POLITECNICO
MILANO 1863

RASD

Requirements Analysis and Specification Document

Authors: **Davide Currò**

Matilde Lombardo

Lorenzo Mondo

Date: 22/12/2023

Professor: Elisabetta Di Nitto

Summary

1 Introduction

1.1 Purpose

The platform CodeKataBattle (CKB) has been designed in order to help students improve their software development skills by training with peers on code Kata [1]. A code kata battle is a programming exercise in a programming language of choice. The principal objective of the platform is to provide a dynamic and engaging environment where students can develop new knowledge, improve their skills and learn how to work in a team. The scope of the educator is to challenge and motivate their students. Furthermore, the introduction of gamification elements encourages students to do their best work to strive for excellence. To summarize, the purpose of CodeKataBattle is to create an educational ecosystem that encourages collaboration, competition, and continuous improvement in the field of software development.

1.1.1 Goal

G1. Help students to improve their software skills.
G2. Allow students to subscribe into a specific tournament.
G3. Allow students to see the corresponding tournament rank.
G4. Allow students to be receive a score for the work they have done.
G5. Help students make a co-working environment by creating and joining teams.
G6. Allow teams to participate to the battles.
G7. Allow the educator to create a CKB tournaments and battles.
G8. Allow the educator to authorize other educators to create and manage battles within a specific tournament.
G9. Allow the educator to setup rules for the battles.
G10. Allow manually evaluation by the educator that check and evaluates the work done by Students.
G11. Allow the educator to define badges and rules to reward students based on their performance.

1.2 Scope

The scope of the project aims to satisfy both students and educators needs. The students will use this platform to improve their skills and their knowledge, on the other hand the educator will use it to test the students' learning level. Therefore, this document is focused on their needs since they are the main consumers of the platform, in describing what CodeKataBattle should provide and how to arrange the platform's functions.

The platform assigns scores to teams and creates competition rankings for tournaments. Students can accumulate points based on their performance in various tournaments after an appropriate registration and submission of their final work related to a particular battle of the tournament.

Students work on projects within GitHub repositories, using GitHub Actions to notify the platform every time they commit. The platform automatically checks the code and calculates the scores based on the fulfilment of the requirements of every battle.

The document describes an educational platform dedicated to hands-on training and assessment of students' software development skills through "code kata" challenges, tournaments, and automated and manual assessments.

The platform also integrates gamification elements to motivate students. In fact, since the whole learning process is managed as a tournament where you must win a battle to gain more points or new badges, the students are more motivated to fulfil the challenge assigned. At the same time, they will learn in an unconventional way and perhaps this will help them to learn more easily and quickly.

Moreover, since the tournament is divided into little projects to code, this will help the student to learn theoretical notions in a more practical way.

1.2.1 World Phenomena

World driven shared phenomena:

Identifier	Description
WP1	Students want to participate to CKB.
WP2	The user visits the CKB website.
WP3	Student forks the GitHub repository of the code kata
WP4	Student sets up an automated workflow through GitHub Actions that informs the CKB platform as soon as students push a new commit into the main branch of their repository
WP5	Student writes his own solutions to pass the required tests
WP6	Student pushes a new commit containing his own solution into the main branch of their repository

1.2.2 Shared Phenomena

System driven shared phenomena:

Identifier	Description	Controller
SP1	Registration of students and educators on the platform	W
SP2	Sign up of students and educators	W
SP3	Educator creates a new tournament	W
SP4	Registration of the student in the tournament	W
SP5	Educator creates a battle that belongs to a specific tournament	W
SP6	Educator grant to other colleagues the permission to create battles within the context of a specific tournament by the platform	W
SP7	Educator set minimum and maximum number of students per group for the creation of a specific battle	W
SP8	Educator set a registration and final submission deadline	W
SP9	Students create teams for the battle	W
SP10	Student uses the platform to join a battle on his/her own or as a team	W
SP11	Students codes a solution to pass the required tests.	W
SP12	Group commits the solution in GitHub and the platform downloads the commit to evaluate the solution.	W
SP13	Educator manually evaluates a student	W
SP14	Educator assigns a badge to a student	W
SP15	Educator closes a tournament	W
SP16	The platform notifies student when a new tournament is created	M
SP17	The platform notifies students after they have subscribed.	M
SP18	The platform notifies the students about all upcoming battles created within the tournament they are subscribed at	M
SP19	The platform notifies the students when their final battle rank becomes available	M
SP20	The platform publishes the final rank of the team's scores	M
SP21	The platform shows the profile of a user and the badges he got	M
SP22	When the registration deadline to the battle expires, the platform creates a GitHub repository containing the code kata and sends the link to all students who are member of subscribed teams	M
SP23	The platform notifies the student when a tournament finishes.	M

1.3 Definitions, Acronymous and Abbreviations

1.3.1 Definitions

Definition	Description
Student	A user aspiring to enhance their development skills by utilizing the platform.
Educator	A user utilizing the platform to assess students participating in their own tournaments.

1.3.2 Abbreviations

Abbreviation	Meaning
CKB	CodeKataBattle
RASD	Requirements Analysis and Specification Document
WPX	World Phenomena number X
SPX	Shared Phenomena number X
GX	Goal number X
DX	Domain assumption number X
RX	Requirement number X

1.4 Revision History

- December 22, 2023: version 1.0

1.5 Reference Document

This document refers to:

- The specification document: “Assignment RDD AY 2023-2024.pdf”;
- Slides of Software Engineering 2 course on WeBeep;

1.6 Document Structure

The document is structured in different sections.

The **first section** introduces the goals of the system we are describing. It explains how the system works and what it’s going to achieve with its functionalities. There are lists of World and Shared phenomena in order to describe what the system is responsible for, and what depends on the world around.

The **second section** describes the possible scenarios that could happen while using the system and how the platform will behave. Furthermore, there is the class diagram, which states which are the major entities in the system, how they relate to each other and how they interact. The statecharts describe in a graphical way the different states that the entity entries into while performing a specific set of actions. In addition, in this section are listed all the functions that the system provides to the users and lastly who are the users that will use this platform.

The **third section** is dedicated to the explanation of the requirements that must be implemented to achieve the goal established in the first chapter of the document. It contains the sequence diagrams as a description of the several use cases described to explain the functions of the system and its behavior in the various scenarios. In addition, it contains details about the hardware and software interfaces and Software System Attributes.

The **fourth section** is dedicated to the formal analysis of the system using Alloy.

