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.

### 1.4.2. Acronyms

- CKB: CodeKataBattle
- CK: CodeKata, Code Kata
- GH: GitHub
- GHA: GitHub Action
- GDPR: General Data Protection Regulation
- 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.

**CKB Network Architecture**

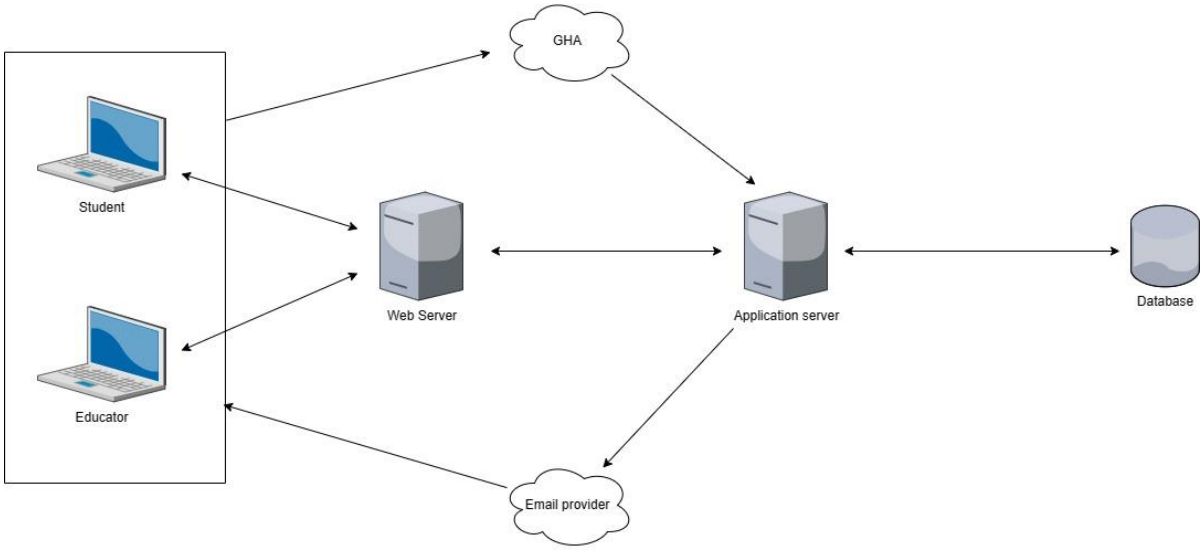



*Figure 1 High Level View*

### 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.

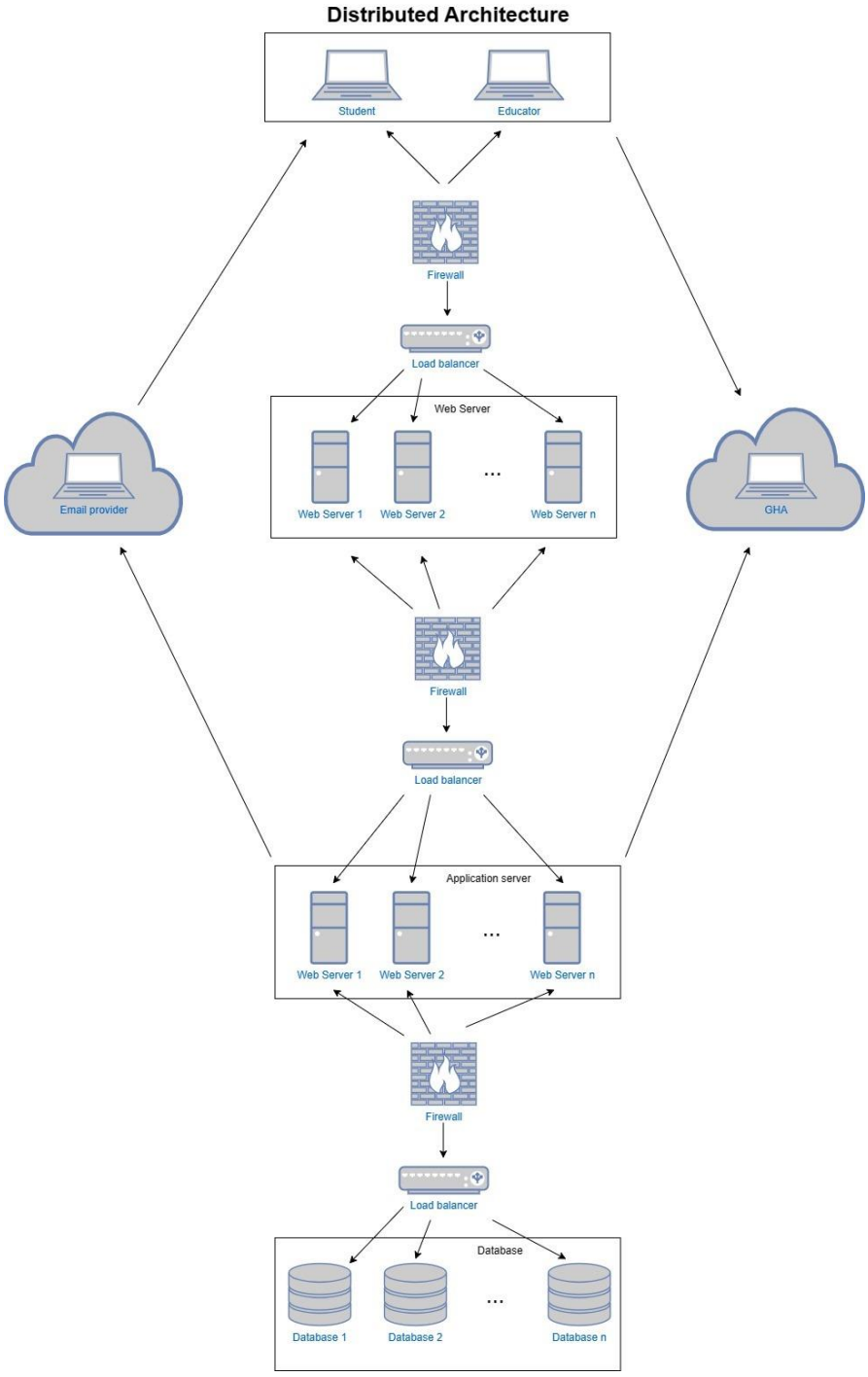



*Figure 2 Distributed View*

## *2.2. Component view*

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

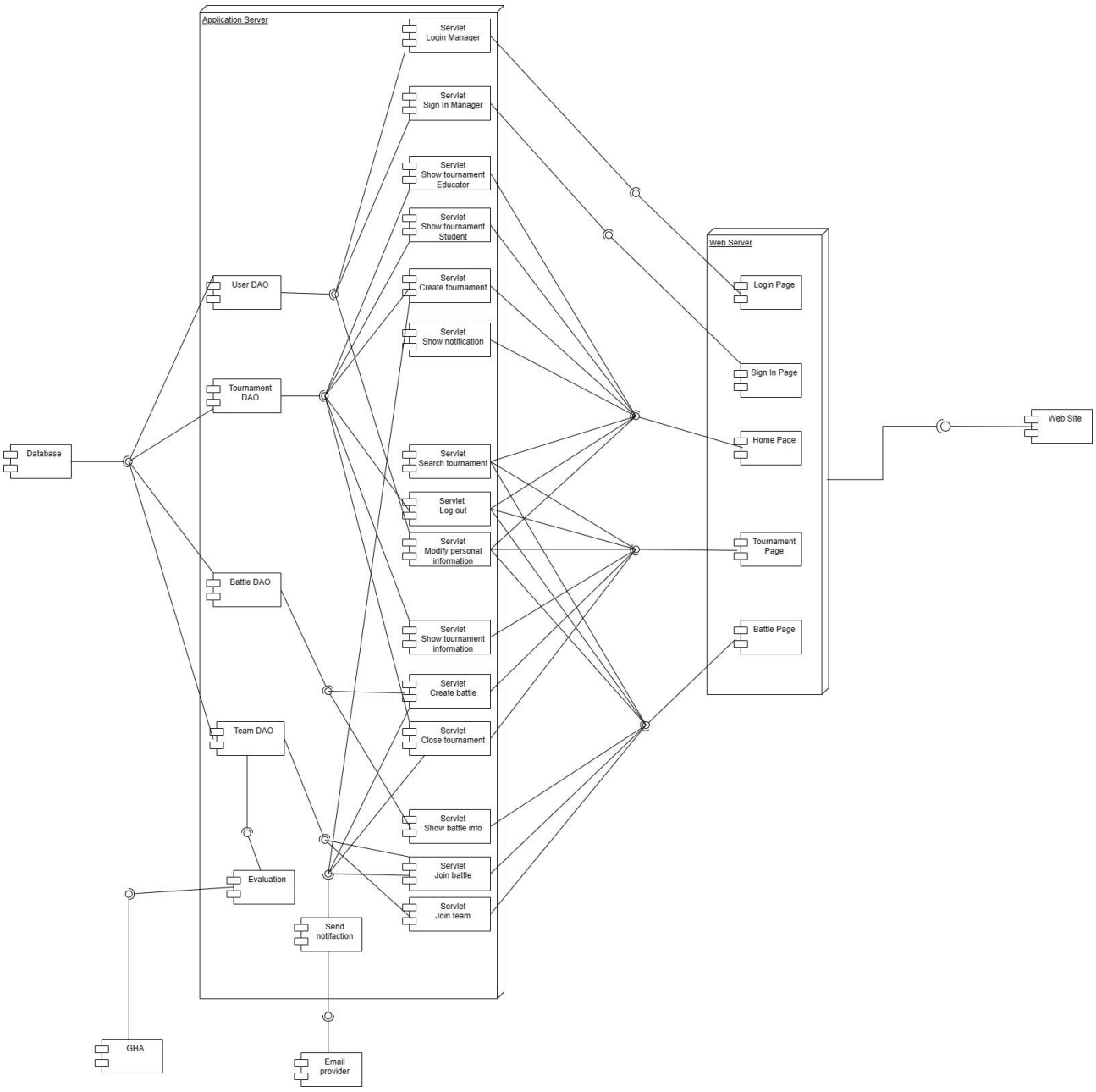



*Figure 3 Component View*

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.

## *2.3. Deployment view*

CKB addresses two types of users: Student and Educator.

### 2.3.1. Student

A Student is a user eager to hone his skills in software development and decides to enrich his education by registering on the CKB application.

After successfully registering on the application, the Student has the opportunity to immerse himself in challenging tournaments. Once registered in these tournaments, he can engage in battles against other Students equally eager to improve their skills. In this competitive environment, the user has the opportunity to compare his results with those of his peers, creating a dynamic and collaborative environment for learning and developing software development skills.

### 2.3.2. Educator

An Educator is a user who shares his or her skills in software development with other users eager to improve their skills.

After registering, the Educator has the ability to create tournaments, organising programming battles within them in which registered Students can participate to hone their skills. The Educator oversees the tournament by monitoring the codes submitted by the Students to the platform, providing additional evaluation in addition to the automatic evaluation. This process creates a collaborative environment where students can receive direct and targeted feedback to further improve their software development skills.

## *2.4. Runtime view*

## *2.5. Component Interface*

## *2.6. Architectural styles and patterns*

## *2.7. Other design decision*

### 2.7.1. Domain Assumption

[DA1]. Users must have a valid email address

[DA2]. Students must have a GH profile

[DA3]. Students need to know how to access GitHub, how to fork a repository and how to enable GitHub Actions

[DA4]. Each user will create only 1 account

[DA5]. Personal data given by User during the registration process are assumed true

[DA6]. User must have an internet connection

[DA7]. Educator must upload the correct CK

### 2.7.2. Dependency

[D1]. The CKB will make use of the GHA service offered by GitHub user account

[D2]. The CKB will access GitHub APIs to create new repositories and add Students

### 2.7.3. Constraint

[C1]. The CKB must abide to the GDPR regulation with regards to the treatment of personal data.

[C2]. The CKB must be implemented as a web site

# 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

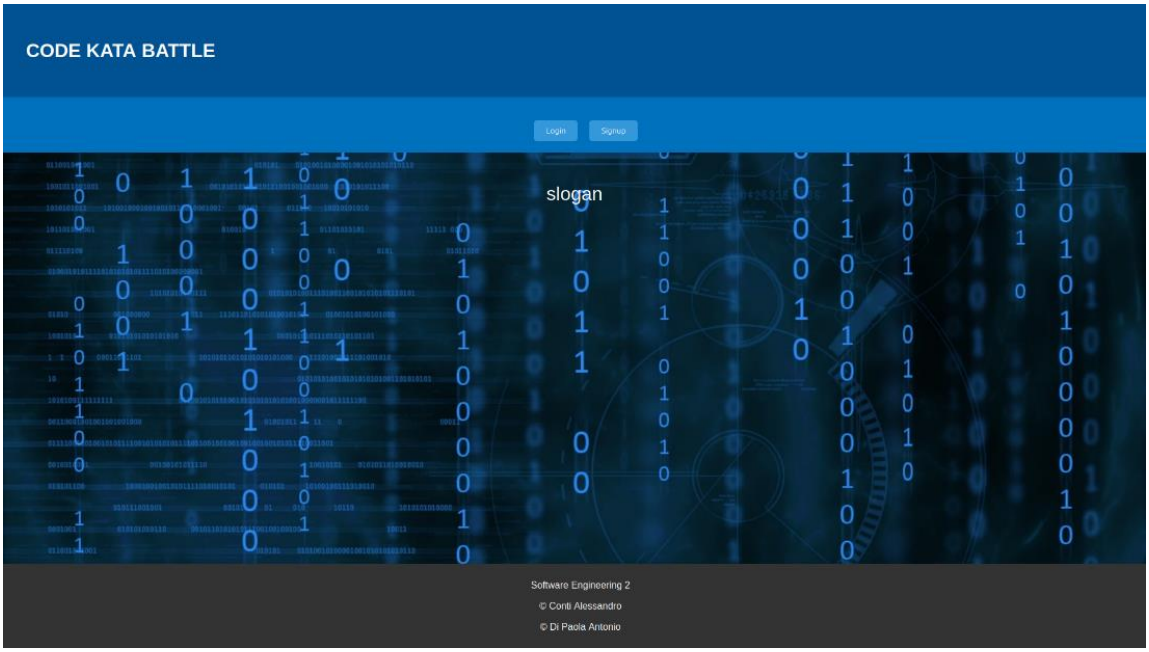



*Figure 2 Main Page*

- 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.

