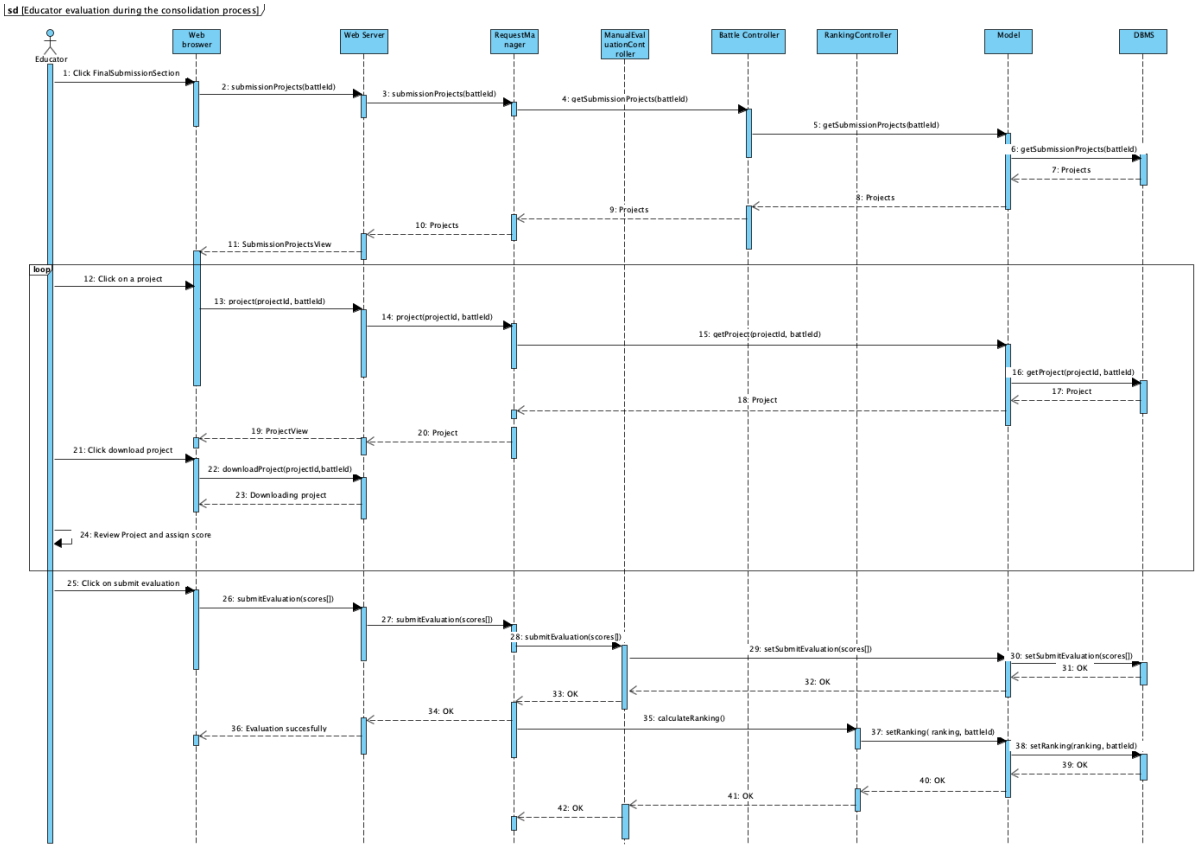
2.4.11 Student start to work



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 AutomaticEvaluationController, interacts with the Model to 'pull' the latest changes from the GitHub repository. The Model then requests the Database Management System (DBMS) to update the latest data, thus keeping the project status synchronised with the latest student submissions. Immediately afterwards, the system proceeds with the calculation of the team score. Once the score has been calculated, the model updates the DBMS with the new team score. Finally, when the submission deadline is reached, the system stops monitoring further code pushes. This moment marks the end of the battle.

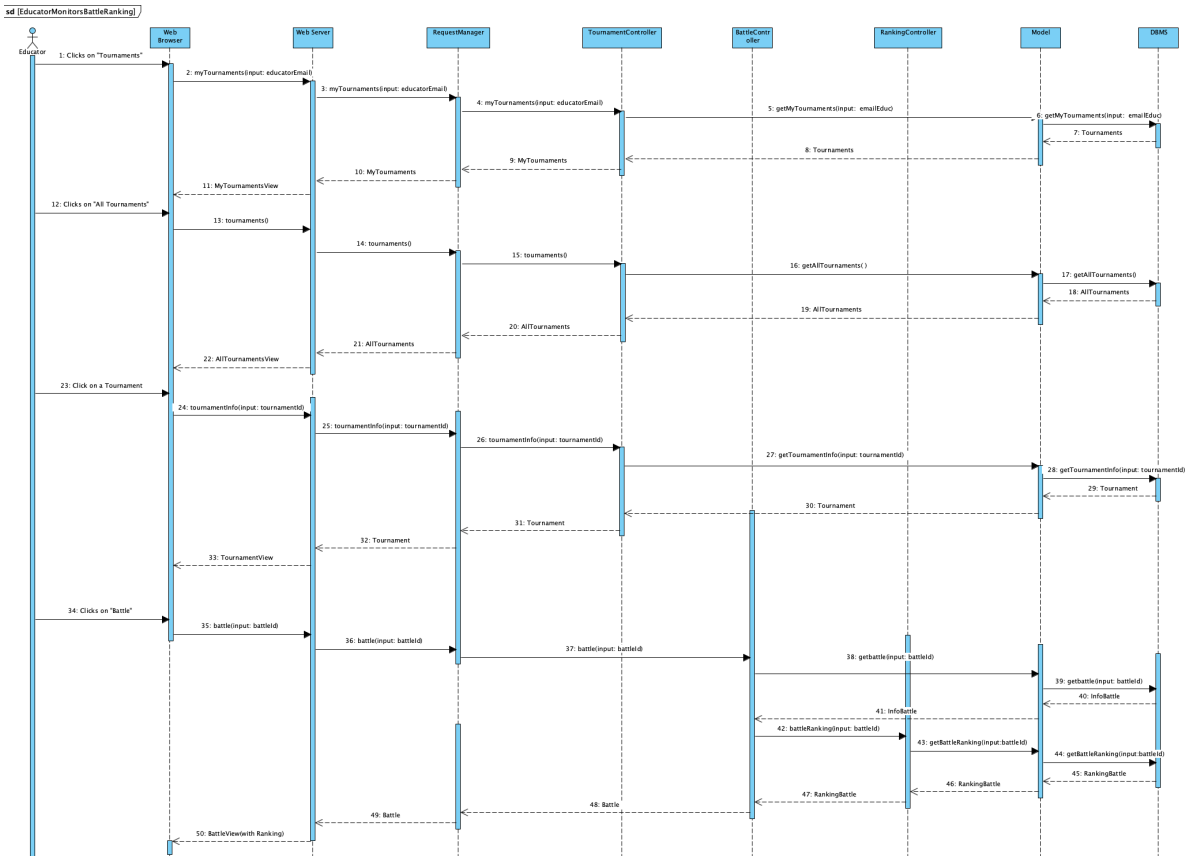
2.4.12 Educator evaluation during the consolidation process