The **last sections** show the total effort spent by each member of the team of the group to complete this document and all the references used to write the document.

2. Overall Description

2.1 Production Perspective

2.1.1 Scenarios

1. **User Registration**
   The student Davide wants to improve their software development skills using a platform called CodeKataBattle. Using his personal device, he opens the platform and decides to sign up as Student. He signs up on the platform, fills up the registration form with the required personal information and creates an account. In the future, if the user wants, he can access his personal information that he wrote during the registration process and update them with the newest one.

2. **Creation of the tournament**
   The educator Gianluca wants to organize a new competition. He signs in the platform; he creates a new tournament by clicking on the apposite button on the home page. Afterwards, he is redirected to the view where it is possible to setup the tournament options by choosing for example the name of the tournament, the description, the deadlines dates for the registration, the max number of participants, etc. When Gianluca is sure about the options he has chosen, he clicks on the confirmation button and the competition is created, the notification is sent to all the student of the CKB platform who are now able to register themselves to the new tournament.

3. **Student registers himself to a new tournament**
   After receiving the notification, student Davide expresses a desire to participate in the tournament. To do so, he searches for interesting tournaments in his notification pool. Reading brief descriptions of each tournament, Davide finds one that particularly interests him. He clicks on it and is redirected to the tournament's registration page. Here, Davide can view a more detailed overview of the tournament, and if the registration deadline has not expired, he decides to sign up for the competition by clicking the confirmation button. Finally, Davide is successfully registered for the new tournament.

4. **Creation of a new battle in the tournament**
   In the context of organizing a code kata battle, an educator John with the necessary permissions logs into the CodeKataBattle (CKB) platform. While navigating through the list of tournaments he is involved in, John selects a specific tournament. He then initiates the creation of a new battle within the chosen tournament, providing details such as code kata description, project specifics, group size constraints, registration and submission deadlines, and additional scoring configurations.

   Once confirmed, the CKB platform notifies all students subscribed to the tournament about the newly created battle. Students, in response, form teams based on the provided details. When the registration deadline expires, the CKB platform generates a GitHub repository for the code kata and shares the link with all registered teams.

5. **Student participates to a new battle and sets up the environment to push his solutions.**
   Lorenzo, a student eager to enhance his programming skills and enjoy a competitive environment, explores the CodeKataBattle (CKB) platform to find exciting challenges. Upon logging into the CKB platform with his student credentials, Lorenzo navigates to his personalized dashboard, where he can view a list of ongoing tournaments he has previously subscribed to.
   Lorenzo identifies a specific tournament and notices a new battle within that tournament that piques his interest. Before the registration deadline for the battle expires, Lorenzo decides to participate.
   He has three options: he can either register as a solo participant, create a team, or accept an invitation to join an existing team. After making his choice, Lorenzo successfully registers for the battle within the specified deadline.
   Upon the registration deadline's expiration, the CKB platform creates a GitHub repository for the battle and notifies Lorenzo about its availability.
   Lorenzo to contribute to the battle, forks the GitHub repository to obtain his own copy. In preparation for submitting his solutions, Lorenzo sets up GitHub Actions within his forked repository. The GitHub Actions are configured to trigger the CKB platform through appropriate API calls whenever Lorenzo pushes new commits to the main branch. Lorenzo is now correctly participating to the battle and he's able to submit his own solutions.

6. **Submission of the solution in the platform.**
   Matilde has just finished developing her solution and has completed the task assigned by the professor and now she wants to submit it in order to be evaluated. To submit her solution, Matilde uses GitHub and navigates in the repository related to the battle and pushes her solution in the main branch of such repository. The commit automatically triggers the CKB platform that, starting from the code written by Matilde, executes the test cases created by the educator. Matilde will then be automatically evaluated based on the test cases she has passed with her solution.

7. **Manual evaluation of the project.**
   The educator wants to manually evaluate a project since he wants to personally check some aspects of the work done by the students. He logs into the platform, access the group solution, and reviews a particular part of the project he wants to check. His final score can be different than the automatic one. He assigns the new score to the team and then the new scores are added to the system and impact the final ranking.

8. **A student wants to see his position in the ranking of the competition.**
   He logs into the platform and views the updated rankings. He can see his own position and the ranking of the other students in the competition. The rankings are available once the battle is over, after that the platform automatically evaluates the teams and assigns them the score. After the possible manual evaluation of the educator the rankings are final and published in the platform.

9. **Assignment of the badges to the students**
   Educator wants to reward a student for the works done. Therefore, he creates the badges and sets the rules to be respected in order to gain them. Based on what achieved by the students, the platform assigns the badge to them. When accessing to the student account everyone can see which badges the student has collected.

2.1.2 Class Diagram

This is the class diagram that describes the entire system. The most important entities described in the diagram are:

- **Student**: It represents the student that wants to participate in a tournament, it has a Name, Surname, Email.
- **Tournament**: It's created by the educator, and it's composed by a set of battles and participants register to it.
- **CodeKataBattle**: It's the platform system that manages all the tournaments.
- **Educator**: It's the teacher who creates the tournament, can assign scores manually and creates the rules for gaining the badges.
- **Battle**: It's the main part of the diagram, it has his deadlines for the registration and submission. Each battle has his teams associated to and the score of the team depends on its result.
- **TestCase**: They are the different sets of test cases created from the educator that the students need to pass in order to gain points into the battles.
- **Team**: It's the entity that is created after the battle is generated. It's composed of some students and has his associated score.
- **Score**: It can be automatically or manually generated, and it depends on a different set of characteristics of the solution of the battle problem.
- **Badge**: It's the entity associated to a user. A user wins a badge only if he complete some specific tasks.
- **Rule**: They are written by the educator. They are the requirements to achieve in order to win a badge.

The majority of the relations in the diagram are compositions, thus the entity cannot exist without the other one that composes it, e.g. the tournament can't exist without the entity educator which created it.

