POLITECNICO DI MILANO

SOFTWARE ENGINEERING II

CodeKataBattle

Requirements Analysis and Specification Document

Version 1.0

Daniel Maurio Ruiz Suarez
Sebastian Enrique Perea Lopez

December 22, 2023

Contents

1. INTRODUCTION

1.1. Purpose

The purpose of the Requirement Analysis and Specification Document (RASD) for the CodeKataBattle (CKB) project is to accurately explain the functional and nonfunctional requirements that are essential for the development of the CKB platform. Additionally, the RASD acts as a contractual foundation for users, providing a detailed overview of the CKB platform's features and constraints.
Unlike the code specific documentation, this document implements user-friendly language, making it more accessible to a general public or those not versed in the subject and the tools that were used to develop the platform.
This RASD emphasizes the importance of precise and clear specifications, offering a roadmap for the successful development and deployment of the CKB platform. It not only presents the core functionalities of the system but also establishes the quality attributes and performance metrics that the platform should follow.

1.1.1. Goal

The following are the goals that we want to implement with our platform:
[G1] Students improve their software development skills with practical coding challenges (battles) and exercises.
[G2] Facilitate the collaboration between students by allowing them to form teams to face the different battles and challenges that compose a tournament.
[G3] Simulate real-world software development practices, including version control, automated testing, and continuous integration.
[G4] Encourage a test-first approach to programming by providing challenges with predefined test cases.
[G5] Create a competitive environment where students can compete in the battles and see their rankings on the leaderboards of the battles and tournaments.
[G6] Provide educators with the tools to create, manage, and evaluate coding challenges, as well as check students progress and programming logic.

1.1.2. Alignment with Educational Principles

CKB is in line with the established educational principles for effective software development teaching. By incorporating practical coding, collaborative learning, and practical challenges, the platform supports a modern approach to software development education.

1.1.3. Evolutionary Nature

As technology and educational practices evolve, the CKB platform is designed to allow continuous adaptation and growth. Regular updates and feedback mechanisms will ensure that the software remains as a dynamic and effective tool for educators and students alike.

1.2. Scope

The CodeKataBattle (CKB) project aims to revolutionize the concept of software development education by addressing the challenges faced by students with the improvement of their coding skills and logic. In the current educational landscape, students often struggle to bridge the gap between theoretical knowledge and practical application. CKB's main purpose is to provide a dedicated platform for collaborative learning, impulsing a more hands-on-code learning aproach while motivating its students to learn and help each other.
The project also introduces a competitive element where students who participate in code kata battles, can be recognized based on their performance. This gamified approach provides motivation for students to continuously improve their coding abilities.
Synthesizing these educational goals and trends, CKB identifies shared phenomena within the educational domain, aiming to provide solutions to challenges commonly faced by students. The platform leverages collaborative learning trends, acknowledging the importance of students working together on coding projects.
The scope of the CKB project extends beyond mere coding exercises; its goal is to establish a dynamic and engaging environment that not only improves coding skills but also encourages collaboration, competition, and a comprehensive understanding of real-world software development practices. Through this multifaceted approach, CKB aspires to reshape the educational process, by making it more practical, competitive, and favorating to the development of well-rounded software developers.

1.2.1. World phenomena

Phenomena events that take place in the real world and that the machine cannot observe.
[W1] Students collaborate actively, forming teams for collective participation in code kata battles, enhancing their software development skills.
[W2] Educators create and organize battles, challenging students to showcase their programming prowess.
[W3] After the creation of a battle, the CKB platform generates a GitHub repository containing the battle, initiating the collaborative coding process.
[W4] Educators may manually evaluate projects, assigning personal scores to recognize and reward exceptional student efforts.

1.2.2. Shared phenomena

Phenomena controlled by the world and observed by the machine.

[SP1] When an educator initiates the creation of a new tournament, the CKB platform records this event. The system generates a new tournament with specified details and configuration settings.

[SP2] Upon the creation of a new tournament, the CKB platform initializes battles within that tournament based on the parameters set by the educator.

[SP3] When a student registers on the CKB platform, the system captures the registration event.

[SP4] As students form teams and join battles within a tournament, the CKB platform registers the participation event.

[SP5] After the submission deadline, if manual evaluation by educators is required, the CKB platform enters a manual evaluation period.

Phenomena controlled by the machine and observed by the World.

[SP6] When an educator concludes a tournament, indicating that no further battles or submissions will be accepted, the CKB platform logs the tournament closure event.

[SP7] The CKB platform sends notifications to users, including students and educators, regarding new tournaments, upcoming battles, rank changes, and other critical updates.

[SP8] While in a tournament, the CKB platform provides real-time updates on rank changes, deadlines, and significant tournament-related information.

[SP9] As the platform computes scores for each battle within a tournament, the CKB system updates the personal tournament scores for each student.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

- **CodeKataBattle (CKB):** The collaborative platform designed to improve the students software development skills through code katas and team-based programming challenges.
- **Code Kata (Battle):** A programming exercise aimed at improving skills through repetitive practice and incremental improvements.
- **Static Analysis:** The process of examining code without executing it, focusing on identifying potential issues and improving code quality.
- **Team Score:** The numerical representation of a group's performance in a code kata, considering various factors such as test case success, timeliness, and code quality.
- **Gamified:** The way to give a normal and common process a more game-like approach to make it more entertaining and engaging for the modern students.
- **Tournament:** An event consisting of student teams working to complete a code kata.
- **Commit:** a change made to the code in the repository by a student.
- **Scoring:** the process of assigning a score to a student and team based on their level of contributions to the code kata.

1.3.2. Acronyms

- **CKB:** CodeKataBattle
- **RASD:** Requirement Analysis and Specification Document.
- **API:** Application Programming Interface
- **IDE**: Integrated Development Environment

1.3.3. Abbreviations

- **[Gn]:** The n-th goal of the system
- **[Wn]:** The n-th world phenomena
- **[SPn]:** The n-th shared phenomena
- **[UCn]:** The n-th use case
- **[Rn]:** The n-th functional requirement

1.4. Revision history

- **Version 1.0 (22/12/2023):**
    - Initial release of the Requirements Analysis and Specification Document (RASD) for CodeKataBattle.

1.5. Reference Documents

This document is strictly based on:

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

1.6. Document Structure

Mainly the current document is divided in 4 chapters, which are:

1. **Introduction:** it aims to describe the environment and the demands taken into account for this project. In particular it's focused on the reasons and the goals that are going to be achieved with its development;
2. **Overall Description:** it's a high-level description of the system by focusing on the shared phenomena and the domain model (with its assumption);
3. **Specific Requirements:** it describes in very detail the requirements needed to reach the goals. In addition it contains more details useful for developers (i.e information about HW and SW interfaces);
4. **Formal Analysis:** this section contains a formal description of the main aspect of the World phenomena by using Alloy;
5. **Effort Spent:** it shows the time spent to realize this document, divided for each section;
6. **References:** it contains the references to any documents and to the Software used in this document.

2. OVERALL DESCRIPTION

2.1. Product perspective

2.1.1. Scenarios

1. **Student joins a battle**

   Emily, a computer science student, logs into the CKB platform and searches through the ongoing tournaments. After watching the different battles available, she chooses a tournament and joins one of its battles, forming a team with her classmates and friends before spending some exciting and joyful hours coding, learning and solving the challenge proposed for the battle.

2. **Educator creates a tournament**

   Professor Rodriguez, an educator passionate about coding practices, logs into the CKB platform. Inspired to challenge his students, he creates a new tournament with intriguing coding exercises. Setting deadlines and scoring parameters, he anticipates the students' creative solutions and the competitive spirit that will unfold.

3. **Real-time rank updates**

   During an ongoing battle, Carlos continuously monitors the updates on the ranking of the platform. As his team makes more progress, he observes the dynamic changes in the rankings, looking at how some students go up and down or even out of the ranks.

4. **Manual evaluation by educator**

   Dr. Patel, an educator with years of experience and familiarity with software quality and good practices, manually evaluates code submissions from his students. Using the CKB platform, he provides constructive feedback, assigns personal scores, and recognizes exceptional coding practices.

5. **Tournament closure notification**

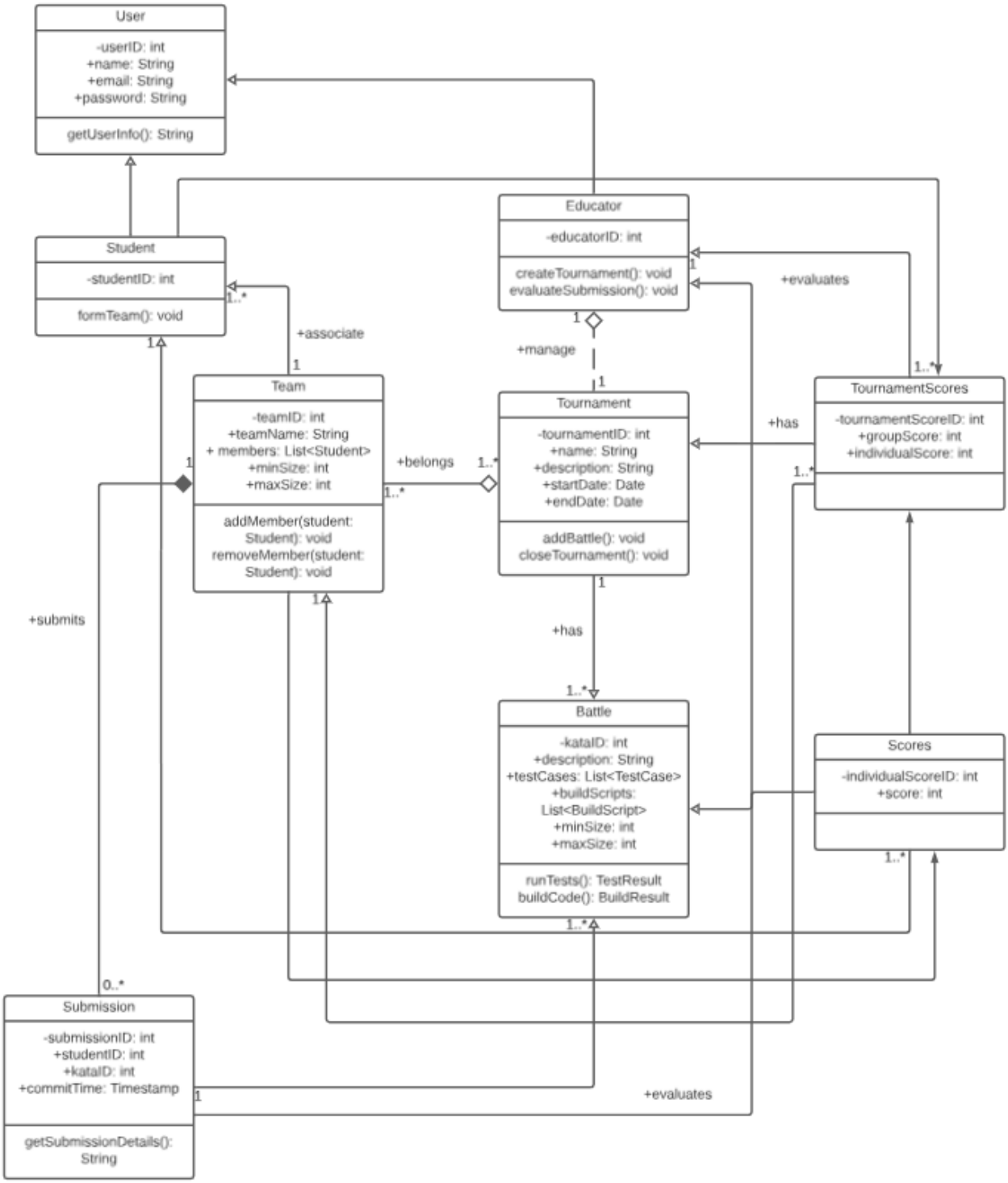
   As the submission deadlines start to approach, all students registered in a battle receive a notification that the battle is close to its end. This prompts a last-minute flurry of activity as participants finalize their submissions.

2.1.2. Domain class diagram

The domain diagram provides a high-level overview of the essential entities and relationships that exist within the CodeKataBattle system. It captures the fundamental components that interact and collaborate to facilitate the coding battles, users management, and score tracking. The diagram serves as a visual representation of the core elements in the system's domain.

- The base class represents all users of the system, including both Students and Educators. It holds common attributes and behaviors.
- Derived from the User class, these entities represent distinct user roles in the CKB system.
    - Students participate in coding battles
    - Educators manage tournaments and evaluate submissions.
- Teams consist of one or multiple students collaborating on coding challenges. Teams are associated with the battles.
- Represents a coding tournament, organized by Educators. It contains multiple battles and associated teams. Tournaments are a crucial aspect of the competitive coding environment.
- Coding battles are individual or team challenges within a tournament. They contain submissions from student teams, undergo automated evaluation, and receive manual evaluations from educators if necessary.

Associations illustrate how entities like Teams, Submissions, and Scores are related to one another and contribute to the collaborative and competitive aspects of coding battles.
This domain diagram provides a foundational understanding of the primary entities and their interactions, laying the groundwork for more detailed discussions about the system's functionality and behavior.

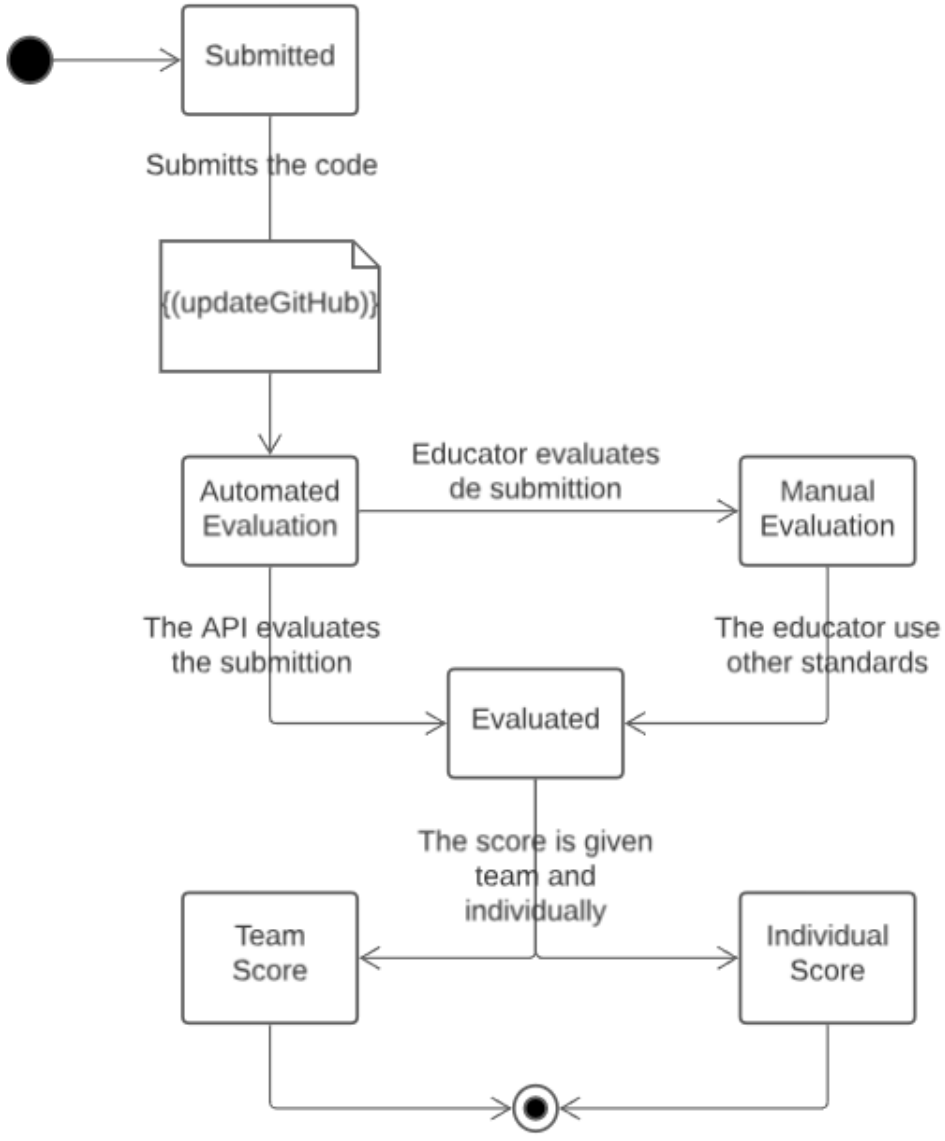


(Figure 1. Domain Class Diagram for CodeKataBattle (CKB))

2.1.3. Statecharts

Statecharts serve as visual guides, representing the specific details of CodeKataBattle. In CKB's dynamic environment, these diagrams illustrate the sequential flow of processes, enhancing clarity for both educators and students. From submission evaluation to tournament lifecycles, statecharts offer a concise roadmap for navigating and comprehending the complexities of CKB.

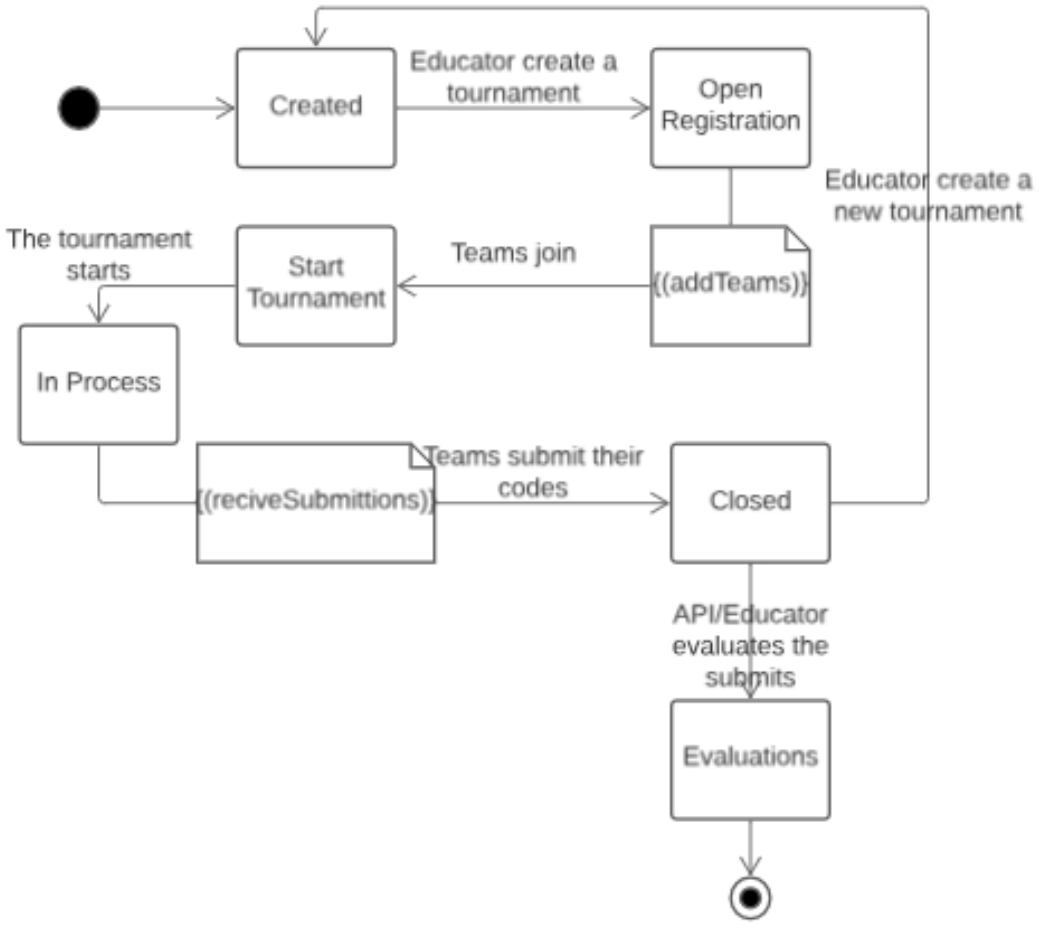
Submission evaluation process



(Figure 2. Submission evaluation process)

The statechart discussed in this section illustrates the submission evaluation process within CodeKataBattle. Starting from the moment a student submits their code, it delineates the journey through automated evaluation, manual evaluation by educators, and concludes with the final evaluated state. This statechart is essential for comprehending the workflow, decision points, and interactions involved in assessing student submissions, providing a valuable tool for system analysis and optimization.