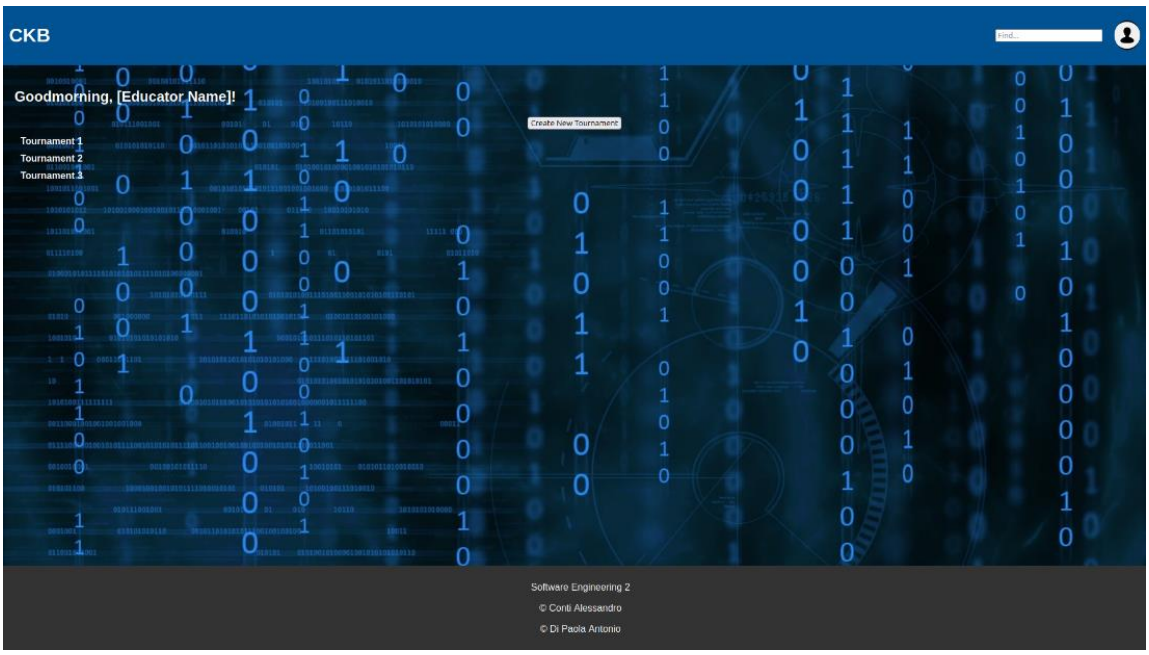



*Figure 3 Educator's Home Page*

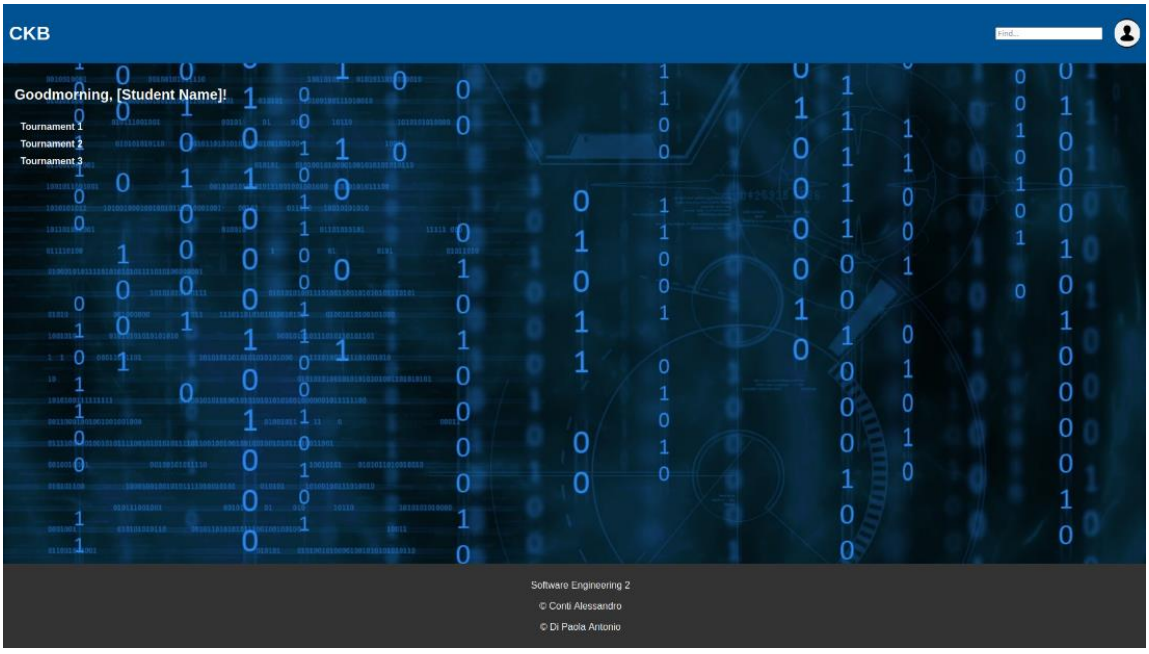



*Figure 4 Student's Home Page*

- 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.

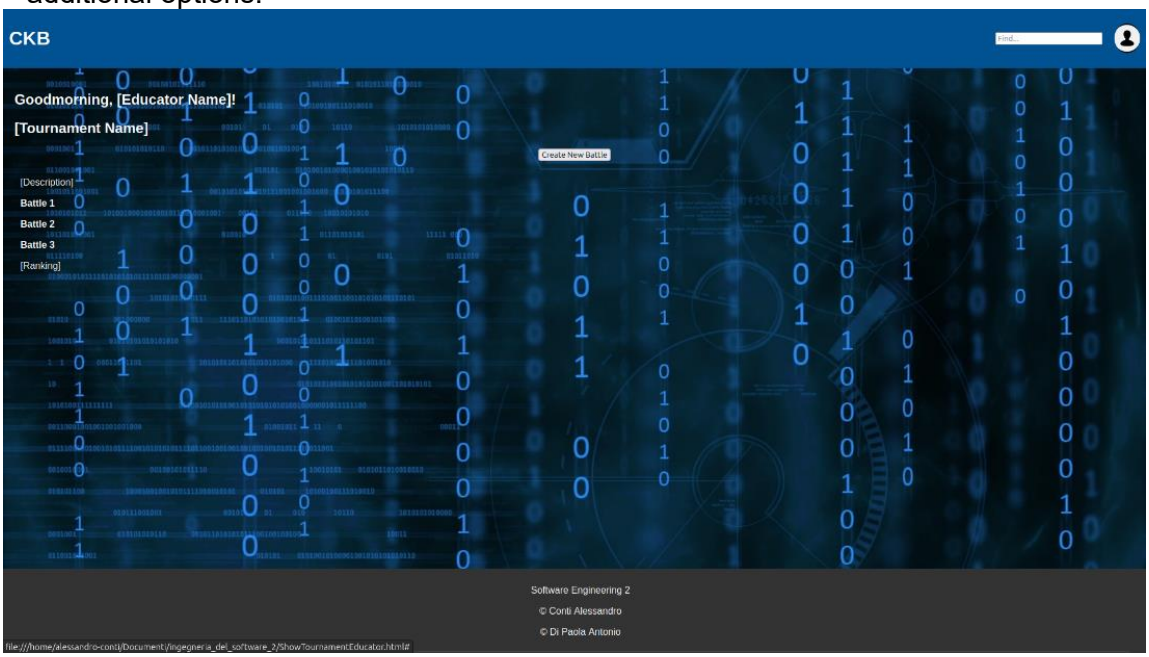



*Figure 5 Page that shows a tournament to which the Educator is responsible for managing.*

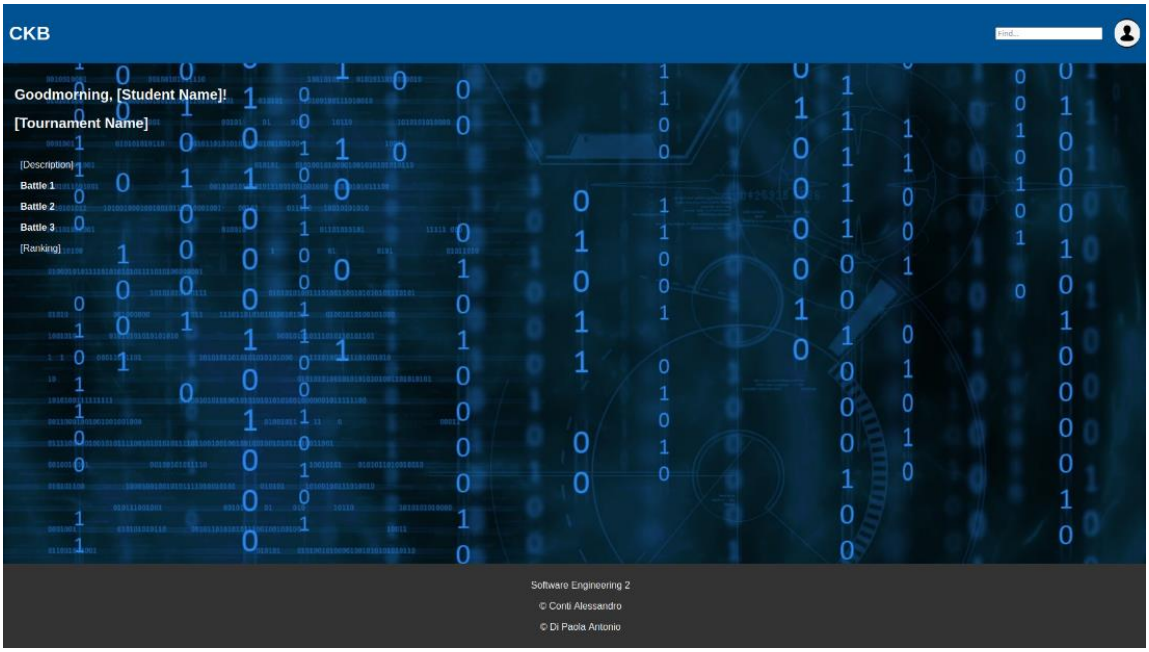



*Figure 6 Page that shows a tournament in which the Student is registered.*

- 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.

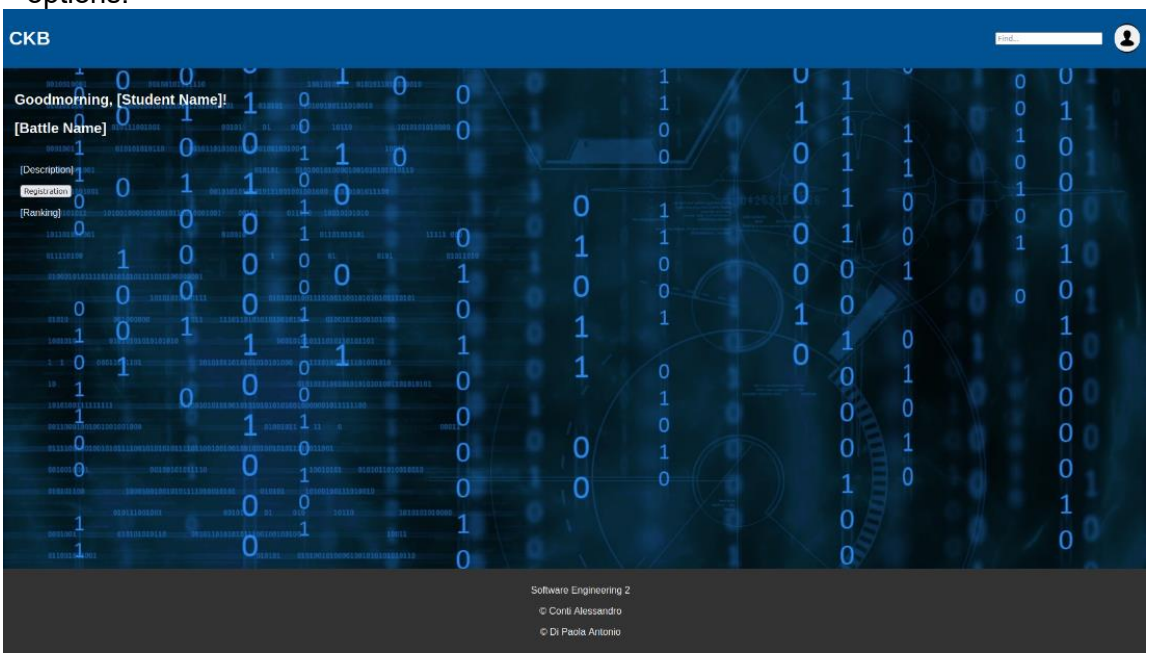



*Figure 7 Page showing a battle that the Student can join.*

- 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.

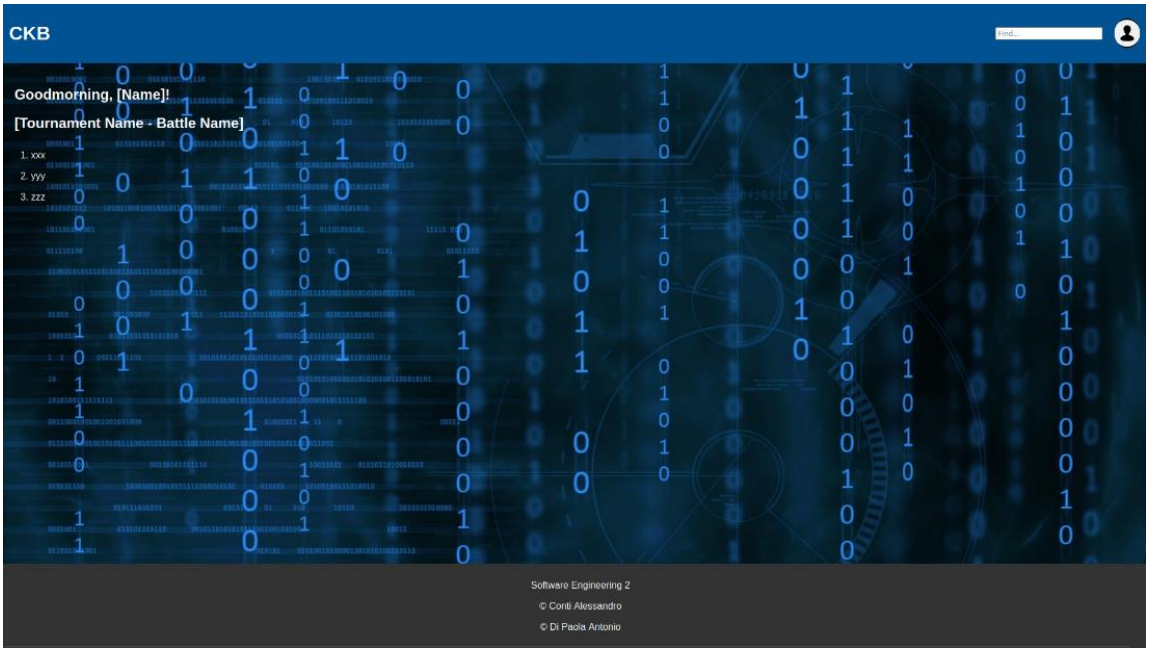



*Figure 8 Page showing a tournament or a battle ranking.*

### 3.1.2. Hardware interface

The application does not require any specific hardware interface as long as you have a computer that connects to an Internet network and a browser that can read and run JavaScript.