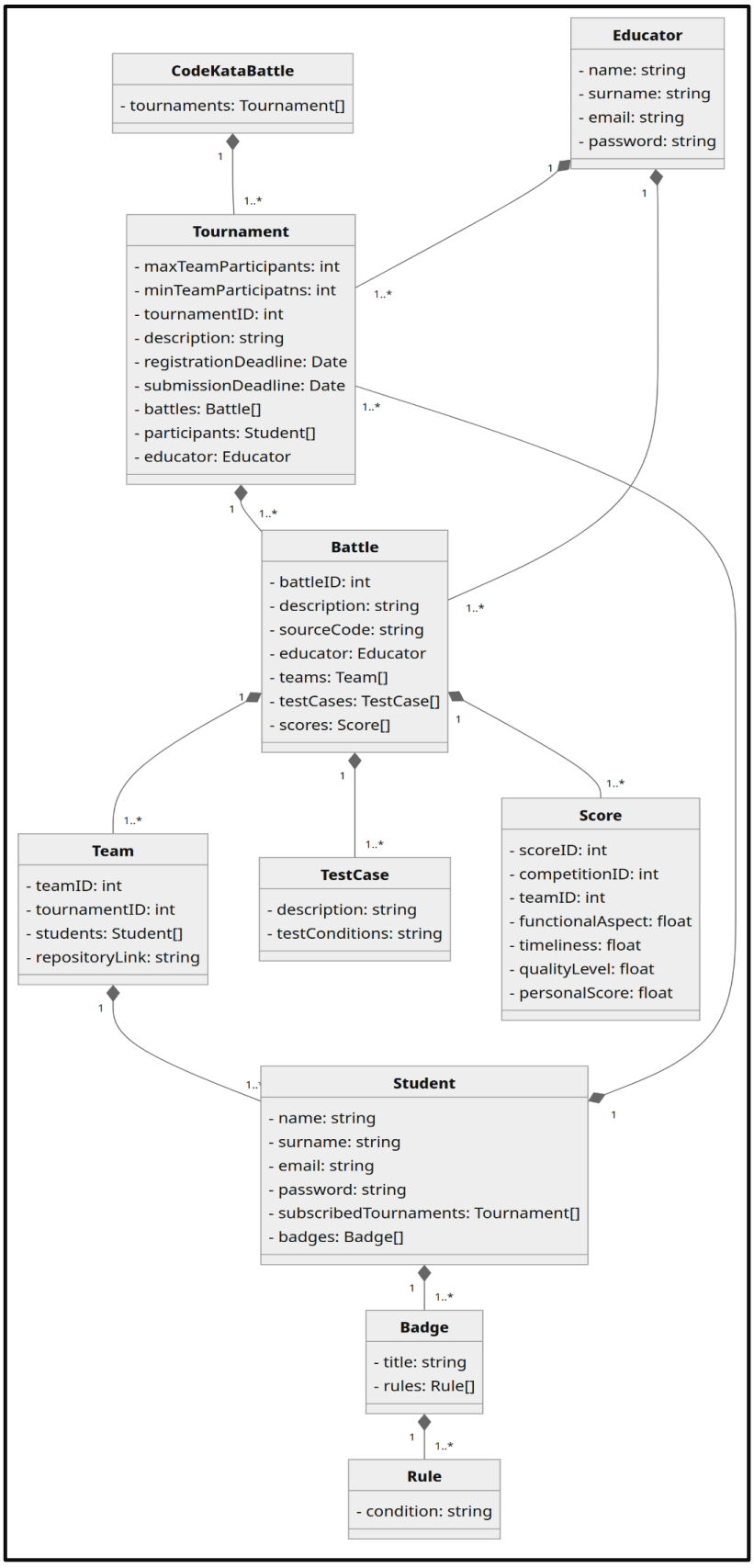


Figure 1 - Class diagram representing the Code Kata Battle system.

2.1.3 Statecharts

This section of the document describes some aspect of the system. It describes the states that some of the most important entities can assume. The examples that are missing were left out due to their simplicity.

The statechart in Figure 1 represents the **registration workflow for new users**, whether they are Students or Educators, on the CodeKataBattle Platform. The StartingView serves as the entry point, accessible from an existing link (i.e codekatabattle.com). This view allows users to choose between registering as a Student or Educator and is considered the initial state in the statechart.

Upon clicking the register button on the StartingView, the user is directed to the RegisterView, presenting a form for data input. After the user fills in the required information and clicks the submit button on the RegisterView, the platform transitions to the CheckData state.

In the CheckData state, the inserted data undergoes verification. If the data is deemed valid, the platform advances to the HomePage state, signifying successful registration. Conversely, if there are issues with the provided data, the platform reverts to the RegisterView state, allowing the user to correct and resubmit the information.

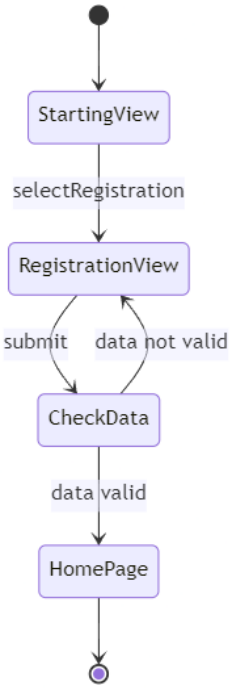


Figure 2: The diagram shows the main steps of the registration process of the user (Student or Educator)

The statechart in Figure 2 outlines the workflow for **user login** on the CodeKataBattle Platform, mirroring the structure of the registration process. Initiated from the StartingView, users can opt to log in as a Student or Educator, considering this as the starting state in the statechart.

When the user clicks on the login button in the StartingView, they are redirected to the LoginView, where a form is provided for entering login credentials. Upon submitting the login data through the submit button on the LoginView, the platform transitions to the CheckLogin state.

In the CheckLogin state, the entered credentials undergo verification. If the login is successful and the credentials are valid, the platform transitions to the HomePage state, indicating a successful login. However, if there are issues with the provided credentials, the platform returns to the LoginView state, allowing the user to rectify and resubmit the information.

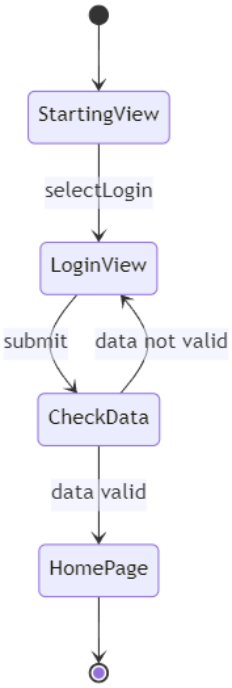


Figure 3: The diagram shows the main steps of the login process of the user (Student or Educator)

The state diagram in Figure 3 captures the **student** journey within the CodeKataBattle Platform, focusing on **tournament exploration, battle selection, team creation, and participation**. The initial state represents the entry point leading to the HomePage.

