Accademic Year: 2023-2024

POLITECNICO
MILANO 1863

Design Document
CodeKataBattle

SOFTWARE ENGINEERING 2
PROFESSOR DI NITTO ELISABETTA

CONTI ALESSANDRO CODICE PERSONA: 10710583 MATRICOLA: 252665
DI PAOLA ANTONIO CODICE PERSONA: 10717589 MATRICOLA: 956038

Contents

1. INTRODUCTION

1.1. Overview

This Document is intended to provide a comprehensive overview of the CodeKataBattle project. This documentation will be a guide for the reader, enabling them to understand the decisions regarding the design of the application. In particular, the architectural structure and design choices will be detailed, allowing developers, stakeholders, and other team members to gain a clear understanding of how all the features previously outlined informally in the RASD will be implemented.

1.2. Purpose

The purpose of our project, named CodeKataBattle (abbreviated as CKB), is to provide a digital platform that enables students to develop and refine their skills in software development through a collaborative experience. The CKB platform allows students to practice in a programming language of their choice. The exercises offered on this platform are created by educators and require students to complete the code or parts of it, adding the missing components they deem appropriate. The code is then executed and subjected to specific tests created by educators to verify its correctness.

Exercises are organized into tournaments by educators, within which 'battles' are held to evaluate the performance of the students. This process culminates in the creation of performance rankings among all students involved in the battles and tournaments. In this way, CKB encourages friendly competition and the growth of students' skills in the field of software development.

1.3. Scope

The CKB application, as defined in the RASD document, is a web application focused to enhancing software development skills for users enrolled in the Students category. Educators, Educators, who have in-depth expertise in the topic, will manage tournaments and battles within the platform to facilitate the improvement of students' skills. Through detailed rankings of battles and tournaments, students can assess their level and monitor progress in enhancing their skills.

1.4. Definitions, Acronyms, Abbreviations

1.4.1. Definitions

- *Commit*: Commits are the core building block units of a Git project timeline. Commits can be thought of as snapshots or milestones along the timeline of a Git project. Commits are created with the `git commit` command to capture the state of a project at that point in time.

- *Fork*: A fork is a new repository that shares code and visibility settings with the original “upstream” repository.
- *Code kata*: Code kata contains the description of a battle and the software project on which the student will have to work, including also test cases and build automation scripts
- *Test Case*: A test case is a singular set of actions or instructions that the Educator wants to perform to verify a specific aspect of the project pushed by the students. If the test fails, the result might be a software defect that students might have not found.
- *Upload*: Upload in which the educator sends to the platform database the code kata for a specific battle.
- *Repository*: A Git repository is the .git/folder inside a project. This repository tracks all changes made to files in your project, building a history over time. Meaning, if you delete the .git/folder, then you delete your project’s history.
- *Branch*: In Git, a branch is a new/separate version of the main repository. Branches allows users to work on different parts of a project without impacting the main branch. That main branch is the one seen as the default one.
- *Push*: The `git push` command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo.
- *Pull*: The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content. Merging remote upstream changes into your local repository is a common task in Git-based collaboration work flows.
- *Demilitarized Zone*: A demilitarized zone is a physical or logical subnet that contains and exposes services to an external network that is not considered secure, such as the Internet. The purpose of a DMZ is to protect an organization's LAN.

1.4.2. Acronyms

- CKB: CodeKataBattle
- CK: CodeKata, Code Kata
- GH: GitHub
- GHA: GitHub Action
- GDPR: General Data Protection Regulation
- DMZ: Demilitarized Zone
- DAO: Data Access Object
- RASD: Requirements Analysis and Specification Document
- DD: Design Document

1.4.3. Abbreviations

- [Gn]: the n-th goal of the system
- [WPn]: the n-th world phenomena
- [SPn]: the n-th shared phenomena
- [Sn]: the n-th scenario
- [DAn]: the n-th domain assumption
- [Dn]: the n-th dependency
- [Cn]: the n-th constraint
- [Rn]: the n-th functional requirement
- [UCn]: the n-th use case
- [SDn]: the n-th sequence diagram

- Repo: Git repository

1.5. Revision history

- Version 1 ()

1.6. Reference documents

This document is strictly based on:

- The specification of the RASD and DD assignment of the Software Engineering 2 course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2022/2023.
- Slides of Software Engineering 2 course on WeBeep.
- Official link of CodeKata: http://codekata.com/.

1.7. Documents Structure

The rest of the document is organized as follows:

- *Architectural Design* (Section 2) contains a detailed description of the system architecture, the definition of the main components and the relationships between them.
  The last part of the section will describe the design choices, models and paradigms used in the implementation.
- *User Interface Design* (Section 3) contains a detailed description of the user interfaces, which are presented in the RASD in the form of an overview.
- *Requirement Traceability* (Section 4) shows the relations between the requirements from the RASD and the design choices of the DD and how they are satisfied by the latter.
- *Implementation, Integration and Test Planning* (Section 5) provides a road-mapping of the implementation and integration process of all components and explains how the integration will be tested

2. ARCHITECTURAL DESIGN

2.1. Overview

In this section we will explain what components will compose our system, the interaction between them, and the replication mechanisms chosen to make the system distributed.

2.1.1. High level view

The CKB application will be implemented following the idea of a three-tier architecture as shown in Figure 1.

The three tiers are the follow:

1. *Presentation Tier*: This tier allows interaction between the user and the application through a user interface based on web pages. These pages dynamically adapt to user requests and application responses.
2. *Application Tier*: This tier handles user requests, retrieves and, if necessary, modifies information in the database.
3. *Data Tier*: This tier constitutes the main database. Its primary function is to store data securely and make it available when requested by the user.

The three-tiered distributed architecture allows efficient division of responsibilities, making it easier to scale and manage components.

The Presentation Tier handles the user interface, the Application Tier handles the application logic, and the Data Tier handles persistence and data access. This subdivision facilitates maintenance, scalability, and optimization of overall system performance.

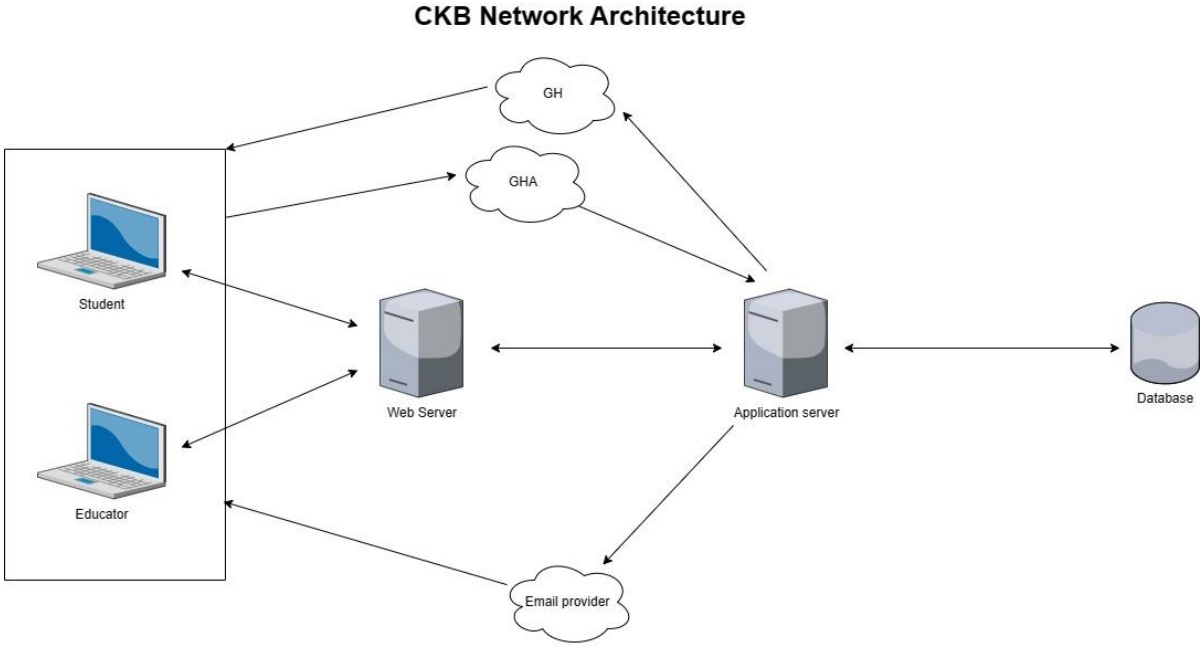


Figure 1 High Level View

2.1.2. Distributed view

The CKB application will be implemented following the idea of distributed system architecture as shown in Figure 2

The components that make it possible to have this programming paradigm are the follow:

- *Firewall*: a network security device that monitors incoming and outgoing traffic through a predefined set of security rules, deciding whether to allow or block certain events. The firewall then acts as a filter for all user requests, allowing only those authorized according to specific access permissions to pass through.
- *Load Balancing*: a system that fairly distributes the workload it receives among various hardware resources in order to optimize system performance. Load balancing aims to ensure fair allocation of user requests across different machines, thus helping to improve service availability so that it is more efficient.

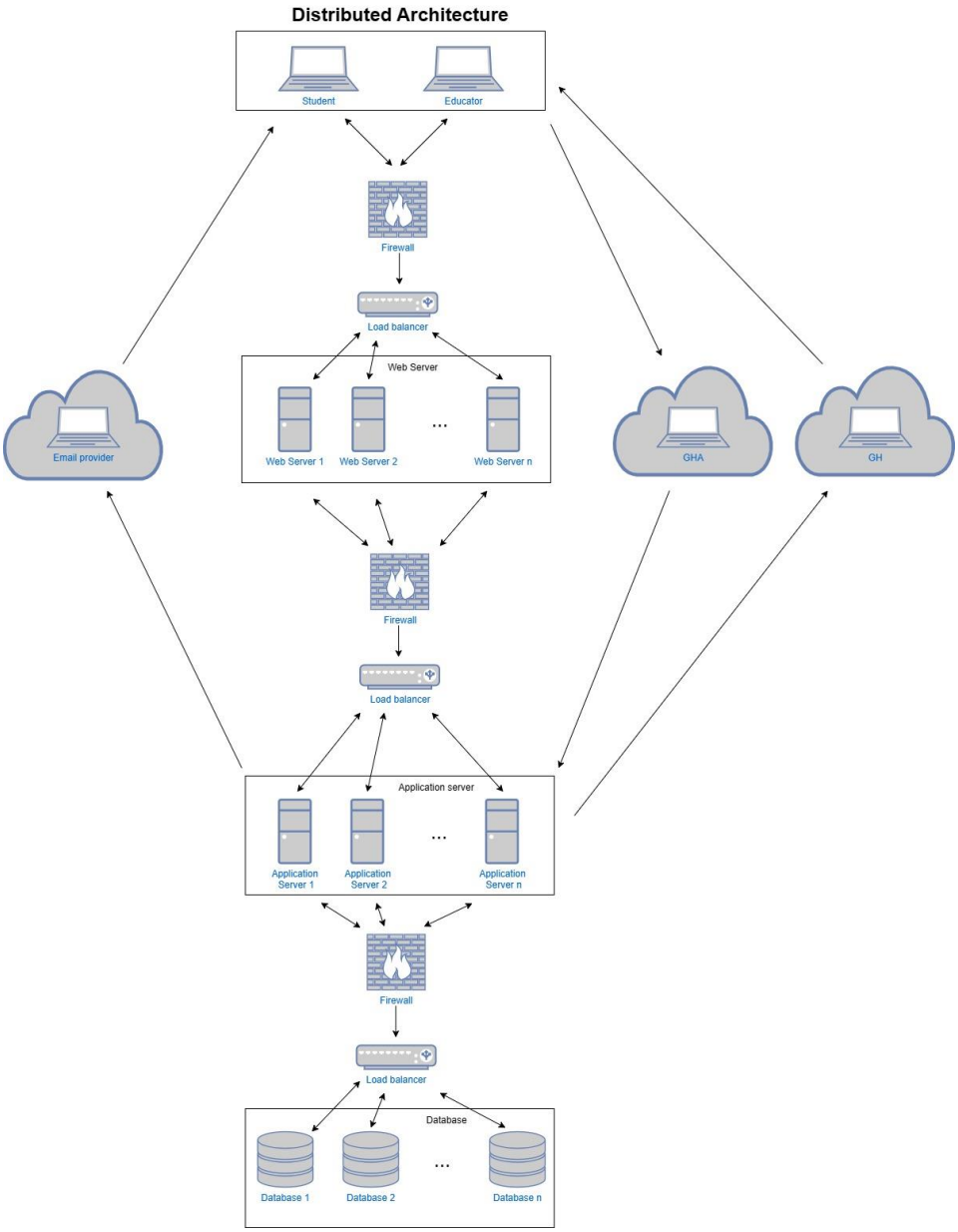


Figure 2 Distributed View

2.2. Component view

This section will show the component view of the system representing the internal architecture of the CKB application.

The diagram we see in figure 2 allows us to identify the three architectural levels of the application, as described above.