### 3.1.3. Software interface

The only software interface that the application requires is the email API.

### 3.1.4. Communication interface

The communication protocol we use is the HTTPS protocol; it is important to use it so that we have secure communication through the Internet.

## *3.2. Functional requirements*

In the following are specified all the requirements that the system has to fulfill. In order to work properly, the system should:

- [R1]. System allows students to sign up
- [R2]. System allows educators to sign up
- [R3]. System allows students to login
- [R4]. System allows educators to login
- [R5]. System allows educator to create a new tournament and to add all the needed information
- [R6]. System allows educator to grant the permission to other colleagues to create battle in a specific tournament
- [R7]. System allows educator to create a battle inside a particular tournament by adding CK and the deadlines
- [R8]. System allows educator to see rankings of each tournament, which he created or in which he has been nominated collaborator
- [R9]. System allows educator to see ranking of a specific battle
- [R10]. System allows educator to modify the evaluation manually of a specific group in a battle
- [R11]. System allows educator to close a tournament
- [R12]. System notifies students about the creation of a new tournament
- [R13]. System notifies students about the creation of a new battle in a specific tournament in which they have subscribed
- [R14]. System notify student about publication of final ranking of a specific battle
- [R15]. System notify student about publication of final ranking of a specific tournament
- [R16]. System allow student to join a tournament
- [R17]. System allow student to join a battle
- [R18]. System allows students to see rankings of each tournament, which they are sub scripted
- [R19]. System allows students to see ranking of a specific battle
- [R20]. System, at end of the registration period of a battle, create a GH repo and send invitation to all member of the group
- [R21]. System automatically evaluates the code in the main branch of the repo after each commit in it
- [R22]. System allows students to search tournament by key words

### 3.2.1. Use cases diagram

The diagram of use cases derived from the scenarios described above is shown below. These diagrams are useful to show the actors interacting with the system and to understand their roles.

### 3.2.1.1. Student use case diagram

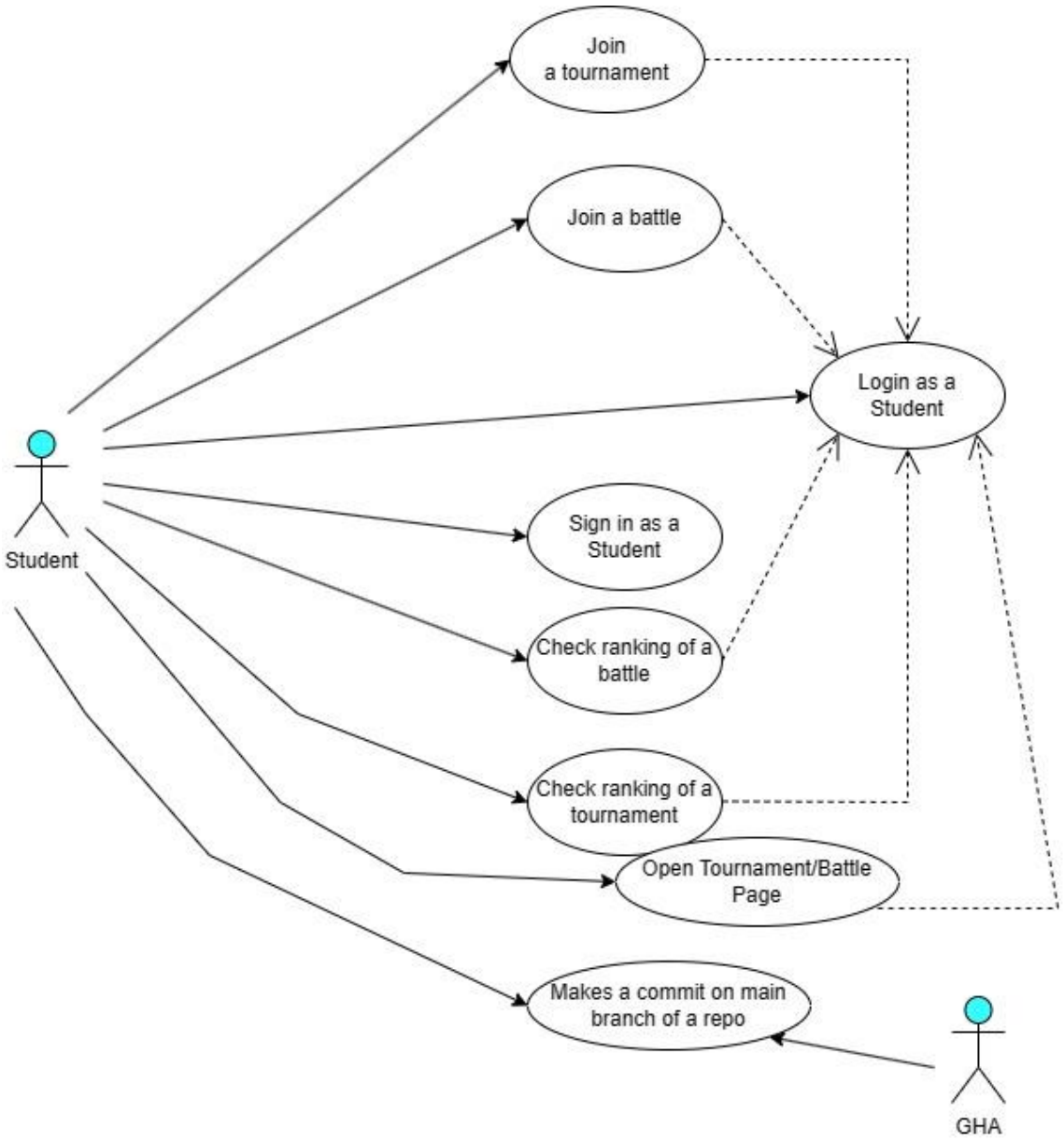



*Figure 9 Student Use Case Diagram*

### 3.2.1.2. Educator use case diagram

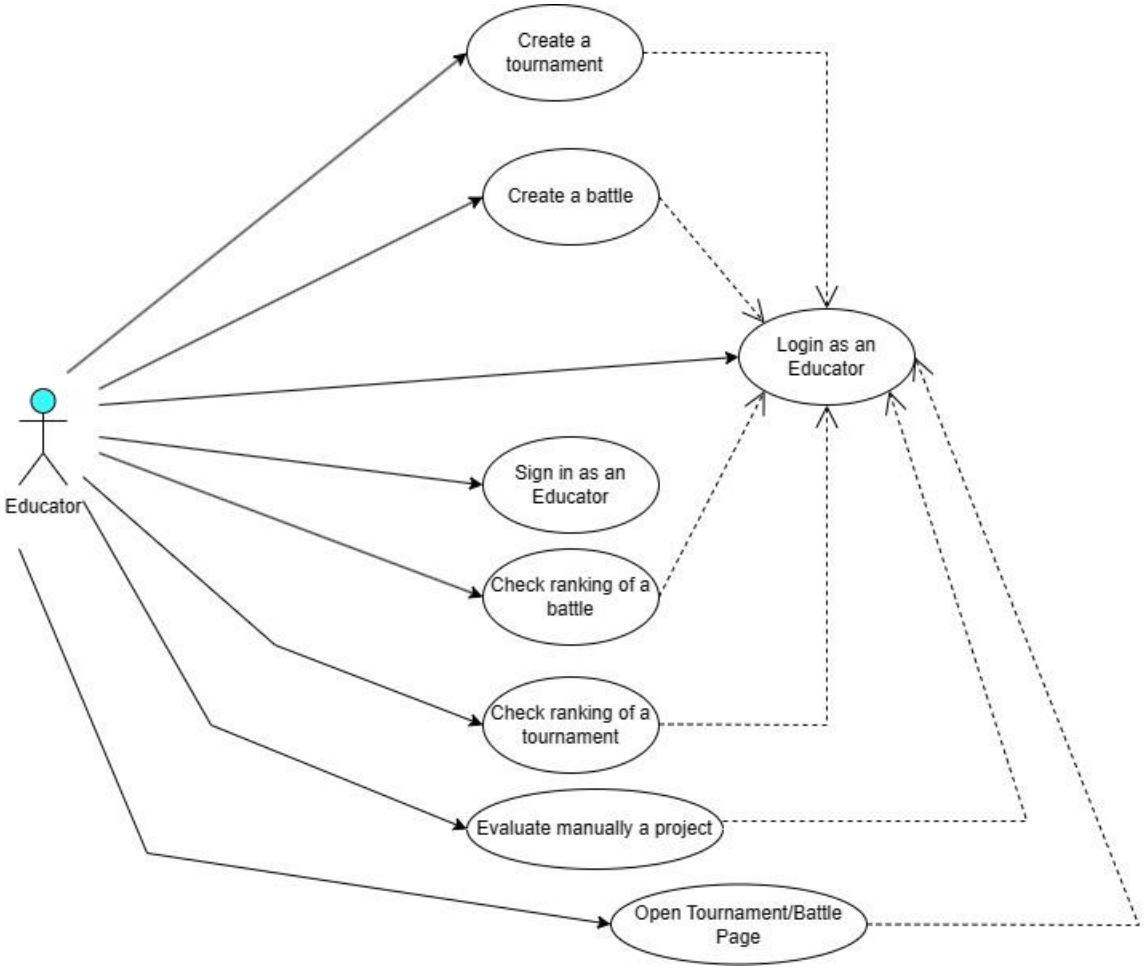



*Figure 10 Educator Use Case*

### 3.2.2. Use cases

[UC1]. Sign in of a new student

| | |
|---|---|
| *Name* | Sign in of a new student |
| *Actors* | Student |
| *Entry Condition* | A student opens the site |
| *Event Flow* | 1. Student opens the site<br>2. System shows the home page<br>3. Student clicks the “Sign in as a student”<br>4. System shows the sign in page<br>5. Student inserts their personal data<br>6. Student press the “Sign in” button<br>7. System checks if the credential inserted in the previous step are not already used by another account<br>8. System sends a confirmation email to the user<br>9. Student confirms the registration by clicking the link received by email |
| *Exit Condition* | Registration has been successful. The student data is stored into the system’s database. The student then will be able to login in the system by using their credentials |
| *Exception* | 7. The username or the email are already used.<br>The system comes back to point 3 adding an error message |

[UC2]. Sign in of a new educator

| | |
|---|---|
| *Name* | Sign in of a new educator |
| *Actors* | Educator |
| *Entry Condition* | An educator opens the site |
| *Event Flow* | 1. Educator opens the site<br>2. System shows the home page<br>3. Educator clicks the “Sign in as an educator”<br>4. System shows the sign in page<br>5. Educator inserts their personal data<br>6. Educator press the “Sign in” button<br>7. System checks if the credential inserted in the previous step are not already used by another account<br>8. System sends a confirmation email to the user<br>9. Student confirms the registration by clicking the link received by email |
| *Exit Condition* | Registration has been successful. The educator data is stored into the system’s database. The educator then will be able to login in the system by using their credentials |
| *Exception* | 7. The username or the email are already used.<br>The system comes back to point 3 adding an error message |

[UC3]. Login of a student

| | |
|---|---|
| *Name* | Login of a student |
| *Actors* | Student |
| *Entry Condition* | A student opens the site |
| *Event Flow* | 1. Student opens the site<br>2. System shows the home page<br>3. Student clicks the “Log in as a student” button<br>4. System shows the login page<br>5. Student inserts their credential<br>6. Student press the “Login” button<br>7. System checks the credentials<br>8. System shows the student’s home page |
| *Exit Condition* | Student has accessed to his personal home page |
| *Exception* | 7. Student inserted invalid credentials.<br>The system comes back to point 3 adding an error message |

[UC4]. Login of an educator

| | |
|---|---|
| *Name* | Login of an educator |
| *Actors* | Educator |
| *Entry Condition* | An educator opens the site |
| *Event Flow* | 1. Educator opens the site<br>2. System shows the home page<br>3. Educator clicks the “Log in as an educator” button<br>4. System shows the login page<br>5. Educator inserts their credential<br>6. Educator press the “Login” button<br>7. System checks the credentials<br>8. System shows the educator’s home page |
| *Exit Condition* | Educator has accessed to his personal home page |
| *Exception* | 7. Educator inserted invalid credentials.<br>The system comes back to point 3 adding an error message |

[UC5]. Creation of a tournament

| | |
|---|---|
| *Name* | Creation of a tournament |
| *Actors* | Educator and student |
| *Entry Condition* | Educator opens the site |
| *Event Flow* | 1. Educators open the site<br>2. System shows the home page<br>3. Educator logs in<br>4. System shows the educator's home page<br>5. Educator fills the form for the creation of a new tournament<br>6. Educator clicks the "Create a new tournament" button<br>7. System saves the tournament<br>8. System shows the tournament details page<br>9. System notifies all students subscribed to CKB about the creation of the tournament<br>10. Educator clicks the "Adds another educator as a collaborator" button<br>11. System shows a form to fill<br>12. Educator fills the form with the username of the educator to add<br>13. System saves the modification at the tournament |
| *Exit Condition* | Tournament has been successful created. The tournament data is stored into the system's database.<br>Now student can join the tournament whenever they want until the registration deadline expires. |
| *Exception* | 9. If the educator does not want to add collaborators. The steps 10 through 13 are skipped |

[UC6]. Creation of a battle

| | |
|---|---|
| *Name* | Creation of a battle |
| *Actors* | Educator and student |
| *Entry Condition* | An educator opens the site |
| *Event Flow* | 1. Educator opens the site<br>2. System shows the home page<br>3. Educator logs in<br>4. System shows the educator's home page<br>5. Educator clicks on the tournament in which he wants to create a battle<br>6. System shows the tournament details page<br>7. Educator press the "Create a new battle" button<br>8. System shows the page for creating the battle<br>9. Educator fills the form<br>10. Educator clicks the "Create" button<br>11. System saves the battle<br>12. System notifies all students that are subscribed to the tournament that a new battle has been created |
| *Exit Condition* | Battle has been successful created. The battle data is stored into the system's database.<br>Now student can join the battle whenever they want until the registration deadline expires. |