From the HomePage, users can initiate a search for tournaments, transitioning to the ListOfAvailableTournaments state. In this state, users can explore available tournaments and choose one. If there are no available tournaments, the system returns to the HomePage.

Upon selecting a tournament, the platform moves to the ListOfBattles state, presenting users with a list of battles associated with the chosen tournament. Users can then select a specific battle, moving to the TeamCreation state.

In TeamCreation, users can submit their team for the selected battle, and the system transitions to IntoTheBattle. Here, participants can submit their solutions to the programming challenges. The system remains in the IntoTheBattle state until the deadline for submissions expires, at which point it moves to the EndBattle state.

In EndBattle, the system evaluates the submissions and presents the ranking view. If there are more battles in the tournament, the system returns to the ListOfBattles state, allowing users to select another battle. If it's the last battle, the system transitions to the final state, indicating the end of the tournament journey.

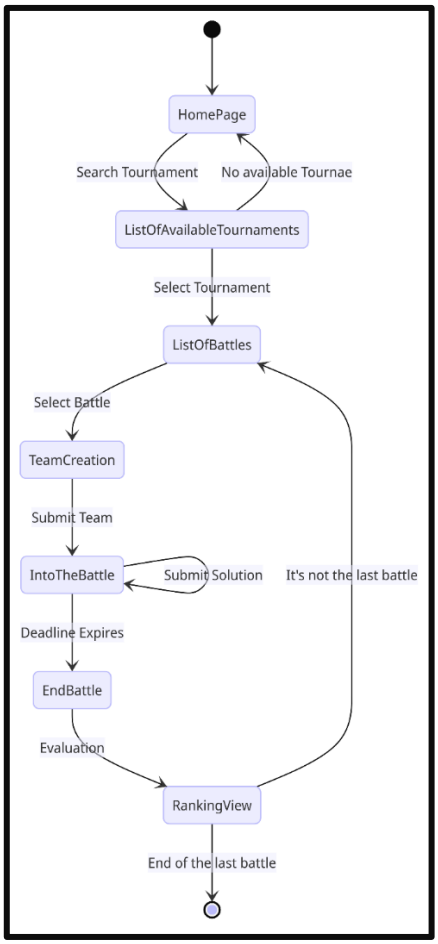


Figure 4: This diagram shows the states the entity of the student can assume when participating in a tournament and a battle.

The diagram in Figure 4 delineates the **educator**'s journey within the CodeKataBattle platform, encompassing the **creation of tournaments, including the setup of battles within the tournament, and culminating in the conclusion of the tournament**.

The *EducatorHomePage* serves as the main hub for educators upon logging into the platform. Here, they can create a new tournament by clicking a dedicated button, initiating a transition to the *CreateTournamentView* state.

In the *CreateTournamentView* state, the platform provides a tailored space for tournament creation. Educators can meticulously configure settings. Upon completion, pressing the "Create Tournament" button triggers the tournament's creation. All subscribed students receive notifications, and the state transitions to *TournamentWaitingView*.

During *TournamentWaitingView*, students receive notifications, enabling their participation throughout the registration period. This state persists until the registration deadline, set by the educator in the previous state. Upon deadline expiration, the tournament shifts to *TournamentManagementView*.

In the *TournamentManagementView* state, accessible only by the creating educator, they can:

- Create new battles within the tournament.
- View the leaderboard.
- Grant permissions to other users for battle creation.
- Monitor the progress of ongoing battles or review results of completed ones.

For the proper tournament workflow, it's crucial to create at least one battle. From *TournamentManagementView*, where tournament management occurs, pressing the "Create new battle" button transitions to *CreateNewBattleView*.

CreateNewBattleView allows the educator to set parameters for a new battle, such as the maximum team size and deadlines. Pressing the "Create Battle" button initiates the creation of the battle, moving the state to *BattleRegistration*.

In *BattleRegistration*, the battle enters a waiting state, allowing students to register. After the registration deadline, the state shifts to *BattleIsGoingOn*.

During *BattleIsGoingOn*, students submit their solutions within the stipulated time frame. Upon the expiration of the submission deadline, the state transitions to *EndBattle*.

In EndBattle, students cannot submit new solutions. The platform performs automatic evaluation, and educators can view the battle ranking, provide manual evaluations, award badges for specific achievements, and decide whether to conclude the tournament or continue with a new battle transitioning again back to the *TorunamentManagementView*.