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 Model 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

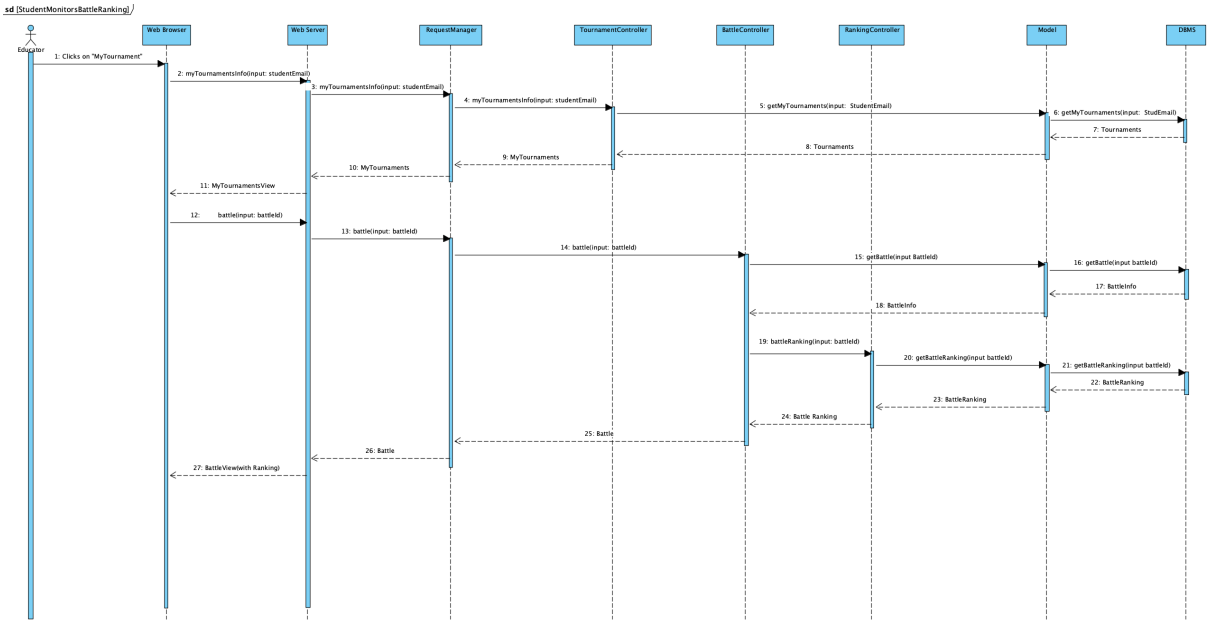
students, providing them with ranking information that is now available.

2.4.13 Educator monitors battle ranking



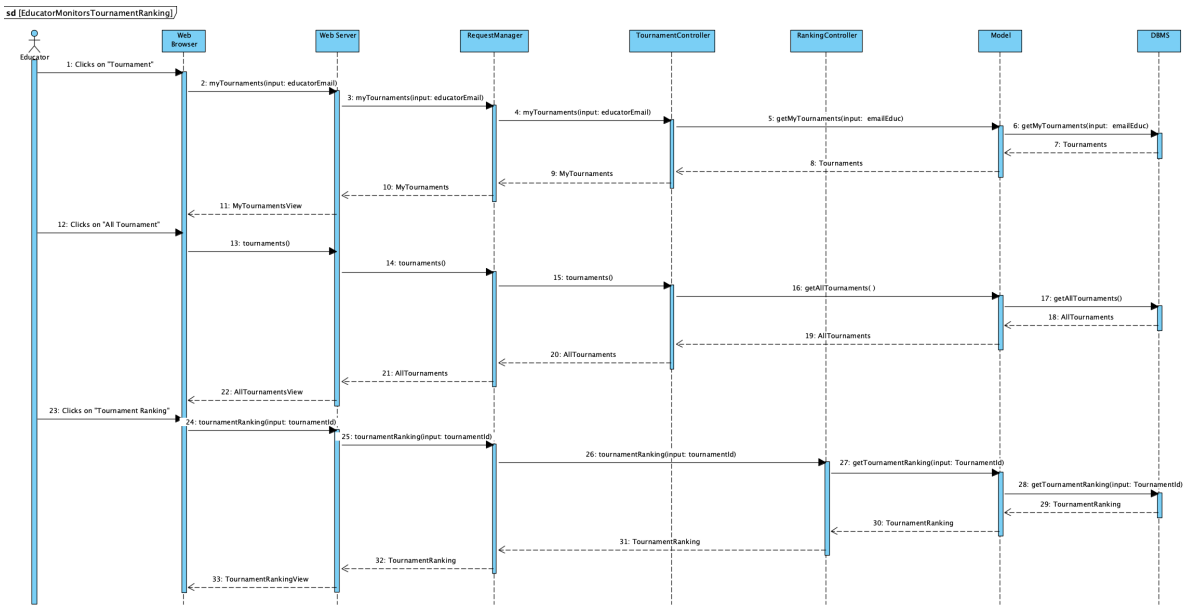
In the sequence diagram, when the educator navigates through the platform to monitor the rankings, he initiates a series of interactions between various system components. After selecting the tournament section, his web browser sends a request to the web server, which is the entry point for all incoming 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 Model 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 Model. The Model 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