[UC7]. Student joins a tournament

| | |
|---|---|
| *Name* | Student joins a tournament |
| *Actors* | Student |
| *Entry Condition* | A student opens the site |
| *Event Flow* | 1. Student opens the site<br>2. System shows the home page<br>3. Student logs in<br>4. System shows the student's home page<br>5. Student searches for a tournament they want to enter<br>6. System shows details about selected tournament<br>7. Student clicks on a specific tournament<br>8. System shows the tournament details page<br>9. Student click the "Join tournament" button |
| *Exit Condition* | Registration has been successful. The subscription is stored into the system's database.<br>Now student will have to wait until the registration deadline expires and the educator creates a new battle. |

[UC8]. Student joins a battle

| | |
|---|---|
| *Name* | Student joins a battle |
| *Actors* | Student |
| *Entry Condition* | A student opens the site |
| *Event Flow* | 1. Student opens the site<br>2. System shows the home page<br>3. Student logs in<br>4. System shows the student's home page<br>5. Student searches for a tournament in which they are registered<br>6. Student clicks on specific tournament<br>7. System shows the tournament details page<br>8. Student click on the battle which he is interested too<br>9. System shows the battle details page<br>10. Student click the "Join battle" button<br>11. Student selects if they participate alone or with a group<br>12. Student invites all mates to the groups<br>13. System waits until registration deadline expires<br>14. System creates a repository for the group<br>15. Student forks the repository and activate GHA |
| *Exit Condition* | Registration has been successful. The subscription is stored into the system's database.<br>Now student is able to modify the project and receive an evaluation by committing the modification into the main branch of the repo. |
| *Exception* | 9. If the deadline for tournament registration has passed.<br>The steps 10 through 15 are skipped.<br>11. If the student decides to participate alone<br>The step 12 is skipped |

[UC9]. Closing a tournament

| | |
|---|---|
| *Name* | Closing a tournament |
| *Actors* | Educator and student |
| *Entry Condition* | And educator opens the site |
| *Event Flow* | 1. Educator opens the site<br>2. System shows the home page<br>3. Educator logs in<br>4. System shows the educator's home page<br>5. Educators clicks on the tournament which he wants to close<br>6. System shows the tournament details page<br>7. Educator clicks the "Close the tournament" button<br>8. System publishes final ranking of the tournament<br>9. System notifies all students subscribed in the tournament the closing of the tournament |
| *Exit Condition* | Tournament has been successful closed. The data is stored into the system's database. |

[UC10]. Evaluation of a code

| | |
|---|---|
| *Name* | Evaluation of a code |
| *Actors* | Student, GHA and educator |
| *Entry Condition* | A student pushes a commit into the main branch of the repo |
| *Event Flow* | 1. Student pushes a commit into the main branch of the repo<br>2. GHA notify the system about the push<br>3. System clones the project in a protected environment<br>4. System tests the code based on the test case and the properties chosen by the educator<br>5. Educator manually evaluates the code and modifies the evaluation made by the system |
| *Exit Condition* | Evaluation has been successful. New data and new ranking are stored in the system database.<br>Now everyone can check the updated ranking |
| *Exception* | 5. This step may not happen because is an optional step. |

[UC11]. Check ranking of a tournament

| | |
|---|---|
| *Name* | Check ranking of a tournament |
| *Actors* | User |
| *Entry Condition* | A user opens the site |
| *Event Flow* | 1. User opens the site<br>2. System shows the home page<br>3. User logs in<br>4. System shows the user's home page<br>5. User searches for the tournament he is interested<br>6. User clicks the interested tournament<br>7. System shows the tournament details page<br>8. User clicks the "Ranking" button<br>9. System shows ranking of the tournament |
| *Exit Condition* | Now user can check the ranking of the tournament |

[UC12]. Check ranking of a battle

| | |
|---|---|
| *Name* | Check ranking of a battle |
| *Actors* | User |
| *Entry Condition* | A user opens the site |
| *Event Flow* | 1. User opens the site<br>2. System shows the home page<br>3. User logs in<br>4. System shows the user's home page<br>5. User searches for the tournament he is interested<br>6. User clicks the interested tournament<br>7. System shows the tournament details page<br>8. User clicks the battle in the tournament that they are interested too<br>9. System shows the battle details page<br>10. User click the "ranking" button<br>11. System shows the ranking of the battle |
| *Exit Condition* | Now user can check the ranking of the battle |

[UC13]. Show tournament details page

| | |
|---|---|
| *Name* | Show tournament details page |
| *Actors* | User |
| *Entry Condition* | A user opens the site |
| *Event Flow* | 1. User opens the site<br>2. System shows the home page<br>3. User logs in<br>4. System shows the user home page<br>5. User searches for the tournament they are interested<br>6. User clicks the tournament they are interested<br>7. System shows the tournament details page |
| *Exit Condition* | Now user has access to all tournament information and functionality |

[UC14]. Show battle details page

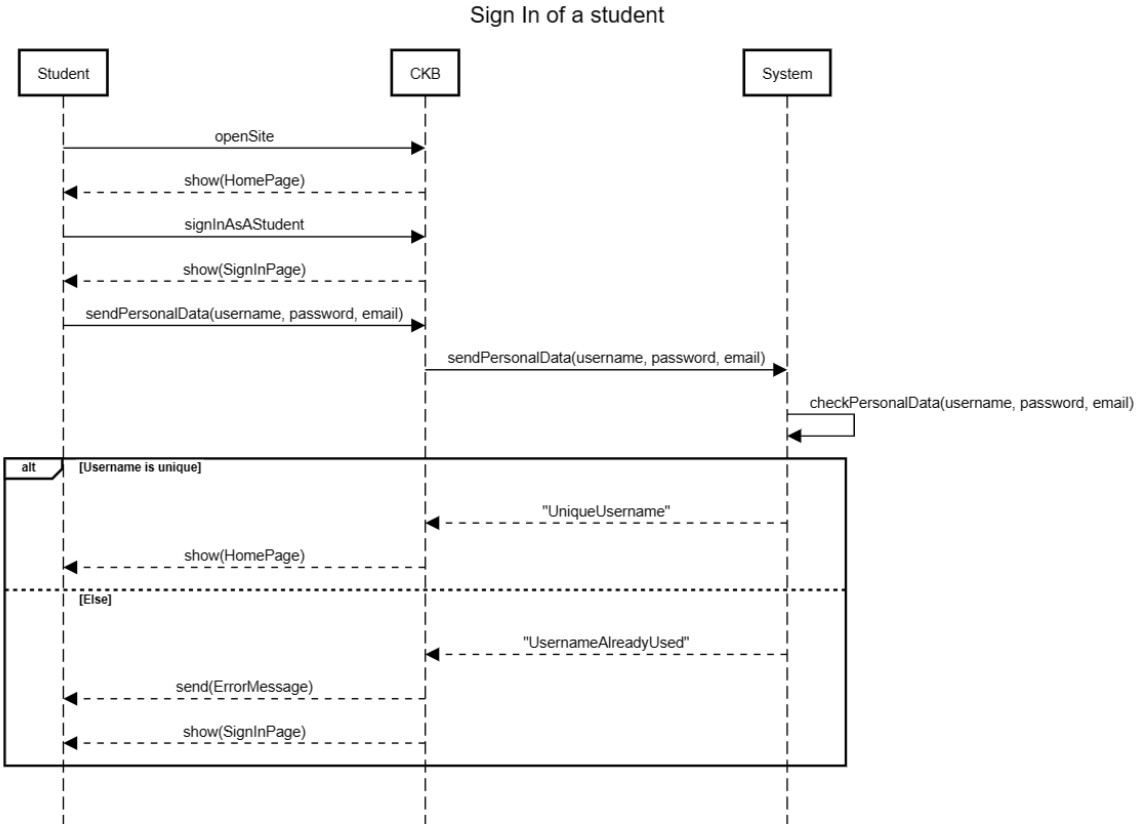
| | |
|---|---|
| *Name* | Show battle details page |
| *Actors* | User |
| *Entry Condition* | A user opens the site |
| *Event Flow* | 1. User opens the site<br>2. System shows the home page<br>3. User logs in<br>4. System shows the user home page<br>5. User searches for the tournament they are interested<br>6. User clicks the tournament they are interested<br>7. System shows the tournament details page<br>8. User clicks the battle in the tournament that they are interested too<br>9. System shows the battle details page |
| *Exit Condition* | Now user has access to all battle information and functionality |

[UC15]. Student accepts an invite for a battle

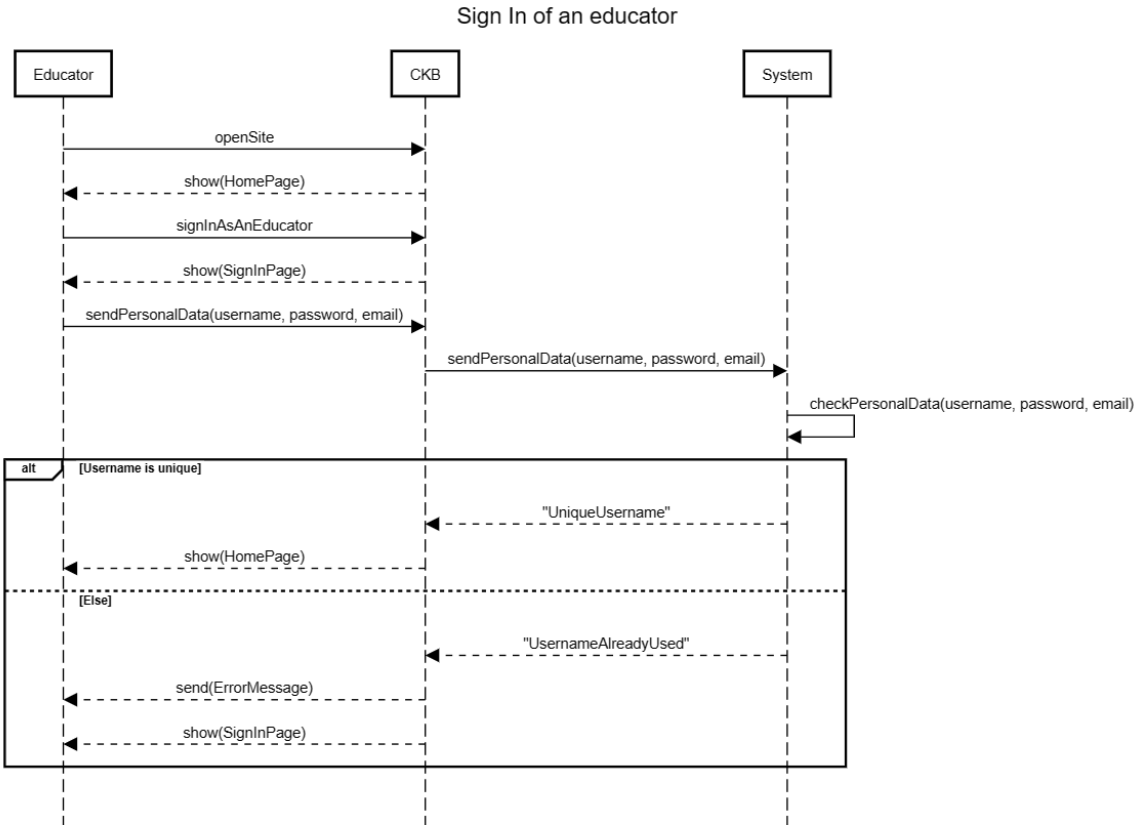
| | |
|---|---|
| *Name* | Student accepts an invite for a battle |
| *Actors* | Student |
| *Entry Condition* | A student opens the site |
| *Event Flow* | 1. Student opens the site<br>2. System shows the home page<br>3. Student logs in<br>4. System shows the student home page<br>5. Student search for a tournament<br>6. Student clicks on a specific tournament<br>7. System shows the tournament details page<br>8. Student clicks on the battle which he is interested too<br>9. System shows the battle details page<br>10. Student clicks on the invitation for the battle<br>11. System shows all the invitation received<br>12. Student clicks “Join with this group” button<br>13. System waits until registration deadline expires<br>14. System creates a repository for the group<br>15. Student fork the repository and activates GHA |
| *Exit Condition* | Registration has been successful. The subscription is stored into the system’s database.<br>Now student is able to modify the project and receive an evaluation by committing the modification into the main branch of the repo. |
| *Exception* | 11. System shows an empty page in case they have not received any invite |