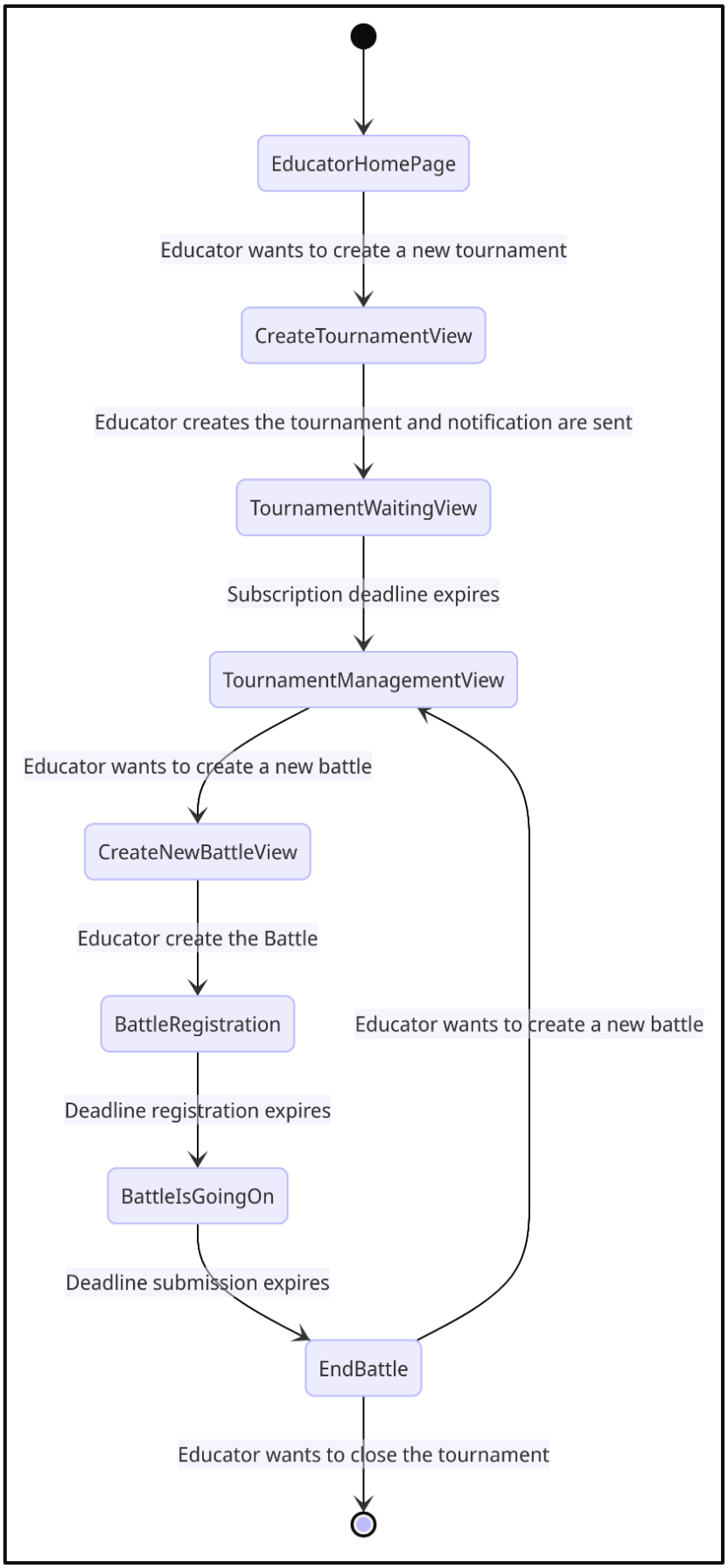


Figure 5: This diagram shows the steps of the entity educator while creating and managing tournament.

The state chart in Figure 5 encapsulates the sequential steps undertaken by an educator in the process of creating, defining rules, and assigning a badge within the CodeKataBattle Platform.

1. **CreateBadge State:**
    - **EnterDetails:** At the outset, the educator enters essential details for the badge, such as the name and description.
    - **ChooseIcon:** Subsequently, the educator selects an icon for the badge, with the option to change it.
    - **SaveDetails:** Upon finalizing the details, the educator saves the badge information.
2. **WriteRules State:**
    - **DefineRules:** In this state, the educator outlines the rules and criteria for earning the badge.
    - **SaveRules:** Once the criteria are defined, the educator saves the rules for the badge.
3. **AssignBadge State:**
    - **SelectStudent:** The educator begins the badge assignment process by selecting a specific student.
    - **ConfirmAssignment:** After choosing the student, the educator confirms the badge assignment.

The state chart concludes with the successful assignment of the badge to the chosen student.

This state chart provides a clear visualization of the educator's workflow, starting from badge creation and rule definition to the final step of assigning the badge to a student. It delineates a structured process for educators to manage badges on the CodeKataBattle Platform.

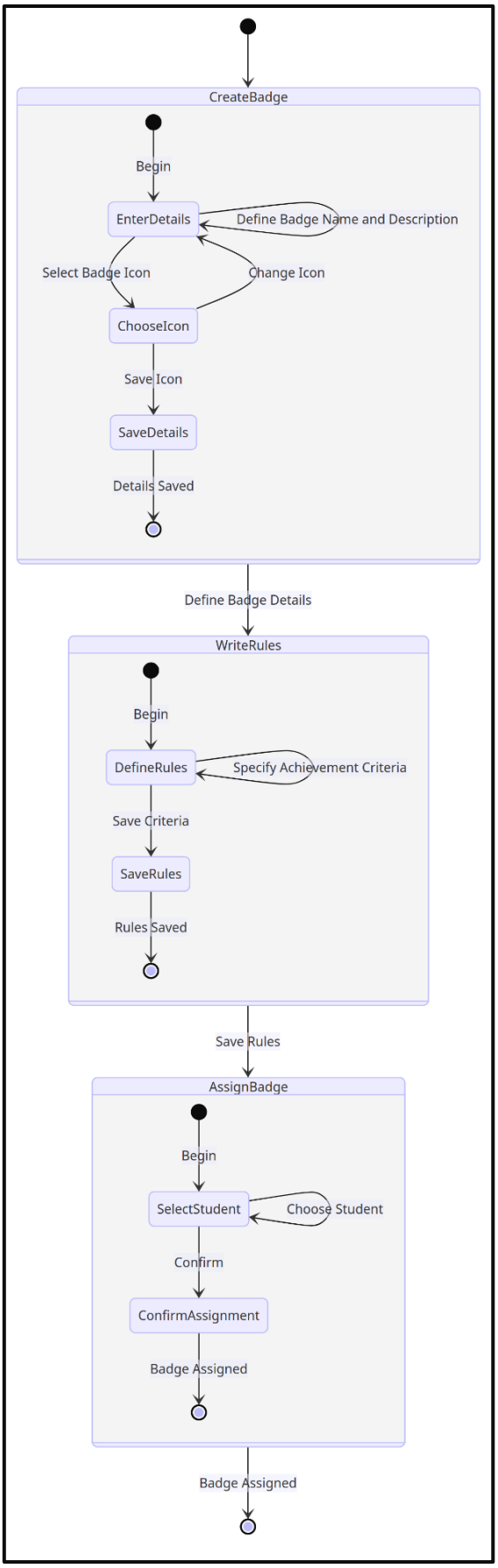


Figure 6: The diagram above shows the steps of the entity educator when creating a badge, writing the rules for gaining it and assigning it to a student.

2.2 Product Functions

In this section the main functionalities of CodeKataBattle are presented and described in more detail.

2.2.1 Management of the Tournament and Battles

One of the most important functions of CodeKataBattle is definitely the creation of the tournament. The system allows the user authenticated as an Educator, after an appropriate registration and login, to create a tournament and define the various battles in it. On the other hand, after the sign up and login in, the user Student is allowed to participate in a tournament and create teams for the battles. The system must manage these functionalities in order to avoid issues, such as having the same student registered multiple times in the same battle, or not giving the students the possibility to create tournaments and battles. It ensures that the deadlines defined by the educator are valid and gives the educators the possibility to invite other teachers to create new battles within the same tournament. The system notifies the students when a new tournament has been created or when there is a new battle incoming.

2.2.2 Assignment of Badges

The CKB platform also includes gamification badges. The platform allows the Educator to create and define rewards that represent the achievements of individual students. The badge is defined when the educator creates a tournament and each badge has a title (e.g. “top committer”) and one or more rules (i.e., simple Boolean properties) that must be fulfilled to achieve the reward. At the end of the tournament, the system checks the score of each student and assigns to one or more students the badge (depending on the skills reached). Finally, both student and educator can see collected badges when they visualize the profile of a student. The system ensures that the student doesn’t receive the same badge multiple times.