1. The presentation tier, represented by our *Web Site*, deals with dynamic interaction with the user. Whenever a change to the web page occurs, this layer of the architecture communicates with the *Web Server*. This interaction ensures a timely and accurate response to user requests, helping to provide a dynamic web experience aligned with user interactions. In this process, the presentation tier acts as a key interface between the user and the system, ensuring that changes made to the page are reflected consistently and immediately on the website that the user views.
2. The application tier, jointly represented by the *Web Server* and the *Application Server*, handles requests from users and interacts with the database containing the application data.

   When a user sends a notification to the server, the *Web Server*, via the web page displayed to the user, forwards the request to the *Application Server* in the appropriate format. The *Application Server* is the key component that, upon receiving the request in the correct format, forwards it to the database to retrieve and/or modify the data within. This process constitutes an effective communication flow that allows user requests to be fulfilled efficiently and consistently, while ensuring proper access and manipulation of data in the database.
3. The data tier is represented by the *database*, which is responsible for storing and making accessible the data essential to the proper functioning of the application.

   The *database* ensures that the information stored in it can be accessed by the *Application Server* whenever it is necessary to retrieve and/or modify data.

All the components that will compose the system will be explained in detail below.

- Web Site

  This component takes care of user interaction.
- Web Server

  All of these components handle the pages that user sees and can interface with.
  - Home Page

    On this page, the user is given a choice between two options: registration or access to the application. Selecting one of these options automatically causes the corresponding page to change.
  - Login Page

    This component handles the login page.
  - Sign In Page

    This component handles the sign in page.
  - Personal Home Page

This component is responsible for managing the home page.

This page shows the specific home page of each specific user.

- Tournament Page

  This component handles the tournament page.

- Battle Page

  This component handles the battle page.

- Application Server

  This component takes care of the interaction between the user and the database.

  - Servlet Login Manager

    This component handles the login process of a user. It checks whether the user is registered in the database and, if so, redirects the web page to its designated home page.

  - Servlet Sign In Manager

    This component handles the registration process of a user. It checks whether the user is not registered in the database and, if so, saves the user.

  - Servlet Show Tournament Educator

    This component is responsible for searching the database in order to retrieve all the tournaments associated with the educator who logged in. It then displays those tournaments on the home page.

  - Servlet Show Tournament Student

    This component is responsible for searching the database in order to retrieve all the tournaments associated with the student who logged in. It then displays those tournaments on the home page.

  - Servlet Create Tournament

    This component is responsible for creating a tournament and saving it to the database.

  - Servlet Show Notification

    This component is responsible for showing notifications in the application.

  - Servlet Search Tournament

    This component is responsible for searching the database for tournaments that meet the characteristics entered by the user.

  - Servlet Log Out

    This component handles the logout of the user who had logged in.

  - Servlet Modify Personal Information

This component is responsible for modifying the user's personal data in the database.

- Servlet Show Tournament Information

  This component is responsible for searching the database for tournament information and displaying it to the user.

- Servlet Create Battle

  This component is responsible for creating a battle and saving it to the database.

- Servlet Close Tournament

  This component is responsible for closing a tournament and to modify the tournament in the database.

- Servlet Show Battle Information

  This component is responsible for searching the database for battle information and displaying it to the user.

- Servlet Join Battle

  This component is responsible for adding a team to the selected battle, to do this the database is modified.

- Servlet Join Team

  This component is responsible for adding a student to the selected team, to do this the database is modified.

- User DAO

  This component handles all interactions with the database that require access to user information.

- Tournament DAO

  This component handles all interactions with the database that require access to tournament information.

- Battle DAO

  This component handles all interactions with the database that require access to battle information.

- Team DAO

  This component handles all interactions with the database that require access to team information.

- Evaluation

  This component is responsible for grading the work of a team of students according to the requests of the educator who created the battle.

- Send Notification

This component is responsible for interacting with the email provider in order to send notifications to interested users.

The notifications it needs to send are created by the following components: *Servlet Create Tournament*, *Servlet Create Battle*, *Servlet Close Tournament* and *Servlet Join Battle*.

- Database

  This component is responsible for saving the data and making it accessible to each user request.

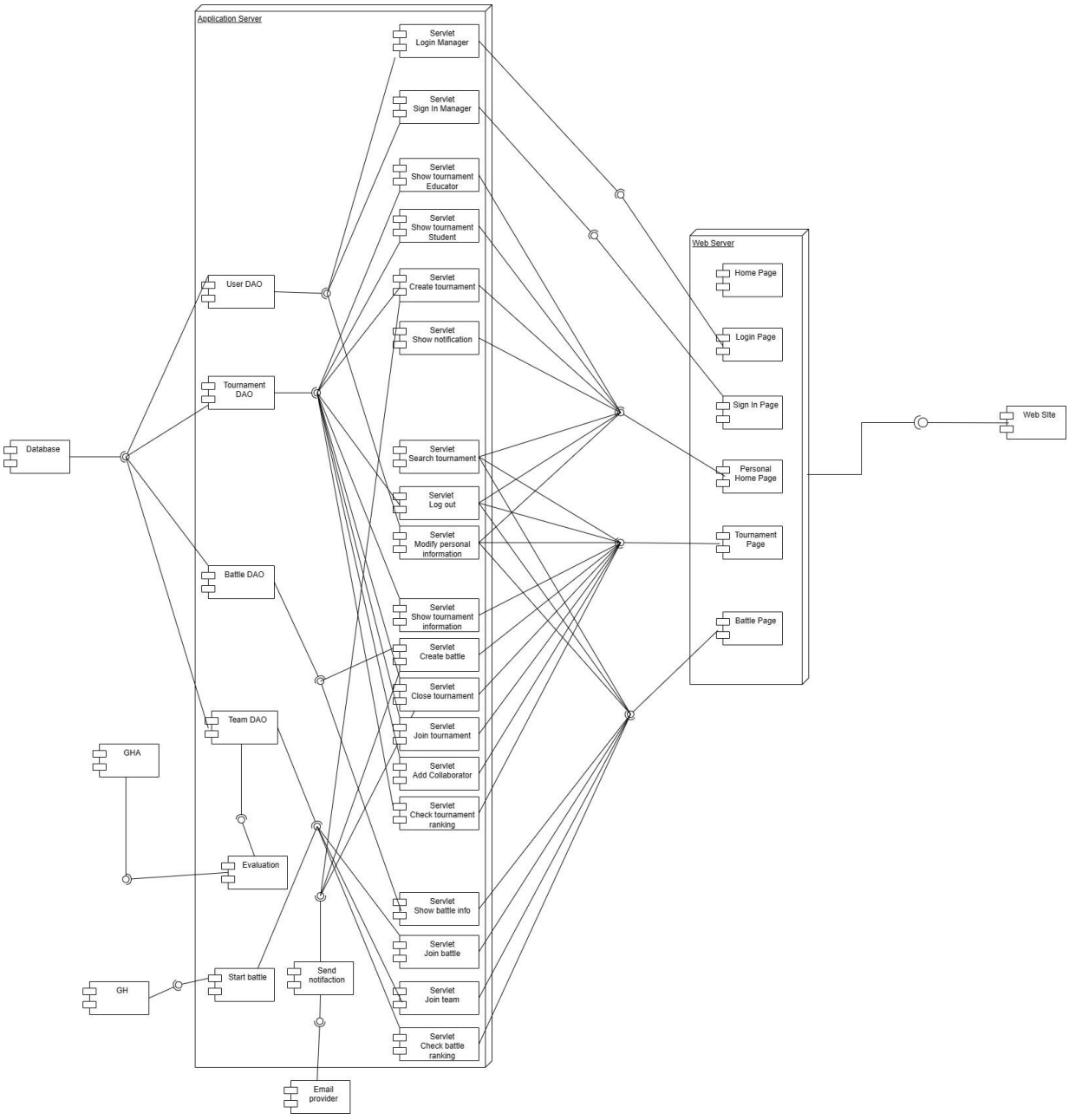


Figure 2 Component View

2.3. Deployment view

In this chapter is described the deployment view for CKB. This view describes the execution environment of the system and the geographical distribution of the hardware components that executes the software composing the application.

Following this, all the components displayed in the figure 3 will be explained in detail.

- *PC Student* and PC *Educator* represent a computer used by a Student and a Educator. The two devices must have a web browser capable of interpreting the Javascript language. This browser is indispensable for browsing the Internet, enabling access to the CKB web page and fluid interaction with the application.
- *Firewall* is a device that monitors packets entering the system, if a packet is potentially dangerous it is not is potentially dangerous it is not forwarded. These components are placed before load balancers, with the goal of carefully filtering all packets entering the network. This configuration ensures that only packets deemed safe are allowed to enter the application's network environment. Through this implementation, a DMZ is established in which security is maximized through strict access control, thus helping to protect the LAN’s application from potential external threats.
- *Load Balancer* is a device which performs the load balancing. This component deals with dividing a group of requests among several machines that specialize in their execution. This approach aims to optimize the overall performance and increase the scalability of the system. Through the intelligent distribution of requests, the component helps ensure efficient use of available resources, greatly improving the responsiveness of the system and its ability to handle increasing workloads in a flexible manner.
- *Web Server* is the component that receives all user requests and passes them to the *Application Server* to execute and return a response. This component is the one responsible for changing the pages or content that the user sees on the screen.
- *Application Server* is the component that receives all user request, processes a response and sends it to the *Web Server* to display. This component is the only one that can process the data, read it and modify it on the database.
- *Database Server* is the component that saves and accessibility the data.