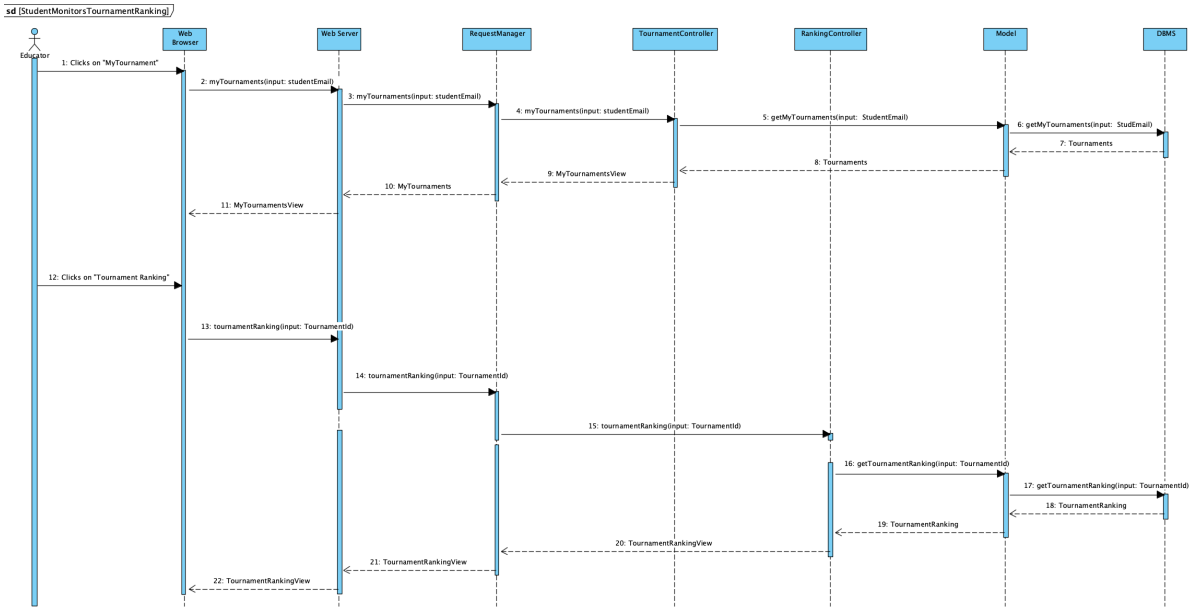
The sequence diagram illustrates how a student can access and view the ranking of a battle within a tournament via an online platform. Starting on the homepage, the student selects the "Tournaments" section, where the web server receives and forwards the request to the Request Manager. This, in turn, 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 Model 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 Model 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 Model and the DBMS. Once the battle information has been retrieved, 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



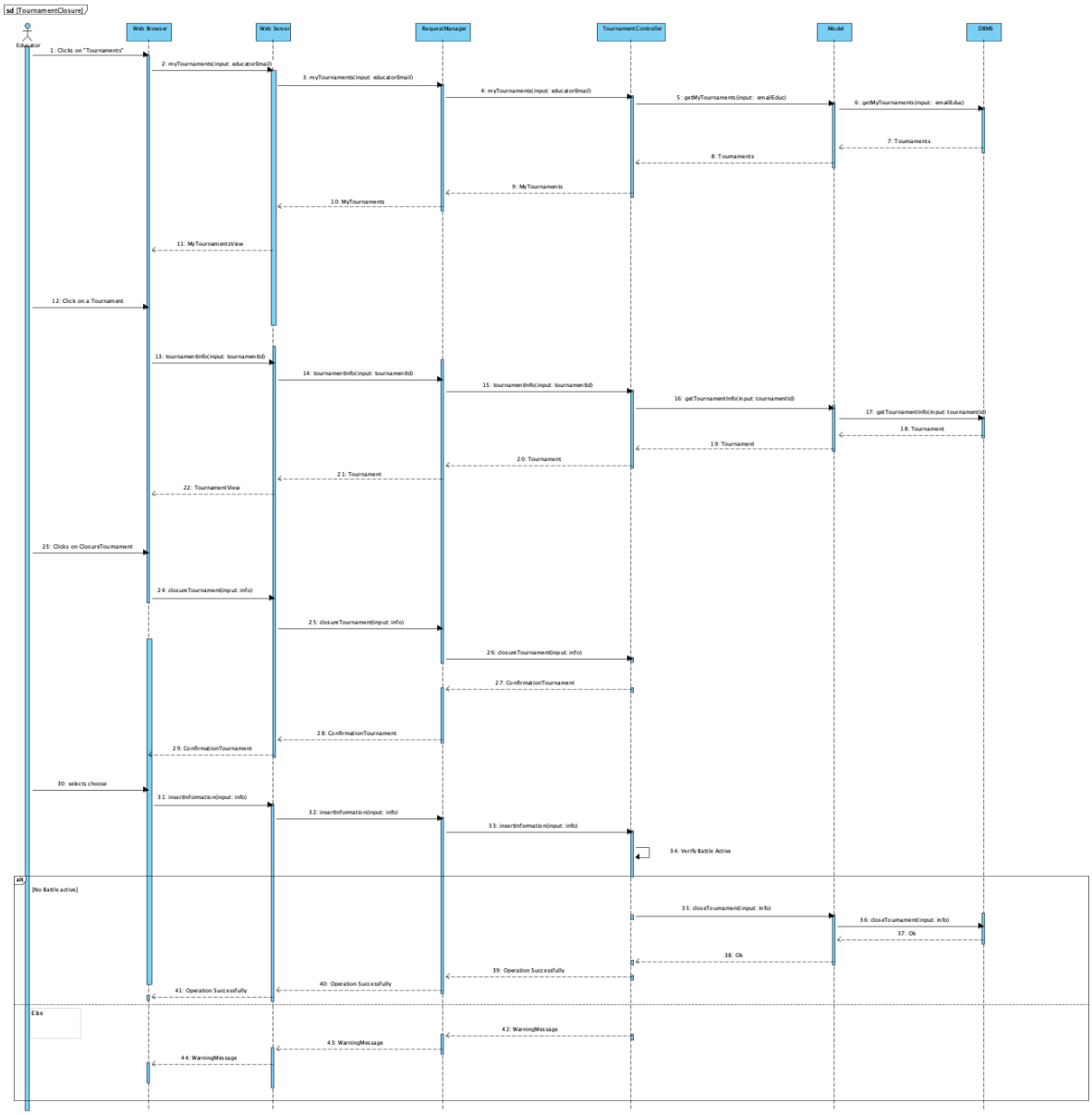
In the sequence diagram, an educator follows a series of steps on the online platform to view the ranking of a tournament. Starting from the homepage, the educator accesses the 'Tournaments' section, which shows him/her a control dashboard. This dashboard provides an overview of the tournaments the educator 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 Model to query the Database Management System (DBMS) for ranking data. The Model 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