2.2.3 Manual and Automatic Evaluation

After the students submit their solutions and the educator closes the battle, the system automatically starts evaluating their final work. The evaluation is made with respect of determined requirements defined by the educator, which are called TestCases, and also with respect to the results of a static analyzer tool that extracts and evaluates the quality level of the sources considering multiple aspects such as security, reliability and mantainability. The system starts testing the solutions of the student, checking if they pass the test cases required. It gives each team a score, which can be temporary if the tournament isn’t over, or final if the tournament has been closed by teacher. The scores will be shown by the system in a ranking table, and each student will be able to see his points in the personal account. The system allows educators to manually evaluate solutions, but this option can only be activated during the battle's creation. In the event that manual evaluation has been set, when the submission deadline is reached, the battle enters the consolidation stage. During this stage, the educator can manually adjust the scores assigned by the platform. This provides the educator with an opportunity to personally review and assess the project submitted by the student, potentially modifying the score originally determined by the system. The system ensures that the scores given will be between 0 and 100 points, and that the student will receive the points only when the battle is over and only one time per battle. It will ensure that the scores will be always available to check.

2.2.4 Communication with GitHub

The connection between the CodeKataBattle (CKB) platform and GitHub is essential for the evaluation of student solutions during a battle. After completing all registration steps of the battle, the platform creates a GitHub repository containing the code kata and then sends the link to all students' members of subscribed team. Afterwards, the students will set up an automated workflow through GitHub Actions, which will inform the CKB platform, through proper API calls, each time the students will push new commit into the main branch of the repository. Each instance of pushing code prior to the deadline prompts the CodeKataBattle (CKB) platform. The platform, in response, fetches the most recent source code, conducts an analysis, and executes tests on the relevant executable files. Subsequently, it computes and refreshes the team's battle score.

2.3 User Characteristics

2.3.1 Student

The student is a user who wants to access and use the services of the CKB platform. The user must register into the system through valid credentials. At the moment of the registration, the CKB platform will check if the data inserted is valid and notifies the student whether the registration is completed, or if some data needs to be re-inserted.

2.3.2 Educator

An educator is a user who registers on the platform with his owns personal data. This user has the goal of organizing tournaments and battles. The educator can delegate authorization to other colleagues for the creation of battles, additionally, the educator is responsible for defining reward in the form of badges. At the end of each tournament, the educator awards badge to students based on their performances.

2.4 Assumptions, Dependencies and Constraints

D1. The users of the CKB platform are assumed to have reliable and consistent internet connectivity.

D2. The assumption that users have valid and secure credentials for authentication on the platform.

D3. The platform assumes a suitable environment for running and evaluating code, including necessary dependencies and testing frameworks.

D4. The students have a basic understanding of programming concepts and practices.

D5. The platform assumes support for a variety of programming languages, allowing students to choose their preferred language for solving code kata.

D6. There exists a stable integration with GitHub, allowing the creation of repositories and the interaction with GitHub Actions.

D7. Educators have the authority to create tournaments, battles and setup rules.

D8. All students must be able to receive each other's notifications.

D9. There exists an external service that provide security measures to protect user data and prevent unauthorized access.

D10. Assume that all users already have a working GitHub account.

D11. Students are expected to participate fairly, adhering to the rules set by educators.

D12. Educators fulfil responsibilities such as manual evaluations, badge creation and tournament closure.

3 Specific Requirements

3.1 External Interfaces

3.1.1 User Interfaces

The CodeKataBattle must be a web application available for every kind of user. It also must be easy to use and intuitive, and for this reason, is important to consider some crucial and specific aspects.

The web application's first page will offer the user the opportunity to register with your credentials. After the registration phases, based on the user type, the main stages of CodeKataBattle will be available. Both users can connect to their GitHub account through an insertion on the homepage, thereby having the environment ready to participate in and create battles.

If you log in as a student, you will see a screen displaying the list of all available tournaments, with the option to choose the battle you want to participate in. Subsequently, the platform will allow you to create/join a team and start the battle.

In the meantime, if logged in as an Educator, the platform will provide a section within the page for a comprehensive setup of tournaments and battles. This involves managing various details in-depth, such as registration deadline, the minimum and maximum number of students, additional configurations for scoring and other tournament-related information.

Another important aspect of CKB platform is the notification management. When the users register themselves on the platform or a battle is created, they are immediately notified via a pop-up. This way, the users can click on the pop-up notification and view all the information within it.

When the team commits on GitHub a new solution, the system automatically evaluates it and then at the end of the battle, it makes available the final ranking score of all the teams on the screen. Furthermore, the student can then see his updated score on his personal homepage, as well as the badges he has received.

3.1.2 Hardware Interfaces

Some key aspects to consider are:

- **Access Devices**: the users need a laptop with IOS or Windows installed and an input device such us mice and keyboards. Users must also be able to properly view the platform's output, including notification messages, rankings, and battle information.

- **Internet Connectivity**: users need to have access to a stable internet connection to interact with the platform, as many features require an online connection.

3.1.3 Software Interfaces

In order to implement the software interface in the CKB platform, several key elements are essential. For example, is important to have in the system a robust backend APIs (Application Programming Interfaces) that are necessary to enable communication between the user interfaces and the underlying system components. In the CKB platform, the use of GitHub APIs allows for a deeper integration with the software development ecosystem on GitHub, simplifying code management, test execution, and the assessment of students' performance.