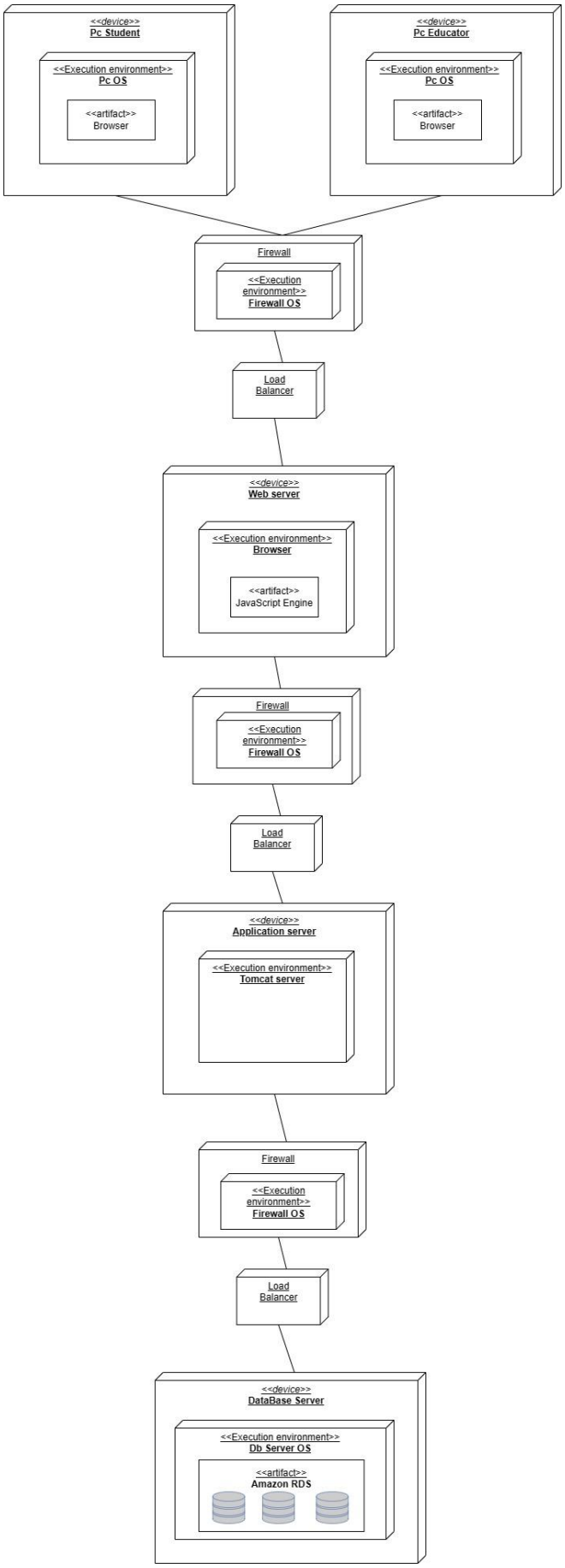


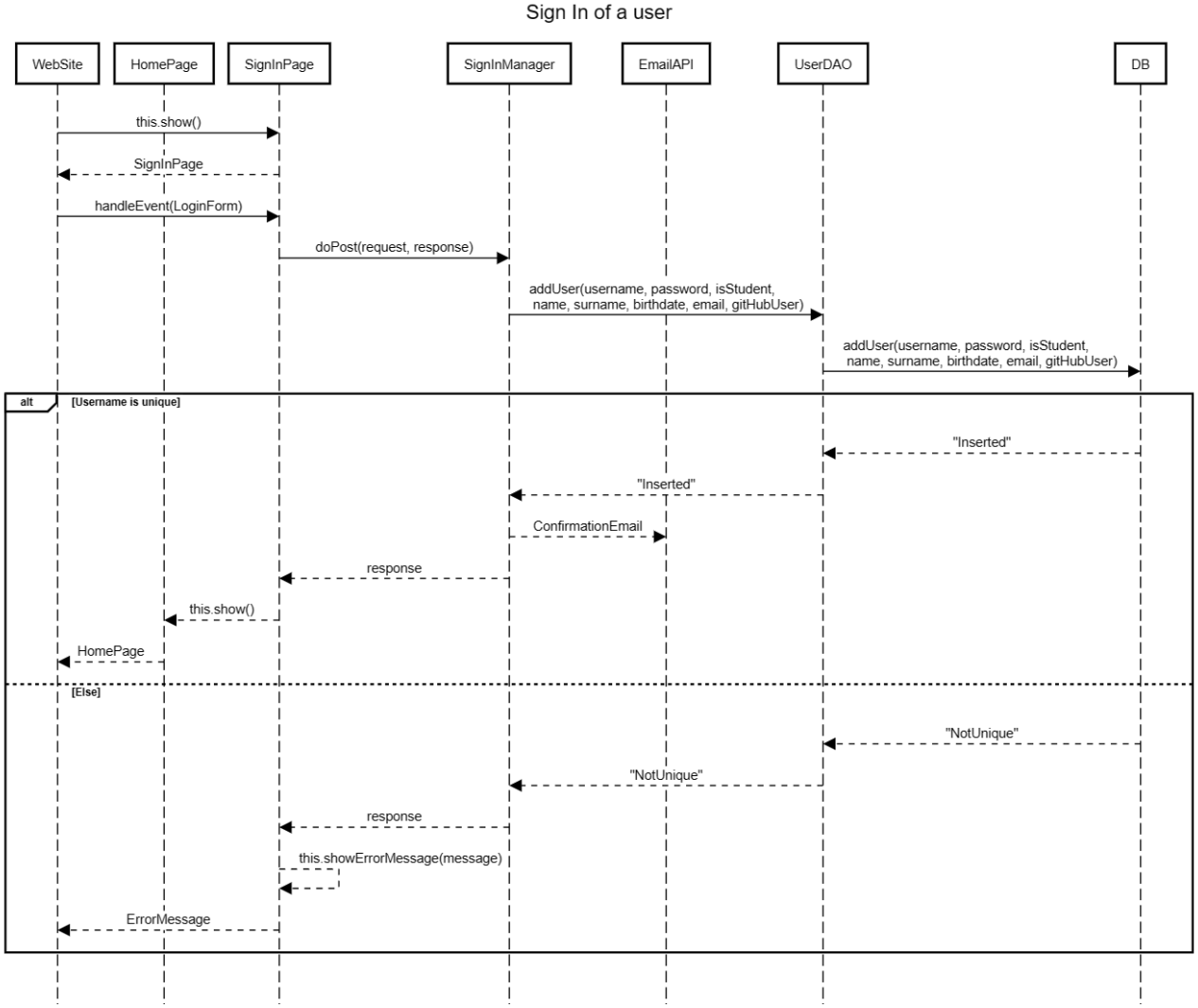
Figure 3 Deployment View

2.4. Runtime view

This section will present all the runtime views associated with the Use Cases found in the RASD. The runtime views show the interactions between the various components of the CKB system.

[SD1]. Sign in of a new user

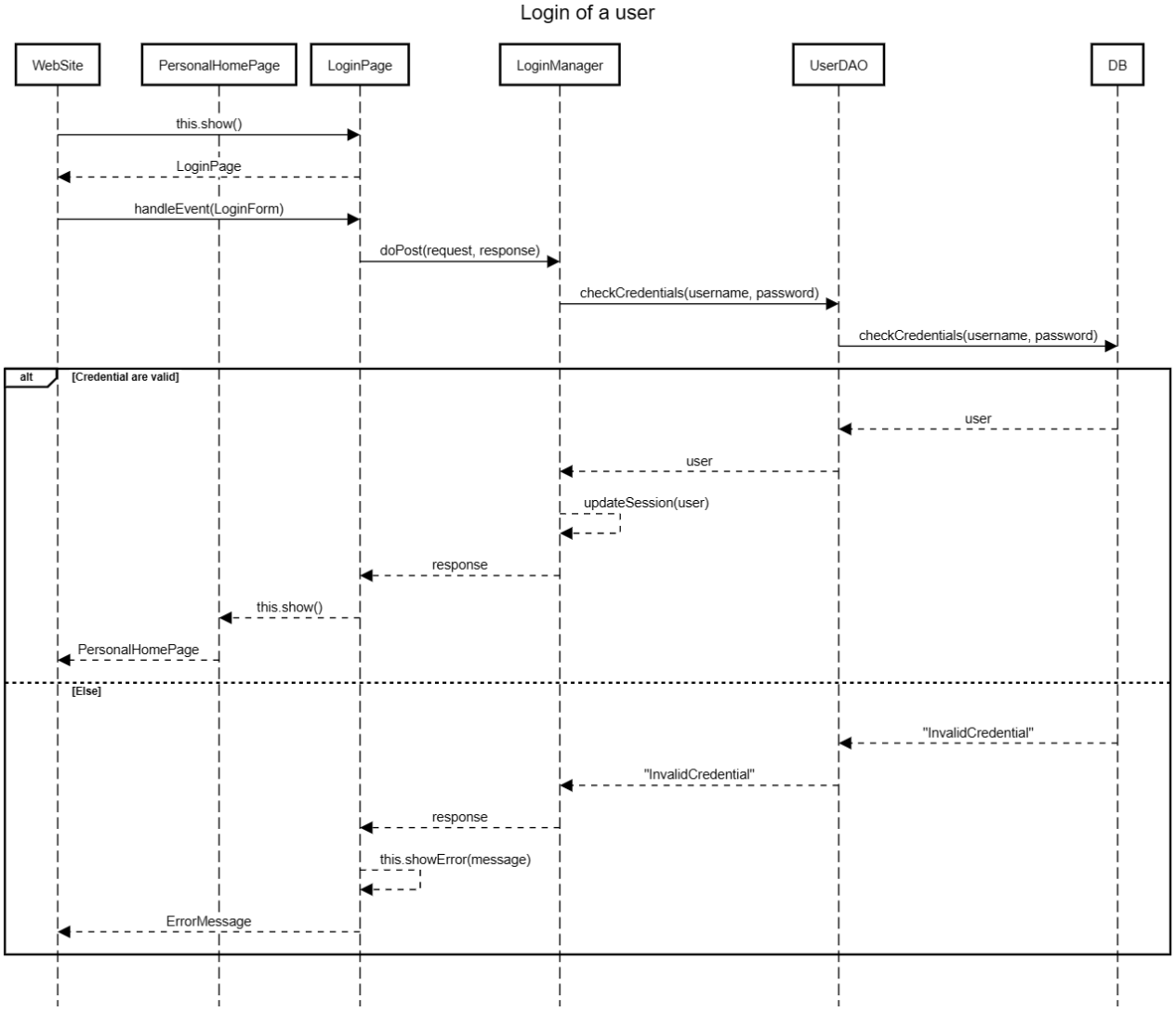
This sequence diagram represents the interaction that occurs when a user wants to register to the CKB application.
When a user wishes to register in the application, they are shown a page to fill in with some personal data. After entering all the required information and pressing the *Submit* button, the data is transmitted to the server responsible for managing the saving and a confirmation email will be sent to them for the successful creation of the account.
In case an error occurs, for example, if the user has already created an account with the same email or username, an error page is displayed. In this scenario, saving the data to the database and sending the confirmation email are not performed.



[SD2]. Login of a user

This sequence diagram represents the interaction that occurs when a user wants to access the CKB application.
When a user wishes to access the application, they enter their login credentials, which are then sent to the server for verification. If the verification is successful, the user is granted access permission and the application will show their customized Home Page. In case the

credentials are incorrect, the user will display an error message prompting thier to provide the correct credentials.

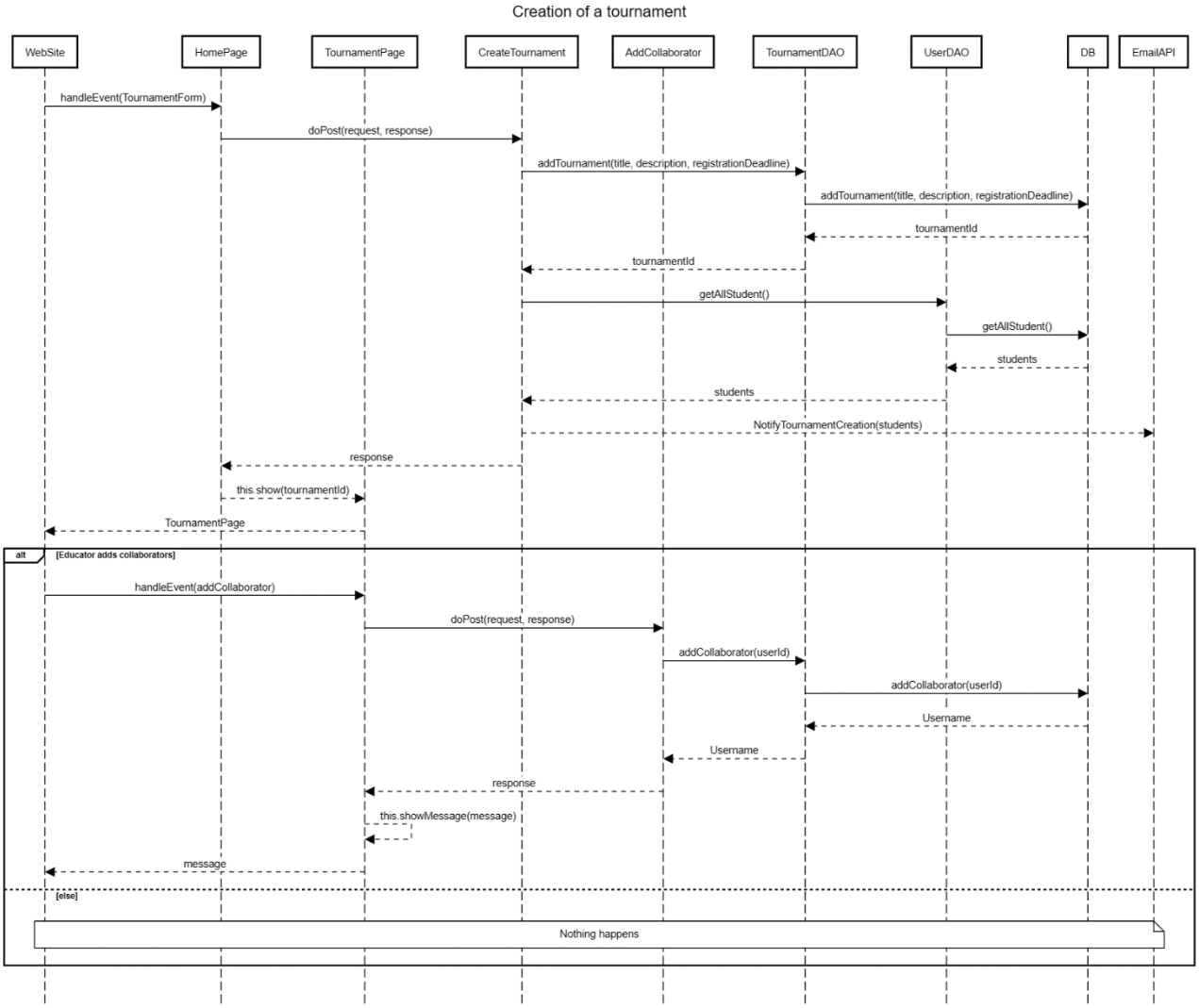


[SD3]. Creation of a tournament

This sequence diagram represents the interaction that occurs when a Educator wants to create a tournament in the CKB application.

When an Educator wishes to create a tournament, they fill in the required fields and send the request to the server, which saves it and sends an email notification to all students enrolled in the application to inform them of the creation of the new tournament. Next, the application shows the educator the page dedicated to the newly created tournament.