### 3.2.3. Sequence diagram

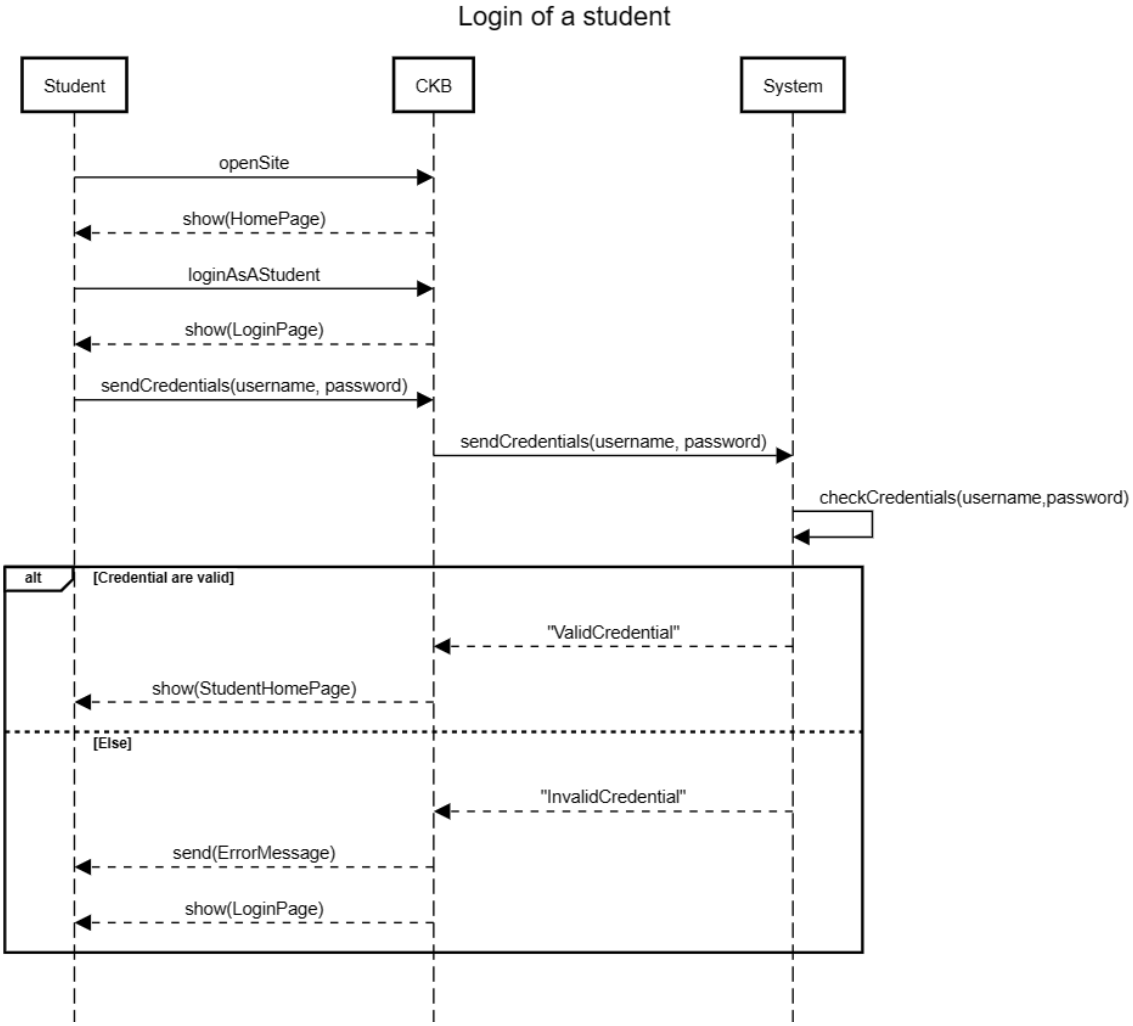
[SD1]. Sign in of a new student



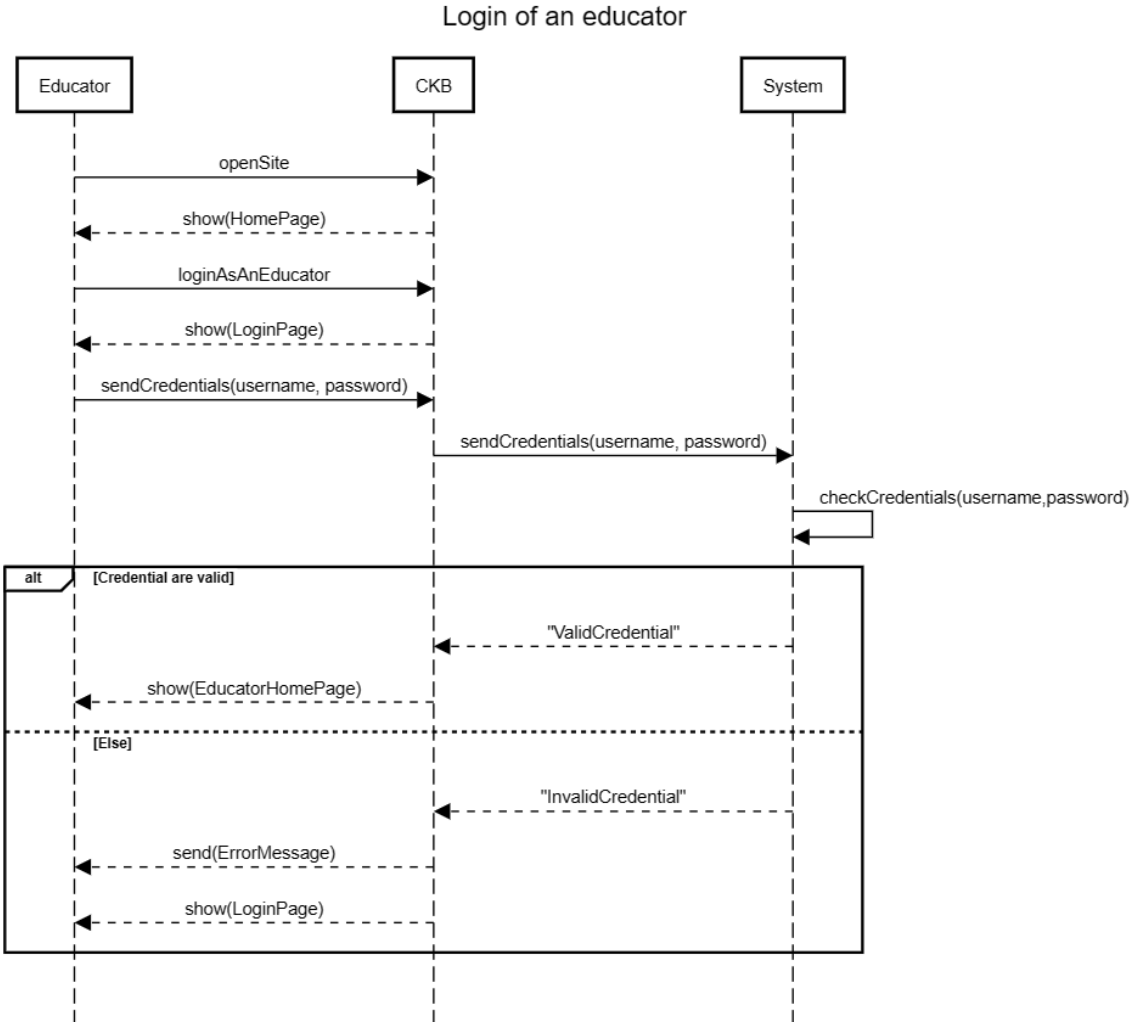

[SD2]. Sign in of a new educator



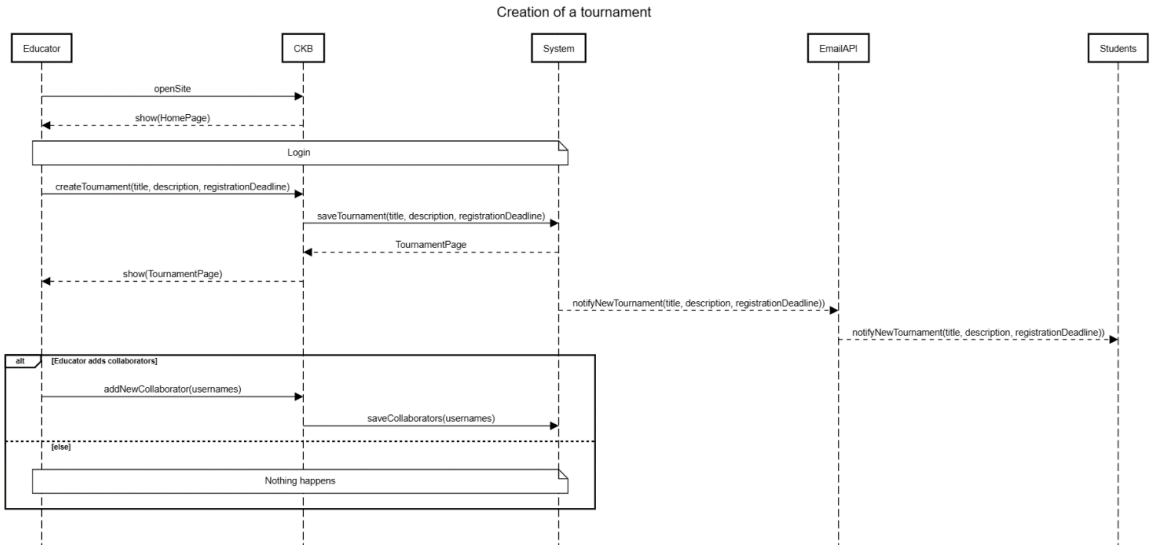
[SD3]. Log in of a student



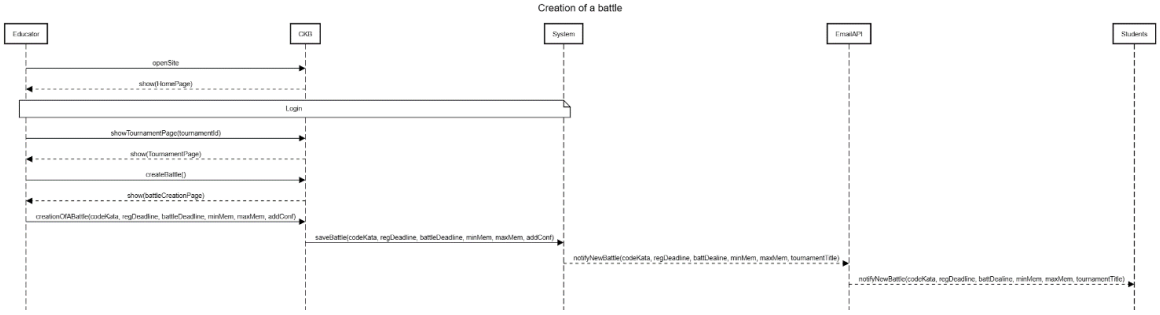
[SD4]. Log in of an educator



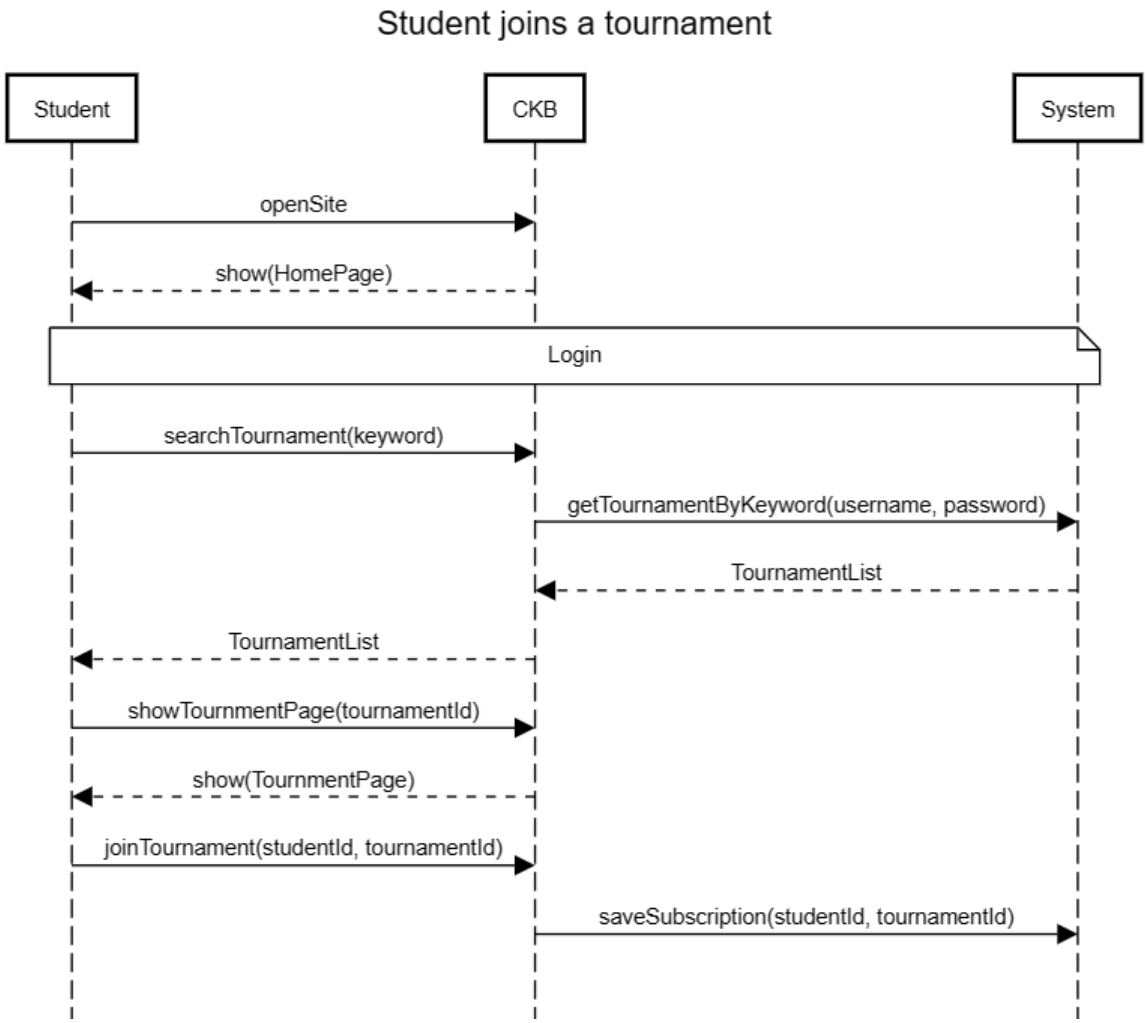

[SD5]. Creation of a tournament



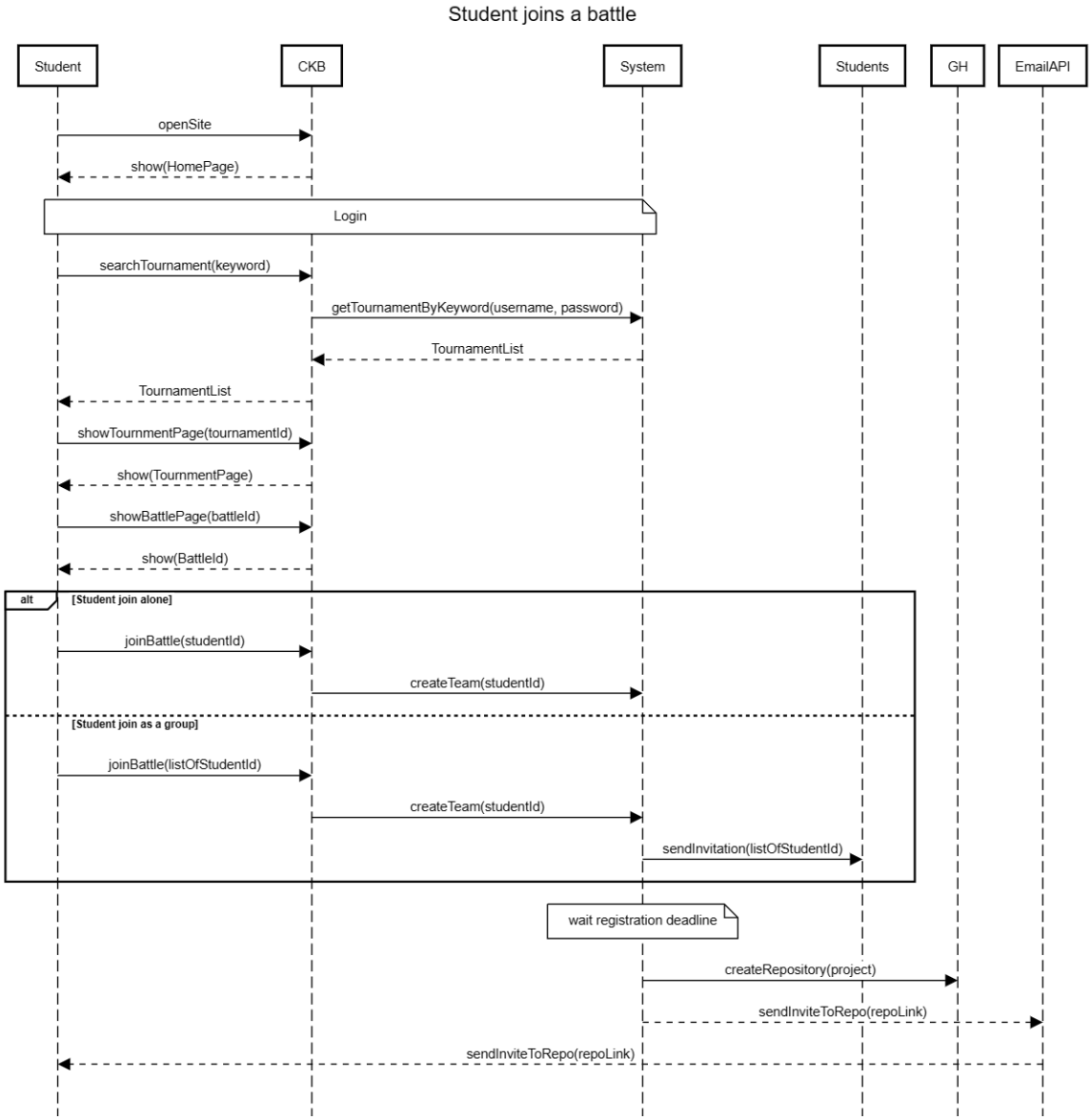
[SD6]. Creation of a battle



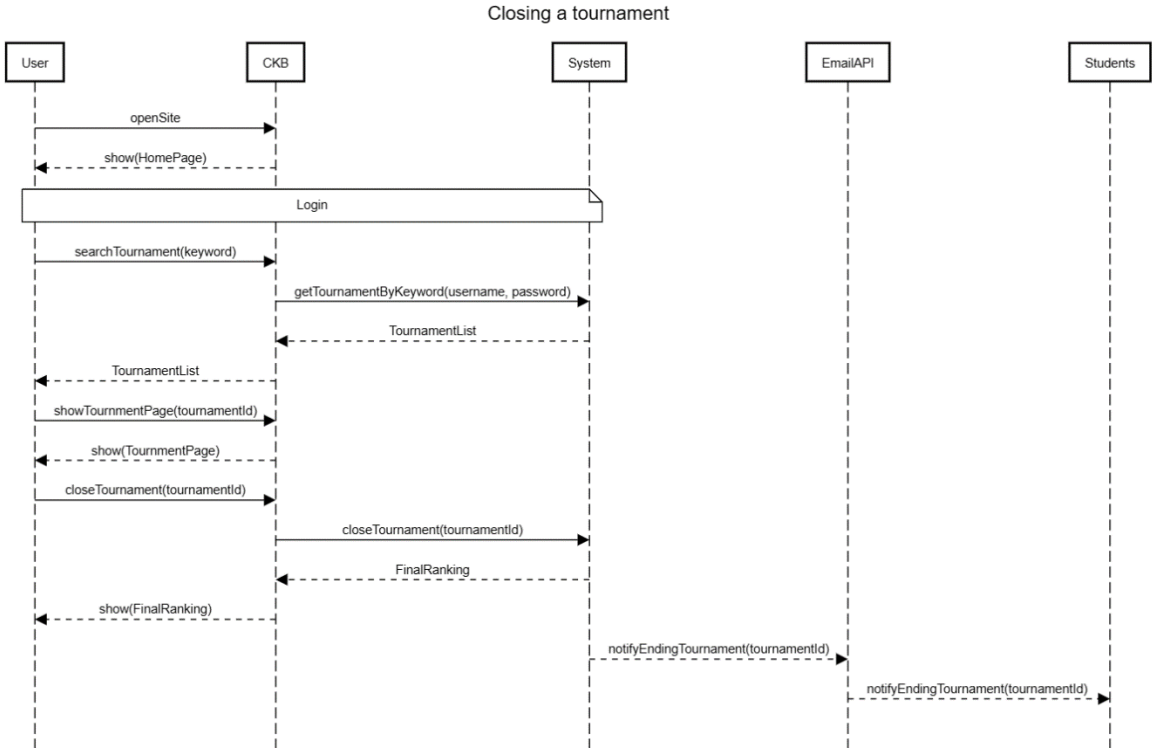

[SD7]. Student joins a tournament



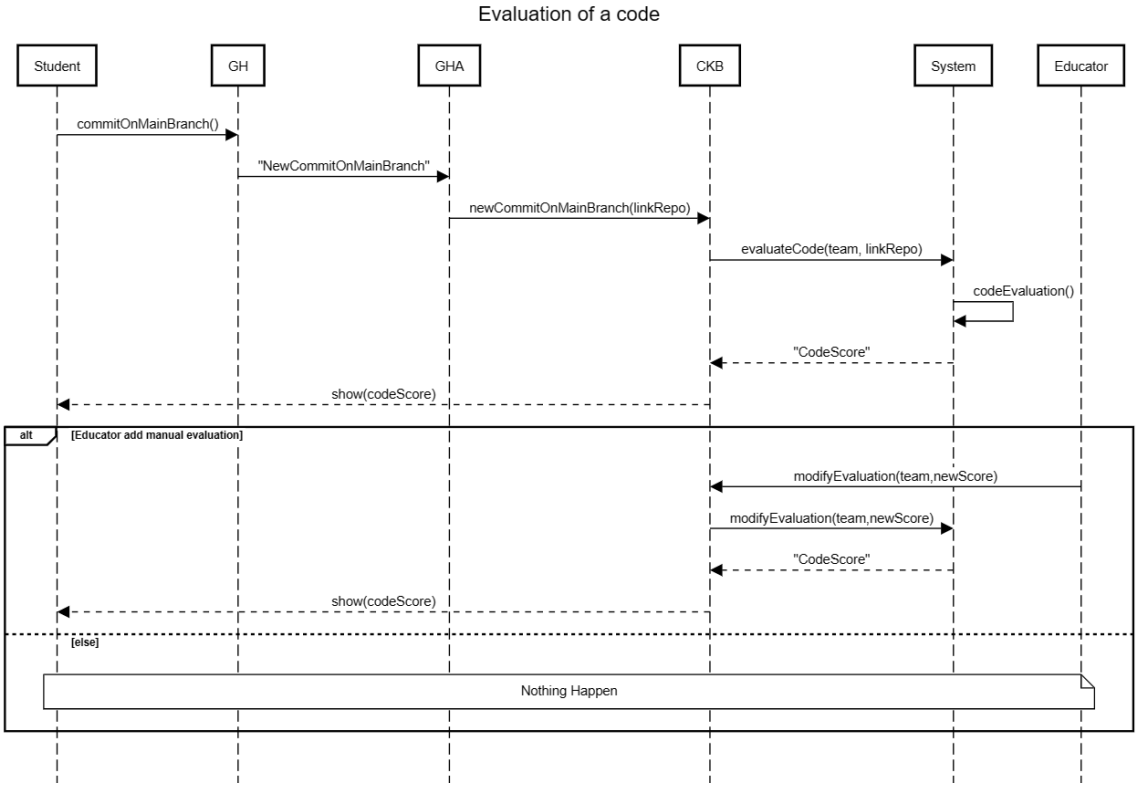
## [SD8]. Student joins a battle



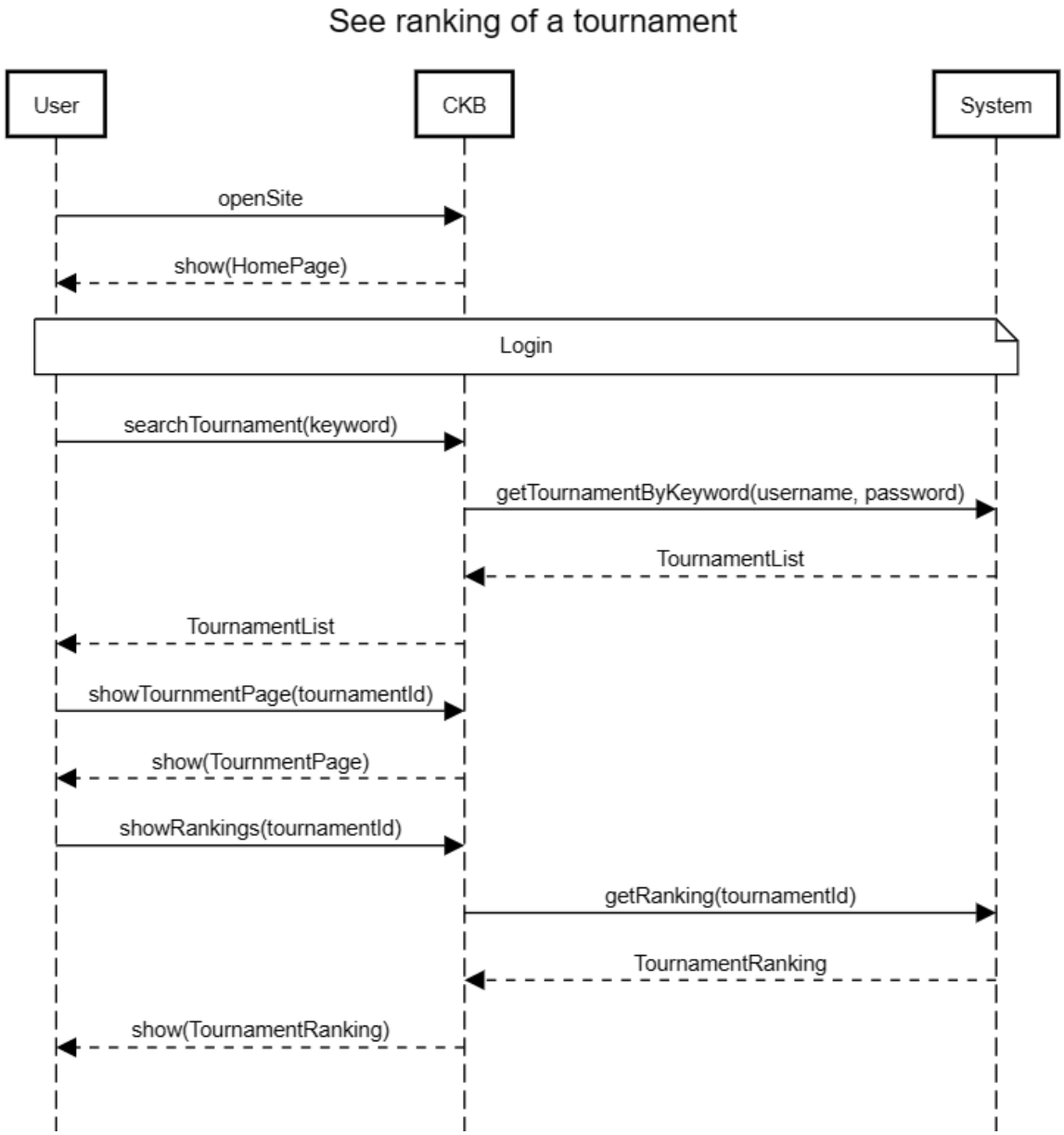
## [SD9]. Closing a tournament



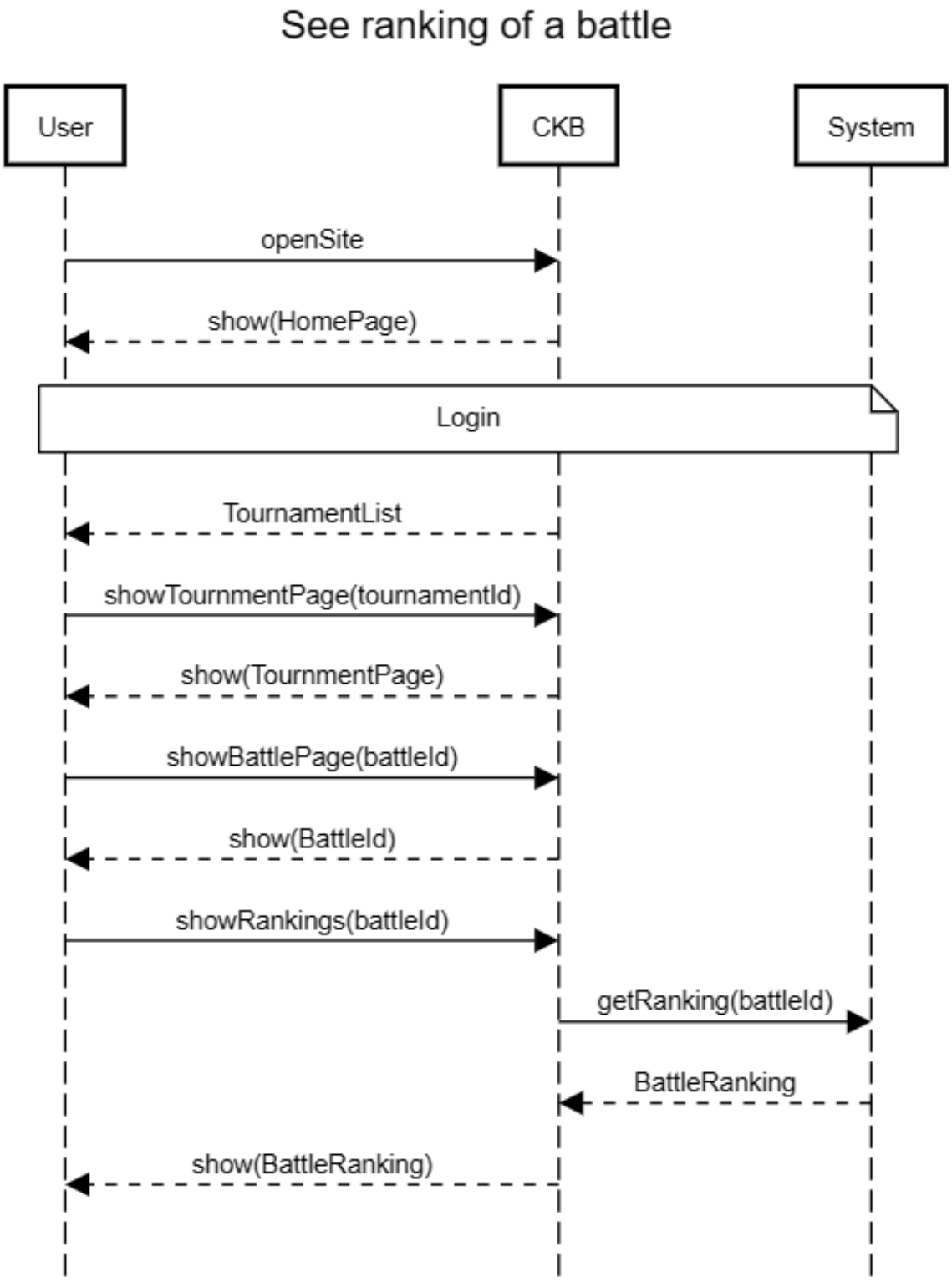

## [SD10]. Evaluation of a code



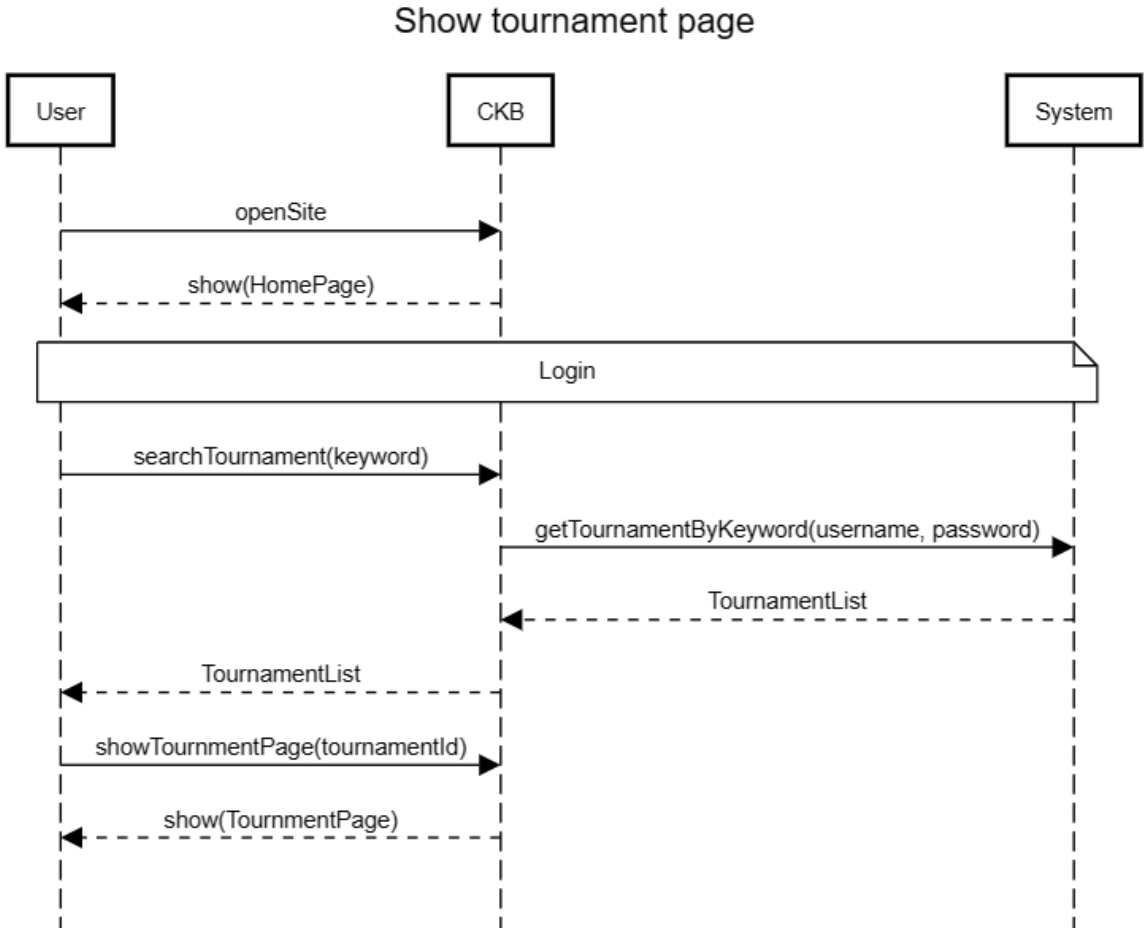
[SD11]. Check ranking of a tournament



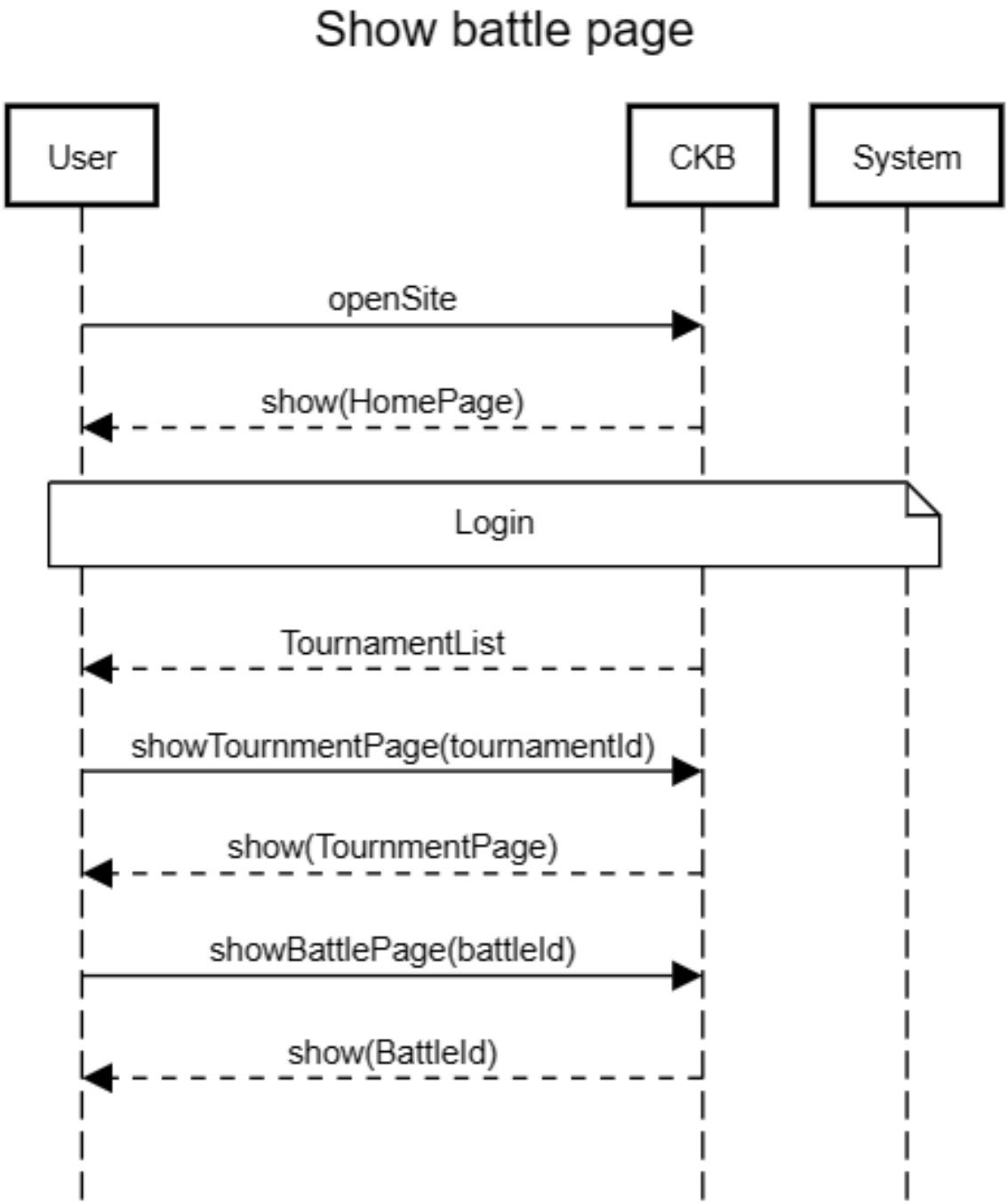
[SD12]. Check ranking of a battle



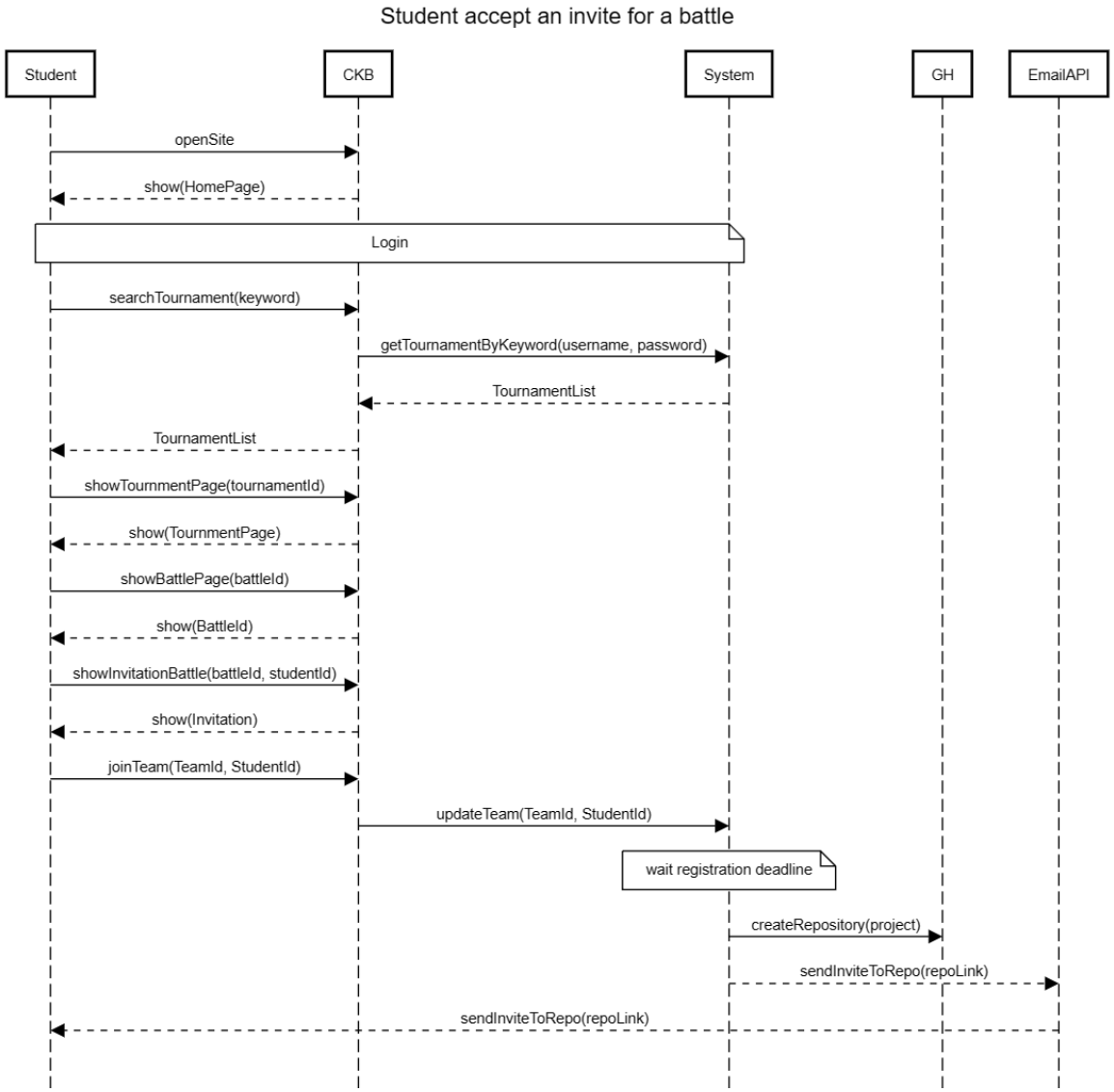
[SD13]. Show tournament details page



[SD14]. Show battle details page



[SD15]. Student accepts an invite for a battle




### 3.2.4. Requirement mapping

- Educator creates a tournament and within it, he creates one or more battle.

| *Requirements* | *Domain Assumption* |
|---|---|
| [R2]. System allows educators to sign up<br>[R4]. System allows educators to login<br>[R5]. System allows educator to create a new tournament and to add all the needed information<br>[R6]. System allows educator to grant the permission to other colleagues to create battle in a specific tournament<br>[R7]. System allows educator to create a battle inside a particular tournament by adding CK and the deadlines | [DA1]. Users must have a valid email address<br>[DA4]. Each user will create only 1 account<br>[DA5]. Personal data given by User during the registration process are assumed true<br>[DA6]. User must have an internet connection<br>[DA7]. Educator must upload the correct CK |

- Student registers for a tournament and will receive all notifications related to it.

| *Requirements* | *Domain Assumption* |
|---|---|