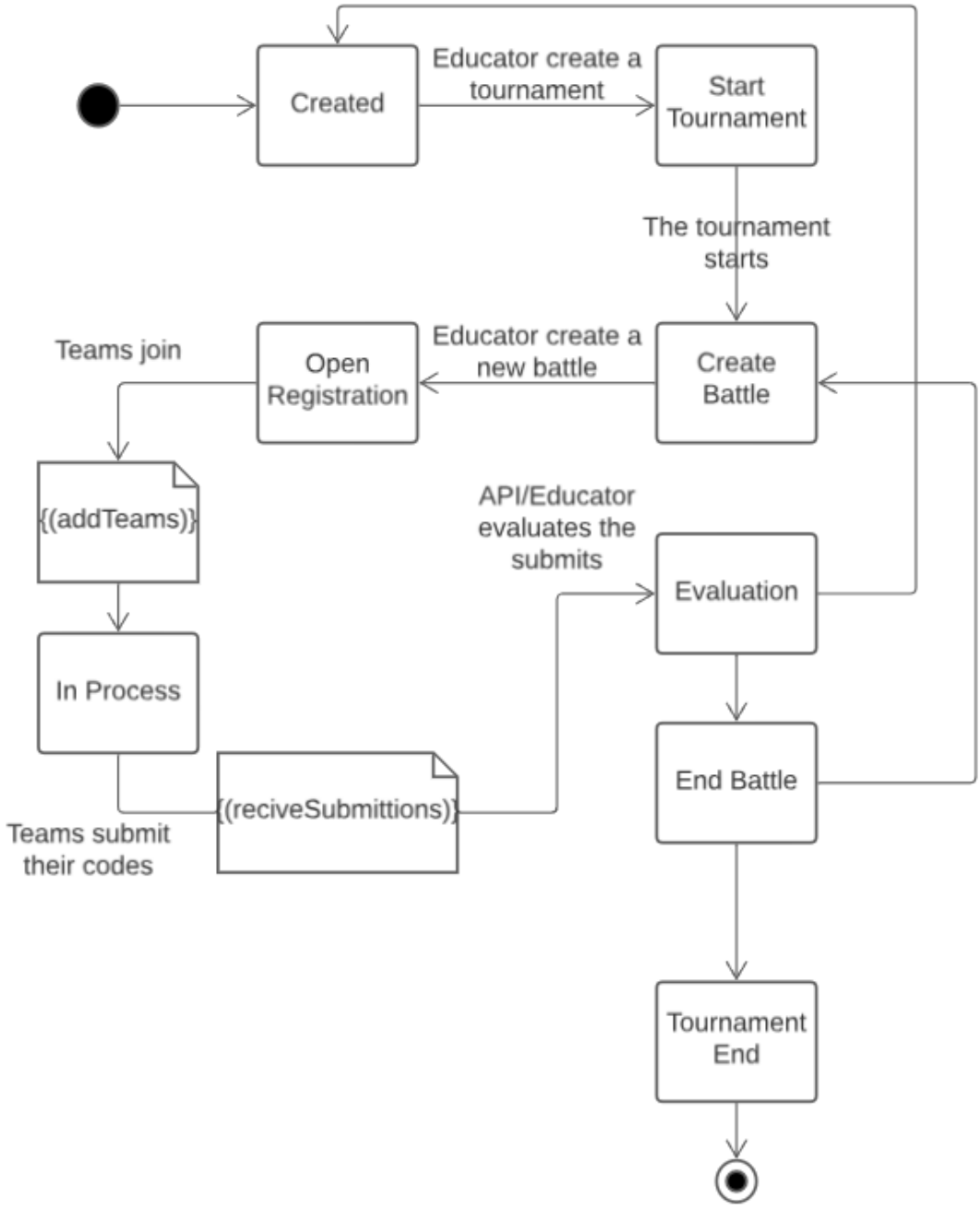
Tournament lifecycle



(Figure 3. Tournament lifecycle)

The statechart presented here provides an in-depth exploration of the tournament lifecycle within CodeKataBattle. Beginning with the creation of a tournament by educators, it navigates through phases such as open registration, battles in progress, and eventual closure. Understanding this lifecycle is crucial for educators to manage tournaments effectively, ensuring a seamless and organized experience for both students and educators participating in CodeKataBattle.

Student participation in tournament



(Figure 4. Student participation in tournament)

In this section, the statechart explores the student's journey, from registration to active participation in a battle within CodeKataBattle (CKB). It covers states such as registered, team formation, coding in progress, and awaiting evaluation. This statechart serves as a visual guide, shedding light on the steps involved in student engagement, team collaboration, and the transition phases during a coding battle, aiding both students and educators in navigating the CodeKataBattle platform effectively.

2.2. Product functions

2.2.1. User Registration and Profile Management:

The CKB platform offers a seamless registration process for both students and educators. Upon registration, users can create and manage their profiles, providing personal details and customizing preferences to enhance their platform experience.

2.2.2. Educator Functions:

Educators wield powerful tools within the CKB platform, allowing them to craft engaging code kata battles. They can meticulously define battle details, including project specifications, test cases, and build automation scripts. Educators also orchestrate tournaments, setting deadlines and configuring scoring parameters. The platform empowers educators to share their responsibilities, granting colleagues permissions for battle and tournament creation.

2.2.3. Student Functions:

Students navigate the CKB landscape with the ability to explore and subscribe to tournaments crafted by educators. Whether participating individually or forming teams within specified limits, students engage in the challenge. The platform facilitates team formation, providing features for invitations and acceptances among peers.

2.2.4. GitHub Integration:

The CKB platform seamlessly integrates with GitHub, automatically creating dedicated repositories for each battle to host code submissions. This integration extends to GitHub Actions, triggering automated workflows upon code pushes, ensuring a streamlined and efficient coding environment.

2.2.5. Automated Scoring:

The platform implements automated scoring, evaluating functional aspects such as the number of passed test cases. Timeliness is factored in by considering the time elapsed from registration to the last commit. Static analysis tools assess the quality of source code, contributing to a comprehensive and automated scoring mechanism.

2.2.6. Manual Evaluation:

Educators hold the authority to manually evaluate and assign personal scores to teams after the completion of a battle. The consolidation stage allows educators to review and score source code submitted by teams, ensuring a holistic evaluation process.

2.2.7. Real-time Rank Updates:

Providing a dynamic experience, the platform continuously updates and displays current battle ranks based on both automated and manual evaluations. This real-time feedback keeps teams and educators informed of their performance throughout the battle.

2.2.8. Tournament Score Calculation:

Individual battle scores are aggregated to calculate a student's overall tournament score. The platform generates a tournament rank, offering a comprehensive view of students' performances within the context of the tournament.

2.2.9. Tournament Closure and Notifications:

Educators have the capability to close tournaments, initiating the finalization of scores and ranks. The platform ensures timely notifications to all students involved in a tournament, keeping them informed about the availability of final tournament ranks.

2.2.10. User Notifications:

The platform employs a robust notification system, keeping users informed about upcoming battles, tournament updates, and finalized scores. This ensures that users stay connected and engaged with the evolving dynamics of the CodeKataBattle environment.

2.2.11. Documentation and Help Center:

Comprehensive documentation serves as a guide for users, elucidating platform features and functionalities. A dedicated help center caters to user inquiries and challenges, providing assistance to enhance the user experience on the CKB platform.

2.3. User characteristics

Understanding the diverse user characteristics within the CodeKataBattle (CKB) platform is crucial for tailoring the learning environment to the unique needs of educators, students, GitHub users, gamification enthusiasts, notification-responsive users, and documentation seekers.

2.3.1. Users

Users, both educators, and students are required to have a basic understanding of version control using GitHub. This includes familiarity with forking repositories, committing code changes, and understanding GitHub Actions for workflow automation. Collaborative development practices on GitHub are essential for effective participation in the platform, where code hosting and automated evaluations rely on the GitHub ecosystem.

- **Educators**

  Educators play a pivotal role in guiding students through coding exercises and assessing their performance. They are expected to possess a strong understanding of software development concepts, programming languages, and coding best practices. Additionally, educators should demonstrate administrative abilities, managing and organizing coding challenges, tournaments, and scoring configurations within the CKB platform. Their mentoring skills contribute to creating a supportive learning environment.

- **Students**

  Students on the CKB platform exhibit diverse skill levels, ranging from beginners to advanced programmers. The platform caters to this variety, encouraging collaboration among students, who form teams to participate in code kata battles. Successful students are those with a keen interest in enhancing their software development skills, embracing a test-first approach, and enjoying the challenges presented for continuous learning.

2.4. Assumptions, dependencies and constraints

2.4.1. Regulatory policies

The CodeKataBattle (CKB) platform adheres to regulatory policies encompassing data protection, privacy, and educational standards. Compliance with legal and ethical norms guides the secure handling of user information, ensuring fairness, and equal opportunities. The platform's development aligns with prevailing laws, necessitating transparent communication on data usage, intellectual property rights, and code ownership. Adapting to regulatory changes is crucial for maintaining ethical practices within the educational landscape.

2.4.2. Domain assumptions

The following domain assumptions serve as foundational principles for the CodeKataBattle (CKB) platform, guiding its development and usage within the educational context. These assumptions encompass users' educational affiliations, technical proficiencies, and collaborative attitudes. Understanding these inherent conditions is essential for effectively leveraging the platform's capabilities, fostering a collaborative and educational environment for both students and educators.

[D1]: Users in an educational setting.
[D2]: Stable internet access for users.
[D3]: Basic GitHub proficiency.
[D4]: Educators' competence in creating coding exercises.
[D5]: Students' willingness to collaborate.
[D6]: Dependence on GitHub API.
[D7]: User reliance on stable internet service.
[D8]: Integration of code quality analysis tools.
[D9]: Educator-configured scoring parameters.

3. SPECIFIC REQUIREMENTS

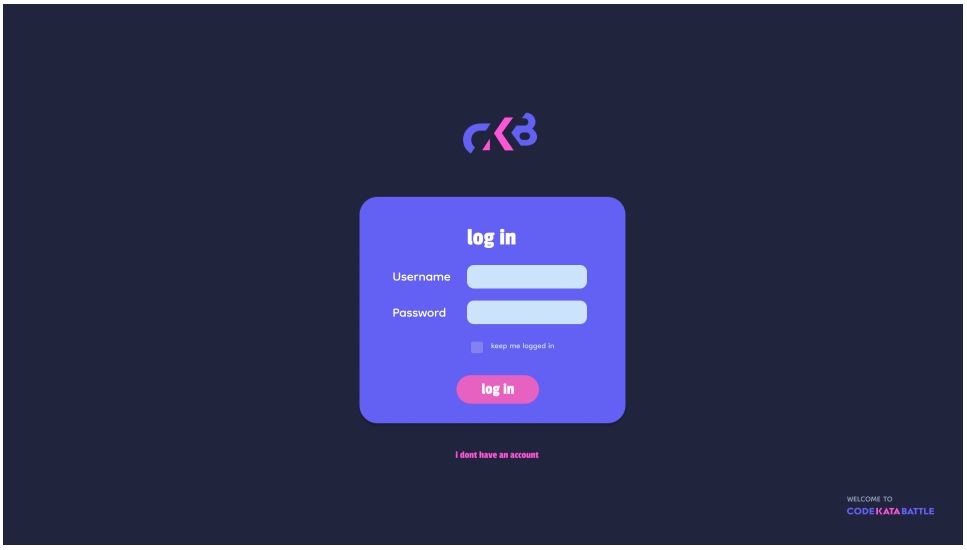
3.1. External Interface Requirements

3.1.1. User Interfaces

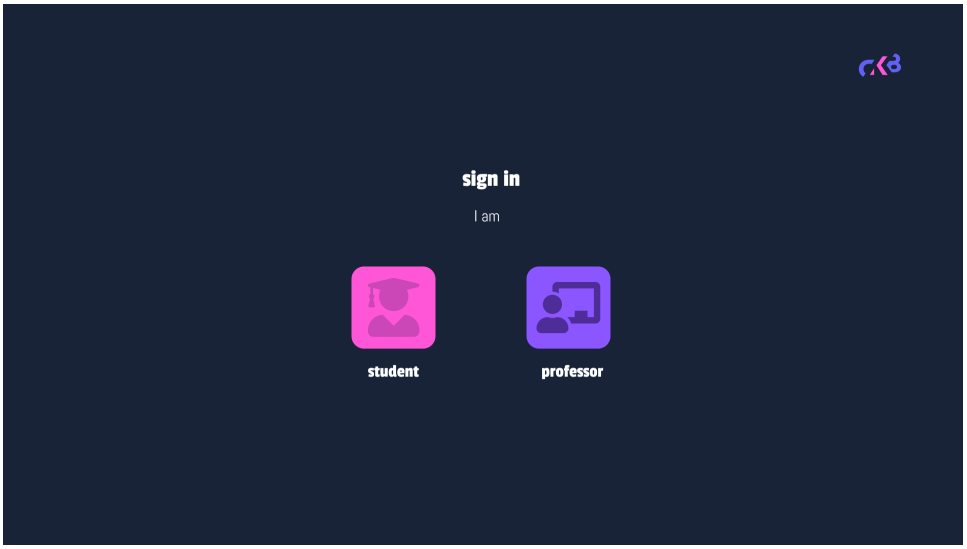
The platform requires user interfaces for various functionalities, accessible to both students and educators.
Features:

- Student Interface:
  - Dashboard for viewing ongoing tournaments and battles.
  - Battle registration and team formation.
  - Submission interface for code implementation and GitHub integration.
  - Personal profile displaying tournament scores and badges earned.
- Educator Interface:
  - Creation of tournaments and battles with specific configurations.
  - Manual scoring interface and evaluation tools.
  - Access to consolidated battle scores and rankings.

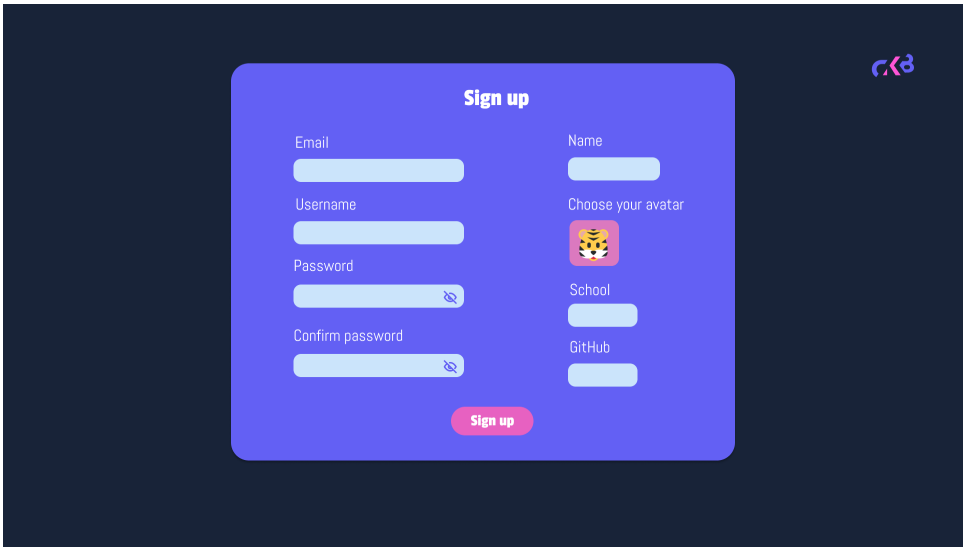
Some wireframes of this interfaces can be viewed below



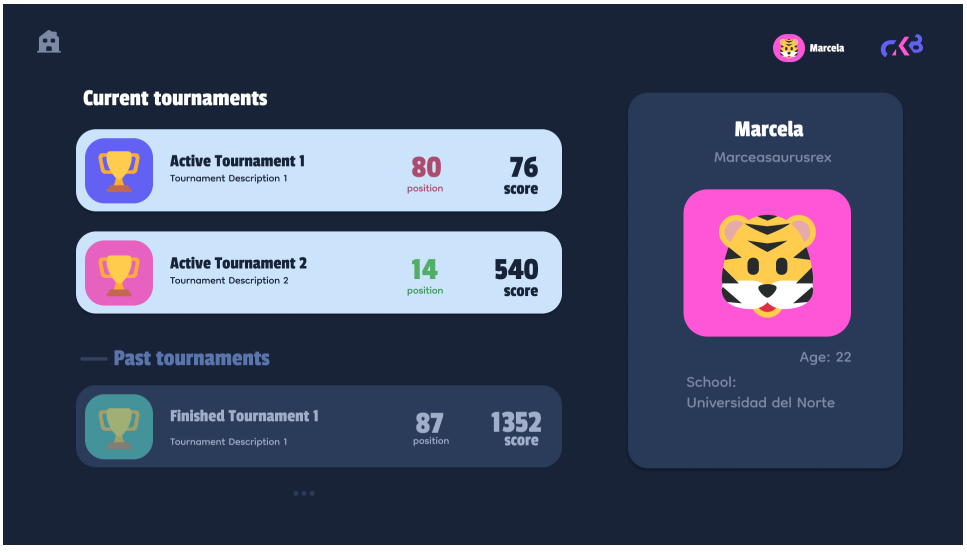
(Figure 5. Login Screen)



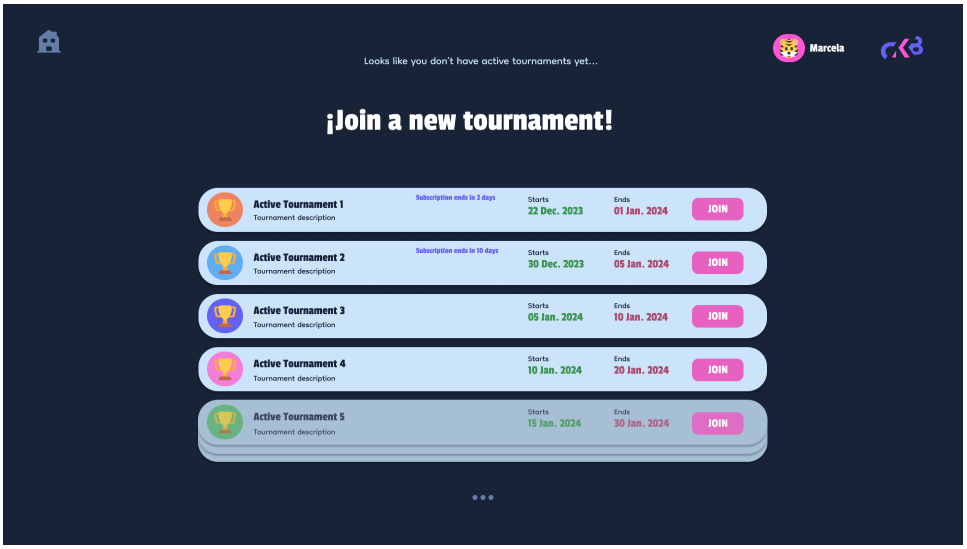
(Figure 6. User type selection)



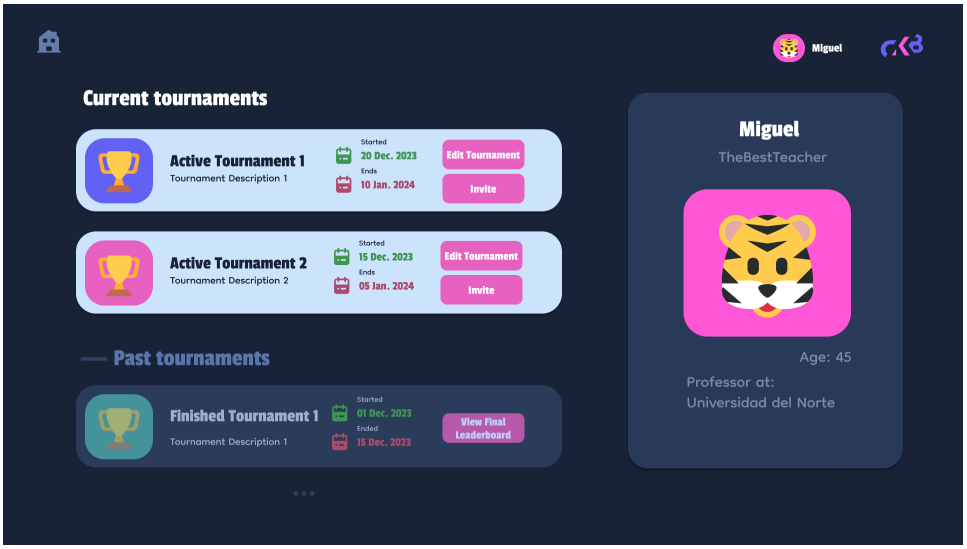
(Figure 7. Sign Up)



(Figure 8. Student Profile)



(Figure 9. Joining tournament)



(Figure 10. Educator Profile)

(Figure 11. Student Dashboard)

3.1.2. Hardware Interfaces

The CodeKataBattle (CKB) platform operates as a web application, thereby necessitating access via devices equipped with standard hardware configurations capable of browsing the internet. The following hardware interfaces are required for the user roles defined in section 2.3:

- For Students:
  - Students engaging with the CodeKataBattle platform will require access to devices capable of internet browsing to participate effectively. Primarily, they will utilize standard desktops or laptops to access the CKB platform via web browsers. While not mandatory, the availability of smartphones with internet connectivity can offer added flexibility for students, enabling access to the platform on-the-go.

- For Educators:
  - Educators, responsible for managing tournaments, battles, and scoring on the CKB platform, will need access to desktops or laptops equipped with web browsers for comprehensive platform management. Additionally, while optional, educators might benefit from using smartphone applications tailored for essential functionalities, offering flexibility and convenience in overseeing the platform's management tasks.

- Automated Processes:
  - The CodeKataBattle platform's automated processes rely on robust server infrastructure. This infrastructure comprises reliable servers capable of hosting the web application, ensuring seamless access, scalability, and efficient handling of concurrent connections and data processing.

- Additional Insights:
  - The platform's design prioritizes accessibility, with no specialized or unique hardware interfaces mandated for either students or educators.

3.1.3. Software Interfaces

For the platform, several software interfaces are integral to its functionality, facilitating various features and interactions within the system:

- **GitHub API:** Integration with GitHub API to manage repositories, automate evaluation triggers, and analyze student commits for continuous integration during battles.
- **Static Analysis Tools:** Utilization of third-party static analysis tools like SonarQube, to assess code quality, including security, reliability, and maintainability aspects in student-submitted code.
- **Webhooks and Custom APIs for Notifications:** Use of custom APIs and webhooks to send real-time notifications to students and educators regarding battle scores, rank updates, deadlines, and crucial tournament information.

- **Web Application Frameworks and Libraries:** Application of various web frameworks and libraries React, Django to create a user-friendly interface for smooth interactions among students and educators accessing the platform.

3.1.4. Communication Interfaces

- GitHub Webhooks: Integration with GitHub's webhook system forms a vital communication link, allowing real-time updates and notifications between the CKB platform and students' repositories on GitHub. These webhooks trigger automated evaluations, enabling instant feedback on student commits and facilitating continuous integration.

- RESTful APIs: The platform employs RESTful APIs, acting as communication channels between the frontend and backend components. These APIs power various functionalities, including user authentication, seamless data retrieval, handling of submissions, and dynamic score updates. They enable efficient data transfer and interaction between different parts of the CKB system.

- Email Notifications: Complementing other communication channels, the platform utilizes email services to deliver essential notifications and updates to users. Email communication serves as an additional means to inform students and educators about battle registrations, impending deadlines, rank changes, and significant announcements, ensuring comprehensive user engagement.