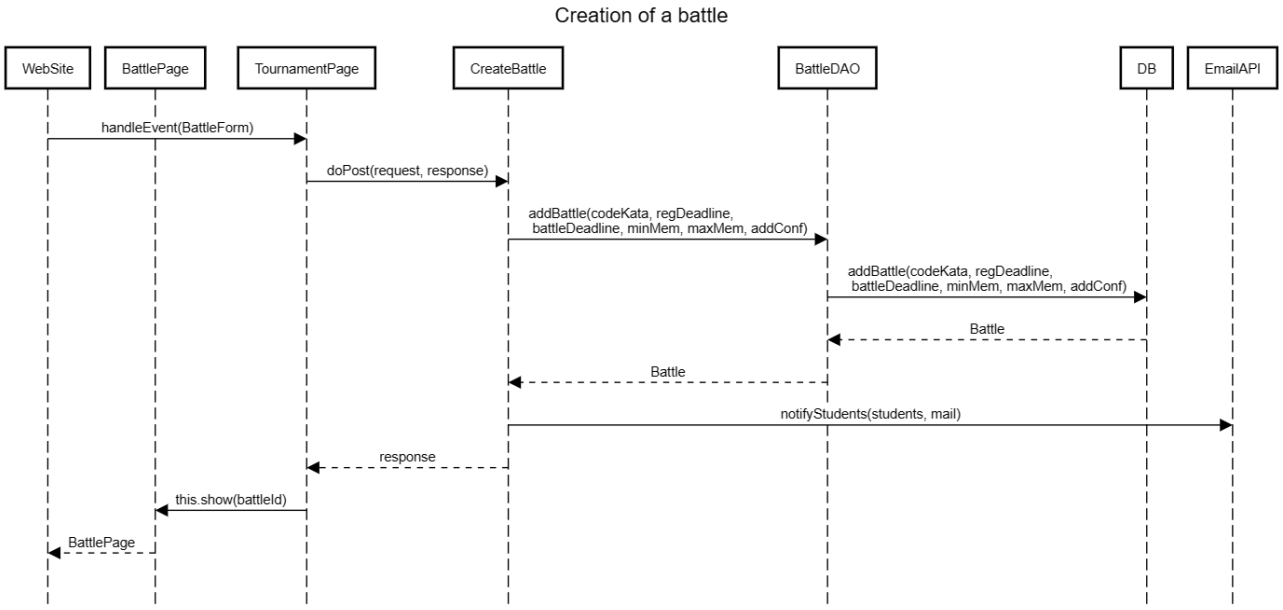
If the educator would like to add other educators as collaborators in managing the tournament, they can easily do so by clicking on the *Add Collaborator* button on the newly created tournament page and selecting the collaborators they would like to include.



[SD4]. Creation of a battle

This sequence diagram represents the interaction that occurs when an Educator wants to create a battle in the CKB application.

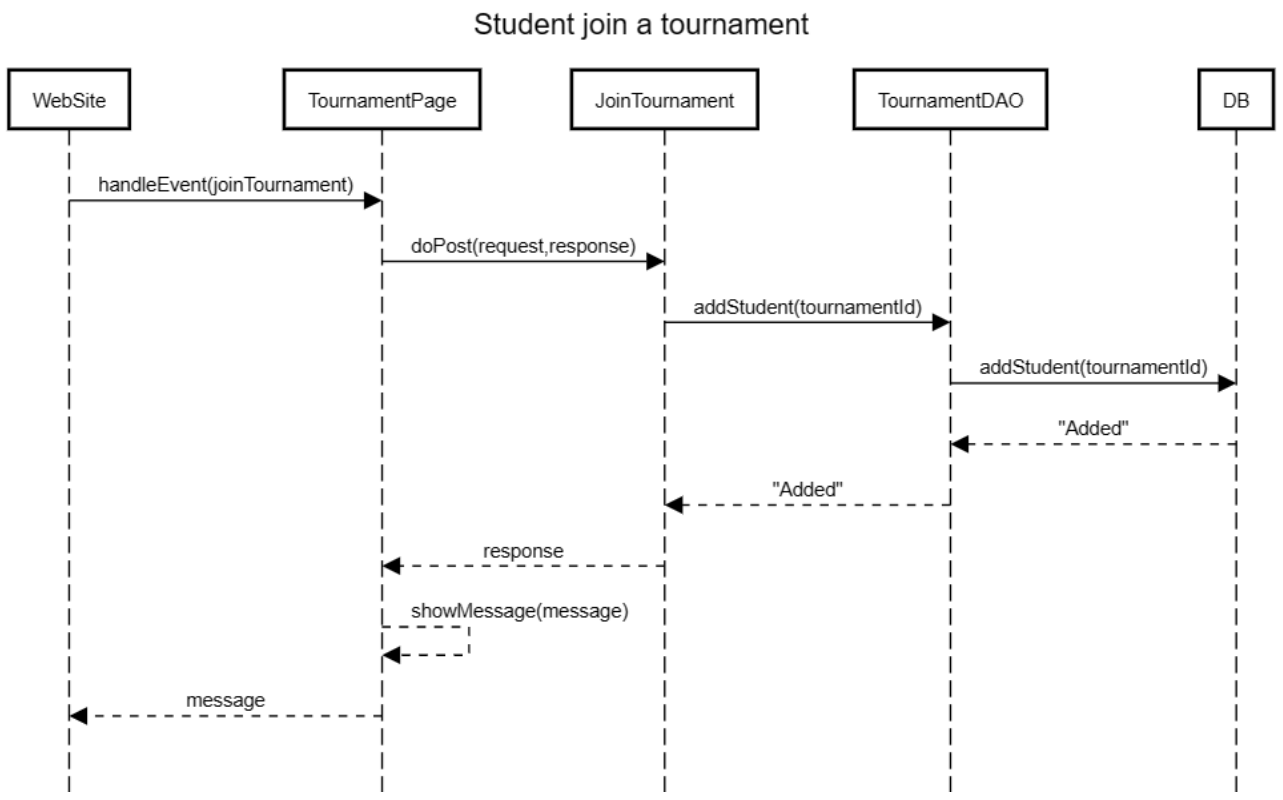
When an educator wishes to create a battle within a specific tournament, they fill in the required fields and sends the request to the server, which saves it and sends a notification via email to all students enrolled in the tournament to inform them of the creation of a new battle. Next, the application shows the educator the page dedicated to the newly created battle.



[SD5]. Student joins a tournament

This sequence diagram represents the interaction that occurs when a Student wants to join in a tournament in the CKB application

When a student wants to participate in a tournament, they access the page of the tournament of interest and click the *Join Student* button. Once the application displays an "Added" message, the student is officially enrolled in the tournament.

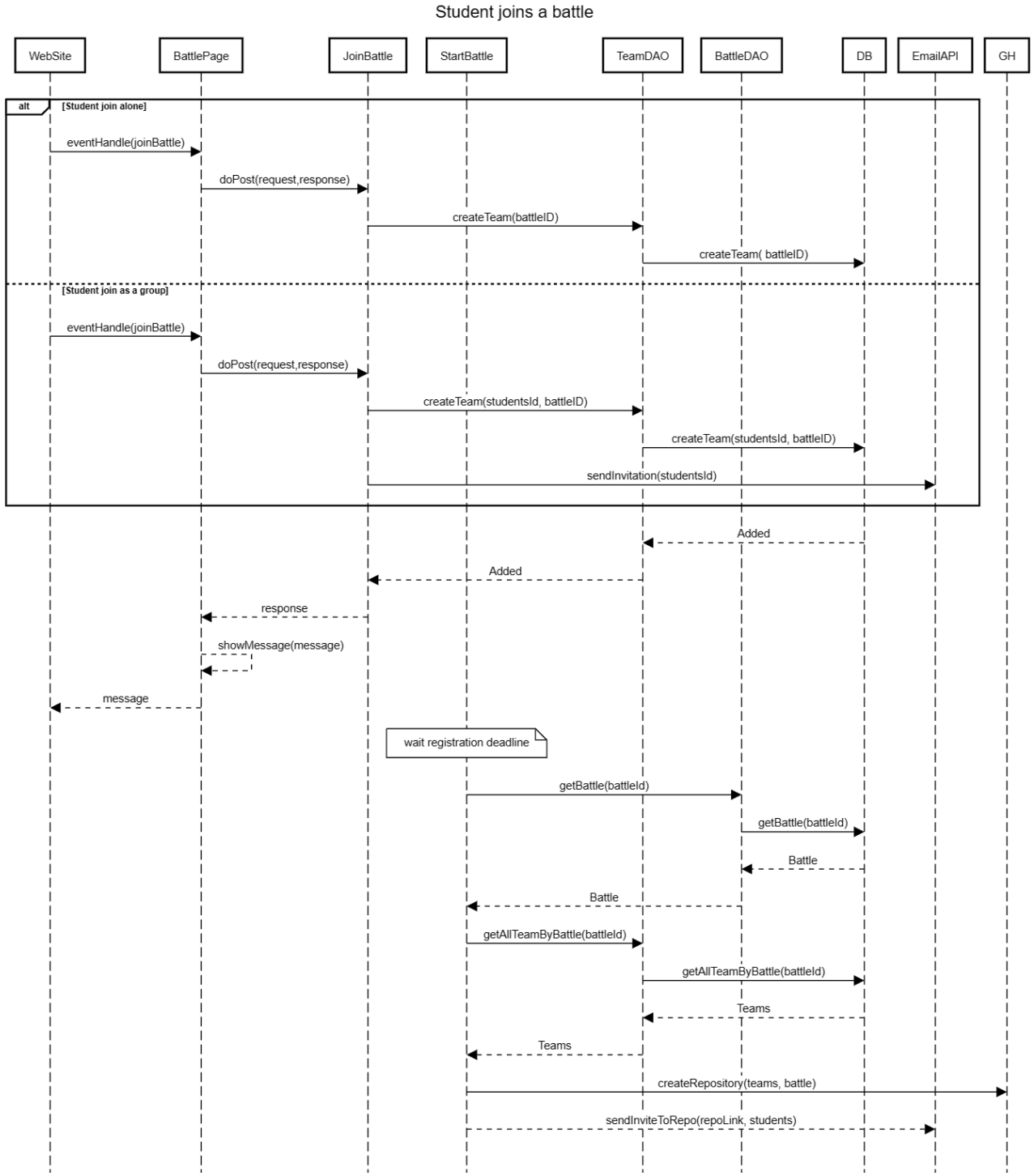


[SD6]. Student joins a battle

This sequence diagram represents the interaction that occurs when a Student wants to join in a battle in the CKB application

When a student wishes to participate in a battle, they access the page of the battle they are interested in. Within this page, there are two buttons: the first allows individual entry, if the battle allows it, while the second allows entry as a team. If the student opts for the second option, they will have to select the other students who will form their team.

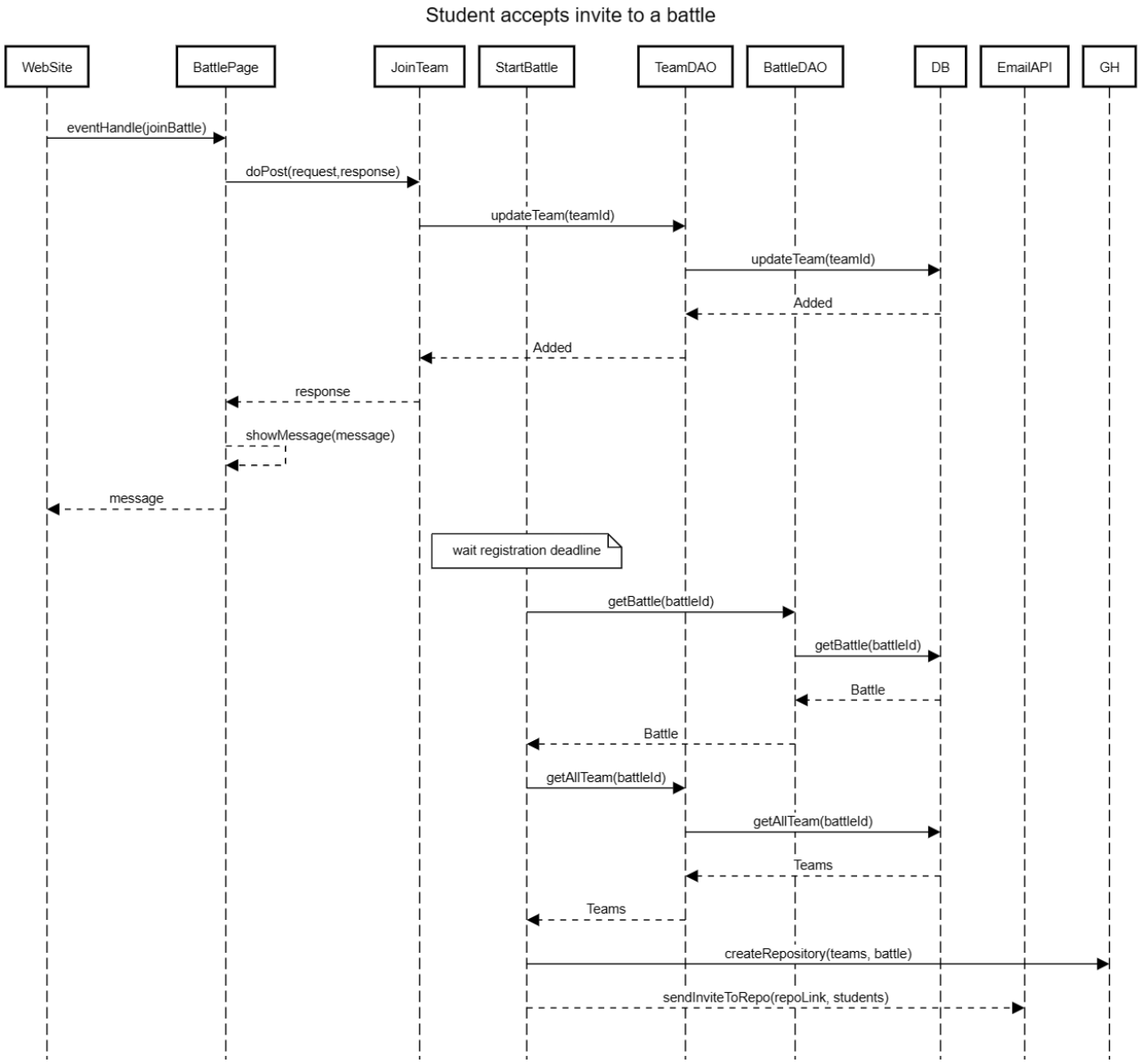
After confirming the team members, the application will save the team in the database and send an email notification to all selected students, informing them of the possibility of participating in the battle together with the student who selected them.