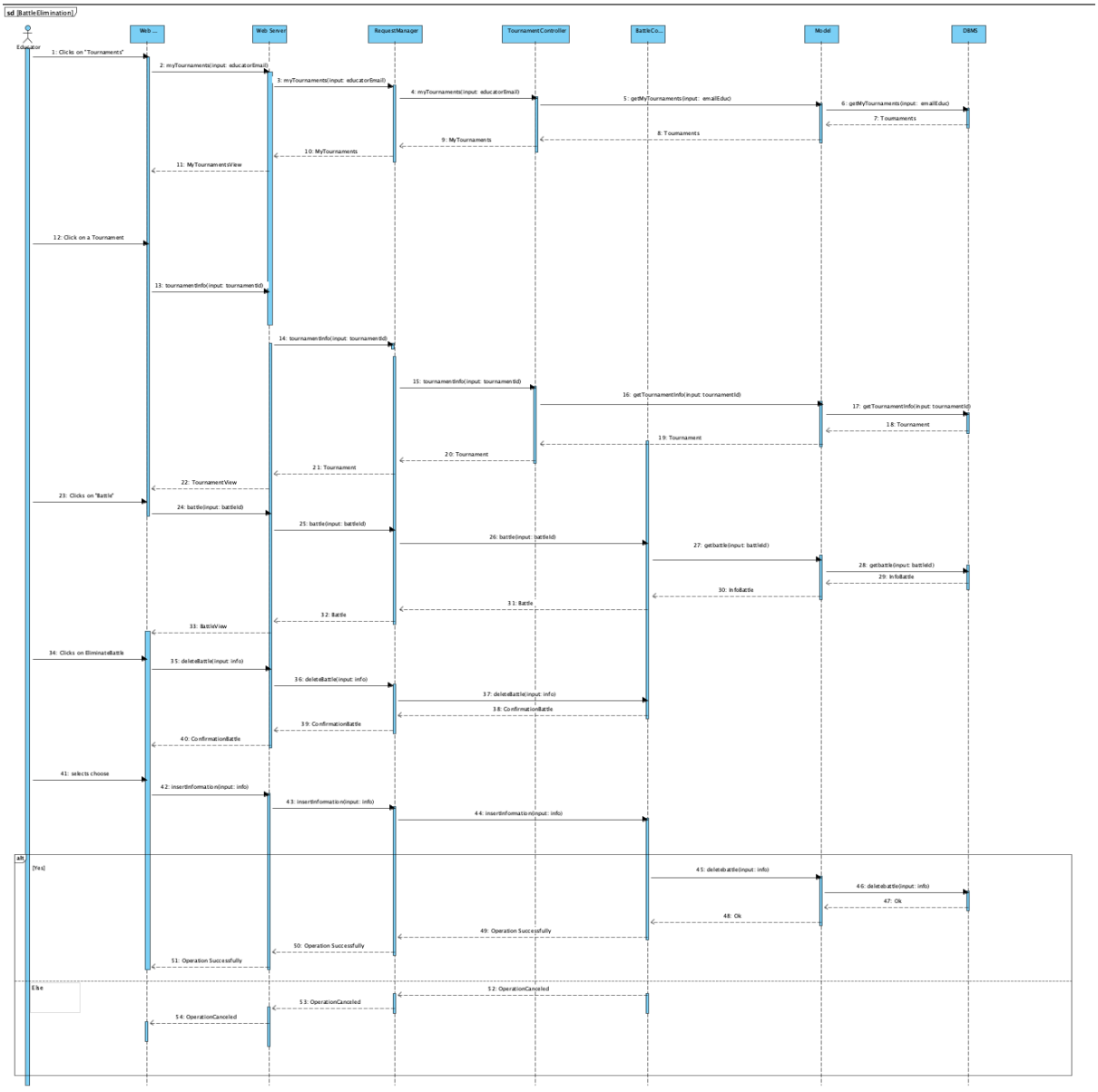
In the sequence diagram, a student follows a procedure to view the ranking of a tournament within CKB. It all starts on the homepage, where the student clicks on the "Tournaments" section, triggering a series of requests through the platform. The first request is sent from the student's web browser to 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 Model 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 Model 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



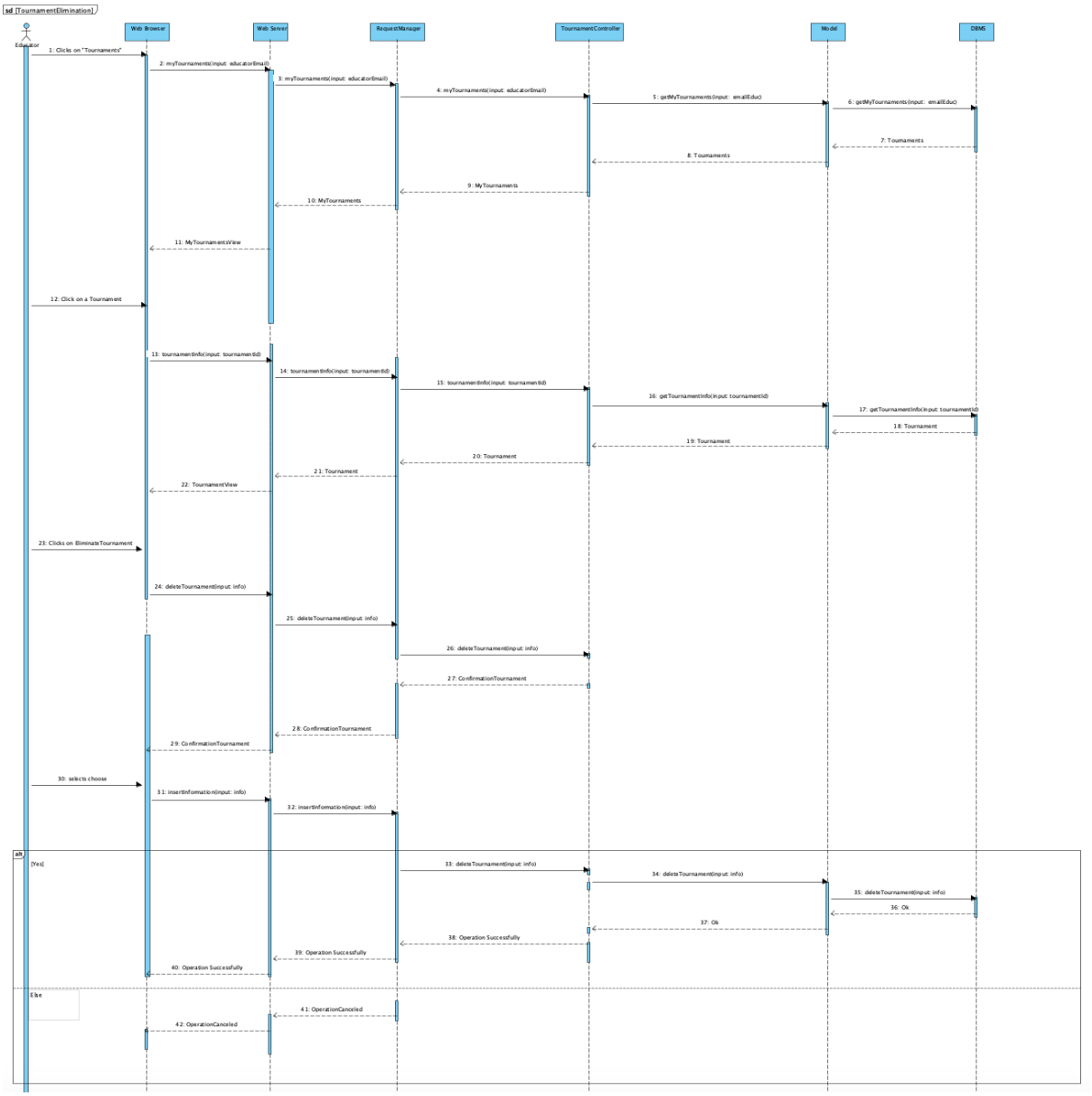
In this scenario, the educator takes a series of actions on the platform to close a tournament. After selecting the 'Tournaments' section from the homepage, a control dashboard is displayed through his web browser. This dashboard is the result of a series of interactions between the browser, the web 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 Model 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 Model to change the tournament status from 'Open' to 'Closed' in the DBMS. The Model 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 Model 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 Model, 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 Model. 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 Model, 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 model. 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