| [R1]. System allows students to sign up<br>[R3]. System allows students to login<br>[R12]. System notifies students about the creation of a new tournament<br>[R13]. System notifies students about the creation of a new battle in a specific tournament in which they have subscribed<br>[R15]. System notify student about publication of final ranking of a specific tournament<br>[R16]. System allow student to join a tournament<br>[R22]. System allows students to search tournament by key words | [DA1]. Users must have a valid email address<br>[DA2]. Students must have a GH profile<br>[DA4]. Each user will create only 1 account<br>[DA5]. Personal data given by User during the registration process are assumed true<br>[DA6]. User must have an internet connection |

- Student joins a battle within a tournament they are register for and invite some friends to participate with them, thus creates a team for the battle.

| *Requirements* | *Domain Assumption* |
|---|---|
| [R1]. System allows students to sign up<br>[R3]. System allows students to login<br>[R14]. System notify student about publication of final ranking of a specific battle<br>[R17]. System allow student to join a battle<br>[R20]. System, at end of the registration period of a battle, create a GH repo and send invitation to all member of the group | [DA1]. Users must have a valid email address<br>[DA2]. Students must have a GH profile<br>[DA3]. Students need to know how to access GitHub, how to fork a repository and how to enable GitHub Actions<br>[DA4]. Each user will create only 1 account<br>[DA5]. Personal data given by User during the registration process are assumed true<br>[DA6]. User must have an internet connection |

- Student receives an evaluation for the code he pushed to their GitHub repository.

| *Requirements* | *Domain Assumption* |
|---|---|
| [R21]. System automatically evaluates the code in the main branch of the repo after each commit in it | [DA1]. Users must have a valid email address<br>[DA2]. Students must have a GH profile<br>[DA3]. Students need to know how to access GitHub, how to fork a repository and how to enable GitHub Actions<br>[DA4]. Each user will create only 1 account<br>[DA5]. Personal data given by User during the registration process are assumed true<br>[DA6]. User must have an internet connection |

- Educator reviews all the evaluated code, and if they deem it necessary, he can modify the evaluation they have received.

| *Requirements* | *Domain Assumption* |
|---|---|