[SD7]. Student accepts an invite for a battle

This sequence diagram represents the interaction that occurs when a Student wants to accept an invite for a battle in the CKB application

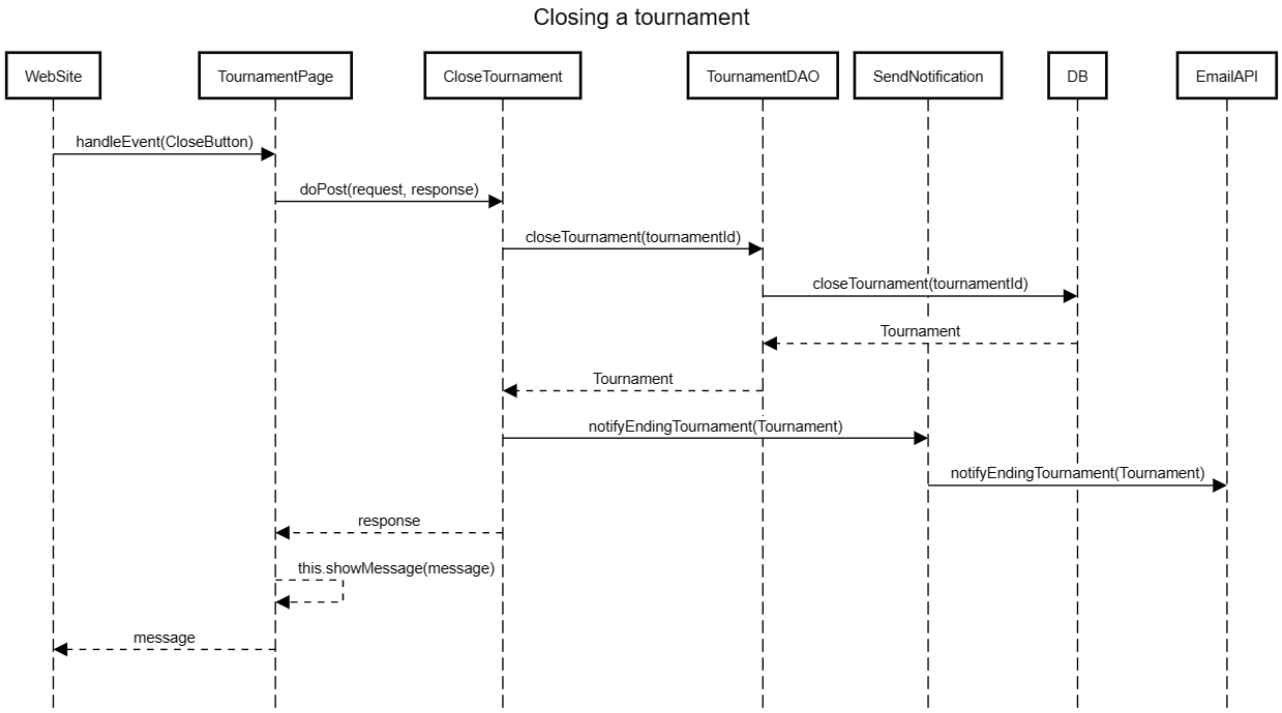
When a student wishes to accept an invitation to participate as a team together with a student who has invited them, they should go to the specific battle page and click the *Join Team* button that will show them all the teams that have invited them and they will select the one they prefer.



[SD8]. Closing a tournament

This sequence diagram represents the interaction that occurs when a Educator wants to close a tournament in the CKB application

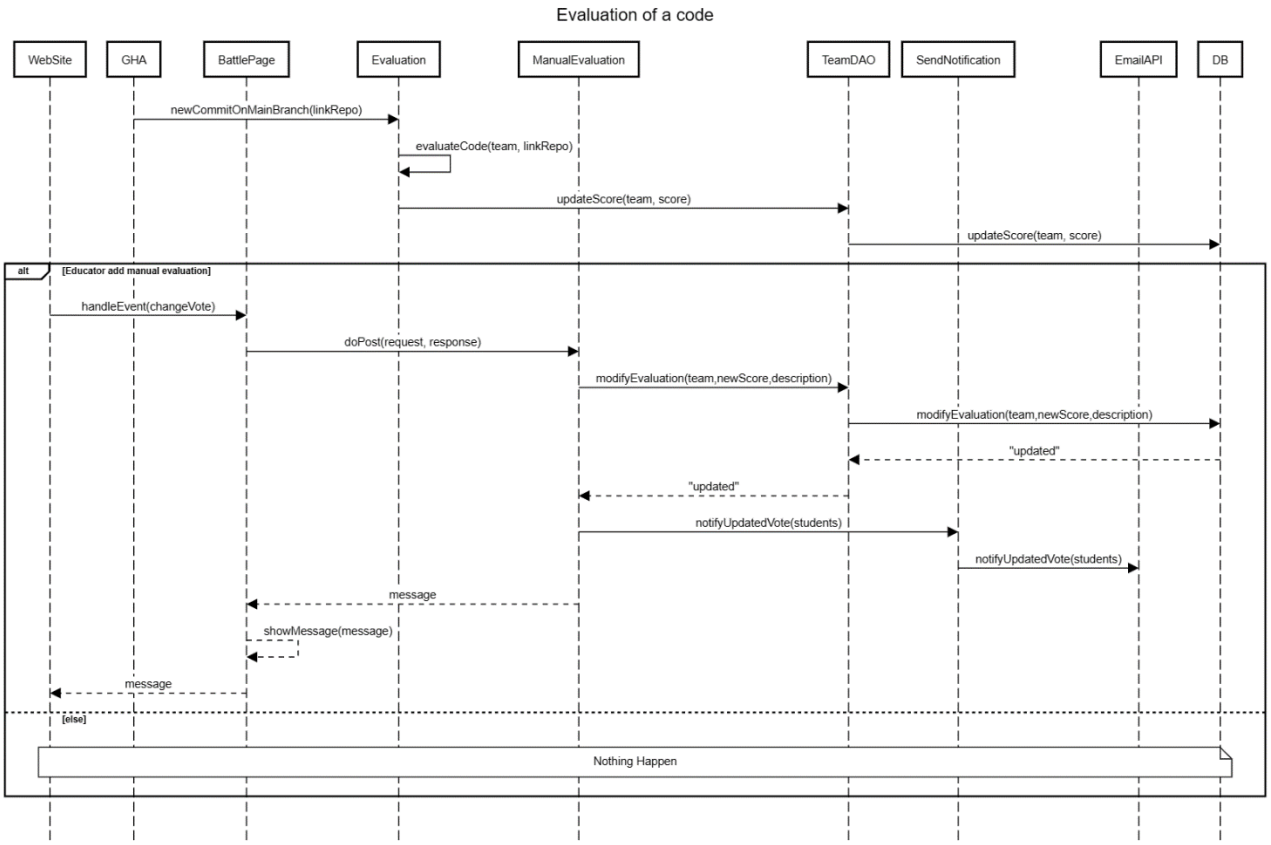
When an educator wants to close a tournament they manage, they should go to the specific tournament page and click the *Close Tournament* button. By doing so, the application will save the status change for the tournament and send the students enrolled in it a notification regarding its closure.



[SD9]. Evaluation of a code

This sequence diagram illustrates the interaction that occurs during an evaluation, whether automatic or manual, of a commit submission made by a team in the CKB application. Whenever GHA sends a notification to the application regarding a commit made by a team to the main branch of their GH repo relating to a specific battle that is still in progress, the application will automatically perform an evaluation of the submission and assign a grade to it and store it in the database.

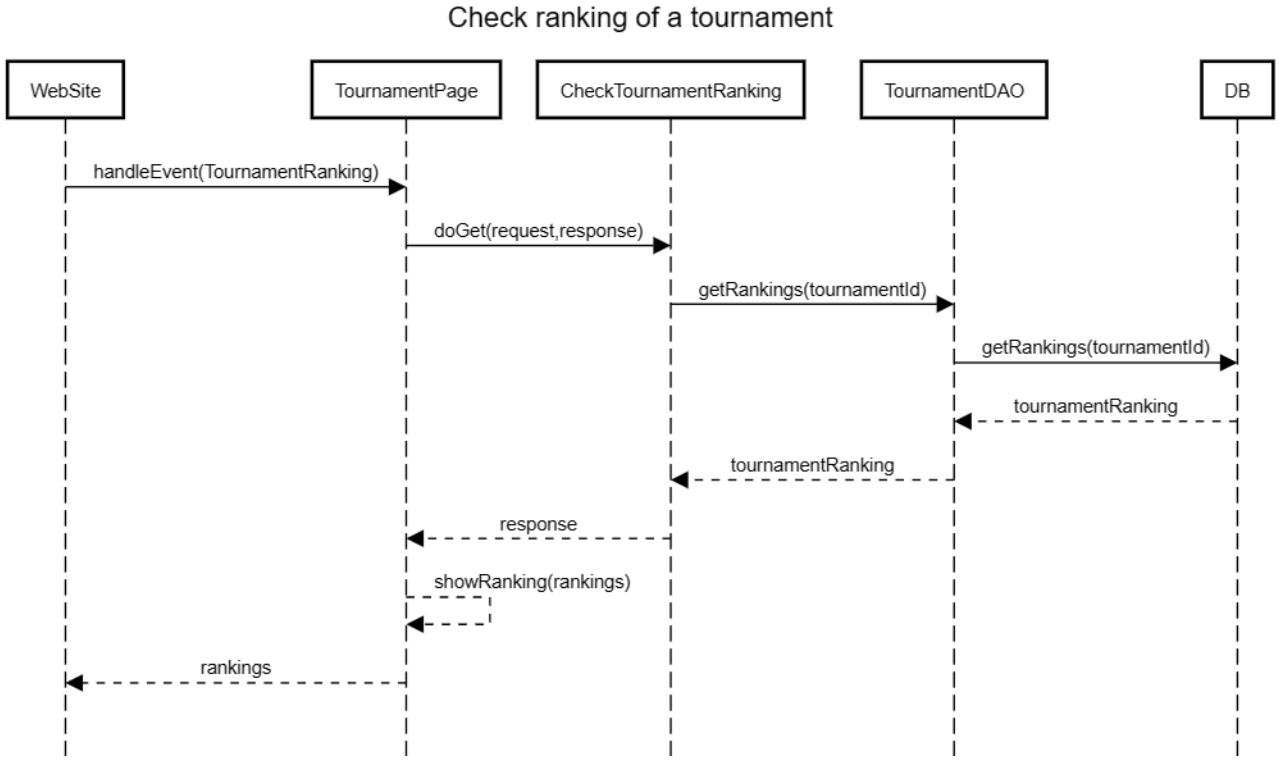
Should an educator wish to change the grade of a team entered in a battle, they may do so by going to the specific battle page and clicking on the *Manual Evaluation* button, this will display a form to allow the educator to select the team to be evaluated manually. Once the evaluation is complete, the application will save the grade and send a notification to the team, informing them that the educator has manually made changes to the grade.



[SD10]. Check ranking of a tournament

This sequence diagram represents the interaction that occurs when a User wants to check the ranking of a specific tournament in the CKB application.

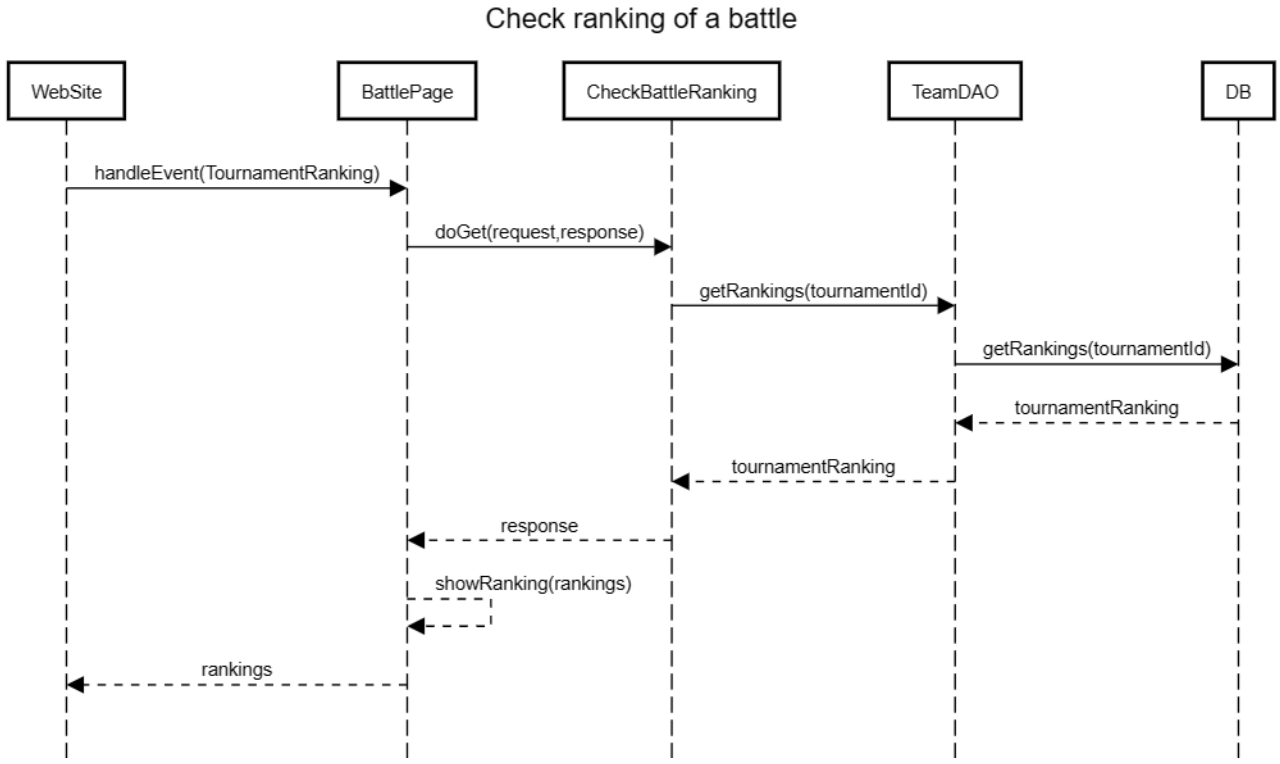
Each time a user enters the page of a specific tournament, it will show, in addition to the details of the tournament, the updated standings relating to the tournament's progress.