3.2 Functional Requirements

Requirements	Description
R1	The system shall allow user to register a personal account.
R2	The system shall allow users to login.
R3	The system shall allow users to modify personal information into account.
R4	The system must check the validity of the information inserted by the users.
R5	The system shall allow the Educator to create tournaments.
R6	The system shall allow the Educator to setup deadlines for the tournament.
R7	The system shall allow the Educator to create battles.
R8	The system shall allow the Educator to grant to other colleagues the permission to create battle.
R9	The system shall allow the students to view the list of available tournaments.
R10	The system shall allow the students to subscribe for a tournament.
R11	The system shall allow the students to view the list of battles.
R12	The system shall allow the students to create/join a team.
R13	The system shall allow the students to select a battle.
R14	The system shall allow the team to participate in the battle.
R15	The system must create a GitHub repository containing the code kata.
R16	The system should send the link of the GitHub repository to all student who are members of subscribed team.
R17	The system must pull and analyse the latest source from GitHub repo.
R18	The system must run the tests on the corresponding executables.
R19	The system must be able to automatically evaluate the job done by the students.
R20	If manual evaluation is required (by the Educator), the system shall allow the Educator to check and evaluate the work done by students.
R21	The system must update the battle score of a team as soon as new push action on GitHub are performed.
R22	The system shall allow the users to see the current rank evolving during the battle.
R23	If manual evaluation is required (by the Educator), the system shall allow the Educator to through the sources produced by each team.
R24	At the end of each battle, the system must update the personal tournament score of each student.
R25	The system shall allow the Educator to create the badge and rules to achieve a badge in the Tournament.
R26	The system must automatically assign the badges at the end of each tournament.
R27	The system shall allow the users to visualize the badges in their profile.
R28	The system must notify students registered on the platform.
R29	The system must notify students every time a battle is created within the tournament.
R30	The system must notify students when the battle registration time is about to expire.
R31	The system must notify students when the final submission is about to expire.
R32	The system must notify users when the final battle rank becomes available.
R33	The system must notify students involved in a tournament when the educator closes that specific tournament.

3.2.1 Use Case Diagram

Student Use Case Diagram

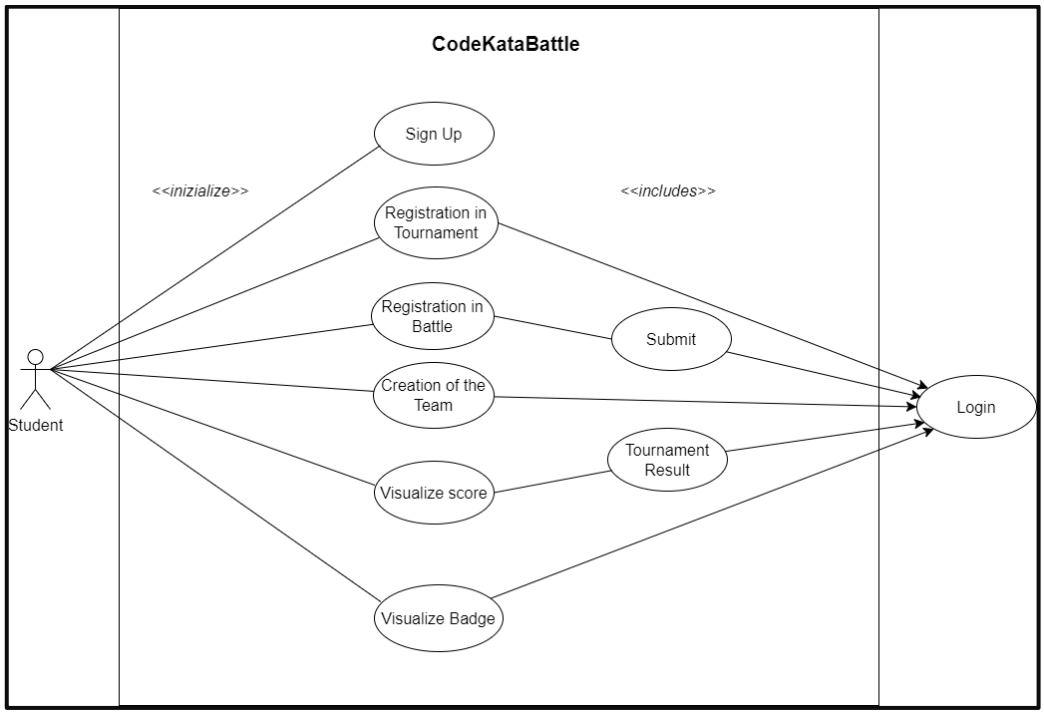


Figure 7: Use case diagram for a student.