| [R2]. System allows educators to sign up<br>[R4]. System allows educators to login<br>[R10]. System allows educator to modify the evaluation manually of a specific group in a battle. | [DA1]. Users must have a valid email address<br>[DA4]. Each user will create only 1 account<br>[DA5]. Personal data given by User during the registration process are assumed true |

- Educator closes a tournament and the platform publishes the ranking.

| *Requirements* | *Domain Assumption* |
|---|---|
| [R2]. System allows educators to sign up<br>[R4]. System allows educators to login<br>[R11]. System allows educator to close a tournament<br>[R15]. System notify student about publication of final ranking of a specific tournament | [DA1]. Users must have a valid email address<br>[DA4]. Each user will create only 1 account<br>[DA5]. Personal data given by User during the registration process are assumed true<br>[DA6]. User must have an internet connection |

- Both students and educators check the update ranking of the battle and the tournament.

| *Requirements* | *Domain Assumption* |
|---|---|
| [R1]. System allows students to sign up<br>[R2]. System allows educators to sign up<br>[R3]. System allows students to login<br>[R4]. System allows educators to login<br>[R8]. System allows educator to see rankings of each tournament, which he created or in which he has been nominated collaborator<br>[R9]. System allows educator to see ranking of a specific battle<br>[R18]. System allows students to see rankings of each tournament, which they are sub scripted<br>[R19]. System allows students to see ranking of a specific battle | [DA1]. Users must have a valid email address<br>[DA2]. Students must have a GH profile<br>[DA3]. Students need to know how to access GitHub, how to fork a repository and how to enable GitHub Actions<br>[DA4]. Each user will create only 1 account<br>[DA5]. Personal data given by User during the registration process are assumed true<br>[DA6]. User must have an internet connection |

### *3.3. Performance requirements*

Our application, designed to enhance software programming skills, commits to ensuring a timely and reliable response at every stage of the process. Notifications regarding the creation of new tournaments will be sent in real-time to all subscribers, while notifications related to battles will be limited to participants in the corresponding tournament. This selection will occur swiftly, allowing all users to have an ample time frame to register.