[SD11]. Check ranking of a battle

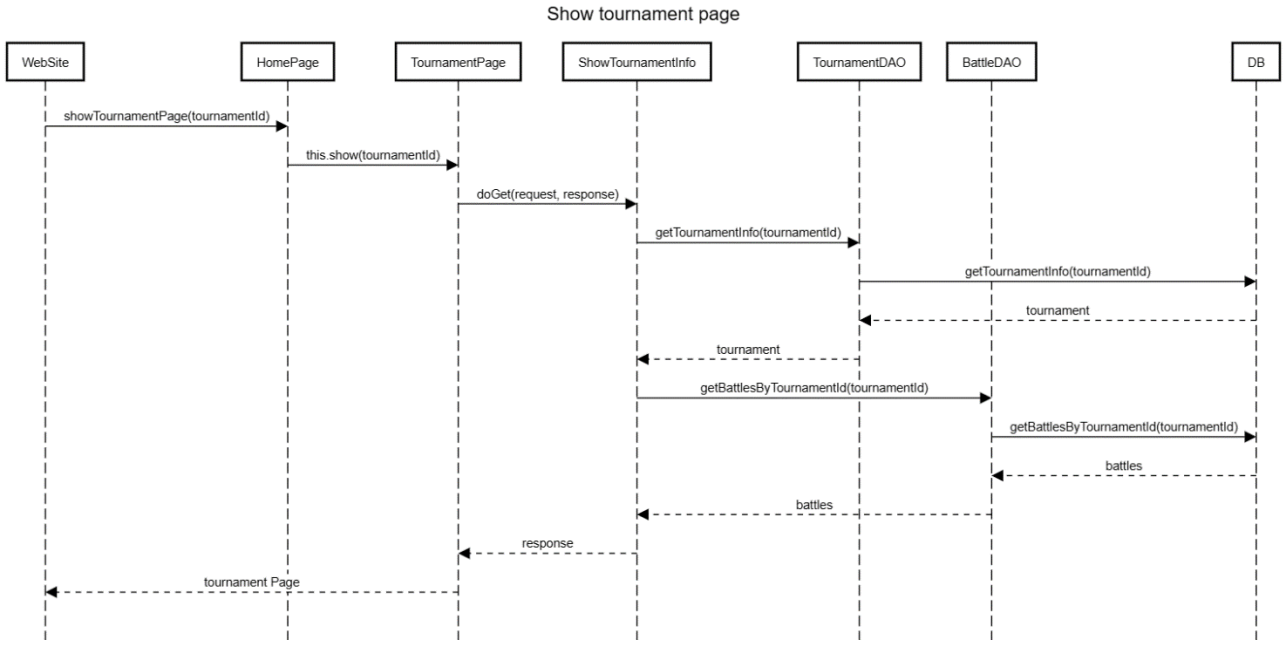
This sequence diagram represents the interaction that occurs when a User wants to check the ranking of a specific battle in the CKB application.

Each time a user enters the page of a specific battle, it will show, in addition to the details of the battle, the updated standings relating to the battle's progress.



[SD12]. Show tournament page

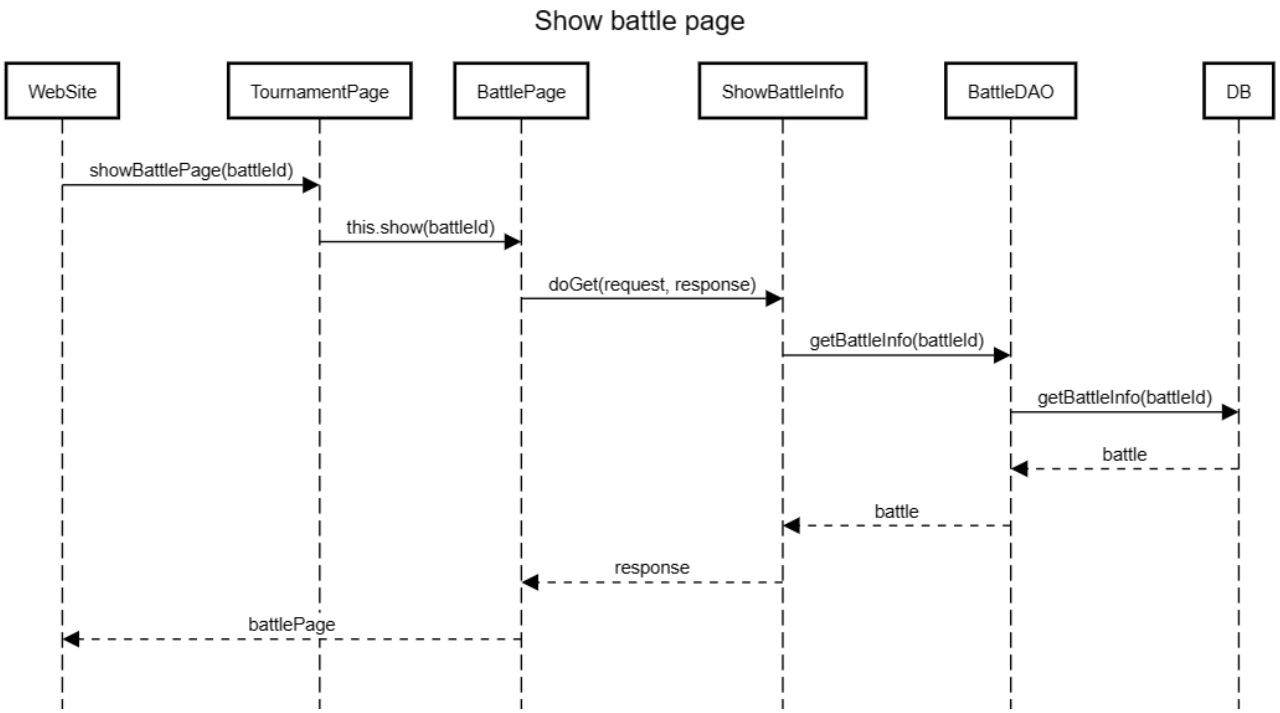
This sequence diagram represents the interaction that occurs when a User wants to show the specific tournament page in the CKB application.
Whenever a user enters a specific tournament page, it will display all the details of a tournament according to the user's role.



[SD13]. Show battle page

This sequence diagram represents the interaction that occurs when a User wants to show the specific battle page in the CKB application.

Whenever a user enters a specific battle page, it will display all the details of a battle according to the user's role.



2.5. *Component Interface*

This section will list the interfaces of all the methods that each component provides.

- *Bean*: is an object that stores data that is retrieved from the database.
  - User
  - Student
  - Educator
  - Tournament
  - Battle
  - Team
- *DAO*: is an object that is responsible for interfacing with the database to request and modify data in it.
  - UserDAO
    - Boolean addUser(String username, String password, int role, String name, String surname, Date birthdate, String email, String githubUser)
    - int checkCredential(String username, String password)
    - List<Student> getAllStudent()
    - List<Educator> getAllEducator()
  - TournamentDAO
    - int addTournament(String name, String description, DateTime registrationDeadline)
    - String addCollaborator(int userID)
    - Boolean addStudent(int tournamentID)

        - Tournament closeTournament(int tournamentID)
        - Tournament getTournamentInfo(int tournamentID)
        - List<User> getRankings(int TournamentID)
    - BattleDAO
        - int addBattle(File codeKata, DateTime registrationDeadline, DateTime battleDeadline, int minMemberPerTeam, int maxMemberperTeam, List<String> addConf)
        - Battle getBattle(int battleID)
        - List<Battle> getBattleByTournamentID(int tournamentID)
        - List<User> getRankings(int battleID)
    - TeamDAO
        - Boolean createTeam(int battleID, List<int> studentID)
        - List<Student> getAllStudentByTeam(int teamID)
        - List<Team> getallTeamByBattle(int battleID)
        - void updateScore(int teamID, int score)
        - Boolean modifyEvaluation(int teamID, int score, String description)
        - Boolean updateTeam(int teamID)
- *Servlet*: is a Java object that extends the functionality of a Web server to provide dynamic services to users through interaction based on requests and responses.
    - SignInManager

Type	POST
Req body	username: [String], password: [String], role: [int], name: [String], surname: [String], birthdate: [Date], email: [String], githubUser: [String]
Success code	code 200: OK message: {Registration has gone successfully you will receive a confirmation email}

Error code	code: 400: Error body : { ”error” : ”Malformed request” } code 401: Error body : { ”error” : ”Not existing mail” } code 401: Error body : { ”error” : ”Not existing GH account”} code 401: Error body : { ”error” : ”Different password” } code: 403: Error body : { ”error” : ”Already registered” }

- LoginManager

Type	POST
Req body	username: [String], password: [String]
Success code	code 200: OK
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code 401: Error body : { ”error” : ”Mail not confirmed” } code 401: Error body : { ”error” : ”Wrong password” }

	code: 404: Error body : { ”error” : ”User not registered” }

- CreateTournament

Type	POST
Req body	Title: [String], Description: [String], RegistrationDeadline: [DateTime]
Success code	code 200: OK body: {tournamentId: int}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 401: Error body : { ”error” : ”User not authorized” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” }

- AddCollaborator

Type	POST
Req body	userID: [int]
Success code	code 200: OK body:{message: String}
Error code	code: 400 Error body : { ”error” : ”Malformed request” }

	code: 400 Error body : { ”error” : ”User selected can’t be a collaborator” } code: 401 Error body : { ”error” : ”User not authorized” } code: 403 Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 404 Error body : { ”error” : ”User not found”}

- CreateBattle

Type	POST
Req body	CodeKata: [File], regDeadline: [DateTime], battleDeadline: [DateTime], minMem: [int], maxMem: [int], addConf: [List<String>]
Success code	code 200: OK body:{message: String}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 401: Error body : { ”error” : ”User not authorized” }

	code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” }

- JoinTournament

Type	POST
Req body	TournamentId: [int]
Success code	code 200: OK body:{message: String}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 400 Error body : { ”error” : ”You have already joined this tournament” } code: 401: Error body : { ”error” : ”User not authorized” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 404 Error body : { ”error” : ”Tournament not found”}

- JoinBattle

Type	POST
Req body	battleID: [int], studentID: [List<int>]

Success code	code 200: OK body:{message: String}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 400 Error body : { ”error” : ”You have already joined this battle” } code: 401: Error body : { ”error” : ”User not authorized” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 403: Error body : { ”error” : ”Student must be subscripted also to the tournament” } code: 404 Error body : { ”error” : ”Battle not found”}

- AcceptInvite

Type	POST
Req body	teamID: [int]
Success code	code 200: OK body:{message: String}
Error code	code: 400: Error body : { ”error” : ”Malformed request” }

	code: 400 Error body : { ”error” : ”You have already joined this team” } code: 400 Error body : { ”error” : ”You don’t have an invite for this team” } code: 401: Error body : { ”error” : ”User not authorized” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 403: Error body : { ”error” : ”Student must be subscripted also to the tournament” } code: 404 Error body : { ”error” : ”Team not found”}

- CloseTournament

Type	POST
Req body	tournamentID: [int]
Success code	code 200: OK body:{message: String}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 400 Error

	body : { ”error” : ”This tournament can’t be closed” } code: 400 Error body : { ”error” : ”This tournament is already closed” } code: 401: Error body : { ”error” : ”User not authorized” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 404 Error body : { ”error” : ”Tournament not found”}