- **LoginManager**: This component offer, through interface VisitorInterface this functions:
  1. registration/login()
  2. login(input: email, password)
  3. logout()
- **SignupManager**: This component offer, through interface VisitorInterface this functions:
  1. registration/login()
  2. signup(input: name, surname, email, password, userType)
- **BattleManagementController** : This component offer, through interface StudentInterface this functions:
  - battle(input: battleId)
  - createTeam(input: emailCreator, battleId)
  - joinTeam(input: teamId, battleId, emailUser)
  - initiateRegistration(input: battleId)
  - insertInformationTeam(input: emailStudent[],info)
- **BattleManagementController**: This component offer, through interface EducatorInterface this functions:
  1. insertInformation(input: info)
  2. deleteBattle(input: battleId, emailEducator, tournamentId)
  3. createBattle(input: battleInfo, emailEducator, tournamentId)
  4. deleteBattle(input: battleId, emailEducator, tournamentId)
  5. battle(input: battleId)
  6. getSubmissionProjects(input: battleId)
  7. getProject(input: projectId,battleId)
- **TournamentController**: This component offer, through interface StudentInterface this functions:
  1. tournamentInfo(input: tournamentId)
  2. joinTournament(input: tournamentId, emailStudent)
  3. tournaments()
  4. myTournaments(input:studentEmail)
- **TournamentController**: This component offer, through interface EducatorInterface this functions:
  1. insertInformation(input: info)
  2. tournaments()
  3. myTournaments(input: educatorEmail)
  4. tournamentInfo(input: tournamentId)
  5. createTournament(input: tournamentInfo)
  6. deleteTournament(input: tournamentId, emailEducator)
  7. closureTournament(input: tournamentId, emailEducator)
  8. verifyBattleActive()
- **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[])
  2. downloadProject(input: projectId)
- **AutomatedEvaluationController**: This component offer, this functions:
- computeScoreTeam(input: teamId)
  1. setEvaluation(input: parameters, Project)
- **NotificationManager**:
  1. sendEmail(input: emailUser, content)
- **Model**:
  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. getBattleRanking(input: battleId)
  16. getTournamentRanking(input: tournamentId)
  17. closeTournament(input: tournamentId, emailEducator)
  18. getAllTournaments()
  19. deleteBattle(input: battleId, emailEducator, tournamentId)
  20. deleteTournament(input: tournamentId, emailEducator)
- **WebServer**:
  1. registration/login()
  2. login(input: email, password)
  3. signup(input: userInfo)
  4. myTournaments(input: educatorEmail)
  5. createTournament(input: tournamentInfo)
  6. createBattle(input: battleInfo, emailEducator, tournamentId)
  7. tournaments()
  8. tournamentInfo(input: tournamentId)
  9. joinTournament(input: tournamentId, emailStudent)
  10. battle(input: battleId)
  11. initiateRegistration(input: battleId)
  12. createTeam(input: emailCreator, battleId)

13. insertInformationTeam(input: emailStudent[],info)
14. sumbissionProjects(input: battleId)
15. project(input: projectId, battleId)
16. downloadProject(input: projectId, battleId)
17. submitEvaluation(input: scores[])
18. tournamentRanking(input: tournamentId)
19. closureTournament(input: tournamentId, emailEducator)
20. insertInformation(input: info)
21. deleteBattle(input: battleId, emailEducator, tournamentId)
22. deleteTournament(input: tournamentId, emailEducator)

2.6 Architectural Styles and patterns

2.6.1 Four-tier system architecture

This system has a 4 tiers architecture. The benefit from this choice are:

- **Scalabilità**: Con i componenti logicamente divisi, è più semplice scalare il sistema. Se, per esempio, il carico sul server web aumenta, puoi aggiungere più server web senza modificare gli altri tier.
- **Flessibilità**:Ogni tier può essere modificato, aggiornato o sostituito indipendentemente dagli altri.

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 simplifies 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**: Questa parte gestisce i dati e le regole dell'applicazione, offrendo metodi per manipolare questi dati.
- **View**: È responsabile della presentazione visiva dei dati. Può esistere in molteplici forme, consentendo diverse rappresentazioni dell'informazione a seconda delle necessità dell'utente.
- **Controller**: Funziona come mediatore tra Model e View. Risponde agli eventi, come l'interazione dell'utente con l'interfaccia (ad esempio, la pressione di un pulsante), elaborando i dati nel Model che poi saranno riflessi nella View.

2.7 Other design decisions

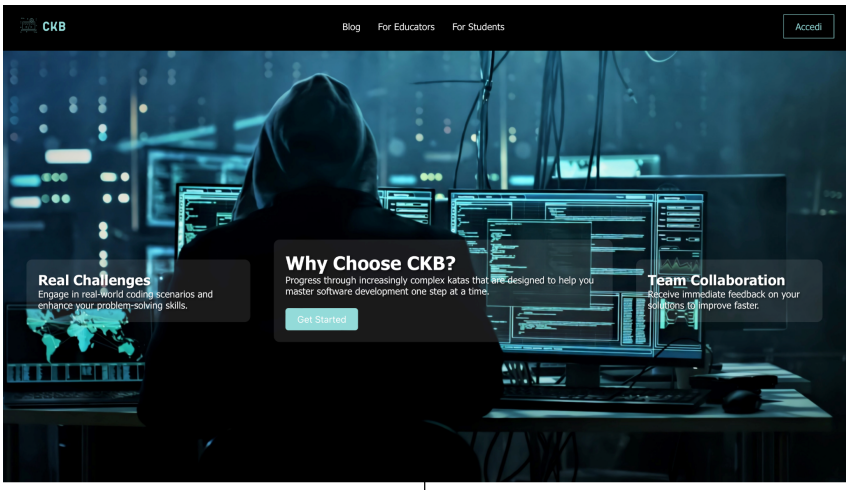
In this part, we discuss various design choices implemented in the system to ensure its optimal functioning.