User interactions with rankings, tournament registrations, and battles will proceed seamlessly, even in high-load situations, ensuring an optimal user experience.

A crucial aspect is the efficient management of the creation of repositories on GitHub, occurring at the end of the registration period for a battle. The system must create these repositories swiftly and efficiently, adapting to simultaneously manage various battles without negative impacts on overall performance.

Commit monitoring and automatic code testing are critical phases in our system. It is essential to ensure that notifications are timely whenever a commit is made to the battle repositories and that code testing occurs swiftly and efficiently, providing immediate feedback to students.

During tournaments, the system will maintain high performance, simultaneously managing communications, rankings, and repository creation. The display of results and rankings for ongoing tournaments will occur with rapid response times.

Furthermore, our platform will guarantee high reliability and availability. During critical phases, such as repository creation and the execution of automatic tests, the system will minimize downtime. In case of spikes in participation or high loads, it will dynamically adapt to maintain optimal performance levels, preserving the quality of service offered to users.

## *3.4. Design constraints*

### 3.4.1. Standards compliace

Regarding data privacy, the CKB adheres to the GDPR, a European Union regulation establishing rules for the protection of personal data and privacy for individuals within the European Union and the European Economic Area. Therefore, users' personal data and information will not be used for commercial purposes.

The system will be fully compliant with the D0-178C standard. This implies that all software tests will be conducted in accordance with specified requirements, enabling the association of each requirement with a dedicated test case used to verify whether it has been satisfied.

Additionally, the system must adopt international standards concerning the use and representation of date and time. This will ensure consistency and international compatibility in interpreting temporal information, contributing to a coherent and accurate management of temporal data within the system.

### 3.4.2. Hardware limitations

The system requires a robust hardware infrastructure to ensure reliable and timely performance during daily activities and potential usage spikes. The server's processing capacity must be sufficiently scaled to handle large volumes of data, ensuring smooth operations even during high-traffic situations. Resilience to peak requests is crucial to ensure the application can concurrently manage a significant number of users and activities without compromising performance.

Simultaneously, the server's storage capacity must be adequate to efficiently handle user data, including details related to tournaments and battles. Efficient data management necessitates a solid storage capacity to ensure the availability and accessibility of information at all times. This aspect is critical to support daily operations and provide a strong foundation for future expansion.

On the user's end, it is essential to use a browser enabled for JavaScript management. This technological dependency is necessary to fully leverage the interactive and dynamic features of the application, offering an optimal user experience. Ensuring that the user has a JavaScript-compatible browser is therefore a fundamental prerequisite for the proper functioning of the system and accessing all its advanced features.

## *3.5. Software system attributes*

### 3.5.1. Reliability

To ensure high reliability, the system will implement advanced mechanisms for real-time error detection and management. Clear and contextualized alerts will be adopted, providing users with precise information in case of any issues. This approach ensures immediate and targeted intervention to resolve problems, preserving the integrity and reliability of operations.

Furthermore, the system's architecture will be designed with a distribution across multiple servers and the implementation of data replication strategies. This design aims to prevent single points of failure that could compromise the system's integrity as a whole. Distribution across multiple servers and data replication will contribute to ensuring operational continuity even in the presence of hardware failures or malfunctions.

To reduce the risk of errors during data-saving operations in the database, the system will use dedicated triggers for error management. These triggers will provide active control during critical phases, intervening promptly to correct any inconsistencies and ensure the coherence of stored data.

Additionally, if a comprehensive traceability of system activities and in-depth analysis of errors are desired, it will be possible to implement a logging system. This tool will allow for detailed recording of all operations, facilitating the understanding and resolution of any issues. Overall, this integrated strategy of active error management, distribution across multiple servers, data replication, and detailed logging of activities will contribute to ensuring high reliability of the application over time.

### 3.5.2. Availability

System availability is a fundamental requirement to ensure uninterrupted operation over time. To optimize availability and ensure operational continuity even in the face of failures, the server implementation will be designed with a distributed approach. This means that the system will be deployed across multiple machines that constitute the server, allowing load balancing and mitigating potential impacts from hardware failures or malfunctions in a single machine.

Data replication will be implemented to ensure redundancy of critical information. In the event of machine malfunctions or data loss, data replicas will be readily available, ensuring service continuity without significant interruptions. This approach will help minimize the risk of data loss and maintain high availability standards, even in scenarios of unforeseen failures.

Additionally, the system will be designed to automatically detect faults and respond accordingly, activating failover mechanisms to ensure that user requests can be handled without interruptions. Efficient management of distribution, replication, and data redundancy is essential to ensure high system availability, minimizing downtime, and overall enhancing user experience.

### 3.5.3. Security

In accordance with security practices, the system will take several measures to safeguard sensitive information and maintain data integrity.

All communication between the client and the server, as well as with the database management system (DBMS), will be via HTTPS, so that there will be a secure, encrypted transfer of data.

Passwords will be hashed before being stored in the database, thus reducing the risk associated with storing passwords in plain text. In addition, data transmission from client to server will be encrypted to protect against potential eavesdropping and unauthorized access.

To further enhance data security, sensitive information will be encrypted before storage, minimizing the impact of potential data leaks. In addition, strict access controls will be implemented to prevent third parties from accessing sensitive data through group requests,

strengthening the overall resilience of the system against unauthorized access and potential security breaches.

### 3.5.4. Maintainability

Code maintainability is a fundamental cornerstone in our approach to software development.

To ensure a code structure that is easily manageable and adaptable over time, we will adopt well-known and established design patterns. The use of these patterns provides a coded architecture that reflects best practices and promotes an intuitive understanding of the code by developers.

Simultaneously, our focus on maintainability is reflected in extensive documentation detailing the implemented functions. This documentation not only provides clarity on the behaviour of functions but also on the broader context in which they are integrated into the system's architecture.

Comprehensive documentation facilitates the work of developers making changes, aiding in understanding the relationships between different parts of the code and thus contributing to the long-term maintainability and improvement of our software.

### 3.5.5. Portability

The developed CKB application is designed to run on any operating system, ensuring significant flexibility in access across various platforms.

The only required condition is the presence of a browser capable of running JavaScript. This design choice aims to maximize the accessibility of the application, enabling users to enjoy its features regardless of the operating system used. Thanks to this approach, users will experience a seamless and comprehensive interface, irrespective of their chosen device or operating system, ensuring greater adaptability and accessibility of the application.

# 4. REQUIREMENT TRACEABILITY

# 5. IMPLEMENTATION, INTEGRATION AND TEST PLANNING

## *5.1. Signatures*

- sig DateTime {} //it represents a couple <data, time>
- sig CodeKata {} //it represents a CK
- abstract sig User {} //it represents a user
- sig Student extends User {} //it represents a student
- sig Educator extends User {} //it represents an educator
- sig Tournament { //it represents a tournament
  registrationDeadline: one DateTime, //the tournament's registration deadline
  students: some Student, //the students registered for the tournament
  educators: some Educator //the educators that are responsible for managing
  }
- sig Battle { //it represents a battle
  registrationDeadline: one DateTime, //the battle's registration deadline
  submissionDeadline: one DateTime, //the deadline by which possible solutions to the battle must be administered
  tournament: one Tournament, //the tournament of which the battle is a part
  code: one CodeKata //it represents the battle's CK
  } {
  registrationDeadline != submissionDeadline and
  registrationDeadline != tournament.registrationDeadline and
  submissionDeadline != tournament.registrationDeadline
  }
- sig Team { //it represents a team
  students: some Student, //students make up the team
  battleT: one Battle //the battle in chich the team is enrolled
  }
- sig Evaluation { //it represents an evaluation
  time: one DateTime, //the time at which the assessment was made
  team: one Team, //the team that received the evaluation
  battleE: one Battle, //the battle in which the team that received the evaluation is enrolled
  grade: one Int, //the team grade
  code: one CodeKata //it represents the evaluation's CK
  } {
  time != battleE.registrationDeadline and
  time != battleE.submissionDeadline and
  time != battleE.tournament.registrationDeadline and
  grade > 0
  }

### 5.2. *Facts*

- fact EachBattleHasAnEvaluation { //each battle has an evaluation
  all e1, e2: Evaluation |
  (e1 != e2) implies ((e1.battleE !=e2.battleE) or (e1.team!= e2.team))
  }

- fact EachTeamInABattleHasDifferentStudent { //each team in a battle has different student
  all t1, t2: Team,
  b: Battle |
  (((t1 != t2) and (b in t1.battleT) and (b in t2.battleT)) implies
  (all s: Student |
  (s in t1.students and s not in t2.students) or (s not in t1.students and s in t2.students)
  or (s not in t1.students and s not in t2.students)))
  }

- fact EachStudentInABattleIsInTheTournament { //each student in a battle is in the tournament
  all t:Team,
  s:Student |
  (s in t.students) implies (s in t.battleT.tournament.student)
  }

- fact BattleAndEvaluationHasSomeCodeKata { //each evaluation with its corresponding battle have the same CodeKata
  all e: Evaluation |
  e.battleE.code = e.code
  }

- fact TeamAndEvaluationHasSomeBattle { //each evaluation with its relative team have the same battle
  all e: Evaluation |
  e.team.battleT = e.battleE
  }

- fact EachTeamHasAnEvaluation{ //each team has an evaluation
  all t:Team |
  #team.t = 1
  }

- fact EachBattleHasDifferentCodeKata{ //each battle has different CodeKata
  all b1, b2: Battle |
  (b1 != b2) implies (b1.code != b2.code)
  }

### 5.3. *Predicate*

- pred show {
  #Battle = 2
  #Team = 4
  #Student = 6
  #Tournament = 2
  }

- run show for 8

Below we find two representations of the system as specified above.

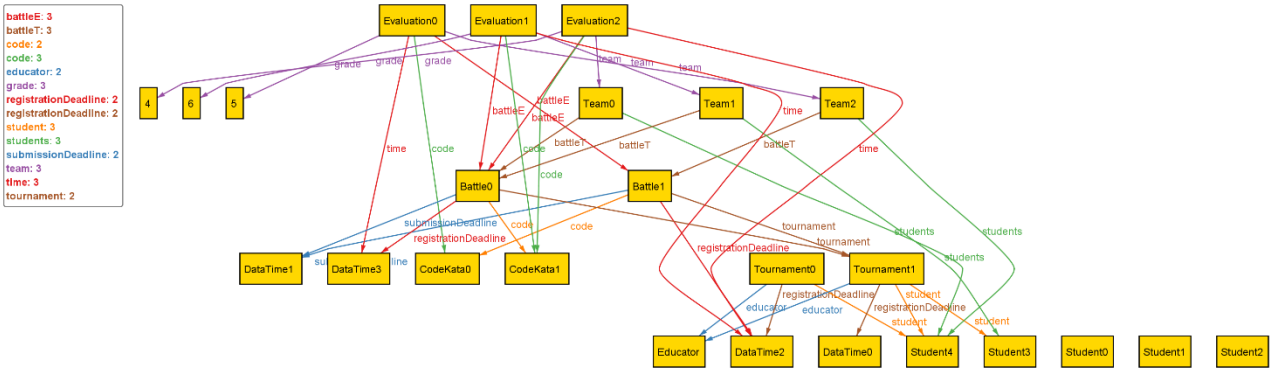



*Figure 11 Alloy Static Model*

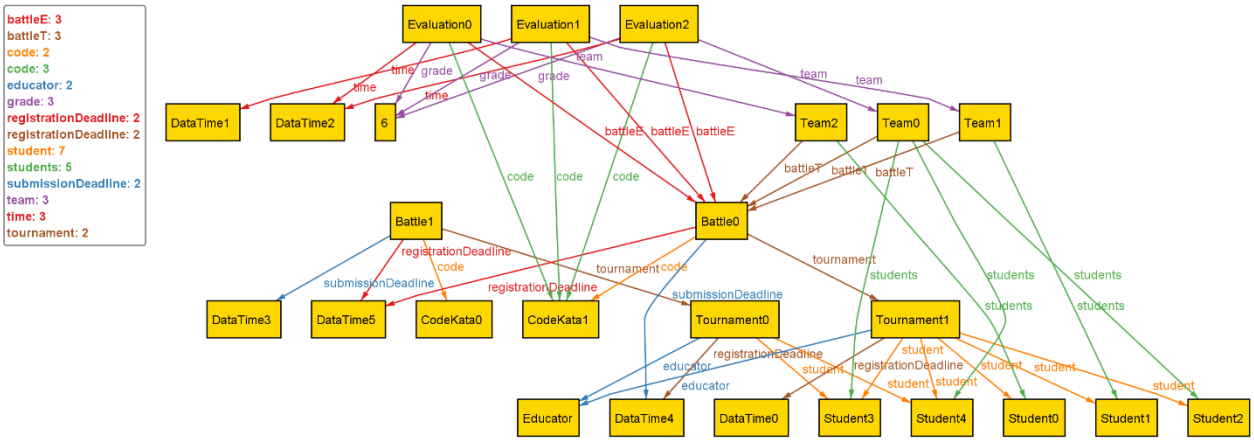



*Figure 12 Alloy Static Model*

# 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*