- ManualEvaluation

Type	POST
Req body	BattleId: [int], TeamId: [int]
Success code	code 200: OK body:{message: String}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 401: Error body : { ”error” : ”User not authorized” } code: 403: Error body : { ”error” : ”Can’t modify this vote” }

	code: 403: Error body : { ”error” : ”Battle has already ended” } code: 404 Error body : { ”error” : ”Battle not found”} code: 404 Error body : { ”error” : ”Team not found”}

- ShowTournamentPage

Type	GET
Req body	tournamentID: [int]
Success code	code 200: OK body:{Tournament: tournament}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 404 Error body : { ”error” : ”Tournament not found”}

- ShowBattlePage

Type	GET
Req body	battleID: [int]
Success code	code 200: OK body:{Battle: battle}

Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 404 Error body : { ”error” : ”Battle not found”}

- CheckTournamentRanking

Type	GET
Req body	tournamentID: [int]
Success code	code 200: OK body:{TournamentRanking: List<User>}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 404 Error body : { ”error” : ”Tournament not found”}

- CheckBattleRanking

Type	GET
Req body	battleID: [int]
Success code	code 200: OK

	body:{BattleRanking: List<User>}
Error code	code: 400: Error body : { ”error” : ”Malformed request” } code: 403: Error body : { ”error” : ”User doesn’t have the authorization to do that” } code: 404 Error body : { ”error” : ”Battle not found”}

- *JavaScript web page*: is an object that manages user interaction on a web page by dynamically changing its visual elements in response to events that occur during the interaction.
    - HomePage
        - this.show()
        - this.showError(String message)
    - SingInPage
        - this.show()
        - this.showError(String message)
    - LoginPage
        - this.show()
        - this.showError(String message)
    - PersonalHomePage
        - this.show()
        - this.showError(String message)
    - TournamentPage
        - this.show(int tournamentID)
        - this.shwoError(String message)
        - this.showMessage(String message)
        - this.showRanking(List<Student> ranking)
    - BattlePage
        - this.show(int battleID)
        - this.shwoError(String message)
        - this.showMessage(String message)

        - this.showRanking(List<Student> ranking)
- *Other classes*:
    - StartBattle
    - Evaluation

2.6. *Architectural styles and patterns*

2.6.1. Architectural style

The implementation of CKB will take place following a three-tier architecture; this structure confers numerous advantages due to the modular division of the system into three distinct and autonomous layers.

1. *Presentation Tier*: the tier responsible for interfacing with the client, by changing or replacing the displayed pages, will receive the data sent by the *Business Logic Tier*. This data will be adapted so that it is visible and understandable to the user.
2. *Business Logic Tier*: this tier includes the application of business logic, which is responsible for performing calculations and making decisions based on the data stored in the data tier. Later, the results will be shown in the presentation tier.
3. *Data Tier*: this tier is responsible for storing, managing, and making accessible the data that will be used by the business logic tier.

By adopting this architecture, it is possible to modify one part of the system without involving the others. Using an intermediate tier to manage the data in the database provides an additional layer of security because users cannot directly modify the saved data. To modify the data in the database they must necessarily interact with the Business Logic Tier which imposes specific formats on the data.

2.6.2. Patterns

2.6.2.1. Model-View-Controller Pattern

We will adopt the Model-View-Controller (MVC) pattern to organize and structure the code. In the context of MVC, the Model assumes the crucial role of storing essential objects during computation and decision making, with responsibilities given to *Data Access Objects* (DAOs). These components ensure the accurate management of the underlying data. On the other hand, *web pages* constitute the components dedicated to presenting information resulting from decisions made by servlets in a clear and understandable way to the user. *Servlets*, grouped under the Controller category, emerge as key figures responsible for reading data from the Model and orchestrating changes in the View. This functional subdivision, typical of MVC, facilitates

maintenance, scalability and code comprehensibility, contributing to effective management of business logic and user interactions within the application.

2.6.2.2. Observer Pattern

We will adopt the Observer design pattern to notify users about relevant changes in the application, such as the creation or the closure of a tournament and creation of a battle. Each class responsible for issuing notifications will send its own messages to the *SendNotification* class, which, acting as an intermediary, will interface with the email API to send notifications to interested clients. The *SendNotification* class acts as a coordination point, playing a key role in orchestrating communication between the application and users, thus ensuring an efficient and effective flow of relevant information.

2.6.2.3. Command Pattern

We will adopt the Command pattern so that each individual action or command is encapsulated in an object that can be handled by distinct classes. This approach offers a significant advantage: the invoker, or caller of the action, does not need to know the implementation details of the command it is calling, only its interface. This separation allows the calling functions to remain independent of the classes involved in creating and handling the commands. In this way, the interface between the invoker and the command remains stable, facilitating code maintenance and allowing new commands to be introduced without modifying existing parts of the system. This design pattern is particularly beneficial in ensuring a clear separation of responsibilities and helps make the code more modular, flexible, and easily extensible.

2.7. Others design decision

In this section are presented some of the design decisions made for the system to make work as expected.

2.7.1. Availability

We opted to implement load balancing and server replication in order to enhance system availability and reliability. The adoption of load balancing allows our system to respond more promptly by distributing requests among less stressed servers. This results in higher operational efficiency which ensures faster response to users. In parallel, through server replication, we are able to improve system resilience: if one server fails, service can be maintained unaffected by using

operational replicas on other servers. This replication strategy helps to ensure continuity of service without significant interruptions, improving the robustness of our system in the face of any hardware failure.

2.7.2. Notification timed

The role of managing the sending of all notifications from the application to users is entrusted to the *SendNotification* component. This component interacts directly with the email API, facilitating the sending of notifications to the specified recipients.

2.7.3. Data storage

The decision to adopt a relational database, rather than a nonrelational one, was driven by the need to efficiently represent the inherent relationships present in the data to be stored. A relational database offers a structured model that lends itself well to the representation and management of connections between different entities. In addition, the use of a relational database greatly simplifies complex operations, such as joining multiple tables and updates within the database itself. The tabular structure of the relational database provides a clear organization of the data, making it easier to navigate and manipulate information. This choice results in more efficient management of complex data relationships, contributing to the consistency and optimal structuring of our storage system.

2.7.4. Security

Our system is provided with three firewalls, which are crucial for the protection of the different components of the application. Given the shared nature of the system, which will not be limited to a single machine, the implementation of these firewalls is essential. The external network, deemed untrusted, could pose a potential threat to the application LAN. The introduction of firewalls creates a demilitarized zone (DMZ), providing security and protection within the application environment. This configuration helps mitigate risks from unauthorized access and provides an additional layer of security, protecting the internal network from potential threats from outside.

3. USER INTERFACE DESIGN

3.1. External interface

3.1.1. User interface

In this paragraph, we will provide an overview of the graphical interface of the application.

- This image represents an overview of how we will then go about developing the Home Page of the application. This page allows you to choose whether you want to register or access the application

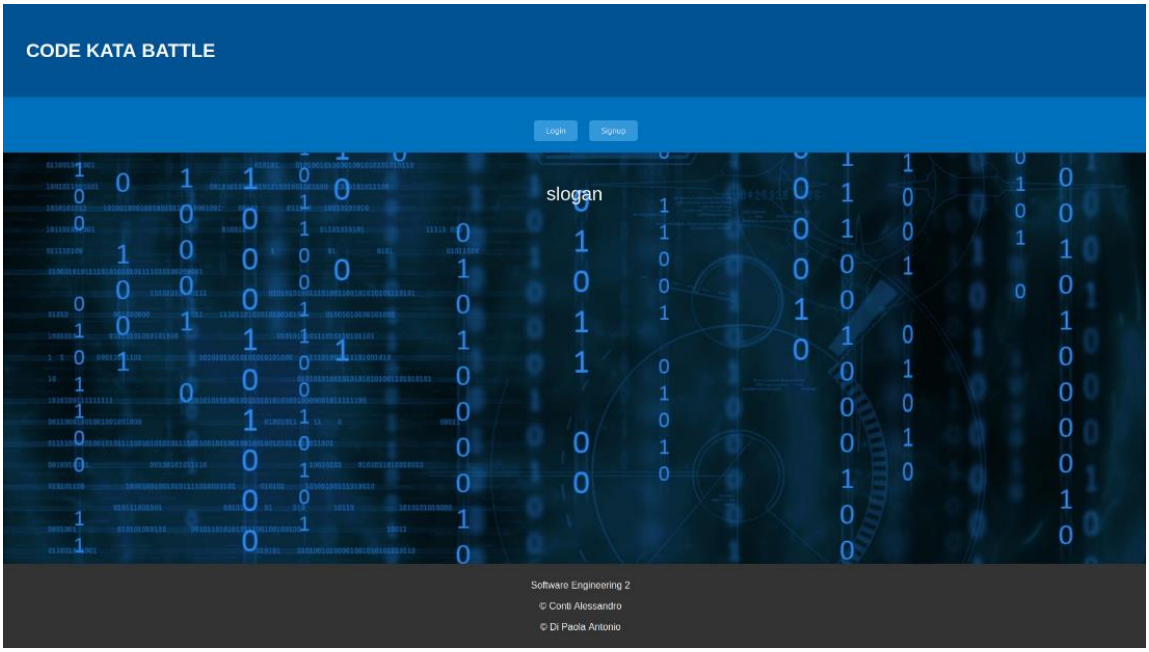


Figure 4 Main Page

- After logging into the application, users will be presented with two similar pages, but with differences based on their roles as Educators or Students. Whether you are an Educator or a Student, the page will always display a search bar, allowing you to search for new tournaments and view all the tournaments you are enrolled in. In the case of a user logging in as an Educator, there is an additional option compared to Students—an extra button that, when clicked, directs them to the page dedicated to creating a new tournament.

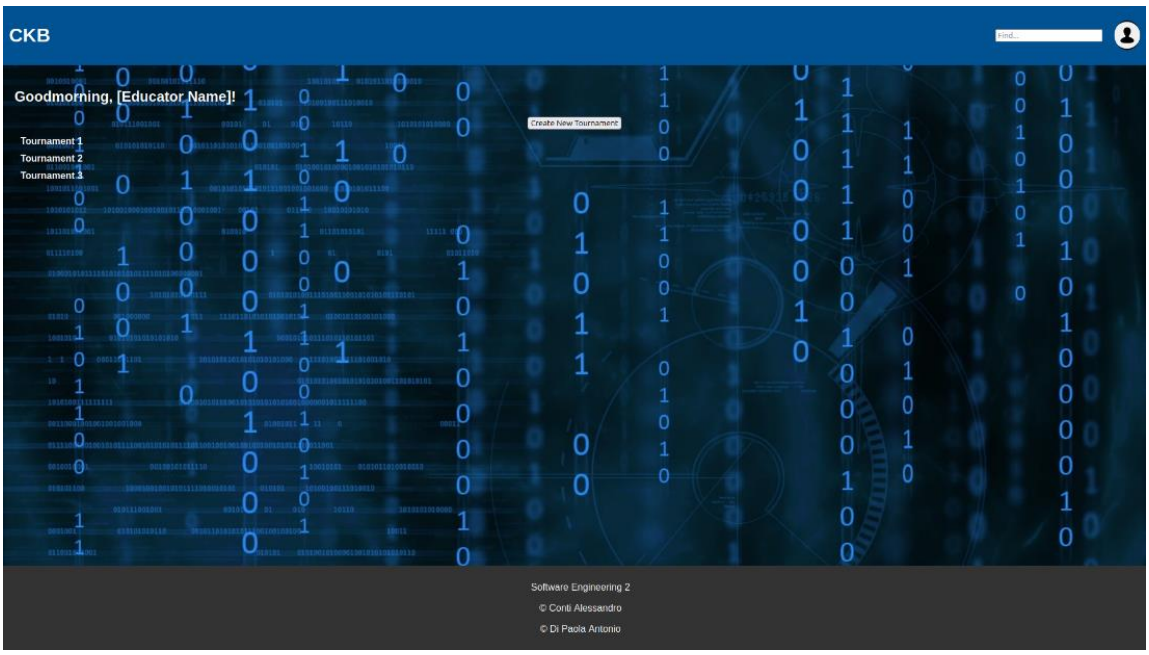


Figure 5 Educator's Home Page

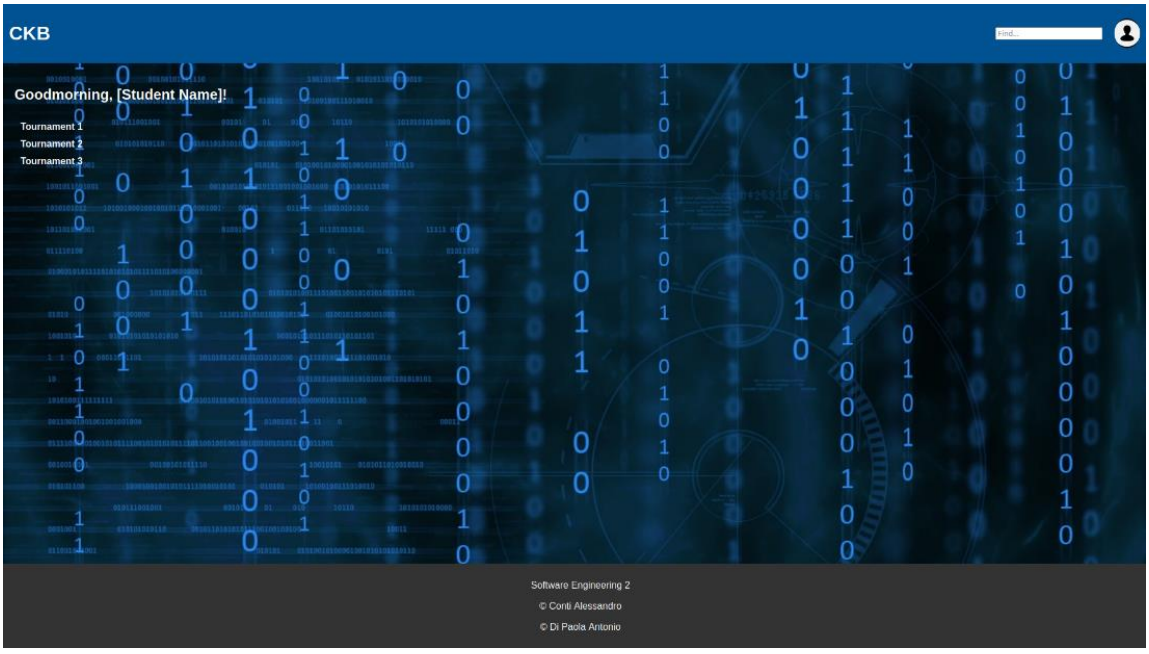


Figure 6 Student's Home Page

- Once a user selects a tournament to view, either through the search bar or via links to tournaments they are enrolled in, they will be redirected to a page displaying all relevant details about that tournament. In the event that the user is an Educator responsible for managing the tournament, their page will include a button enabling the creation of new battles; otherwise, this option will not be available.
  If the user is a Student who has not yet enrolled in the tournament and the registration deadline has not expired, the page will display a button allowing them to sign up.