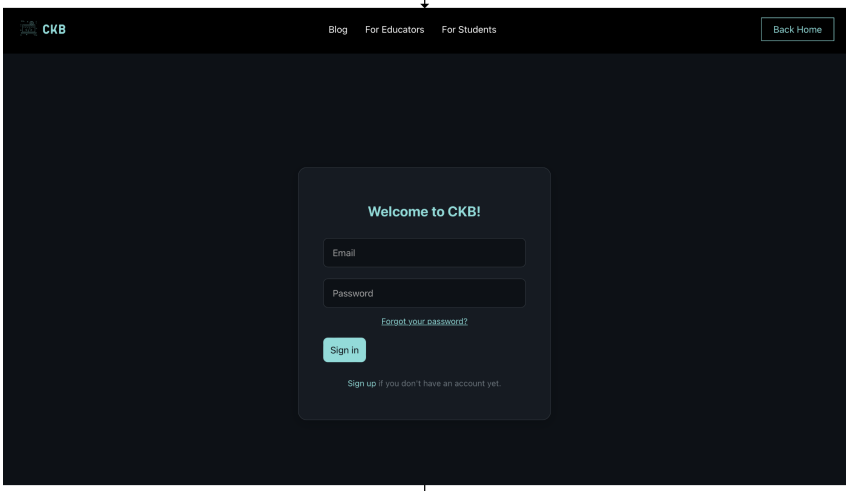
2.7.1 Availability

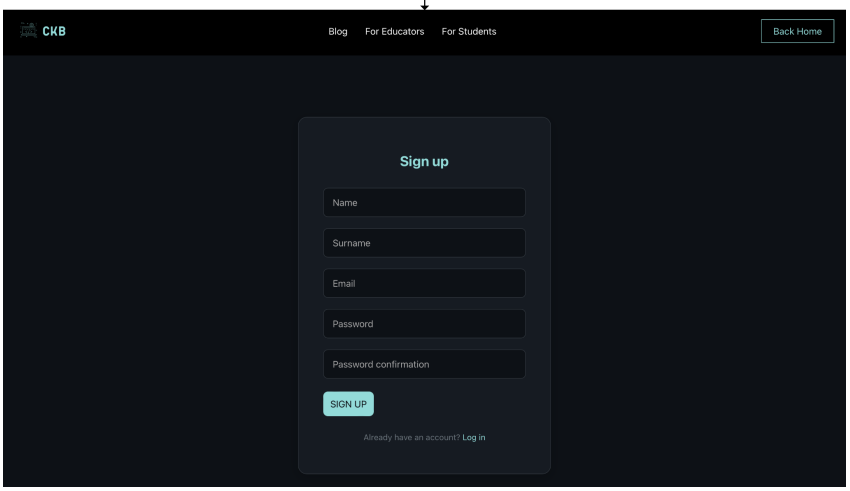
We have added load balancing and replication to make the system more reliable and to keep it running smoothly. This means if one part has a problem, the system can still work because it has backups and shares the work. This helps to make sure the system can handle data safely and stay available for use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

3 User Interface Desgin

These 3 mockups show the flow from the CKB homepage to the login page by clicking on "Sign up". Then from the login page by clicking on "Sign up" it is possible to access the registration 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:*
    - ClientManager:
      - SignupManager
    - NotificationManager
  - *EmailService*
  - *Model*
  - *DBMS*
- **R2**: The system must allow users to login.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - ClientManager:
      - LoginManager
  - *Model*
  - *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
  - *Model*
  - *DBMS*
- **R4**: The system must allow the educator to see which tournaments they have permission for.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Model*
  - *DBMS*
- **R5**: The system must allow educator to insert information about a tournament.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController

  - *Model*
  - *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
  - *Model*
  - *DBMS*
- **R7**: The system must notify via mail subscribed students to the platform upon the creation of new tournaments
  - *Application server:*
    - *NotificationManager*
  - *EmailService*
- **R8**: The system must allow students to subscribe to tournament on the CKB platform before a specific deadline.
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - TournamentController
  - *Model*
  - *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*
  - *EmailService*
- **R10**: The system must allow students to subscribe in the battle
  - *WebBroswer*
  - *WebServer*
  - *Application server:*
    - RequestManager:
      - BattleController
  - *Model*
  - *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
    - *Model*
    - *DBMS*
- **R12**: The system must notify via mail all students who are invited to form a team.
    - *Application server:*
        - *NotificationManager*
    - *EmailService*
- **R13**: The system must allow students to accept an invitation via email to join in a team.
    - *WebBroswer*
    - *WebServer*
    - *Application server:*
        - RequestManager:
            - BattleController
    - *Model*
    - *DBMS*
- **R14**: The system must allow student to decline the invitation via email to join in a team.
    - *WebBroswer*
    - *WebServer*
    - *Application server:*
        - RequestManager:
            - BattleController
    - *Model*
    - *DBMS*
- **R15**: The system must notify via mail the team's creator when an invited student declines his invitation.
    - *Application server:*
        - *NotificationManager*
    - *EmailService*
- **R16**: The system must notify via mail all members of a team when a new student joins their team.
    - *Application server:*
        - *NotificationManager*
    - *EmailService*
- **R17**: The system must allow user to see the list of available tournaments.
    - *WebBroswer*
    - *WebServer*
    - *Application server:*
        - RequestManager:
            - TournamentController
    - *Model*
    - *DBMS*
- **R18**: The system must allow user to see the list of available battles.
    - *WebBroswer*
    - *WebServer*
    - *Application server:*

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

  - *Application server:*
    - *AutomatedEvaluationManager*
  - *GitHub*
- **R26**: The system must calculate battle scores based on functional aspects, timeliness, and quality level of sources.
  - *WebServer*
  - *Application server:*
    - *AutomaticEvaluationManager*
  - *GitHubActions*
- **R27**: The system must update battle scores in real-time as students push new commits.
  - *Application server:*
    - *AutomaticEvaluationManager*
    - *RequestManager*
  - *GitHub*
  - *Model*
  - *DBMS*
- **R28**: The system must consolidate and finalize battle scores after the submission deadline and provide a final battle ranking.
  - *Application server:*
    - *AutomaticEvaluationManager*
    - *RequestManager*
      - RankingController
      - ManualEvaluationController
  - *Model*
  - *DBMS*
- **R29**:The system must notify via mail all students involved in a battle when final battle rank becomes available.
  - *Application server:*
    - *NotificationManager*
  - *EmailService*
- **R30**: The system must calculate and update tournament score at the end of each battle.
  - *Application server:*
    - *RequestManager*
      - RankingController
  - *Model*
  - *DBMS*
- **R31**: The system must create and update the tournament ranking.
  - *Application server:*
    - *RequestManager*
      - RankingController
  - *Model*
  - *DBMS*
- **R32**: The system must create the battle ranking when a new battle is created.
  - *Application server:*