3.2. Functional Requirements

The subsequent list outlines the essential requirements crucial for the system to function effectively. To operate seamlessly, the system must:

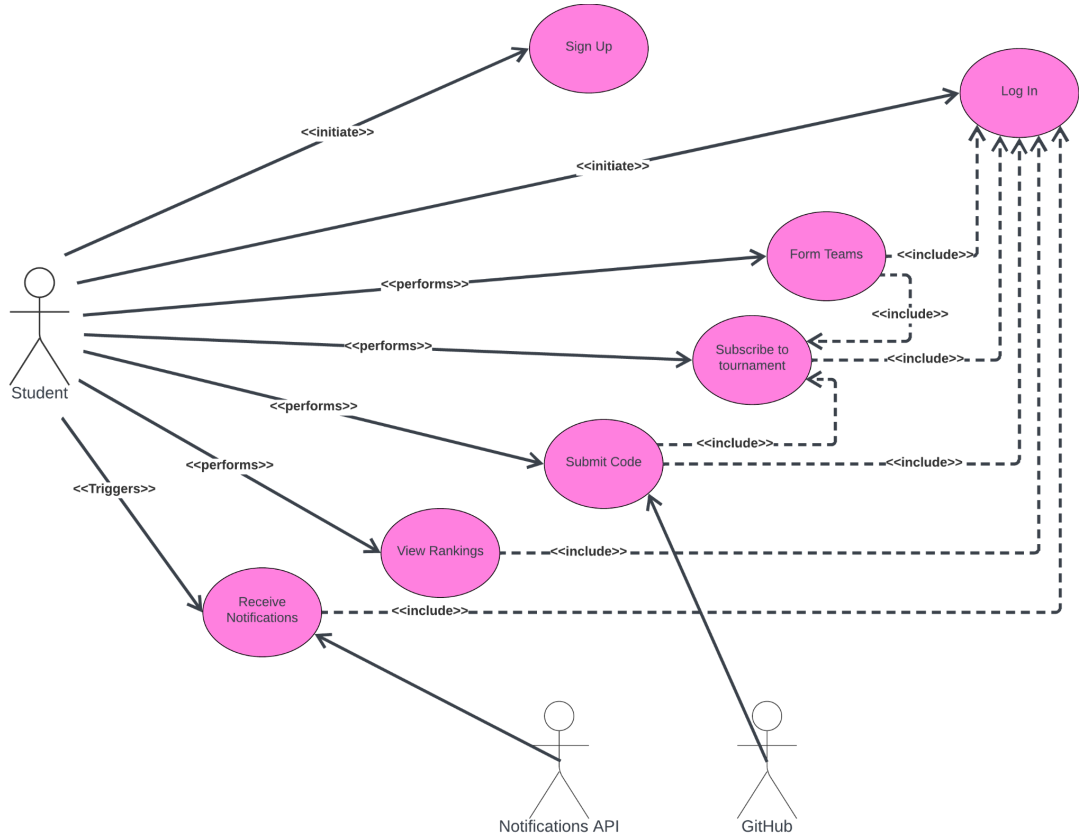
- [R1] The system allows students to sign up and log in.
- [R2] The system allows educators to sign up and log in.
- [R3] Educators can create new tournaments, defining code battles with descriptions, deadlines, and scoring configurations.
- [R4] Educators can set minimum and maximum team sizes for battles within a tournament.
- [R5] Educators can manage and view the list of battles they've created within a tournament.
- [R6] Educators can grant permission to colleagues to create battles within a specific tournament.
- [R7] Educators can upload the code kata with the respective description and software project, including test cases and build automation scripts.
- [R9] Educators can set a registration deadline.
- [R10] Educators can set a final submission deadline.
- [R11] Educators can set additional configurations for scoring.
- [R12] Students can form teams for battles, adhering to set team size limits.
- [R13] The platform automatically generates GitHub repositories for each battle upon creation.

- [R14] The platform performs automated evaluation of submitted code based on test cases, calculating the functionality score.
- [R15] Integration with static analysis tools for quality evaluation, considering security, reliability, and maintainability of code.
- [R16] Educators can manually assess and assign personal scores to student submissions.
- [R17] After the submission deadline, educators review and provide manual evaluations based on code quality and adherence to the test-first approach.
- [R18] The system continuously updates battle scores during the battle period, displaying evolving rankings to students and educators.
- [R19] Upon consolidation after the submission deadline, the final battle rank becomes available to all participants.
- [R20] The platform updates personal tournament scores for each student, presenting the cumulative performance across battles within a tournament.
- [R21] Automated notifications inform students about new tournaments and upcoming battles within subscribed tournaments.
- [R22] Real-time updates notify users (students and educators) about rank changes, deadlines, and significant tournament-related information.
- [R23] Email notifications alert users about critical updates, including new tournaments, battle results, and personal scores.
- [R24] Implement role-based permissions, allowing educators administrative access to manage tournaments, battles, and evaluations, while students have limited access as participants.
- [R25] Enable users (students and educators) to view their profiles, displaying personal information, ongoing tournaments, past performances, and badges earned.

3.2.1. Use Case Diagrams:

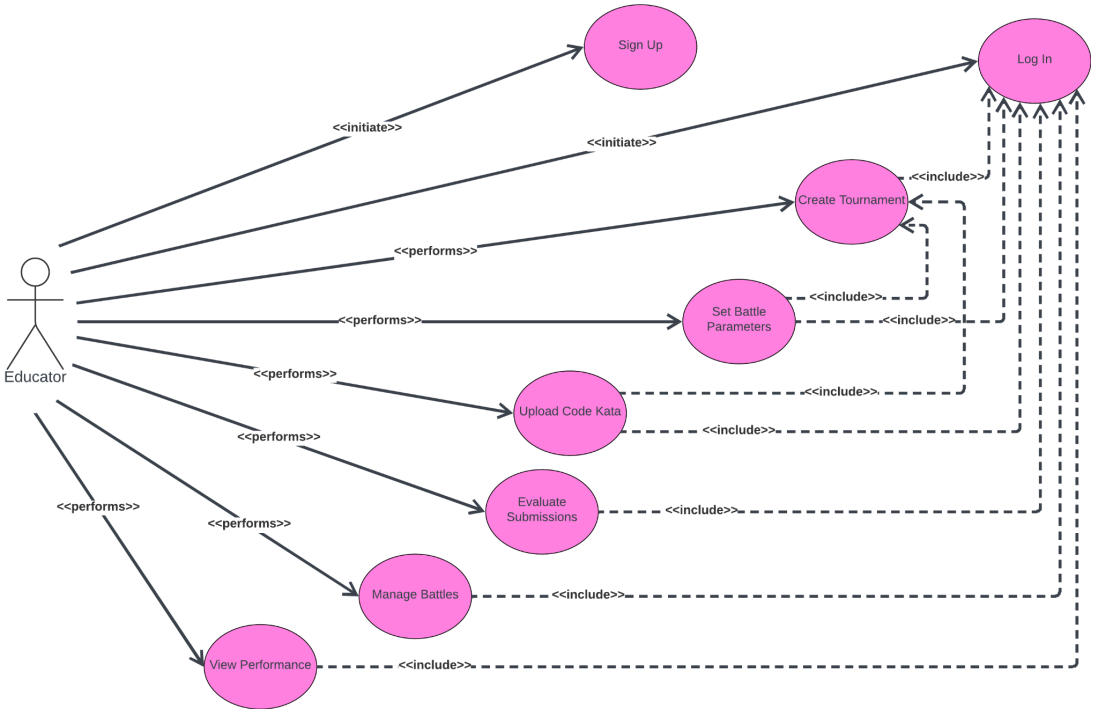
Following, we present use case diagrams to aid in recognizing the actors engaging with the system and their respective roles in each use case.

Student Use Case Diagram



(Figure 12. Student use case Diagram)

Educator Use Case Diagram



(Figure 13. Educator use case Diagram)

3.2.2. Use Cases:

[UC1] - Educator Login

Name	Educator Login
Actors	Educator
Entry Condition	Educator opens the web app and its not logged in
Event flow	1. Educator enters valid login credentials (username/email and password). 2. The system verifies the credentials and authenticates the educator. 3. Upon successful validation, the system grants access to educator-specific functionalities.
Exit condition	Educator gains access to the CKB platform and its associated features.
Exception	2.a: The data inserted is not valid. The system returns to the entry condition.

[UC2] - Student Login

Name	Student Login
Actors	Student
Entry Condition	Student opens the web app and its not logged in
Event flow	1. Student provides valid login credentials (username/email and password). 2. The system verifies the provided details and authenticates the student's identity. 3. Upon successful verification, the system allows access to student-centric functionalities and features.
Exit condition	Student gains access to the CKB platform and its associated functionalities.
Exception	2.a: The data inserted is not valid. The system returns to the entry condition.

[UC3] - Educator Registration

Name	Educator Registration
Actors	Educator
Entry Condition	Educator is not registered in the CKB platform
Event flow	1. Educator navigates to the registration page on the CKB platform. 2. Educator fills in required personal details: name, email, password, institution, and contact information. 3. The system validates the entered information, ensuring the uniqueness of the email and password strength. 4. If validation is successful, the system registers the educator, generating a unique educator ID. 5. An email confirmation is sent to the registered email address for account verification.
Exit condition	Educator completes the registration process and receives a confirmation email for account verification.

[UC4] - Student Registration

Name	Student Registration
Actors	Student
Entry Condition	Student is not registered in the CKB platform
Event flow	1. Student accesses the registration portal on the CKB platform. 2. Student provides personal details: name, email, password, school/college affiliation, and contact details. 3. The system validates the entered data, ensuring the uniqueness of the email and the strength of the password. 4. Upon successful validation, the system registers the student, assigning a unique student ID. 5. An email verification request is sent to the provided email address for account confirmation.
Exit condition	Student completes the registration process and receives an email for account verification.

[UC5] - Educator Creates a Tournament

Name	Educator creates a tournament
Actors	Educator
Entry Condition	The educator is logged into the CKB platform
Event flow	1. The system presents the option to create a new tournament to the educator. 2. Educator provides tournament details: a. Tournament name, description, and duration. 3. The system confirms the successful creation of the tournament.
Exit condition	The educator receives a confirmation message of the newly created tournament.

[UC6] - Educator Uploads Code Kata for Battle

Name	Educator uploads code kata for a battle
Actors	Educator
Entry Condition	The educator is logged into the CKB platform
Event flow	1. Educator selects the option to upload a new code kata for a battle. 2. The system prompts the educator to provide a description and software project details: a. Description of the programming exercise. b. Software project with test cases and build automation scripts. 3. Educator sets deadlines (registration and submission), team sizes, and additional scoring configurations. 4. The system validates and confirms the successful upload of the code kata.
Exit condition	The educator receives a confirmation message indicating successful code kata upload.

[UC7] - Student Submits Code Solution

Name	Student Submits Code Solution
Actors	Student
Entry Condition	The student is logged into the CKB platform and is part of a battle
Event flow	1. Student develops a solution. 2. Upon completion, student pushes the code to the designated GitHub repository. 3. The system triggers an automated evaluation of the submission.
Exit condition	The student receives a confirmation of code submission and awaits evaluation results.

[UC8] - Educator Evaluates Student Submissions

Name	Educator Evaluates Student Submissions
Actors	Educator
Entry Condition	The educator is logged into the CKB platform and has ongoing battle submissions
Event flow	1. Educator accesses the list of ongoing battles and pending student submissions. 2. Educator selects a specific battle and reviews the submitted code of student teams. 3. The educator manually evaluates code quality, adherence to test-first approach, and provides a personal score. 4. The system updates the battle score based on both automated and manual evaluations.
Exit condition	The educator completes the evaluation and submits scores for student submissions.
Alternative	1.a: Educator does not need to make a manual evaluation.

[UC9] - Student Views Battle Rankings

Name	Student Views Battle Rankings
Actors	Student
Entry Condition	The student is logged into the CKB platform and participating in a battle
Event flow	1. Student accesses the ongoing battles section. 2. The system displays the current rankings and scores of all participating teams. 3. Student reviews their team's performance and position in the rankings.
Exit condition	The student completes viewing the battle rankings.

[UC10] - Educator Manages Tournament

Name	Educator Manages Tournament
Actors	Educator
Entry Condition	The educator is logged into the CKB platform and has created a tournament
Event flow	1. Educator accesses the tournament management section. 2. The system displays a list of created tournaments. 3. Educator selects a tournament to manage and views its details (participants, battles, scores). 4. Educator modifies tournament settings, extends deadlines, or adds additional battles. 5. The system updates and reflects the changes made to the tournament.
Exit condition	The educator completes managing the selected tournament.
Alternative	4.a: Educator does not modify the tournament.

[UC11] - Student Joins a Battle

Name	Student Joins a Battle
Actors	Student
Entry Condition	The student is logged into the CKB platform and there's an open battle
Event flow	1. Student explores available battles. 2. The system displays ongoing battles. 3. Student selects a desired battle to participate in. 4. If required, student forms or joins a team according to specified team sizes. 5. The system confirms the student's participation in the selected battle.
Exit condition	The student successfully joins the battle and awaits further instructions.

[UC12] - Educator Grants Permissions

Name	Educator grants permissions to create battles
Actors	Educator
Entry Condition	The educator is logged into the CKB platform and has an active tournament
Event flow	1. Educator opens an active tournament. 2. Educator accesses the permissions settings section. 3. The system asks for the email of the colleagues to be granted permissions. 4. Educator sends an invitation to the colleagues.
Exit condition	The educator completes granting permissions to colleagues.

[UC13] - Student Receives Battle Notification

Name	Student Receives Battle Notification
Actors	Student
Entry Condition	The student is subscribed to the CKB platform and to a tournament
Event flow	1. The system detects the creation of a new battle. 2. The system sends a notification to subscribed students about the newly created battle. 3. Student receives the notification via email.
Exit condition	Student acknowledges the battle notification.

[UC14] - Student Receives Tournament Notification

Name	Student Receives Tournament Notification
Actors	Student
Entry Condition	The student is subscribed to the CKB platform
Event flow	1. The system detects the creation of a new tournament. 2. The system sends a notification to all subscribed students about the newly created tournament. 3. Student receives the notification via email.
Exit condition	Student acknowledges the tournament notification.

[UC15] - Educator Closes Tournament

Name	Educator Closes Tournament
Actors	Educator
Entry Condition	The educator is logged into the CKB platform and has ongoing tournaments
Event flow	1. Educator accesses the tournament management section. 2. Educator selects a tournament that has concluded. 3. Educator initiates the closing process for the selected tournament. 4. The system finalizes the tournament and ends all related activities.
Exit condition	The tournament is officially closed.

[UC16] - Educator Defines Scoring Criteria

Name	Educator defines scoring criteria for a battle
Actors	Educator
Entry Condition	The educator is logged into the CKB platform
Event flow	1. Educator accesses the scoring criteria section for a specific battle. 2. Educator configures criteria such as functional aspects, timeliness, and quality levels. 3. Educator saves or updates the scoring configurations for the selected battle.
Exit condition	The scoring criteria for the battle are defined or modified.
Alternative	2.a: Educator does not modify the configurations.

[UC17] - Student Views Personal Tournament Score

Name	Student Views Personal Tournament Score
Actors	Student
Entry Condition	The student is logged into the CKB platform and has a finalized tournament.
Event flow	1. Student navigates to their profile 2. The system displays the student's cumulative score for ongoing or past tournaments.
Exit condition	Student completes viewing their personal tournament score.

[UC18] - Educator Reviews Tournament Rankings

Name	Educator Reviews Tournament Rankings
Actors	Educator
Entry Condition	The educator is logged into the CKB platform and has a finalized tournament.
Event flow	1. Educator accesses the tournament rankings section. 2. The system displays the overall rankings and scores of all participants in a tournament.
Exit condition	Educator completes reviewing tournament rankings.

[UC19] - Student Cancels Battle Participation

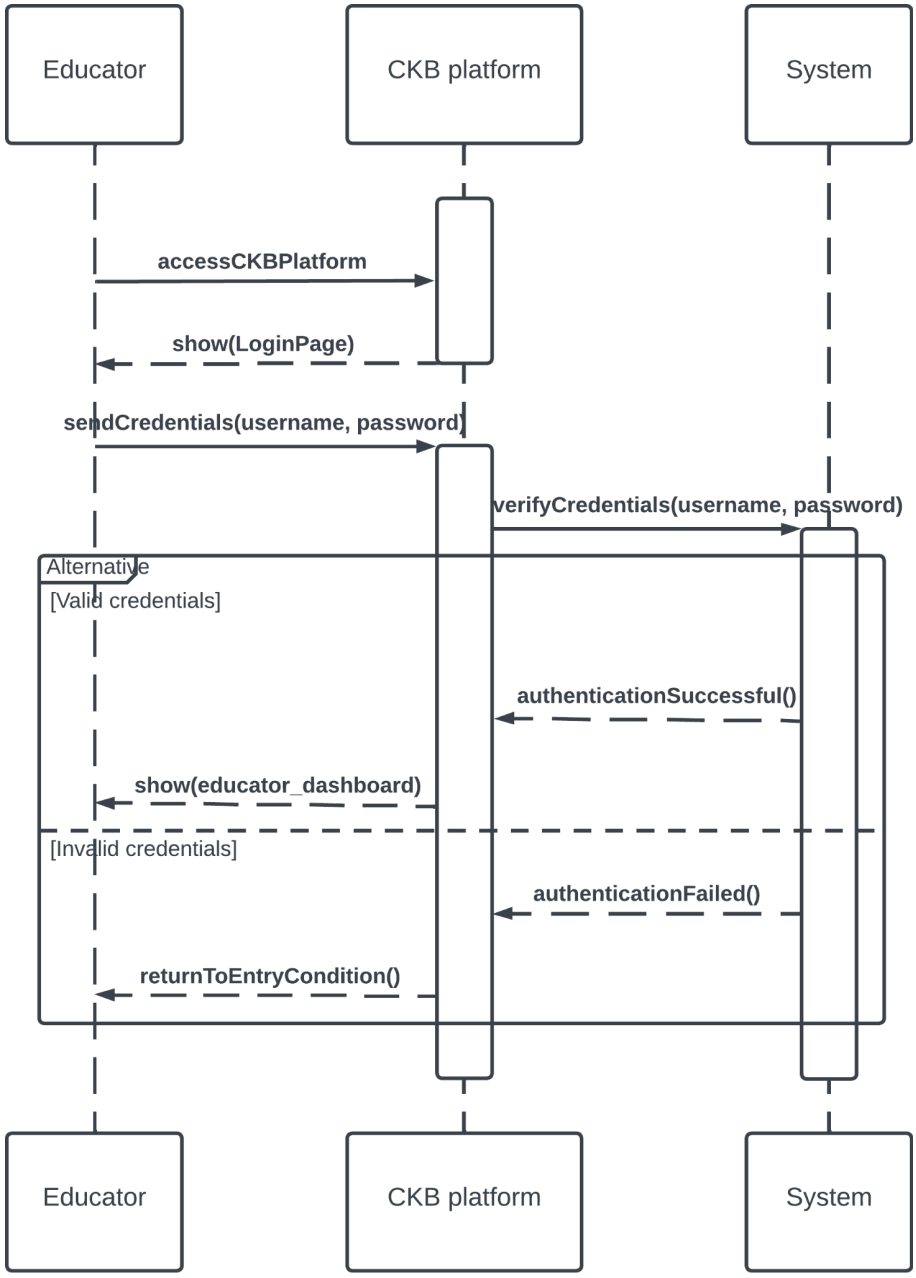
Name	Student Cancels Battle Participation
Actors	Student
Entry Condition	The student is logged into the CKB platform and has joined a battle
Event flow	1. Student accesses the joined battles section. 2. Student selects the desired battle to withdraw participation. 3. The system confirms the cancellation of the student's participation in the selected battle.
Exit condition	Student successfully cancels battle participation.

[UC20] - Educator Extends Battle Submission Deadline

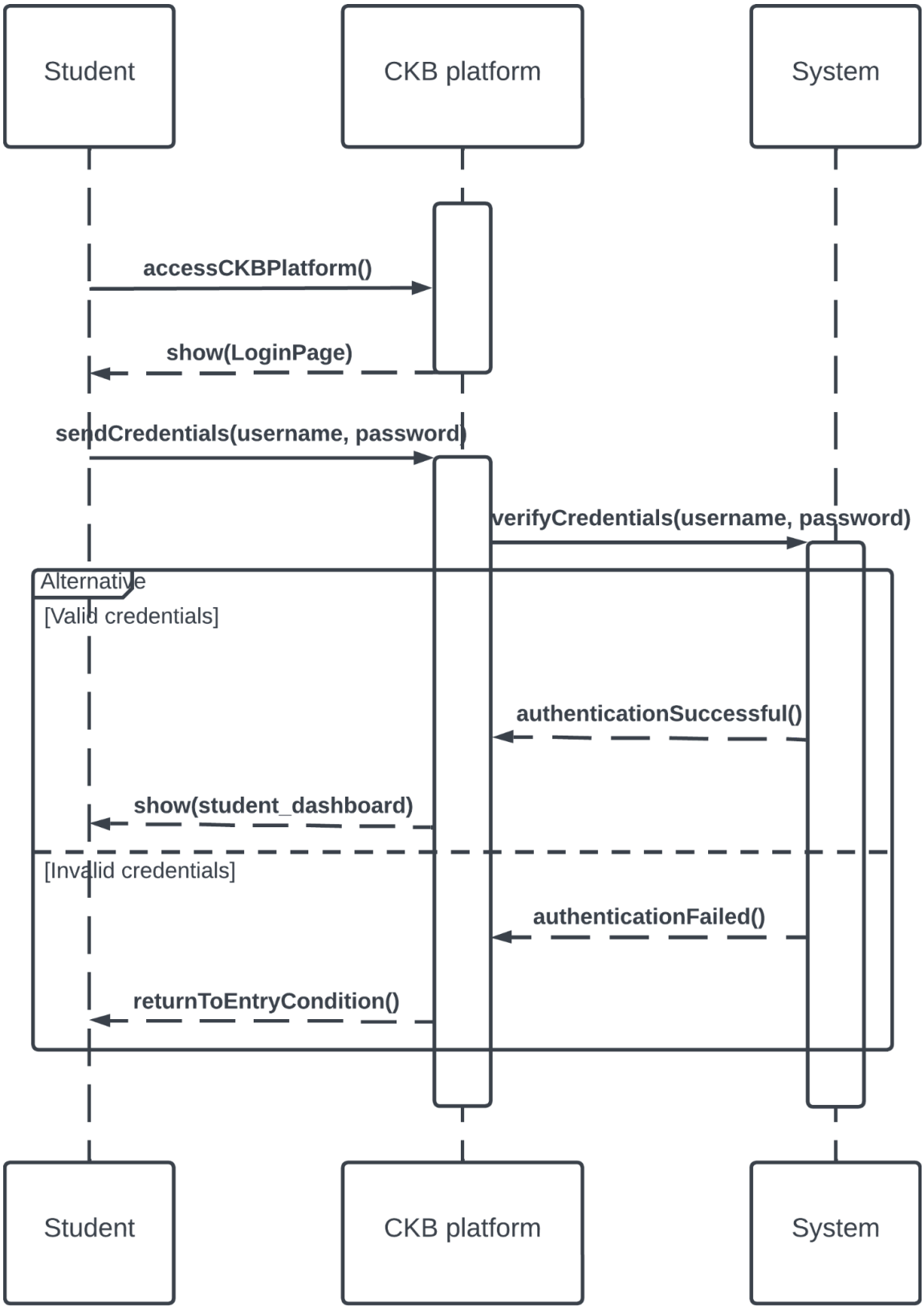
Name	Educator Extends Battle Submission Deadline
Actors	Educator
Entry Condition	The educator is logged into the CKB platform and has ongoing battles
Event flow	1. Educator accesses the ongoing battles management section. 2. Educator selects a battle and extends the submission deadline. 3. The system updates the battle's submission deadline.
Exit condition	The battle's submission deadline is extended.