In other scenarios, the page will only provide details about the tournament without additional options.

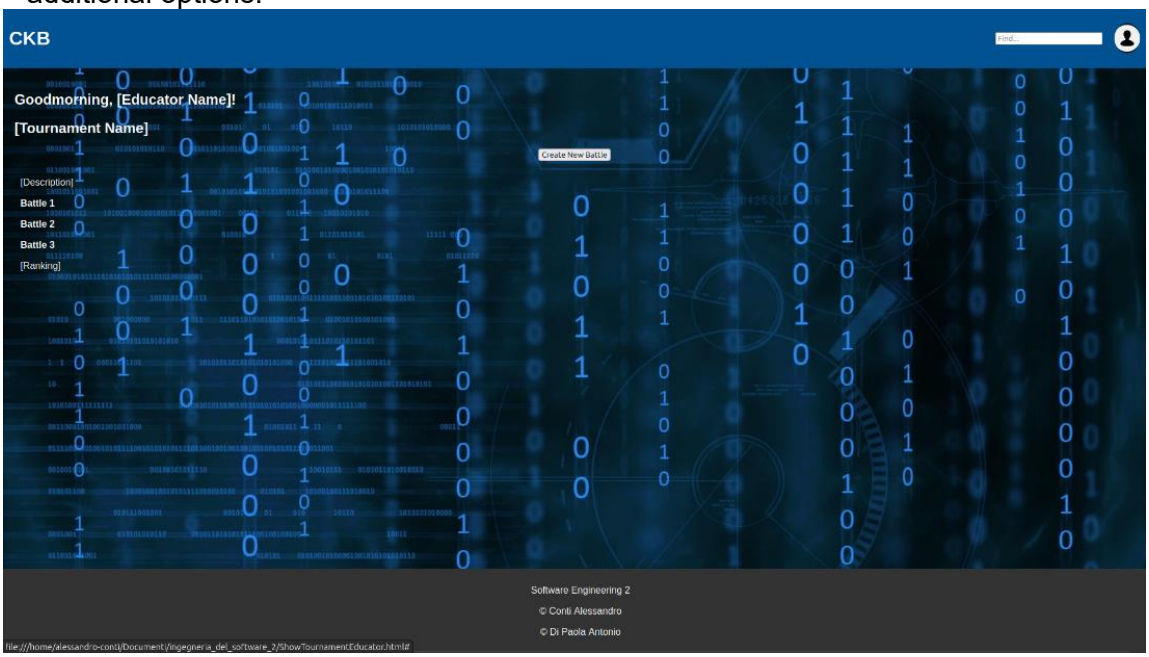


Figure 7 Page that shows a tournament to which the Educator is responsible for managing.

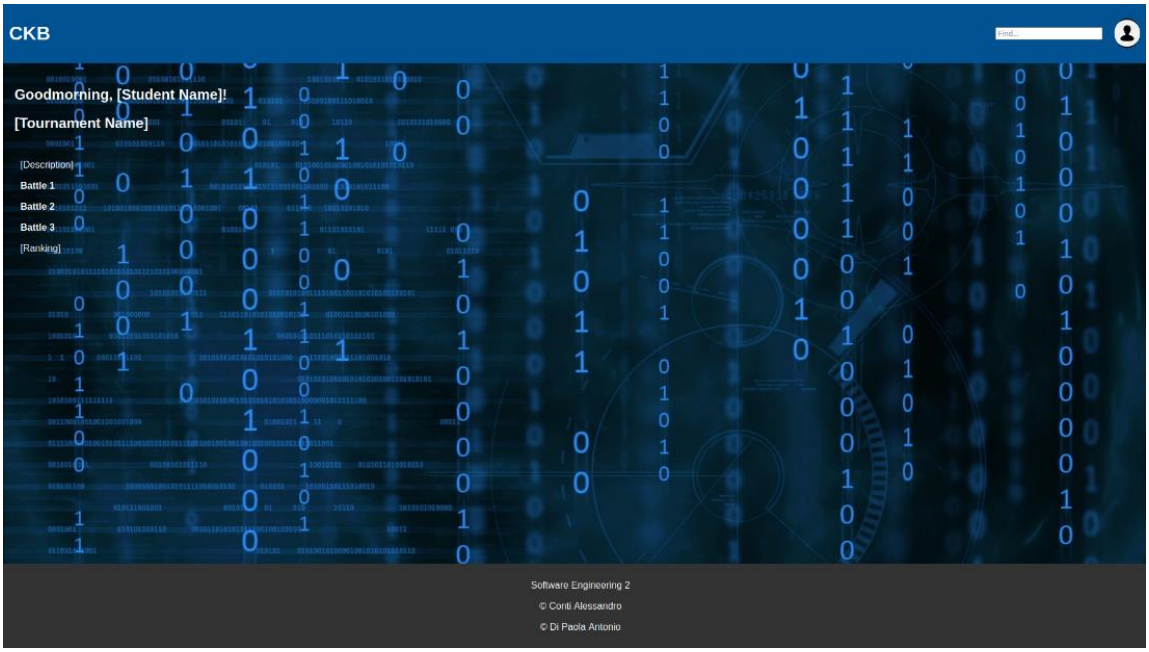


Figure 8 Page that shows a tournament in which the Student is registered.

- Once a user selects a battle to view through its link within their tournament, they will be redirected to a page presenting all the details related to that battle. In the event that the user is a Student enrolled in the tournament but not yet in that specific battle, the page will display the registration button as long as the registration deadline has not passed. In other scenarios, the page will only provide details about the battle without additional options.

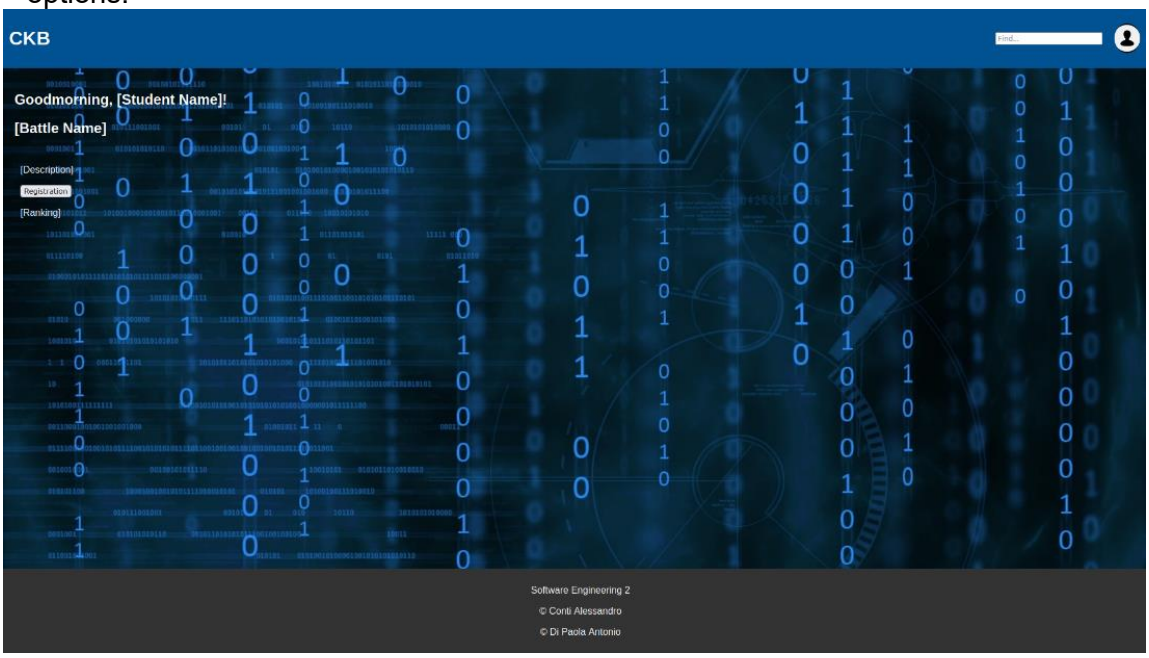


Figure 9 Page showing a battle that the Student can join.

- The page that shows the tournament and/or battle rankings is accessible when a user clicks on the ranking link found on the tournament and/or battle details page.

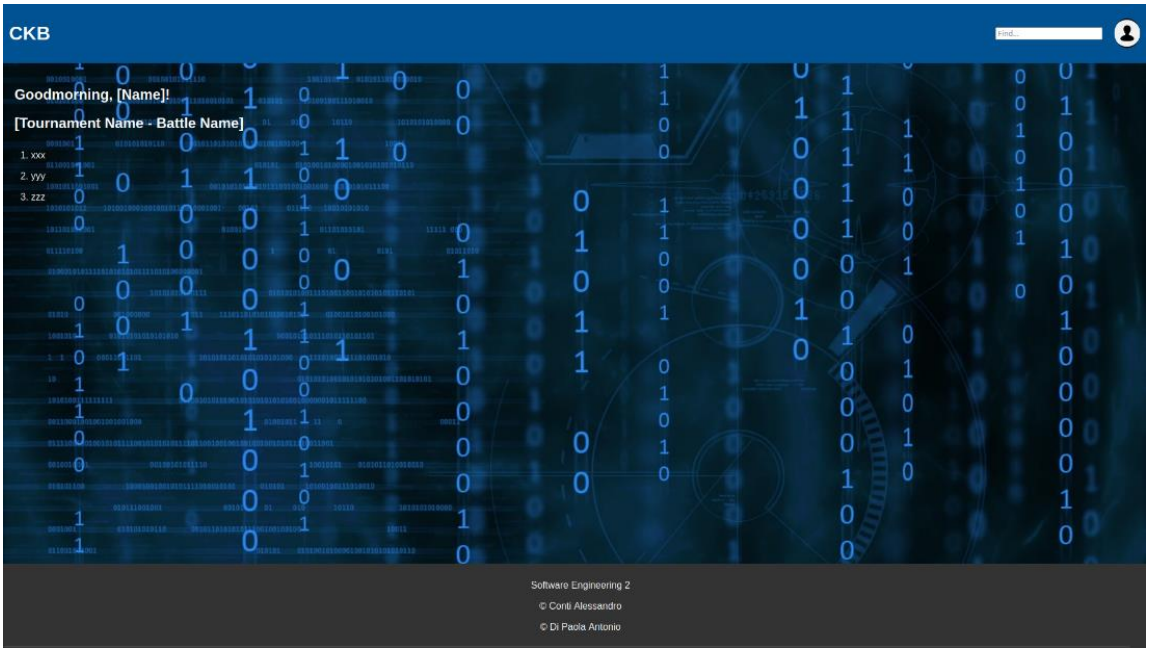


Figure 10 Page showing a tournament or a battle ranking.

4. REQUIREMENT TRACEABILITY

5. IMPLEMENTATION, INTEGRATION AND TEST PLANNING

6. EFFORT SPENT

In the following tables we will summarize the effort spent by each member of the team on the RASD Document

- Conti Alessandro

Chapter	*Effort (in hours)*
1	
2	
3	
4	

- Di Paola Antonio

Chapter	*Effort (in hours)*
1	
2	
3	
4	

7. REFERENCES

- State Diagrams made with: *draw.io*
- Class Diagrams made with: *StarUML*
- Sequence Diagrams made with: *SequenceDiagram.org*
- Alloy models made, ran and checked with: *Alloy 6*
- The User Interface overview was written in HTML with: *Visual Studio Code*