        - *RequestManager*
            - RankingController
            - BattleController
    - *Model*
    - *DBMS*
- **R33**: The system must allow educators with permission on the tournament to close it
    - *WebBroswer*
    - *WebServer*
    - *Application server:*
        - RequestManager:
            - TournamentController
    - *Model*
    - *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
    - *Model*
    - *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
    - *Model*
    - *DBMS*
- **R37**: The system automatically deletes a battle when no teams have subscribed to it by the registration deadline.
    - *Application server:*
        - RequestManager:
            - BattleController
    - *Model*
    - *DBMS*

5 Implementation, Integration and Test Plan

5.1 Overview

Questo capitolo tratta l'implementazione del sistema, l'integrazione dei componenti e il piano di test. L'obiettivo dei test è individuare e correggere la maggior parte dei bug prima del rilascio dell'applicazione. Si sottolinea che l'implementazione e l'integrazione sono processi che vanno di pari passo; spesso l'ordine di integrazione segue quello dell'implementazione. Durante la definizione della strategia di implementazione, si considera anche il piano di test di integrazione.

5.2 Implementation Plan

Il piano di implementazione e testing del nostro sistema seguirà una strategia combinata bottom-up e thread, al fine di sfruttare i vantaggi di entrambi gli approcci.
Strategia Bottom-Up La strategia bottom-up si concentra sull'integrazione incrementale dei componenti del sistema, partendo dalle parti più basilari e procedendo verso le funzionalità più complesse. Questo metodo offre diversi vantaggi:

1. Facilità nel Tracking dei Bug: Poiché l'integrazione avviene progressivamente, è più semplice identificare e risolvere i bug nei moduli di base prima che essi influenzino componenti più ampi.
2. Test Incrementali: Ogni componente può essere testato man mano che viene integrato, permettendo di validare ogni parte del sistema in fasi successive.
3. Flessibilità e Adattabilità: Questo approccio permette una maggiore flessibilità, poiché le modifiche possono essere apportate più facilmente durante le prime fasi di sviluppo.

Strategia Thread La strategia thread, nel nostro contesto, implica l'identificazione e l'implementazione delle funzionalità del sistema tramite diversi thread di esecuzione. I vantaggi includono:

1. Consegne Intermedie Valutabili: Attraverso l'uso di thread, si possono produrre deliverable intermedi che possono essere valutati dai stakeholder, facilitando la validazione del sistema realizzato.
2. Parallelismo nello Sviluppo: Diverse funzionalità possono essere sviluppate in parallelo da diversi team, aumentando l'efficienza e riducendo i tempi di sviluppo.

5.2.1 Features identification

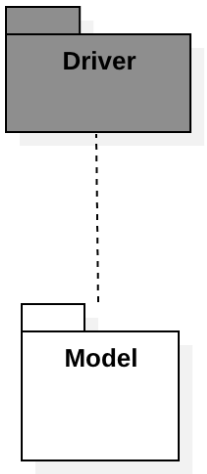
Nella fase di "Identificazione delle Funzionalità", elenchiamo e analizziamo le caratteristiche chiave del sistema. Questo processo è fondamentale per comprendere l'ordine di implementazione dei componenti, dato che alcune funzionalità dipendono da altre precedentemente implementate.
[F1] Sign in and sign up
Queste sono le funzionalità base che consentono agli utenti di accedere e utilizzare la piattaforma. Devono essere implementate per prime, in quanto sono il punto di ingresso per studenti ed educatori e sono prerequisiti per tutte le altre funzionalità.
[F2] Tournament management
Dopo aver stabilito l'accesso degli utenti, il passo successivo è consentire agli educatori di creare e gestire tornei. Questa funzionalità è fondamentale perché struttura l'intero contesto in cui si svolgono le battle. Senza la gestione dei tornei, non sarebbe possibile organizzare e coordinare le sfide di programmazione.
[F3] Battle management
Una volta che i tornei sono configurati, è necessario implementare la gestione delle singole battaglie. Questo include l'upload dei code kata, la definizione dei team, e la configurazione delle scadenze. Le battaglie sono l'elemento centrale dell'interazione degli studenti sulla piattaforma e dipendono dalla previa configurazione dei tornei.
[F4] Evaluation
Con le battaglie in corso, è fondamentale avere un sistema di valutazione che analizzi le soluzioni inviate dagli studenti. Questo processo include la verifica delle soluzioni rispetto ai test case e la valutazione manuale da parte degli educatori. La funzionalità di valutazione deve seguire la gestione delle battaglie, in quanto è essenziale per determinare i punteggi delle squadre.
[F5] Ranking management
Dopo la valutazione delle battaglie, è necessario gestire i ranking sia delle singole battaglie che dei tornei complessivi. Questo permette agli studenti e agli educatori di vedere le prestazioni, creando un contesto competitivo. Questa funzionalità dipende dalla valutazione delle battaglie e dalla gestione dei tornei e

delle battaglie.
Notifications management
This is the feature that manages the notifications for the user. A differenza delle altre funzionalità, la gestione delle notifiche sarà sviluppata e raffinata continuamente durante l'intero processo di sviluppo. Non seguono un ordine fisso ma sono integrate e adattate in base al progresso delle altre funzionalità.

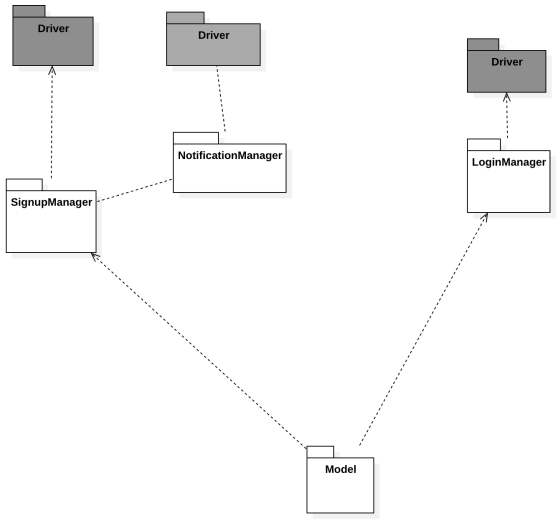
5.3 Component Integration and Testing

Nella sezione successiva, mostreremo per ogni fase quali componenti vengono implementati, come avviene l'integrazione di questi componenti e le modalità di test a cui sono sottoposti, supportati da un diagramma di integrazione specifico per ciascuna fase.