3.2.3. Sequence/Activity Diagrams

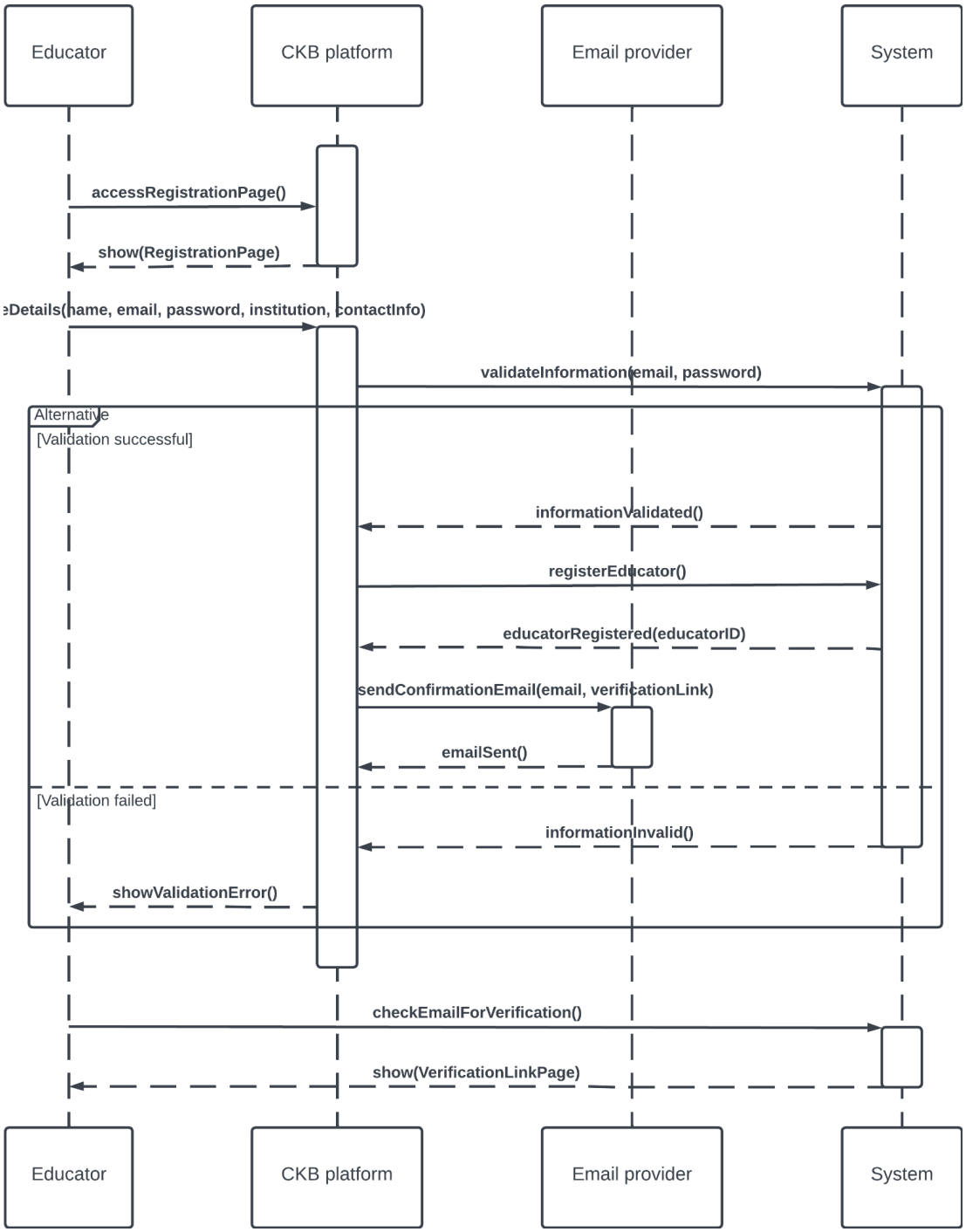
[UC1] - Educator Login



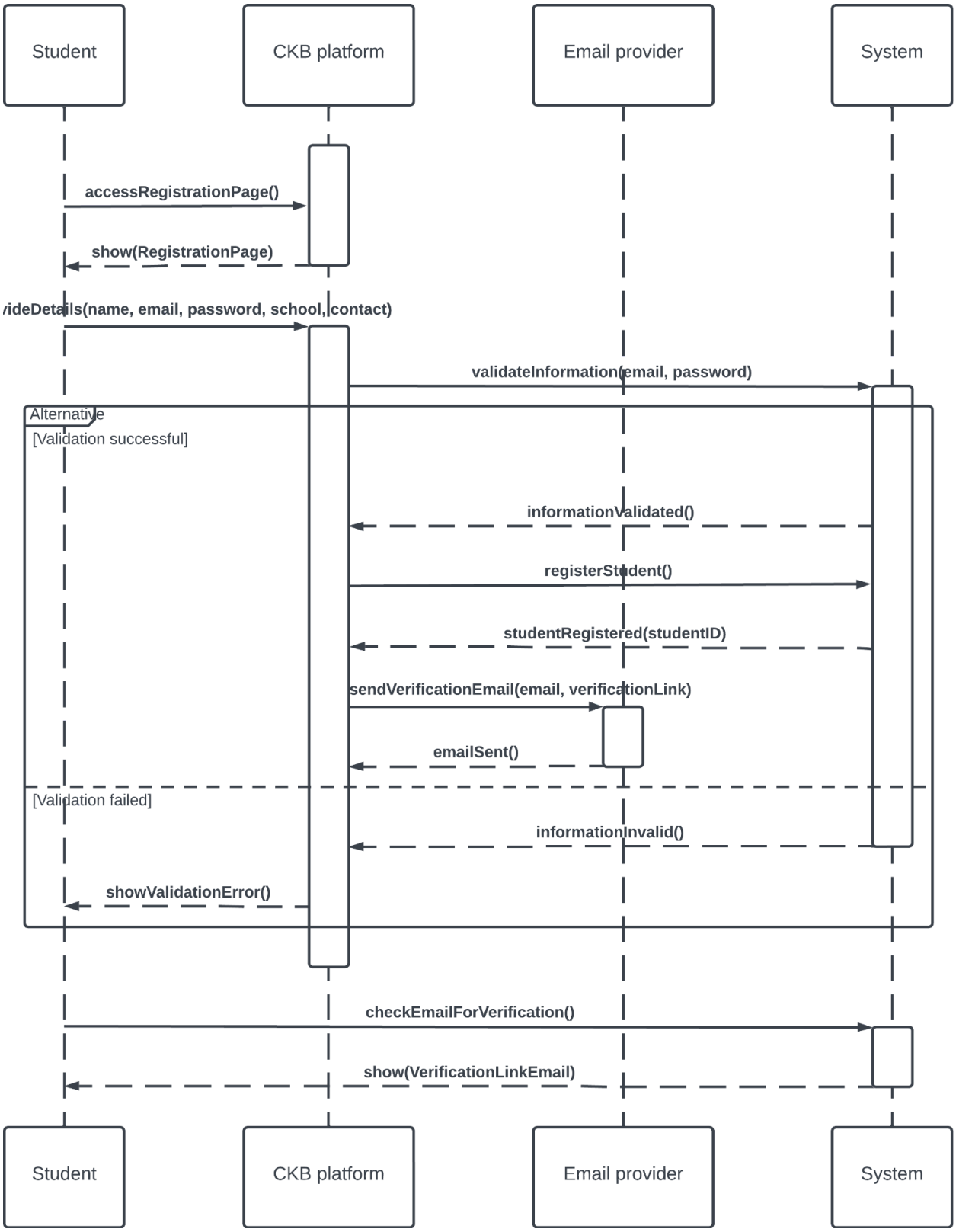
[UC2] - Student Login



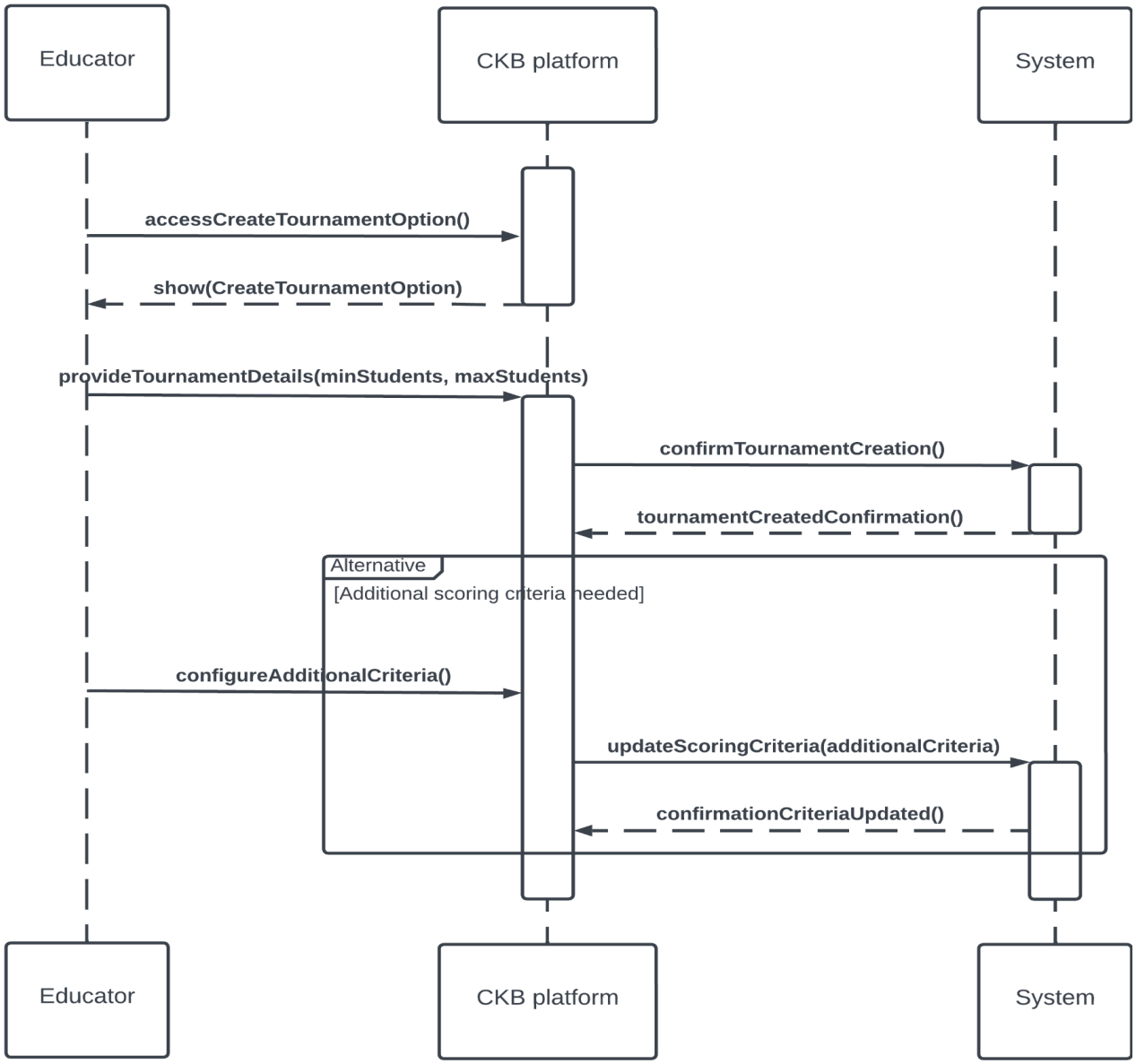
[UC3] - Educator Registration



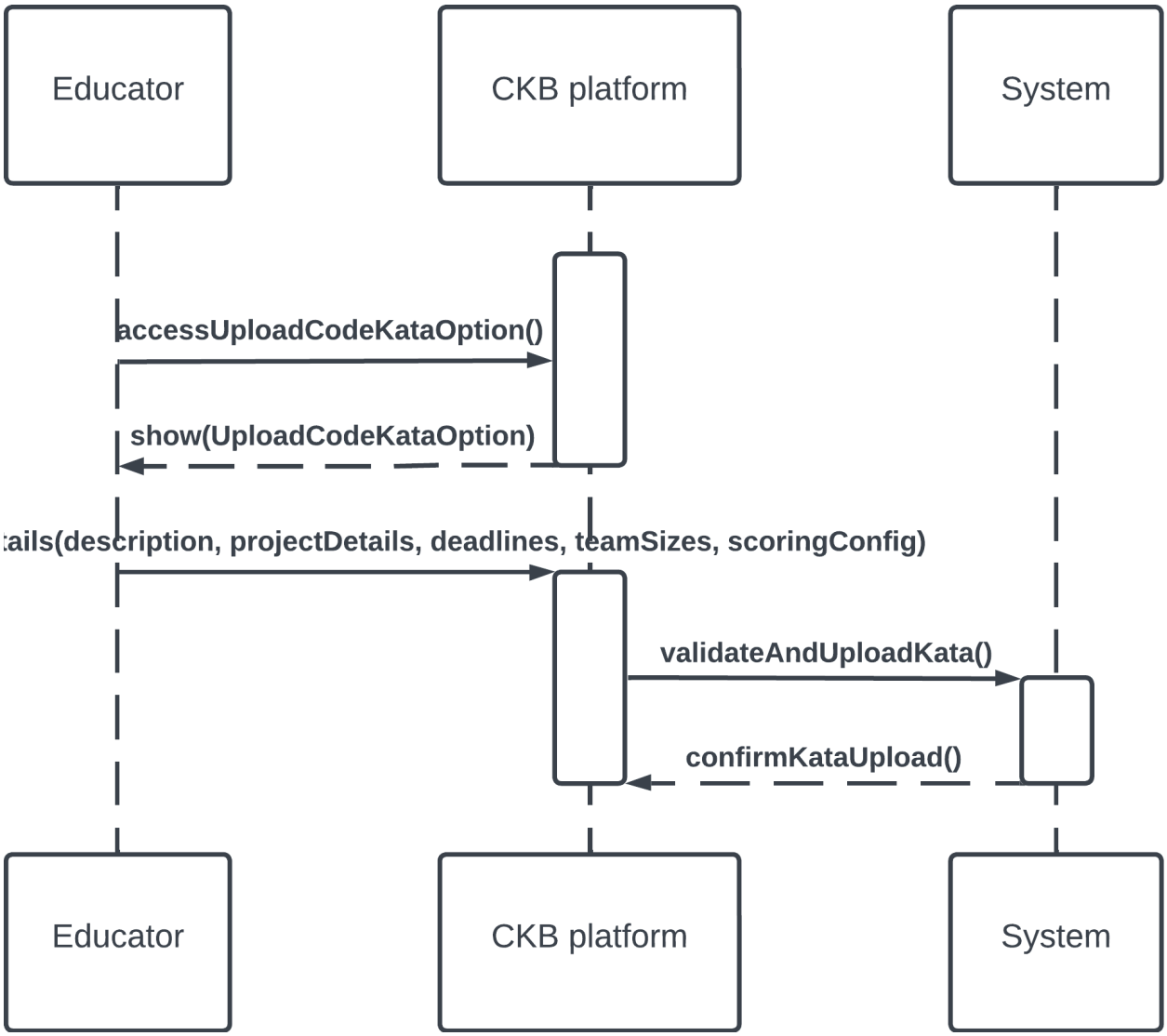
[UC4] - Student Registration



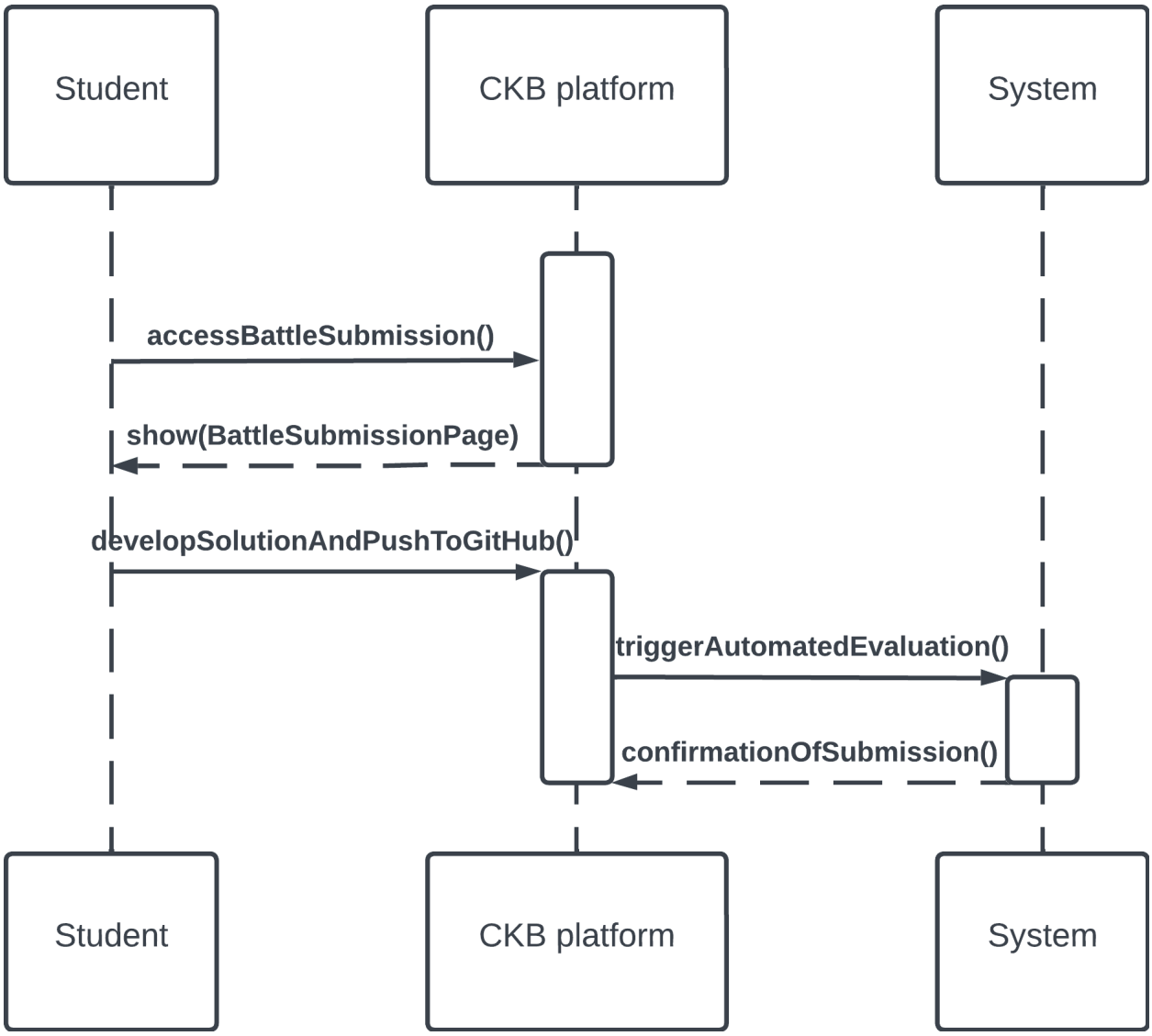
[UC5] - Educator Creates a Tournament



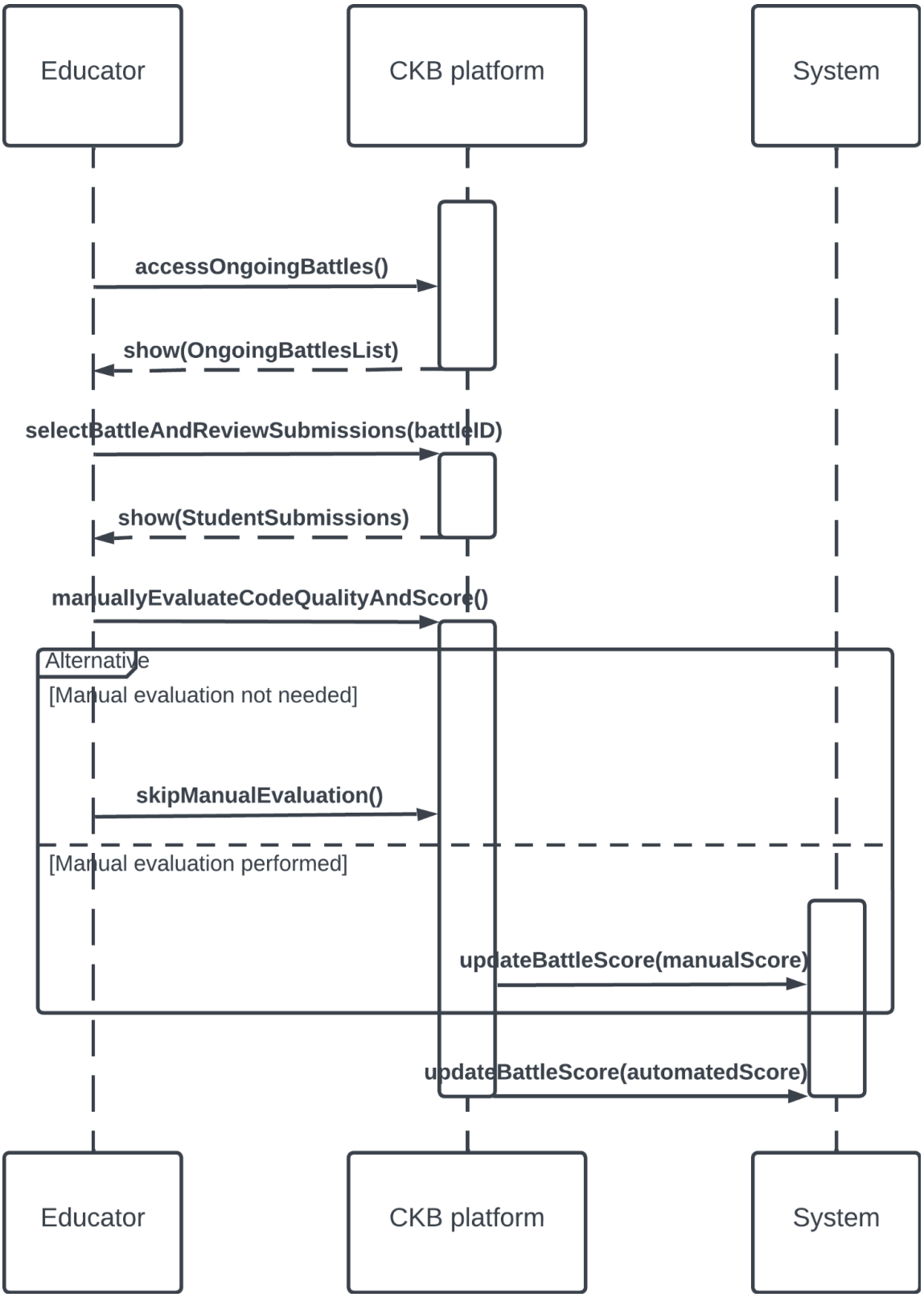
[UC6] - Educator Uploads Code Kata for Battle



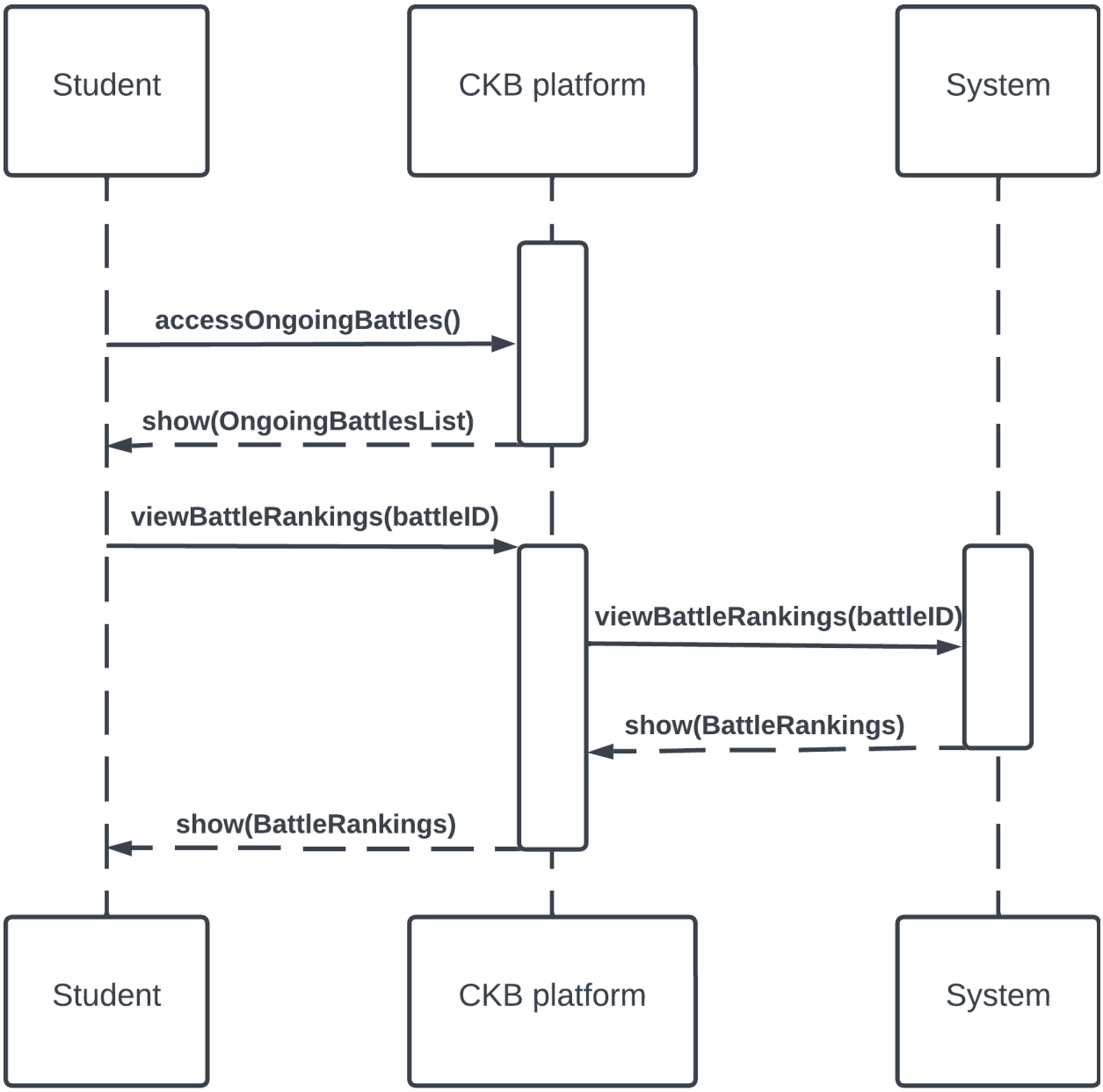
[UC7] - Student Submits Code Solution



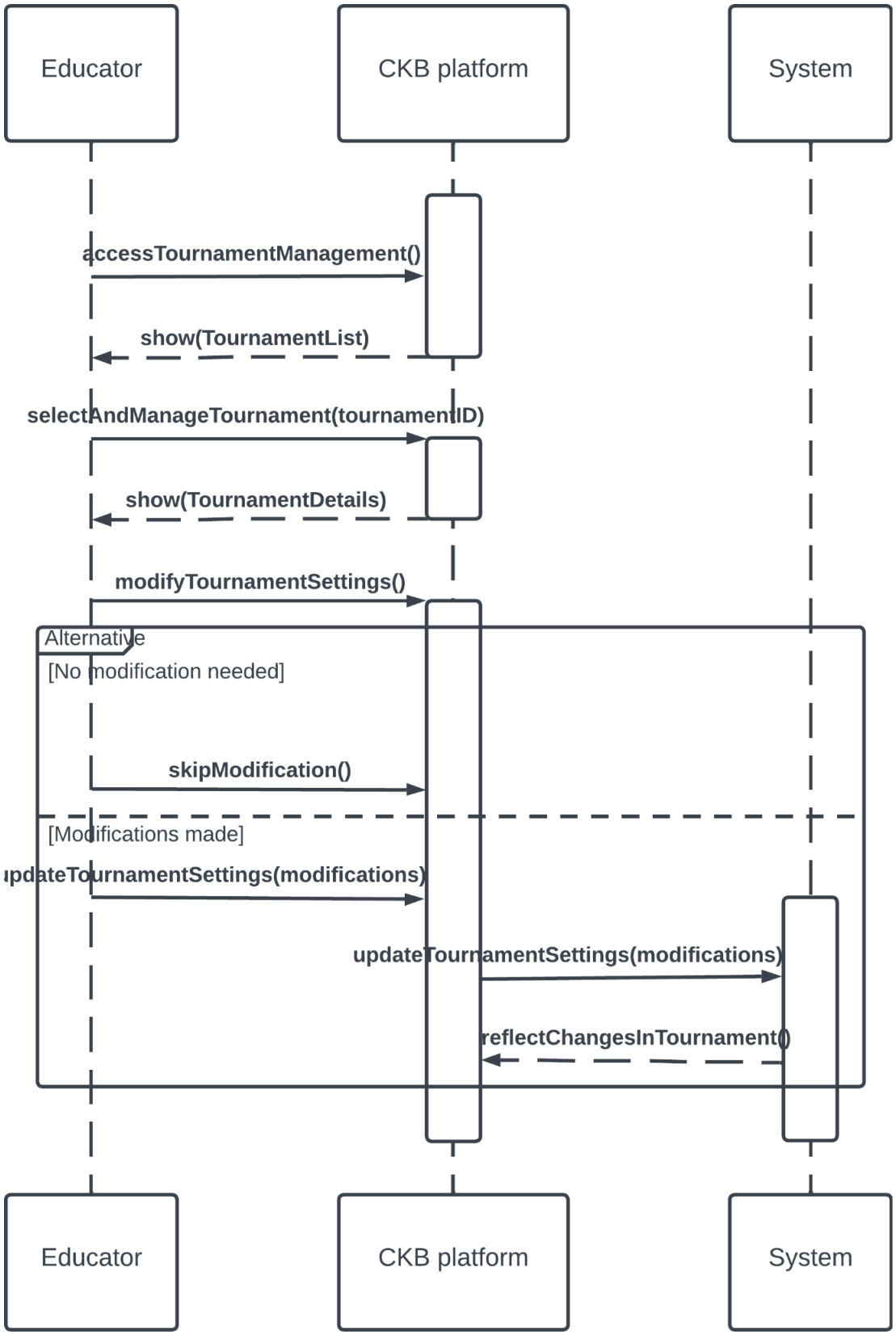
[UC8] - Educator Evaluates Student Submissions