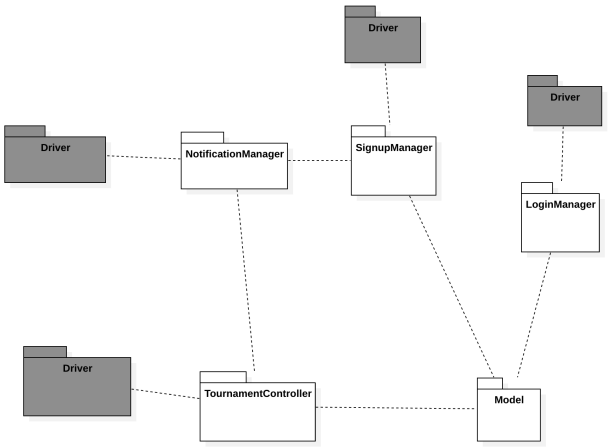
The first step is to implement the Model, which is the component that implements all methods that allow access to the Database and perform queries and updates on it. Since we are following a bottom up approach, we will use drivers to test the integration of the components, as it can be seen from the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



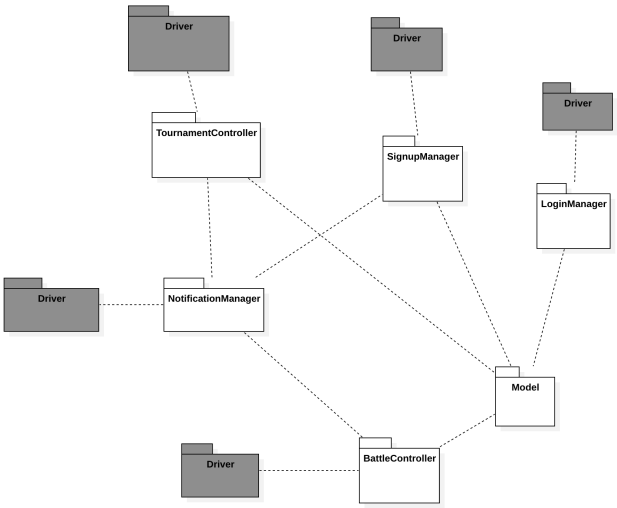
[F1] Sign in and sign up developing Subsequently, we focus on building out the authentication mechanism first and part of the User notification component because the SignUpManager uses it to send email to users.



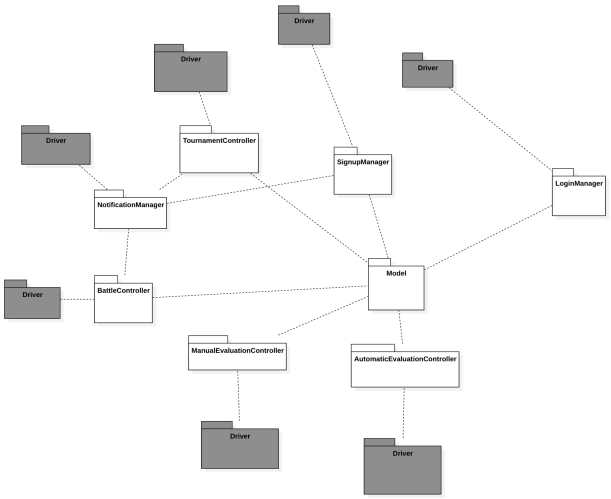
[F2] Tournament management Following the authentication setup, we turn our attention to the tournament management aspect, since it serves as a foundation for subsequent functionalities that hinge on its proper operation. Alongside this, a segment of the Notification Manager is also implemented. This integration is vital as it plays a key role in tournament management, particularly in notifying subscribed users about tournament creation.



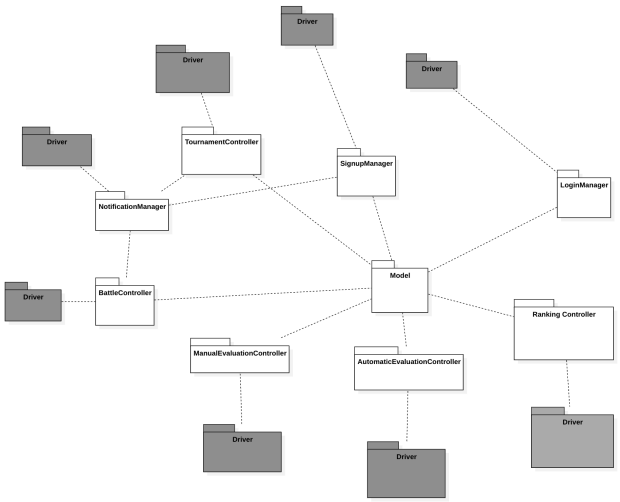
[F3] Battle management Our next focus is on developing the battle management system.



[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

Dopo l'integrazione dei componenti nel sistema CKB, procediamo con una serie di test. L'obiettivo è garantire che le funzionalità rispondano alle aspettative e che l'applicazione si comporti adeguatamente sotto diversi carichi di lavoro. Si delineano quindi vari ambiti di test, ognuno focalizzato su specifiche esigenze:

- **Functional Testing**: Per verificare che ogni singola funzione della piattaforma, dalle notifiche agli aggiornamenti dei punteggi, operi correttamente secondo le specifiche.
- **Performance Testing**:Valuta la velocità, la reattività e la stabilità dell'applicazione sotto vari carichi di lavoro. E' importante per assicurare che le performance dell'applicazione siano all'altezza delle sfide, specialmente quando la piattaforma calcola in tempo reale i punteggi delle competizioni, elemento centrale dell'esperienza utente su CKB.
- **Load Testing**:Verifica la capacità dell'app di gestire un volume elevato di utenti o transazioni simultaneamente.
- **Stress Testing**: Per spingere il sistema oltre i limiti operativi abituali e osservare la sua capacità di mantenere l'integrità e la sicurezza delle operazioni anche in condizioni estreme.

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

5.5 Acceptance Testing

Successivamente ai test descritti in precedenza, procederemo con il testing di accettazione da parte dell'utente. Questo passaggio è cruciale: anche se abbiamo verificato il rispetto dei requisiti funzionali e non-funzionali, resta indispensabile ottenere il benestare finale da parte del committente.

Il testing di accettazione si svolge seguendo un approccio black-box, che coinvolge gli utenti in interazioni con il sistema che riflettono l'utilizzo reale. Questo metodo consente di confermare se il software soddisfa le specifiche desiderate in origine dal cliente. È un controllo definitivo che valida la piena corrispondenza del prodotto alle aspettative e alle necessità per cui è stato commissionato.

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