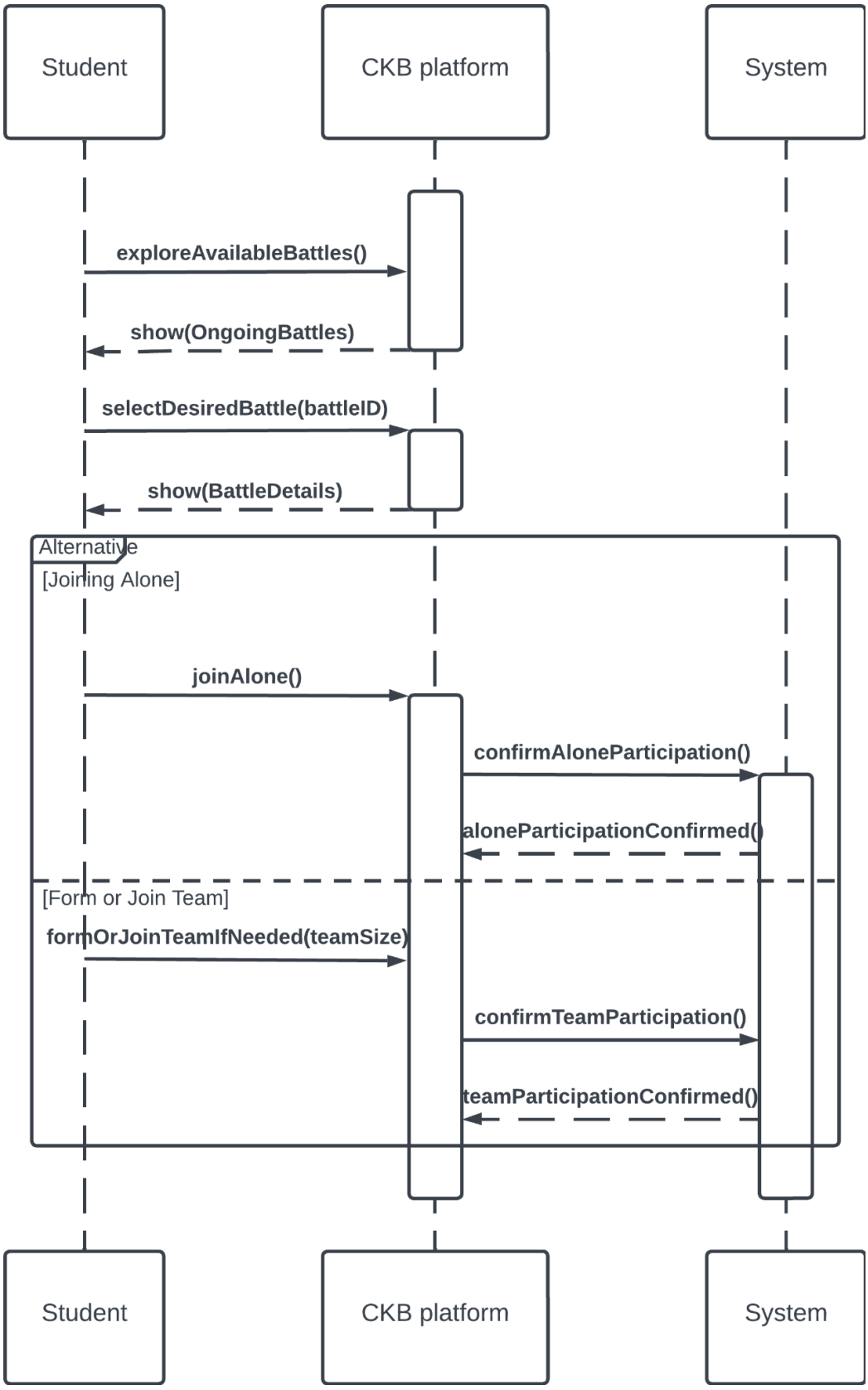
[UC9] - Student Views Battle Rankings



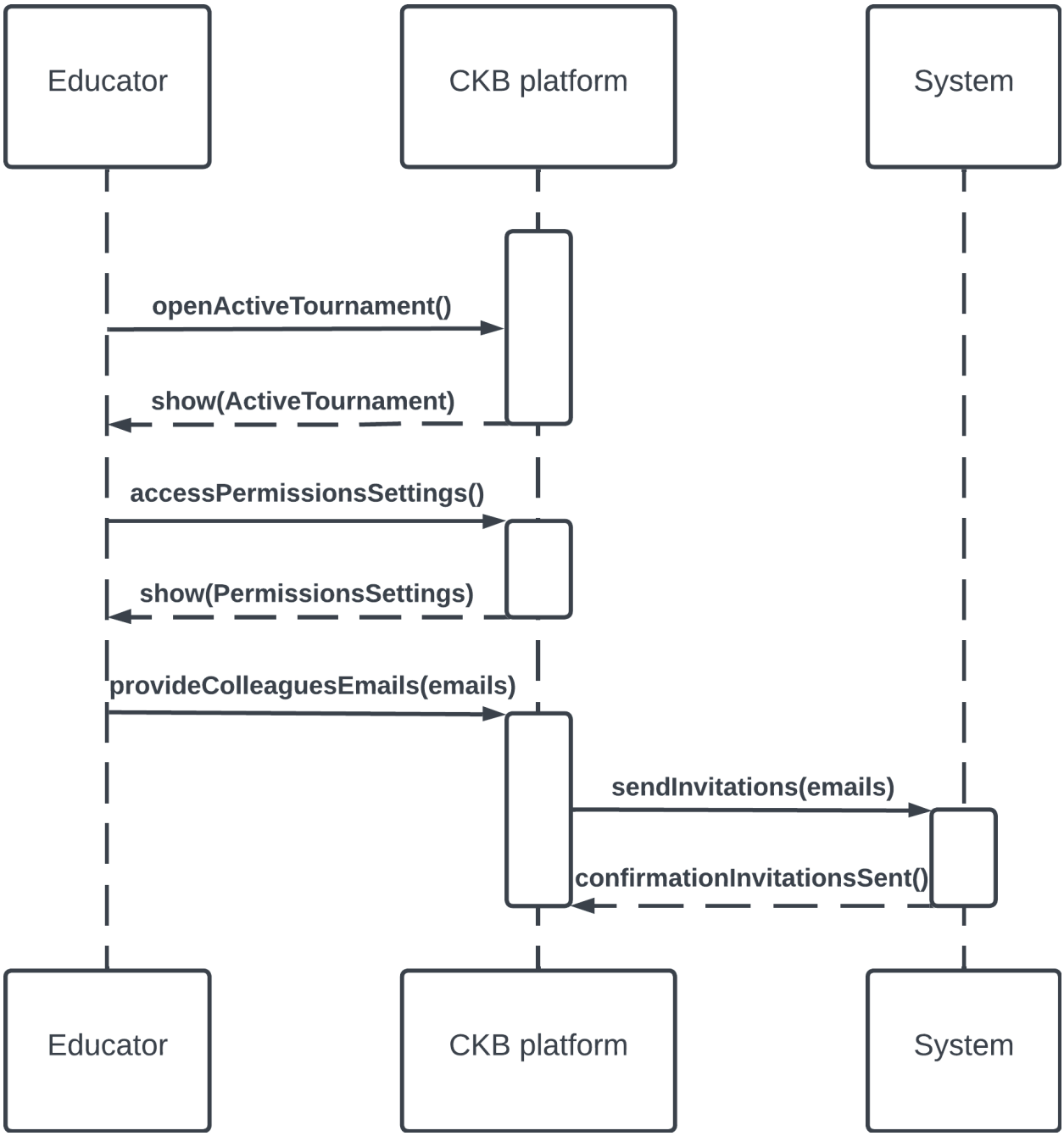
[UC10] - Educator Manages Tournament



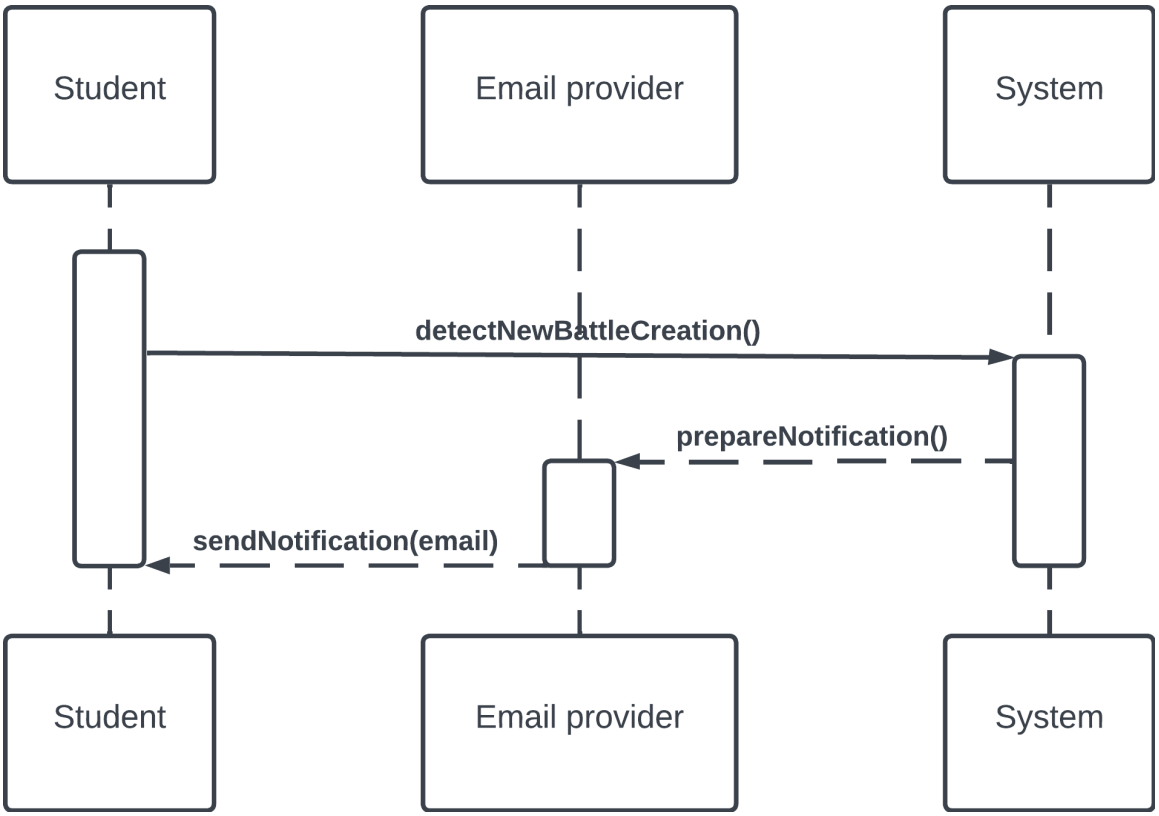
[UC11] - Student Joins a Battle



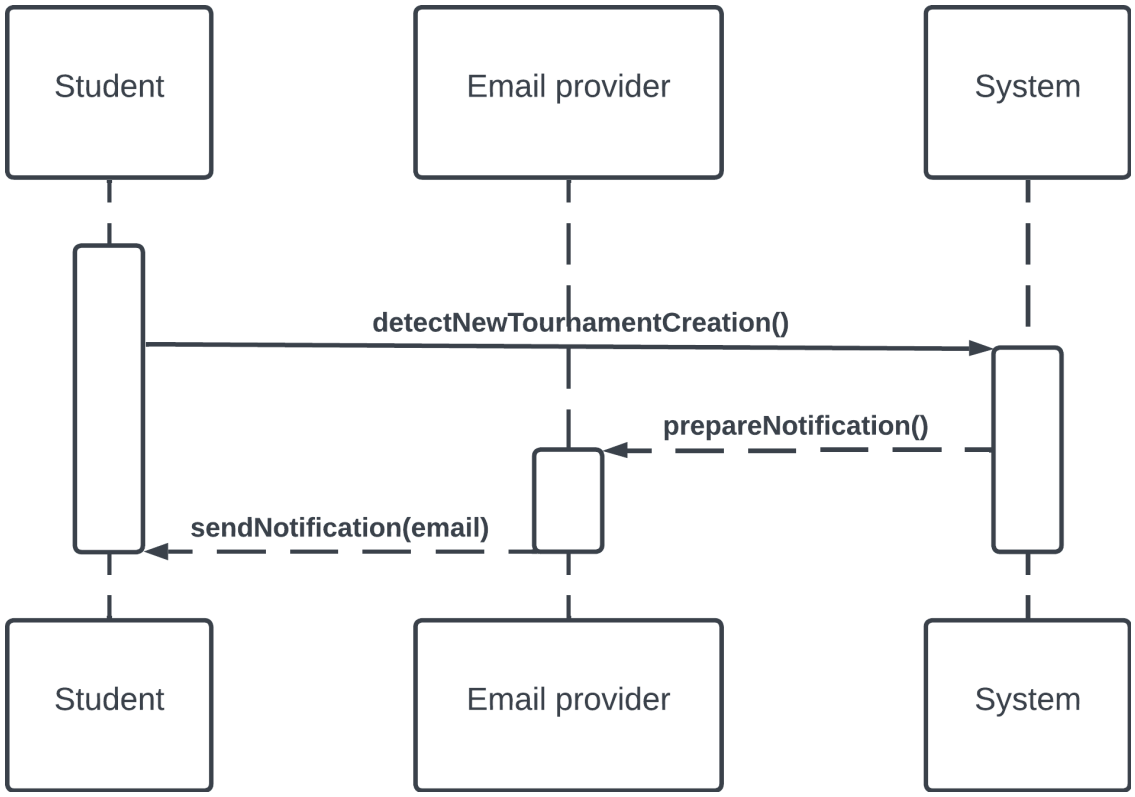
[UC12] - Educator Grants Permissions



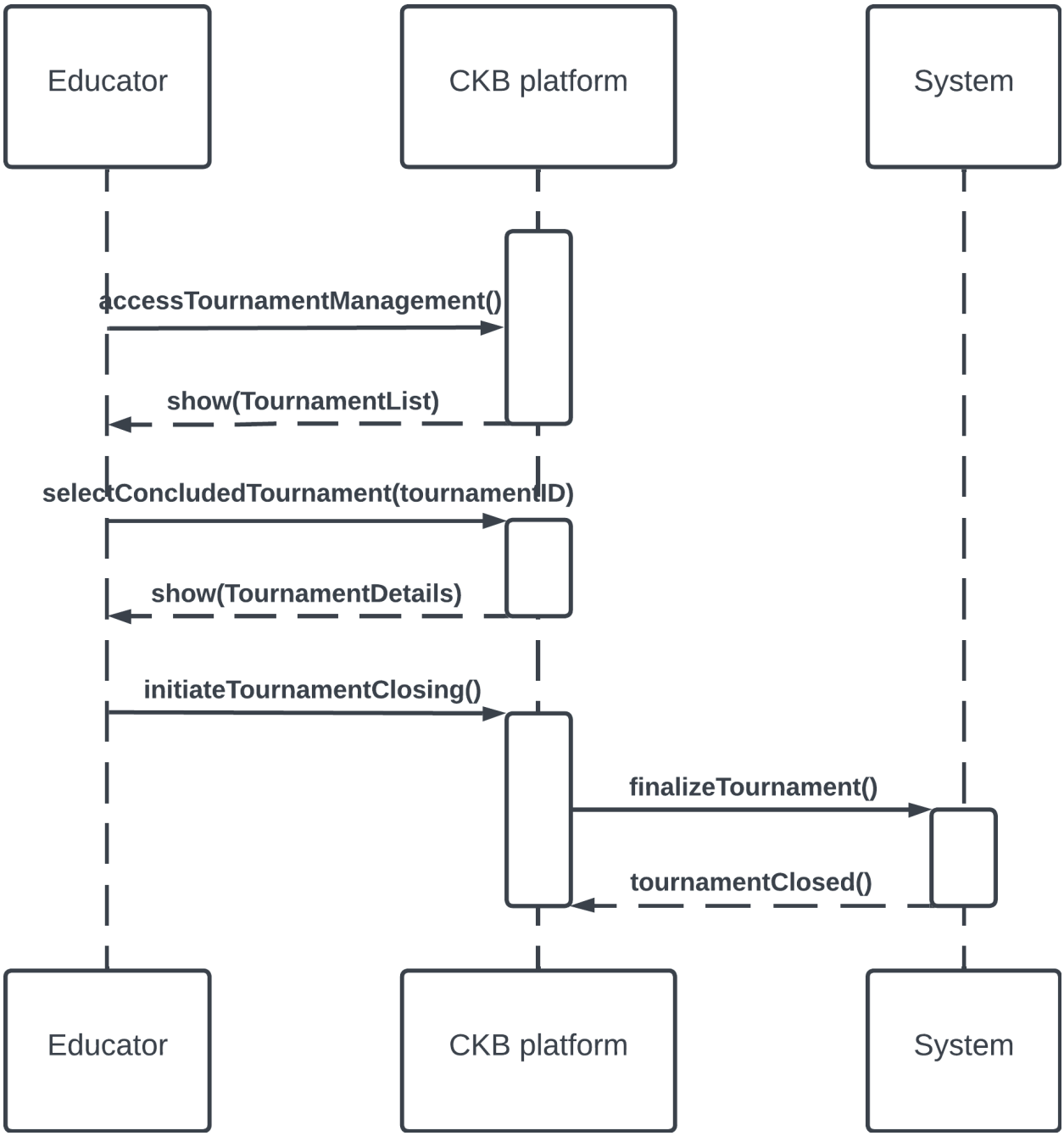
[UC13] - Student Receives Battle Notification



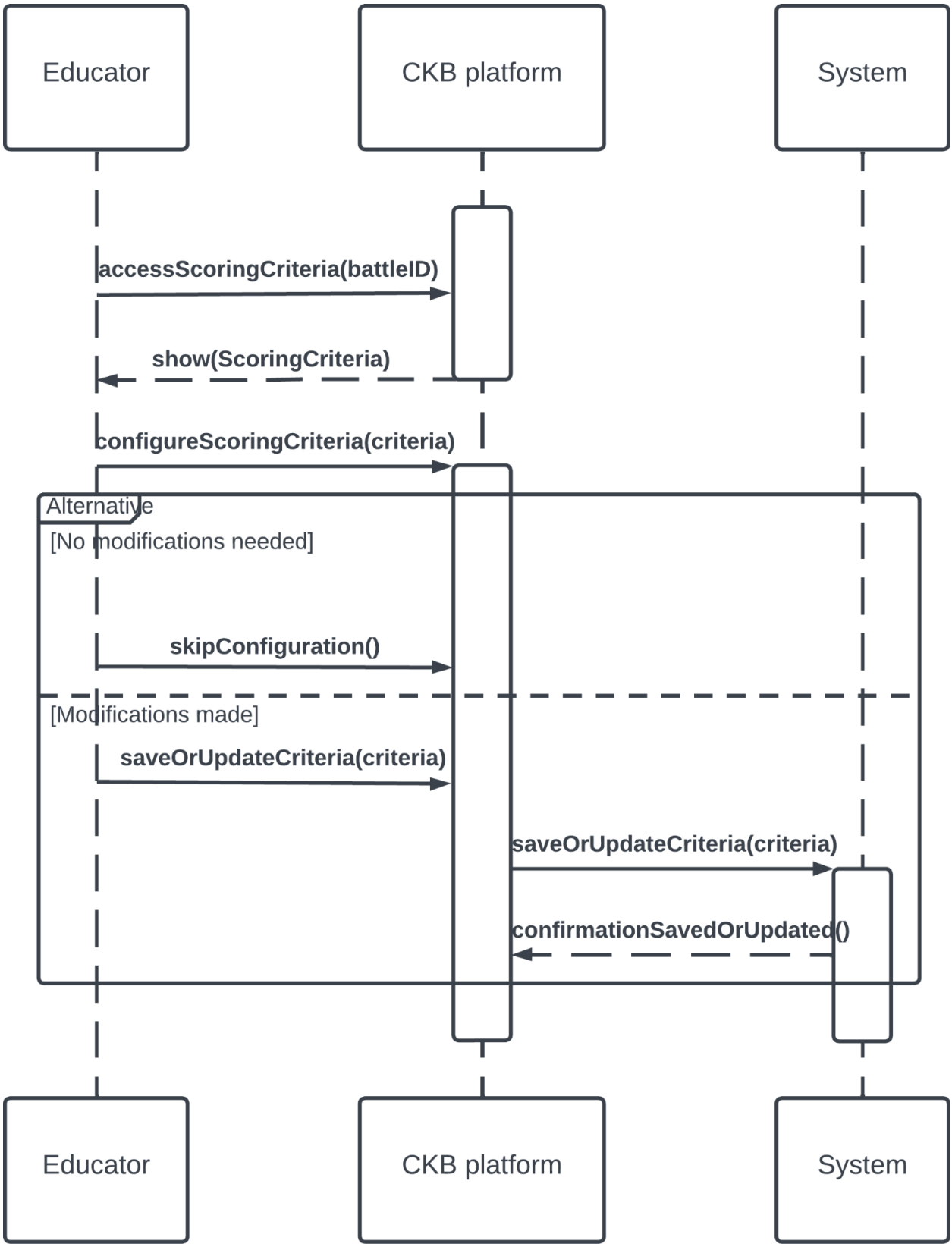
[UC14] - Student Receives Tournament Notification



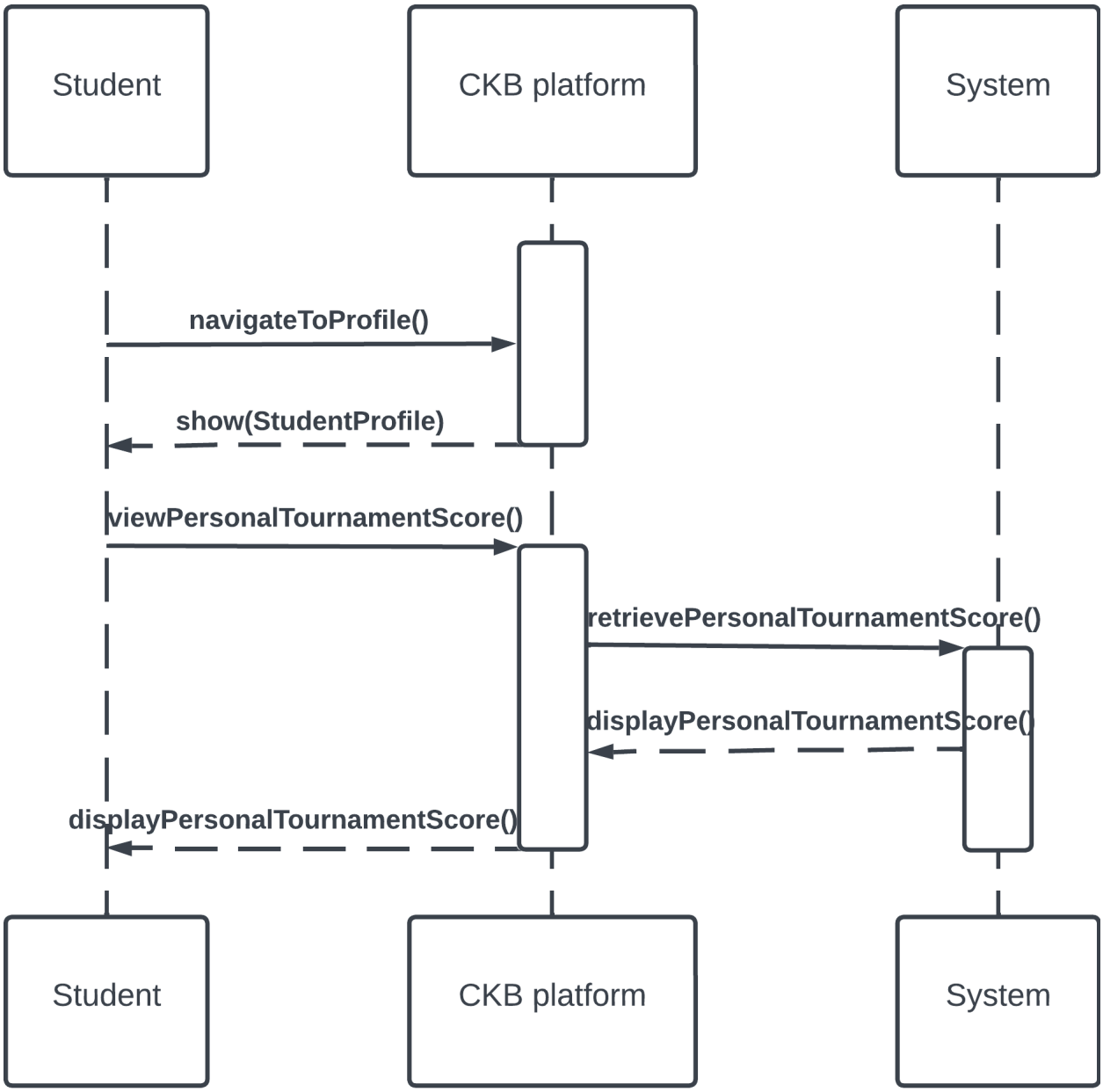
[UC15] - Educator Closes Tournament



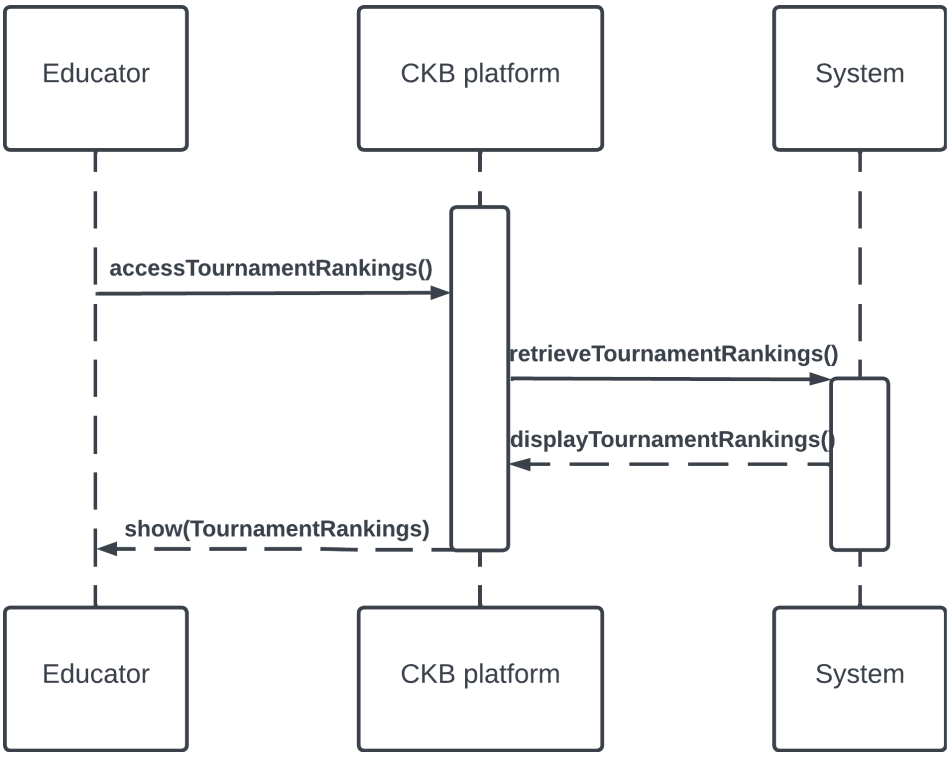
[UC16] - Educator Defines Scoring Criteria



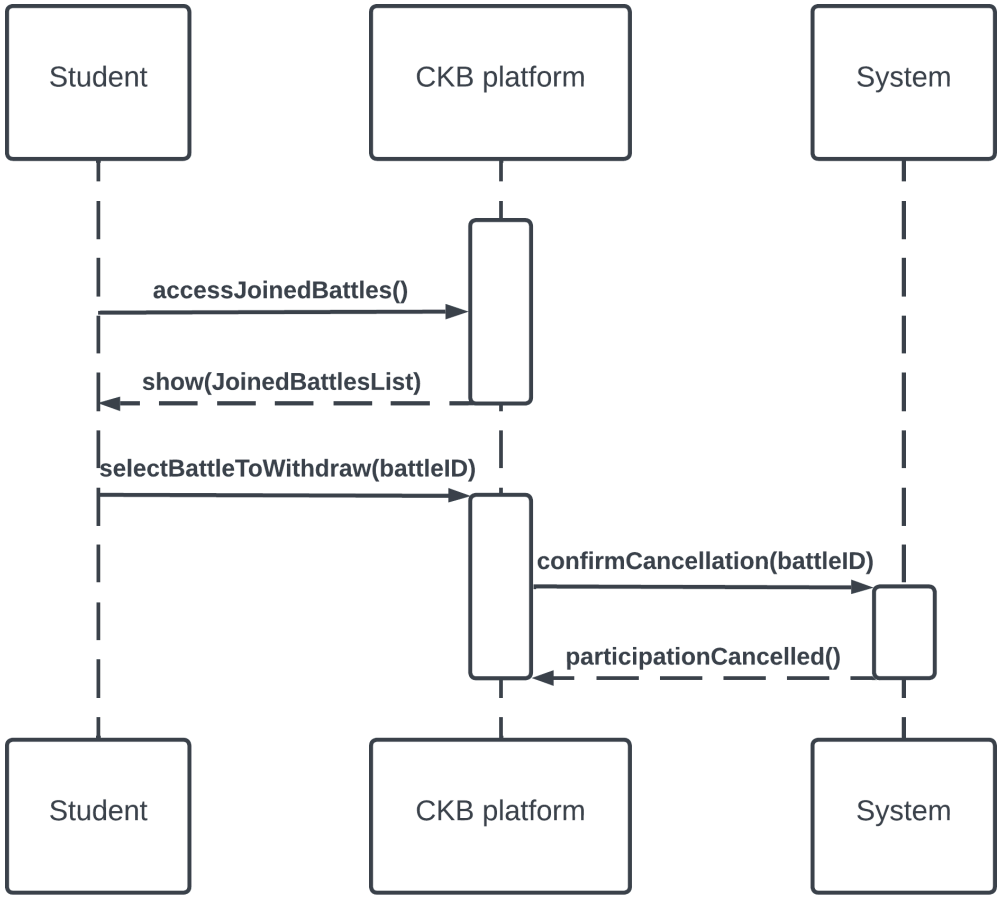
[UC17] - Student Views Personal Tournament Score



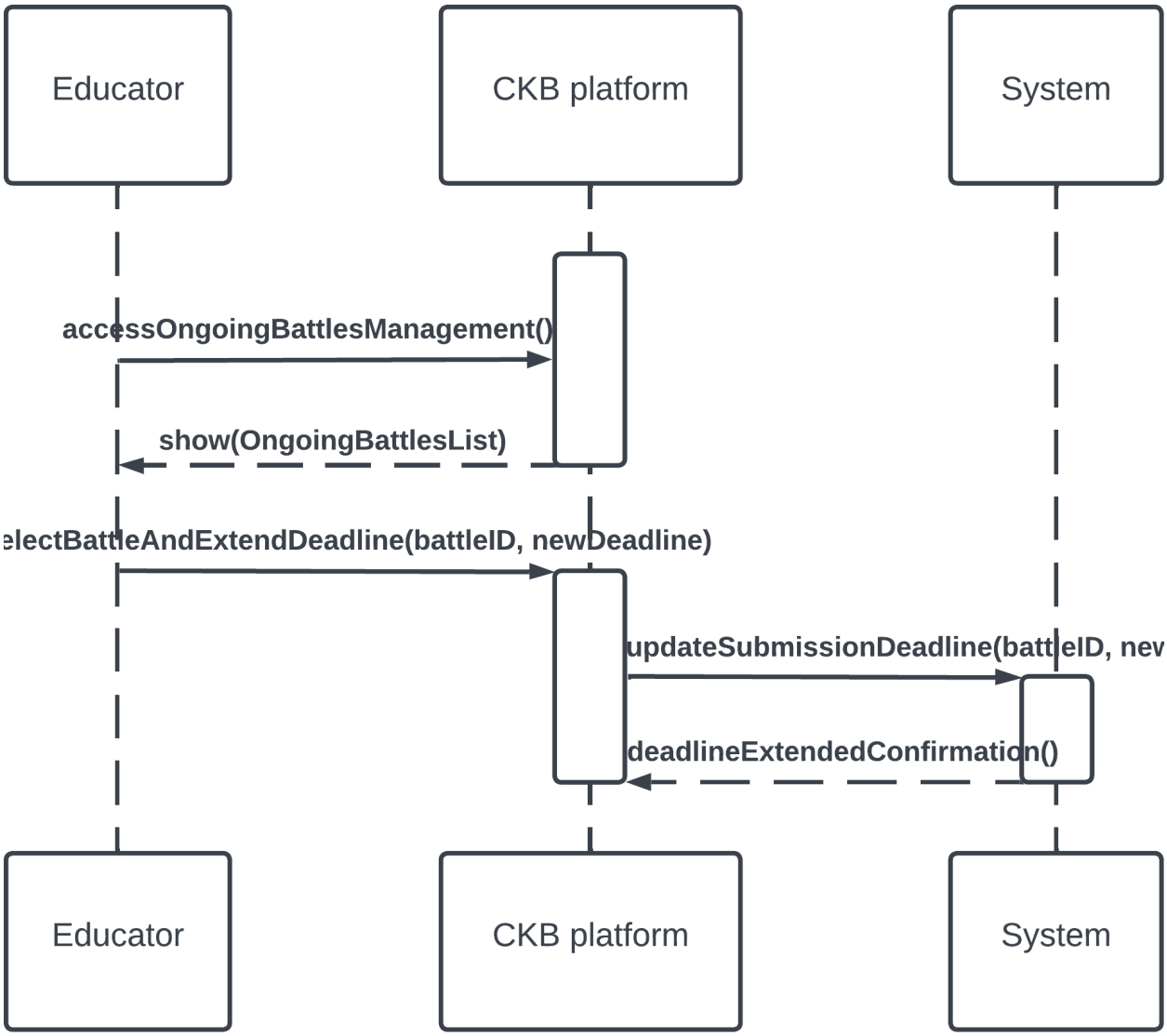
[UC18] - Educator Reviews Tournament Rankings



[UC19] - Student Cancels Battle Participation



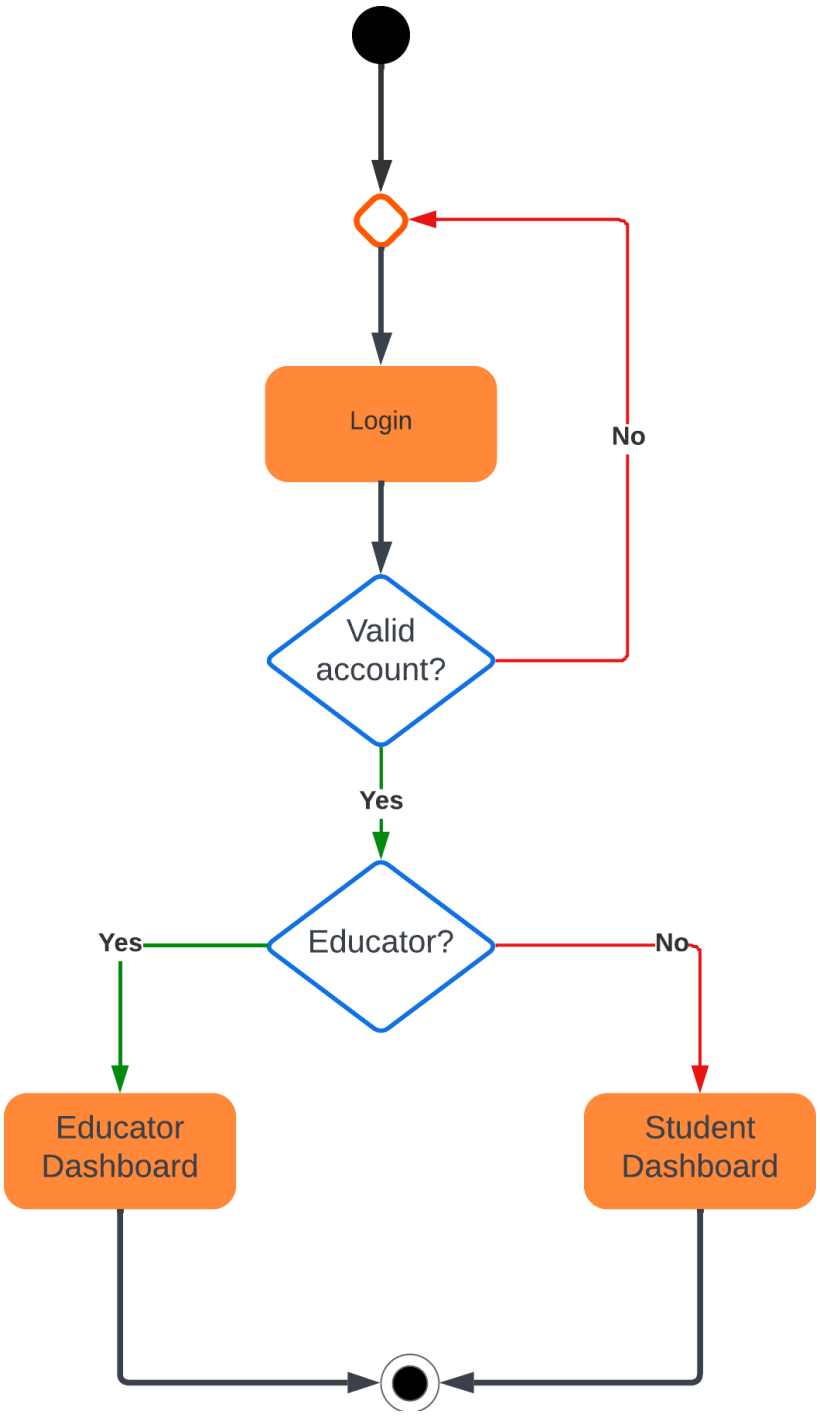
[UC20] - Educator Extends Battle Submission Deadline



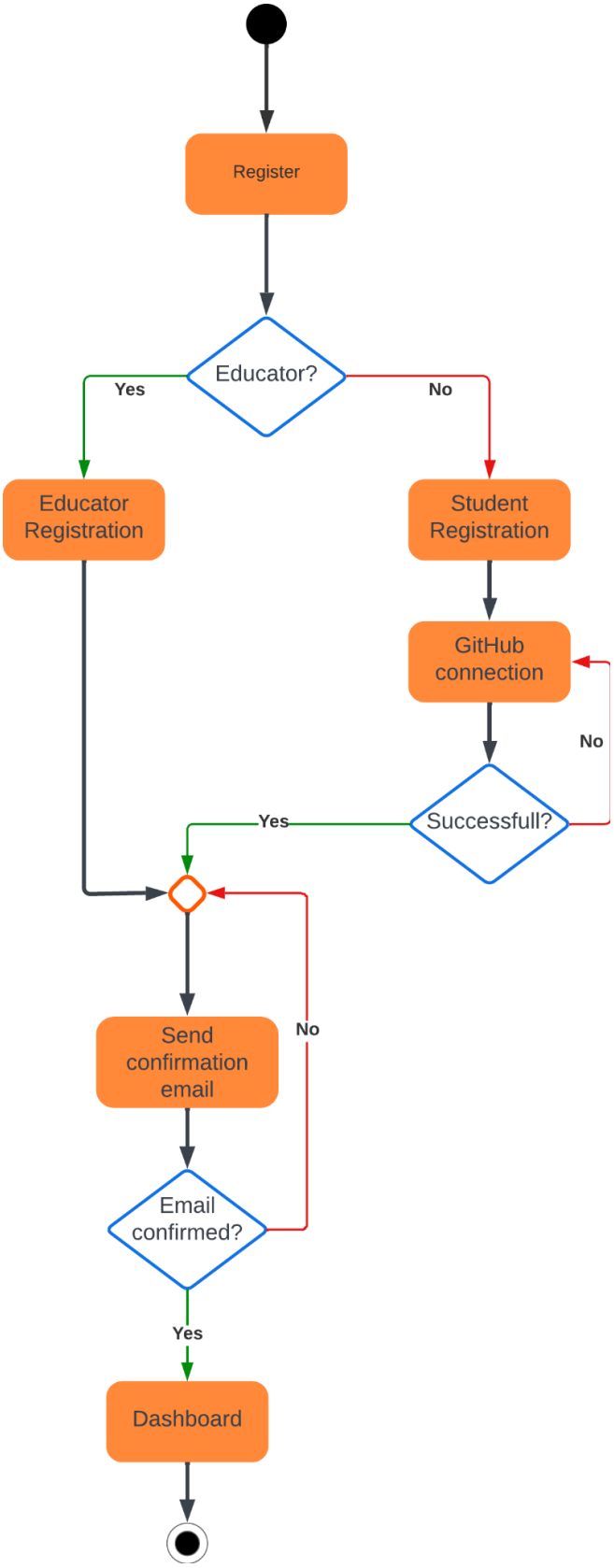
3.2.4. Activity Diagrams

Activity diagrams within the CodeKataBattle (CKB) documentation visually depict the step-by-step processes and interactions occurring within the platform. These diagrams serve as graphical representations illustrating the sequential flow of actions involved in various key functionalities. They provide a clear overview of user-system interactions, showcasing tasks such as user registration, login, tournament management, team formations.

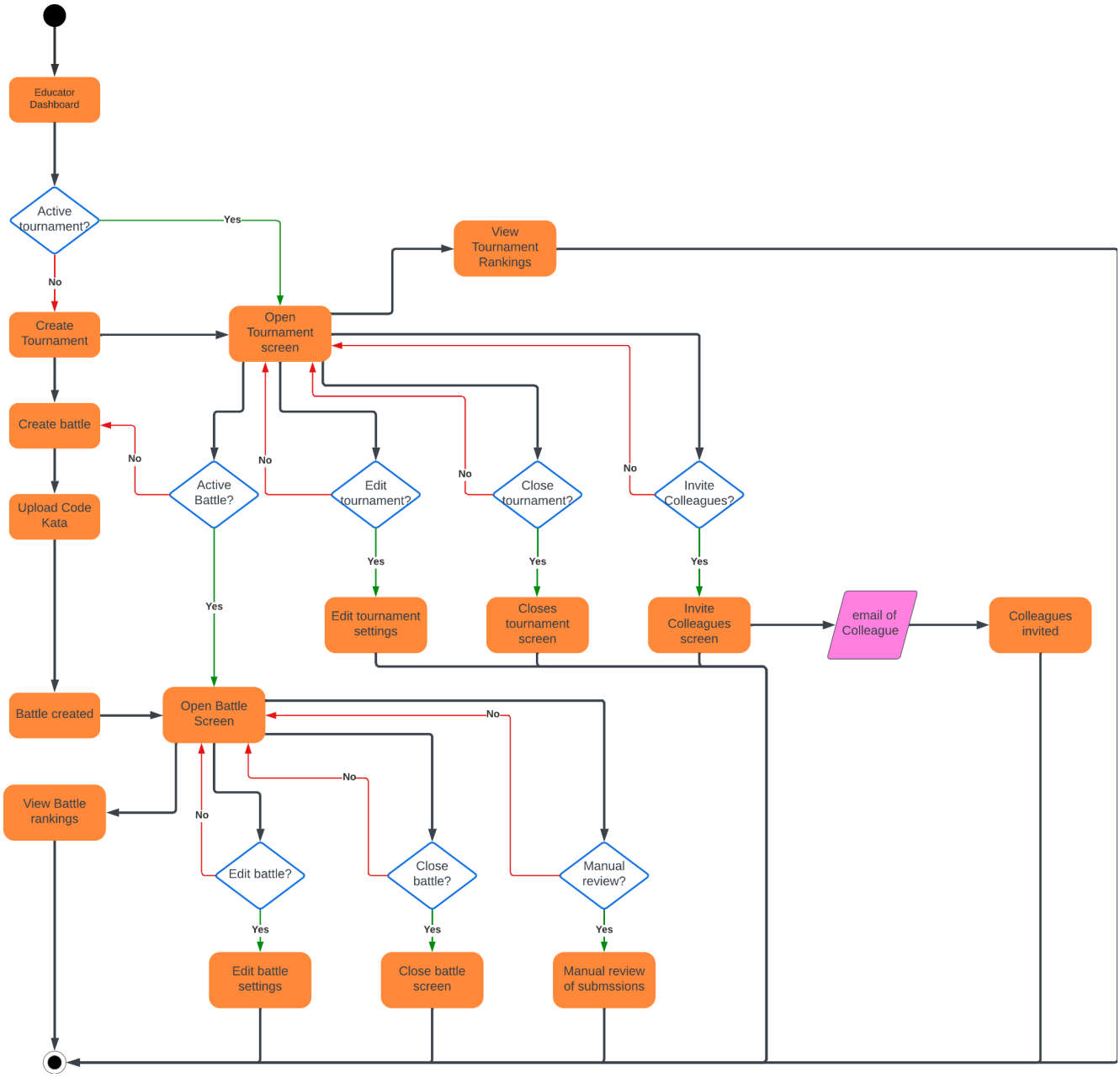
[UC1]-[UC2] Login



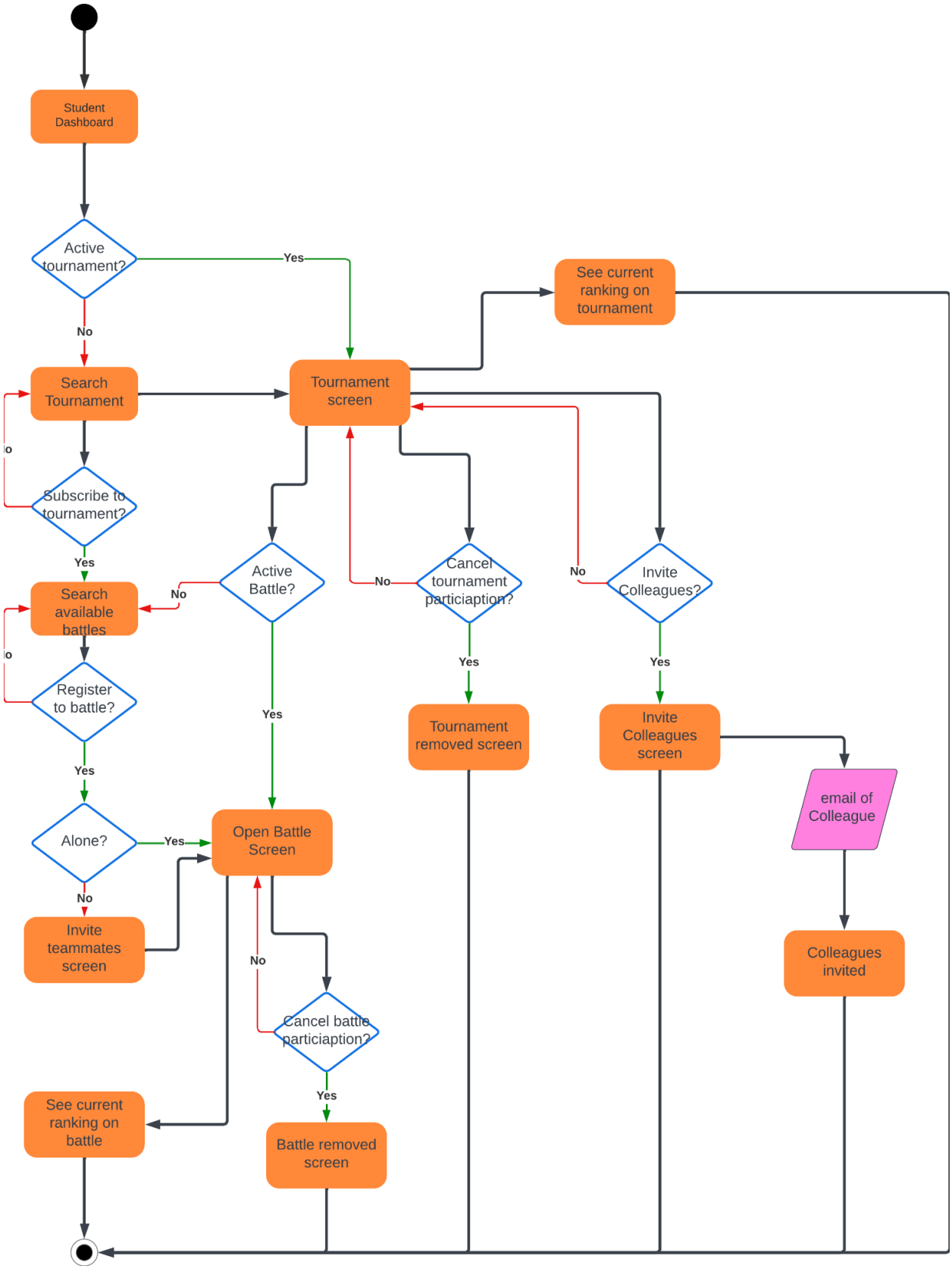
[UC3]-[UC4] Registration



[UC5]-[UC6]-[UC10]-[UC12]-[UC15]-[UC16]-[UC18]-[UC20] Educator Dashboard



[UC7]-[UC9]-[UC11]-[UC13]-[UC14]-[UC17]-[UC19] Student Dashboard



3.2.5. Requirements mapping

Each functional requirement will be mapped to corresponding use cases and diagrams to ensure a comprehensive understanding of the system's behavior and interactions.

Raw ID	Goal ID	Req ID	Use Case ID	Comments
r1	G.1	R.1	UC.1	
r2	G.3	R.2	UC.2	
r3	G.1	R.3	UC.3	
r4	G.1	R.4	UC.4	
r5	G.6	R.5	UC.5	
r6	G.2	R.6	UC.6	
r7	G.5	R.7	UC.7	
r8	G.2	R.8	UC.8	
r9	G.1	R.9	UC.9	
r10	G.5	R.10	UC.10	
r11	G.6	R.11	UC.11	
r12	G.6	R.12	UC.12	
r13	G.2	R.13	UC.13	
r14	G.6	R.14	UC.14	
r15	G.1	R.15	UC.15	
r16	G.2	R.16	UC.16	
r17	G.5	R.17	UC.17	
r18	G.6	R.18	UC.18	
r19	G.5	R.19	UC.19	
r20	G.5	R.20	UC.20	

[G1] Students enhance their software development skills through practical coding exercises and challenges.	
[R1] The system allows students to sign up and log in. [R7] Educators can upload the code kata with the respective description and software project, including test cases and build automation scripts. [R12] Students can form teams for battles, adhering to set team size limits. [R18] The system continuously updates battle scores during the battle period, displaying evolving rankings to students and educators. [R20] The platform updates personal tournament scores for each student, presenting the cumulative performance across battles within a tournament.	[D1]: Users in an educational setting. [D2]: Stable internet access for users. [D4]: Educators competence in creating coding exercises. [D5]: Students willingness to collaborate.

[G2] Foster collaboration among students by allowing them to form teams and work together on code kata battles.	
[R1] The system allows students to sign up and log in. [R4] Educators can set minimum and maximum team sizes for battles within a tournament. [R12] Students can form teams for battles, adhering to set team size limits. [R13] The platform automatically generates GitHub repositories for each battle upon creation. [R18] The system continuously updates battle scores during the battle period, displaying evolving rankings to students and educators.	[D4]: Educators competence in creating coding exercises. [D5]: Students' willingness to collaborate.

[G3]: Simulate real-world software development practices, including version control, automated testing, and continuous integration.	
[R3] Educators can create new tournaments, defining code battles with descriptions, deadlines, and scoring configurations. [R7] Educators can upload the code kata with the respective description and software project, including test cases and build automation scripts. [R13] The platform automatically generates GitHub repositories for each battle upon creation. [R15] Integration with static analysis tools for quality evaluation, considering security, reliability, and maintainability of code.	[D2]: Stable internet access for users. [D3]: Basic GitHub proficiency. [D6]: Dependence on GitHub API. [D7]: User reliance on stable internet service. [D8]: Integration of code quality analysis tools.

[G4]: Encourage a test-first approach to programming by providing challenges with predefined test cases.	
[R3] Educators can create new tournaments, defining code battles with descriptions, deadlines, and scoring configurations. [R5] Educators can manage and view the list of battles they've created within a tournament. [R7] Educators can upload the code kata with the respective description and software project, including test cases and build automation scripts. [R15] Integration with static analysis tools for quality evaluation, considering security, reliability, and maintainability of code. [R16] Educators can manually assess and assign personal scores to student submissions. [R17] After the submission deadline, educators review and provide manual evaluations based on code quality and adherence to the test-first approach.	[D1]: Users in an educational setting. [D4]: Educators' competence in creating coding exercises. [D6]: Dependence on GitHub API [D7]: User reliance on stable internet service. [D8]: Integration of code quality analysis tools. [D9]: Educator-configured scoring parameters.

[G5]: Create a competitive environment where students can compete in code kata battles and see their rankings on leaderboards.	
[R1] The system allows students to sign up and log in. [R15] Integration with static analysis tools for quality evaluation, considering security, reliability, and maintainability of code. [R16] Educators can manually assess and assign personal scores to student submissions. [R17] After the submission deadline, educators review and provide manual evaluations based on code quality and adherence to the test-first approach. [R19] Upon consolidation after the submission deadline, the final battle rank becomes available to all participants. [R20] The platform updates personal tournament scores for each student, presenting the cumulative performance across battles within a tournament.	[D1]: Users in an educational setting. [D4]: Educators' competence in creating coding exercises. [D5]: Students willingness to collaborate. [D9]: Educator-configured scoring parameters.

[G6]: Provide educators with the tools to create, manage, and evaluate coding challenges, as well as monitor students progress.	
[R2] The system allows educators to sign up and log in. [R3] Educators can create new tournaments, defining code battles with descriptions, deadlines, and scoring configurations. [R4] Educators can set minimum and maximum team sizes for battles within a tournament. [R5] Educators can manage and view the list of battles they've created within a tournament. [R6] Educators can grant permission to colleagues to create battles within a specific tournament. [R7] Educators can upload the code kata with the respective description and software project, including test cases and build automation scripts. [R9] Educators can set a registration deadline. [R10] Educators can set a final submission deadline. [R11] Educators can set additional configurations for scoring.	[D1]: Users in an educational setting. [D4]: Educators competence in creating coding exercises. [D6]: Dependence on GitHub API [D9]: Educator-configured scoring parameters.

3.3. Performance Requirements

The CodeKataBattle platform focuses on performance to ensure efficiency and responsiveness. It prioritizes responding quickly to user interactions, optimizing the platform for high loads, and efficiently rendering code versions. Data evaluation methods and benchmarks are explained, and load balancing strategies distribute traffic. Notification delivery, API response, and error handling contribute to a smooth user experience.

- **Response Time:** Ensure the system responds promptly to user interactions, aiming for quick responses across all operations under standard conditions.

- **Scalability:** Design the system to handle increased user loads during peak times without significant performance degradation.

- **Throughput:** Maintain a high throughput capacity to manage a substantial volume of code submissions efficiently, minimizing queuing or processing delays.

- **Database Performance:** Optimize database queries for swift retrieval of user data and battle information to ensure efficient platform operation.

- **Load Balancing:** Implement effective load balancing strategies to evenly distribute incoming traffic, ensuring consistent system performance.

- **Code Evaluation Time:** Streamline code evaluation processes to expedite the assessment of submitted solutions without prolonged delays.

- **Notification Delivery Time:** Ensure timely delivery of notifications (email, platform alerts) to keep users informed without undue delays.

- **API Response Time:** Maintain external API responsiveness within acceptable time frames for seamless integration with the system.

- **Error Handling:** Promptly provide error messages or notifications to users upon encountering issues for quick problem resolution.

- **Platform Uptime:** Ensure high availability of the platform, minimizing downtime to maintain continuous accessibility for users.

3.4. Design Constraints

3.4.1. Standards compliance

The CodeKataBattle platform strictly adheres to industry standards and best practices to ensure a robust, secure and globally compatible system. It adheres to a number of important aspects, including standard coding practices for consistency and maintainability, and strictly adheres to the W3C Guidelines for Web Components to ensure the best user experience. Encryption protocols protect the transmission and storage of sensitive data, while comprehensive documentation aids understanding and future development. Version control and testing procedures follow agreed frameworks and procedures, ensuring robustness and interoperability. Accessibility, compliance and global support mean a commitment to accessibility, data protection and global use. Overall, these design constraints strengthen the CKB platform and its reliability, security, and usability in industrial environments.

3.4.2. Hardware limitations

- **Server Capacities:** The system's performance might be limited by server capabilities, such as processing power, memory, and storage. Ensuring adequate server resources is crucial to handle concurrent user loads during peak usage.

- **Computational Resources:** The computational resources available, including CPUs and GPUs, can affect the speed and efficiency of code evaluation and analysis, especially when dealing with a high volume of submissions.

- **Network Bandwidth and Latency:** The platform's responsiveness and data transfer speeds may be limited by network bandwidth and latency. Slow internet connections could impact users' interactions with the platform.

- **Scalability Challenges:** Hardware limitations might pose challenges in scaling the system to accommodate rapid increases in user traffic or data volume, potentially leading to performance bottlenecks during peak times.

- **Redundancy and Fault Tolerance:** Hardware failures can impact system availability. Implementing redundancy and fault-tolerant mechanisms becomes essential to ensure continuous operation in case of hardware failures.

3.4.3. Any other constraint

- **Time Constraints:** Project deadlines or time-to-market requirements could limit the development cycle, affecting the scope and depth of features or rigorous testing phases.

- **Compatibility Constraints:** Ensuring compatibility across various browsers, operating systems, and devices could be a constraint, particularly in delivering consistent experiences to diverse user bases.

3.5. Software System Attributes

3.5.1. Reliability

CKB prioritizes consistent and error-free operation by employing robust error handling mechanisms and conducting comprehensive testing procedures. Continuous monitoring of system performance allows for swift identification and rectification of potential failure points, ensuring a reliable user experience.

3.5.2. Availability

Ensuring uninterrupted access to the platform is paramount for CKB. To achieve this, redundant systems and failover mechanisms are in place to minimize downtime. Regular maintenance and updates occur during off-peak hours, complemented by load balancing strategies to evenly distribute traffic and prevent overload, ensuring consistent availability.

3.5.3. Security

Protecting user data and system integrity is a key focus. CKB implements stringent authentication and authorization measures, employing encryption techniques for secure data transmission and storage. Regular updates, patching, and security audits are conducted to address vulnerabilities and ensure robust protection against threats.

3.5.4. Maintainability

Adherence to coding standards, strong documentation practices and modular system architecture ensure ease of system maintenance, upgrades and changes. These practices facilitate updates, enhancements, and version control, enabling effective change management.

3.5.5. Portability

CKB is designed to be flexible and deployable in a variety of environments and platforms. System components have been developed to be platform independent and to meet web standards for cross-browser compatibility. By using distributed or virtual technologies, it can be delivered to multiple environments, making it more portable.

4. FORMAL ANALYSIS USING ALLOY

4.1. Signature

The Signature subsection of Alloy establishes the fundamental building blocks of the formal model. It defines the abstract entities, sets, and relations that form the basis of the subsequent Alloy specification. The Signature section delineates the vocabulary and structure necessary to construct a precise and concise representation of the CodeKataBattle platform's key components and their interactions.

```
// Represents a tournament with an owner (an educator) and a set of invited educators
sig Tournament {
   owner: one Educator,
   invited: set Educator,
} {
   no e: Educator | e in invited && e = owner
}

// Represents a battle with an owner (an educator), associated with a tournament,
// and defines minimum and maximum team sizes for the battle
sig Battle {
   ownedBy: one Educator,
   PartOfTournament: one Tournament,
   minTeamSize: one Int,  // Minimum team size for the tournament
   maxTeamSize: one Int,   // Maximum team size for the tournament
} {
   minTeamSize >= 1 && maxTeamSize <= 5 // Ensure reasonable limits
}

// Represents a user (abstract)
abstract sig User {}

// Represents a student who is part of a team, subscribed to tournaments, and has
tournament scores
sig Student extends User {
   partOfTeam: set Team,
   subscribedTo: set Tournament,
   tournamentScore: set TournamentScore
} {
   #tournamentScore = #subscribedTo
}

// Represents an educator who owns battles and tournaments
sig Educator extends User {
   ownBattle: set Battle,
   ownTournaments: set Tournament,
```

```
}

// Represents a team with students, associated with battles, and having team scores
// Limits the amount of students based on the battle settings
sig Team {
   students: some Student,
   isInBattles: one Battle,
   teamScore: one Score,
}{
   #students >= isInBattles.minTeamSize  // Minimum team size
   #students <= isInBattles.maxTeamSize  // Maximum team size
}

// Represents a score associated with a team, battle, and student(s)
sig Score {
   associatedTeam: one Team,
   associatedBattle: one Battle,
   studentScore: some Student
}

// Represents a tournament score associated with a tournament and a student
sig TournamentScore {
   tournament: one Tournament,
   associatedStudent: one Student
}
```

4.2. Facts

In the Facts subsection, concrete instances and constraints about the defined entities in the Signature section are presented. Facts provide specific details and constraints that define the initial state of the system or assert certain properties that hold true throughout the model. This section lays the groundwork for the precise definition of the system's initial conditions and introduces constraints that must be satisfied within the Alloy model.

```
// Fact ensuring the student tournament score is associated with the tournament the student is subscribed to
fact TournamentScoreMatchesSubscribedTournament {
   all ts: TournamentScore |
      ts in ts.associatedStudent.tournamentScore implies
      some t: ts.associatedStudent.subscribedTo |
      ts.tournament = t
}

// Fact ensuring unique association between students and tournament scores
fact TournamentAndStudentUnique {
```

```
  associatedStudent =~tournamentScore
}

// Fact ensuring uniqueness of TournamentScore for each student and tournament
//(i.e. each student has only one tournament score per tournament)
fact UniqueTournamentScorePerStudentPerTournament {
  all t: Tournament, s: Student |
    lone { ts: TournamentScore | ts.tournament = t && ts.associatedStudent = s }
}

// Fact ensuring students are subscribed to tournaments they are part of the team in battles
//(i.e. students are subscribed to tournaments they are competing in)
fact LinkSubscribedToTournaments {
  all s: Student, t: Tournament |
    t in s.partOfTeam.isInBattles.PartOfTournament implies t in s.subscribedTo
}

// Fact ensuring teams and students have a unique relationship
fact teamStudentUnique {
  partOfTeam =~students
}

// Fact ensuring scores are correctly associated with teams and battles
//(i.e. scores are associated with the correct team and battle)
fact TeamScoresAssociatedWithBattle {
  all t: Team, s: Score, b: Battle |
      s in t.teamScore && t.isInBattles = b implies s.associatedTeam = t && s.associatedBattle
= b
}

// Fact ensuring students are correctly linked to their scores
//(i.e. students are linked to the correct scores)
fact StudentsLinkedToScores {
  all t: Team, sc: Score, s: Student |
    sc in t.teamScore && s in t.students implies s in sc.studentScore
}

// Fact ensuring teams are correctly linked to battles
fact TeamsLinkedToBattles {
  all s: Score | s.associatedTeam.isInBattles = s.associatedBattle
}

// Fact ensuring unique team scores
//(i.e. each team has only one score)
fact UniqueTeamScores {
  teamScore = ~associatedTeam
}
```

```
// Fact ensuring unique tournament, battle ownership
//(i.e. each tournament, and battle has only one owner)
fact UniqueTournamentBattleOwnership {
  ownTournaments = ~owner
  ownBattle = ~ownedBy
}

// Fact ensuring battles are part of tournaments owned or invited by educators
//(i.e. battles can only be part of the tournament if the owner of the battle is the owner of the tournament or is invited to the tournament)
fact BattlePartOfTournament {
  all b: Battle |
    let t = b.PartOfTournament |
      (b.ownedBy in t.owner) or (b.ownedBy in t.invited)
}

// Fact ensuring each student belongs to only one team per battle
// (i.e. each student can only be part of one team per battle)
fact EachStudentInOneTeamPerBattle {
  all disj t1, t2: Team, s: Student |
    s in t1.students && s in t2.students && t1.isInBattles = t2.isInBattles implies t1 = t2
}
```

4.3. Models

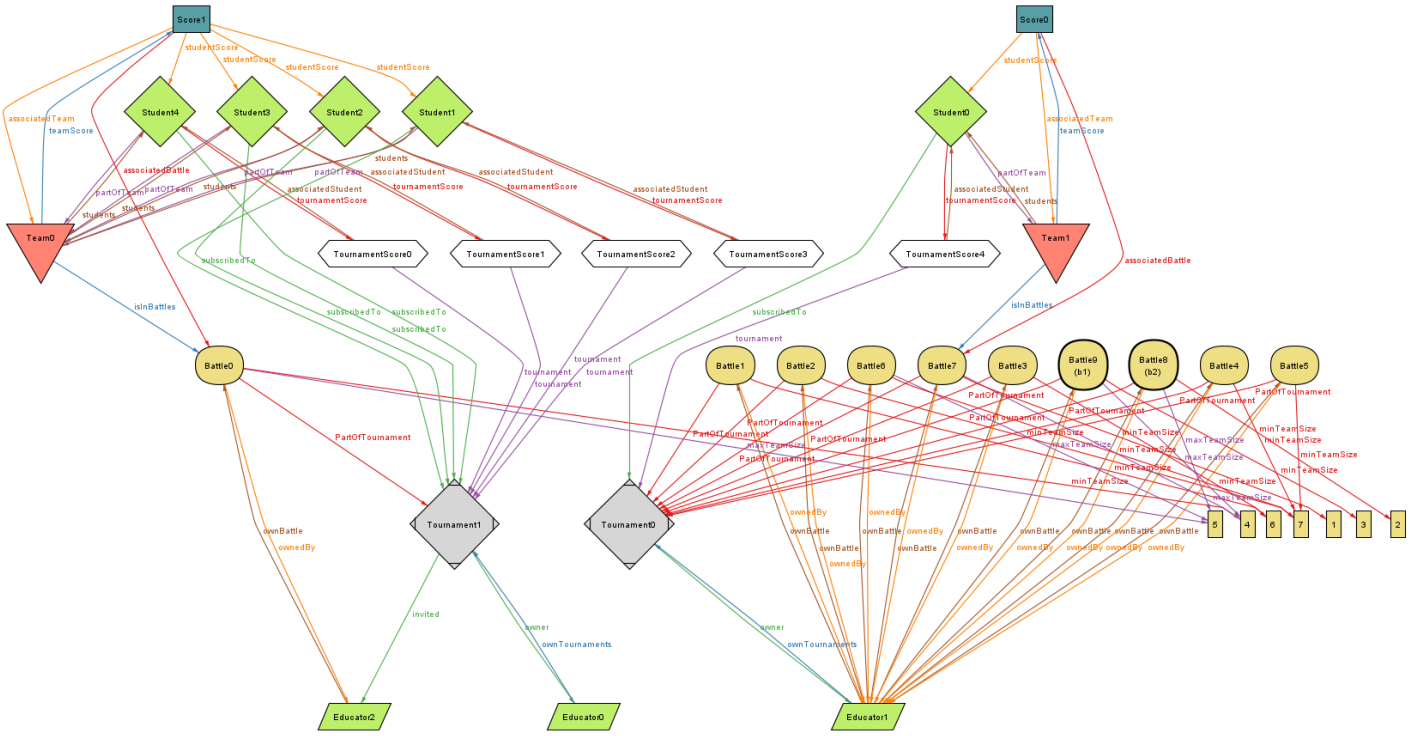
The Models subsection in Alloy involves the formulation of assertions and predicates that represent the system's behavior and properties. This section builds upon the Signatures and Facts, presenting a formalized representation of the CodeKataBattle system's dynamic behavior and functionalities. It encompasses assertions about how the system should behave and predicates that define the desired properties or constraints within the model.

```
// Predicate showing a possible scenario with at least 5 students, 3 tournaments and 4 battles
pred show [] {
  #Battle = 4
  #Tournament = 3
  #Student = 5
}

// Predicate ensuring that there are at least two battles with different team sizes
pred opt1 [] {
    some b1, b2: Battle | b1 != b2 && b1.minTeamSize = 3 && b1.maxTeamSize = 4 && b2.minTeamSize = 2 && b2.maxTeamSize = 5
  #Tournament = 2
```

```
  #Student = 5
}

run show for 10
```

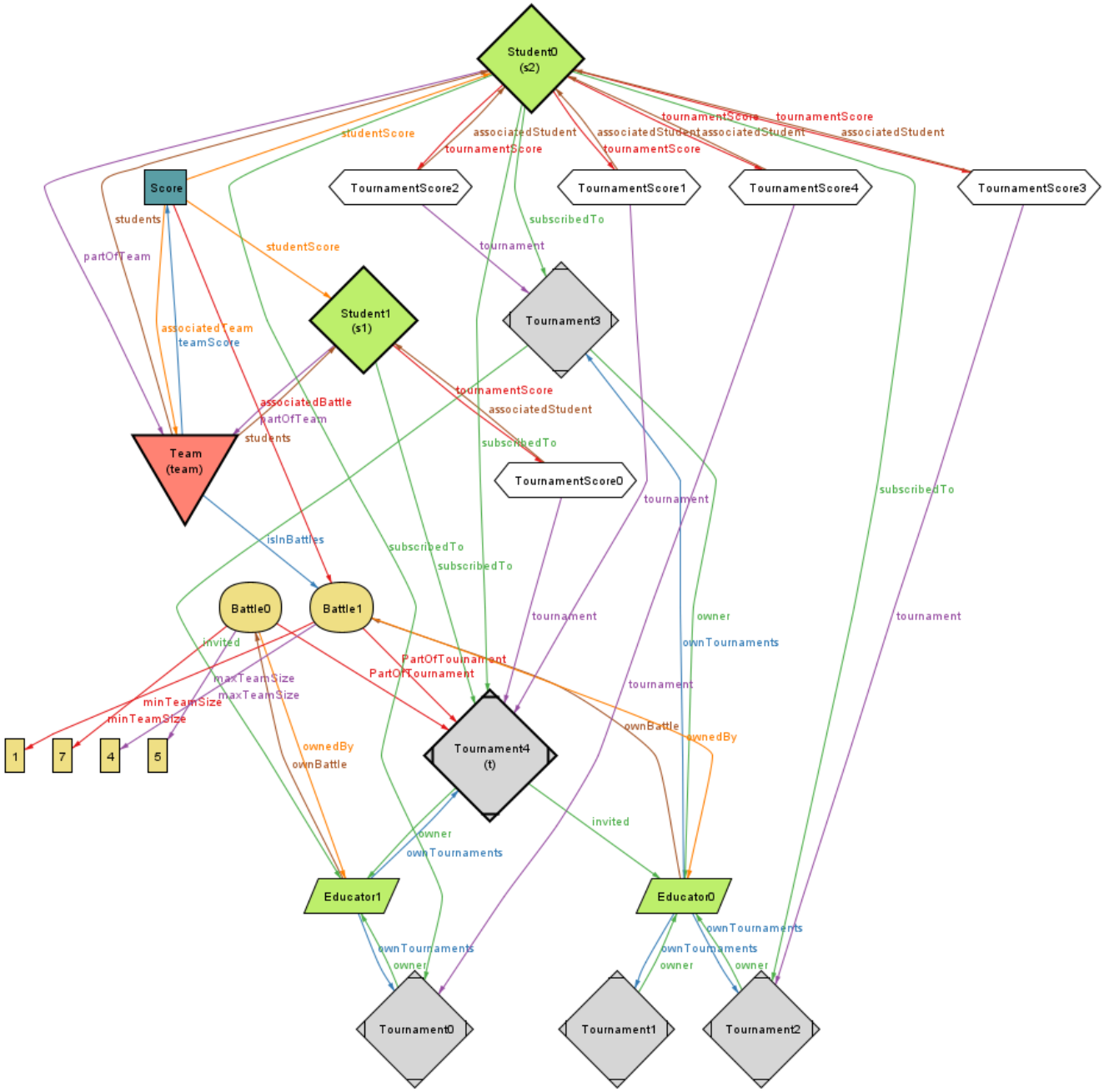


(Figure 14. Alloy Model for opt1)

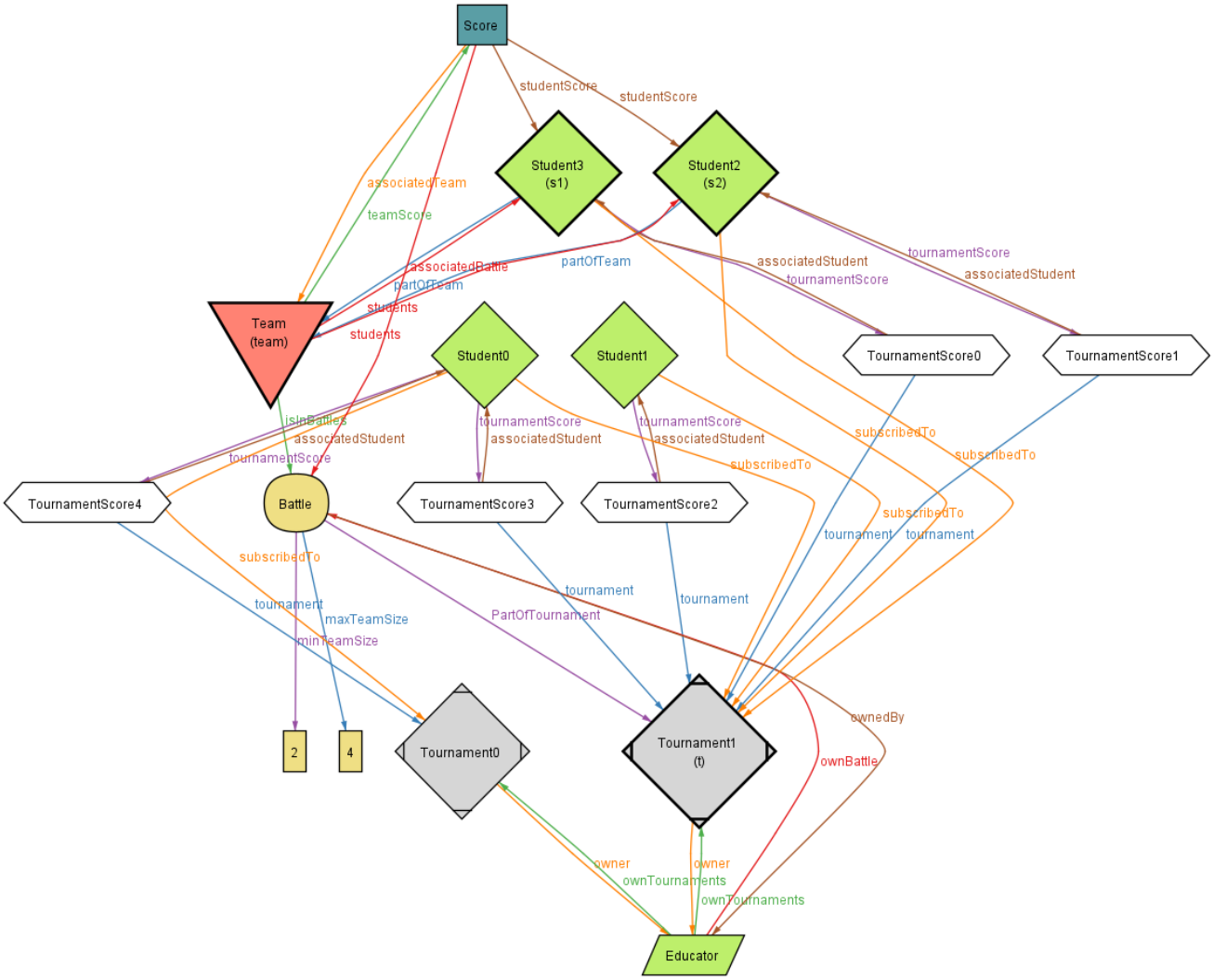
This model provides insight into the interactions and connections within the CKB system concerning Teams, Students, Scores, as well as the associations between Educators, Battles, and Tournaments. It encompasses the configurations of team sizes, their associations with students, and the intricate relationships between various elements within the system.

```
// Assertion proving (by counterexample) that there can be at least two students in the same team in the same tournament
assert NoTwoStudentsConnectedToTournament {
  no t: Tournament |
    some disj s1, s2: Student, team: Team |
                    s1 in team.students && s2 in team.students && s1 != s2 && team.isInBattles.PartOfTournament = t
}

check NoTwoStudentsConnectedToTournament for 5
```



(Figure 15. Alloy Model for NoTwoStudentsConnectedToTournament 1)



(Figure 16. Alloy Model for NoTwoStudentsConnectedToTournament 2)

These models depict scenarios where a student participates in multiple tournaments simultaneously, showcasing the intricate relationships between Tournament Scores and the same student. The first version portrays a student linked to four tournaments, providing a comprehensive yet complex representation. In contrast, the second version simplifies the scenario, illustrating a student's involvement in only two tournaments.

5. EFFORT SPENT

Daniel Mauricio Ruiz Suarez

Chapter	Effort(in hours)
1	6
2	26
3	21
4	4

Sebastian Enrique Perea Lopez

Chapter	Effort(in hours)
1	2
2	3
3	35
4	35

6. REFERENCES

Diagrams:

Lucidchart. (2023). Diagrams. Retrieved from https://www.lucidchart.com

Mockups:

Figma. (2023). Mockups. Retrieved from https://www.figma.com

Alloy Models:

Alloytools. (2023). Alloy models. Retrieved from https://alloytools.org/download.html

Visual Studio Code. (2023). Extension Alloy v0.7.1 by Arash Sahebolamri. Retrieved from https://marketplace.visualstudio.com/items?itemName=ArashSahebolamri.alloy

IEEE Standards:

IEEE. (Year). IEEE Standards. Retrieved from https://grouper.ieee.org/groups/802/22/Documentation/format-rules.html#Document