Educator Use Case Diagram

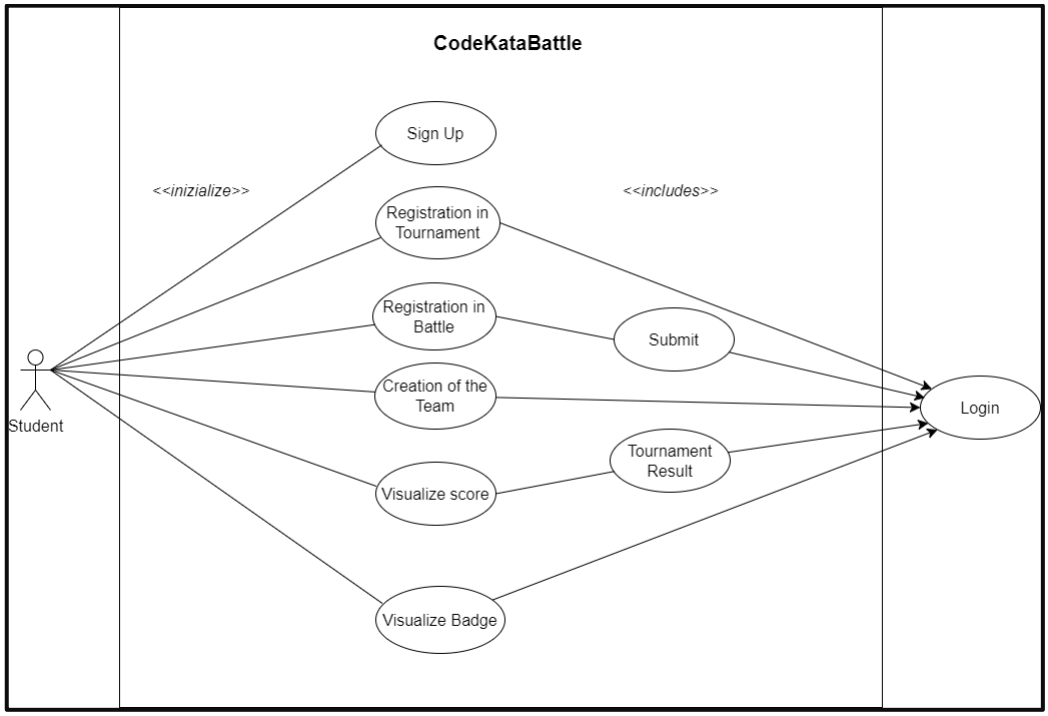


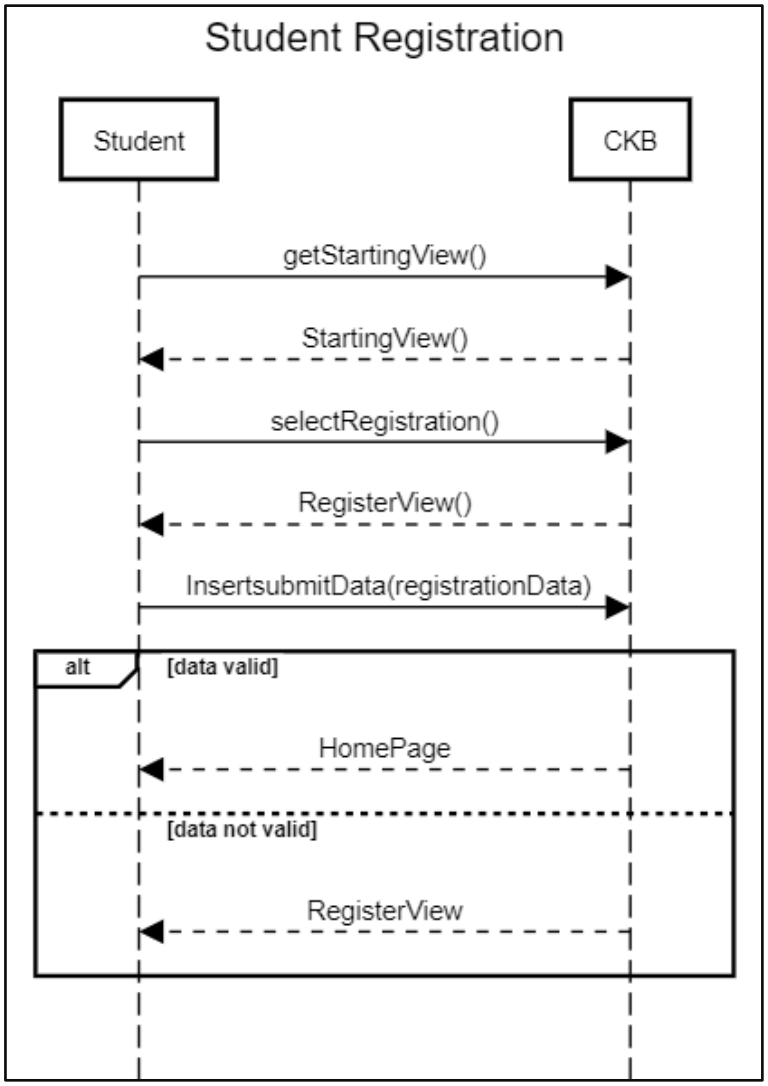
Figure 8: Use case Diagram for the educator.

Sequence diagram and Use Cases

In this section we report the description of all the major use cases of the system, as well as all the related sequence diagrams, describing all the calls and requests that take place for every action.

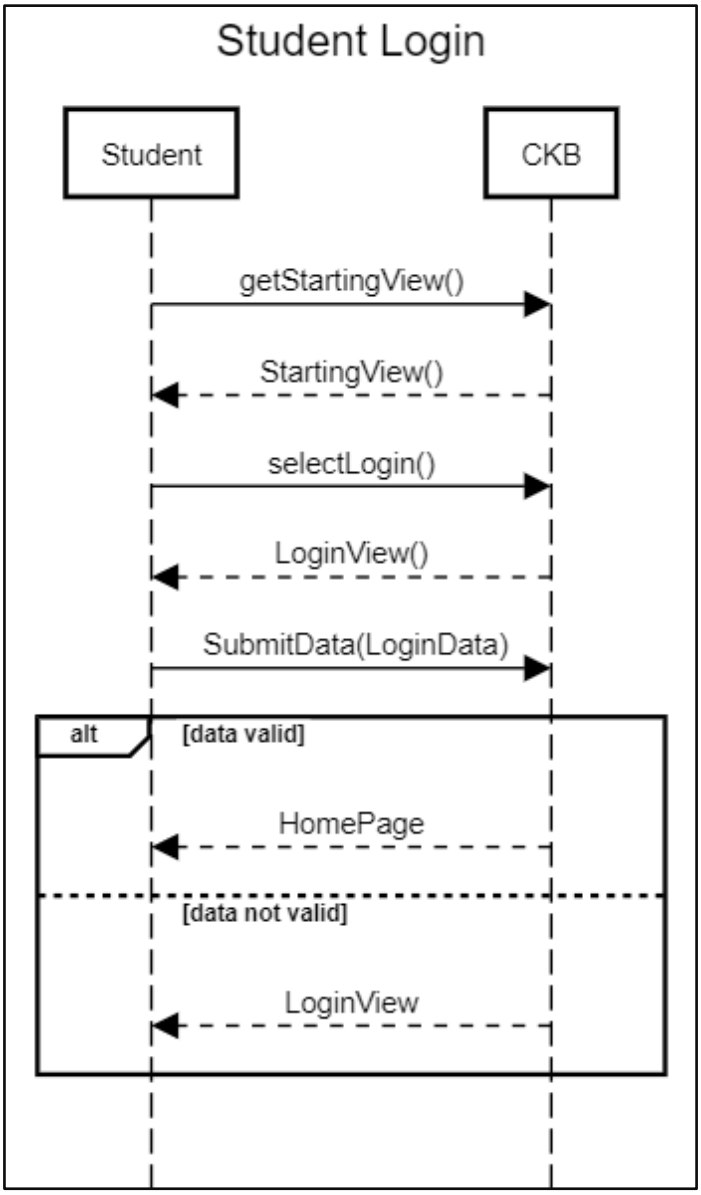
1. UC1 – Student Registration

Name	Sign Up
Actor	Student
Entry Condition	The user doesn't have an account and is on the login view
Event Flow	1. The user clicks on the "Sign up" button. 2. The user fills up the fields required, inserting name, surname, username, date of birth, password, e-mail address. 3. The system checks whether the same credentials have been already used. 4. The system saves the inserted information. 5. The system sends a confirmation email. 6. The student clicks on the link in the email for confirmation. 7. The system shows the home page.
Exit Condition	The user registration is successful, and the student's account is created
Exceptions	1. The student didn't insert valid credentials. 2. The date of birth isn't valid. 3. The student doesn't click on the confirmation link. 4. The student was already registered with the same email address. 5. The fields "Password" and "Confirm Password" don't match.



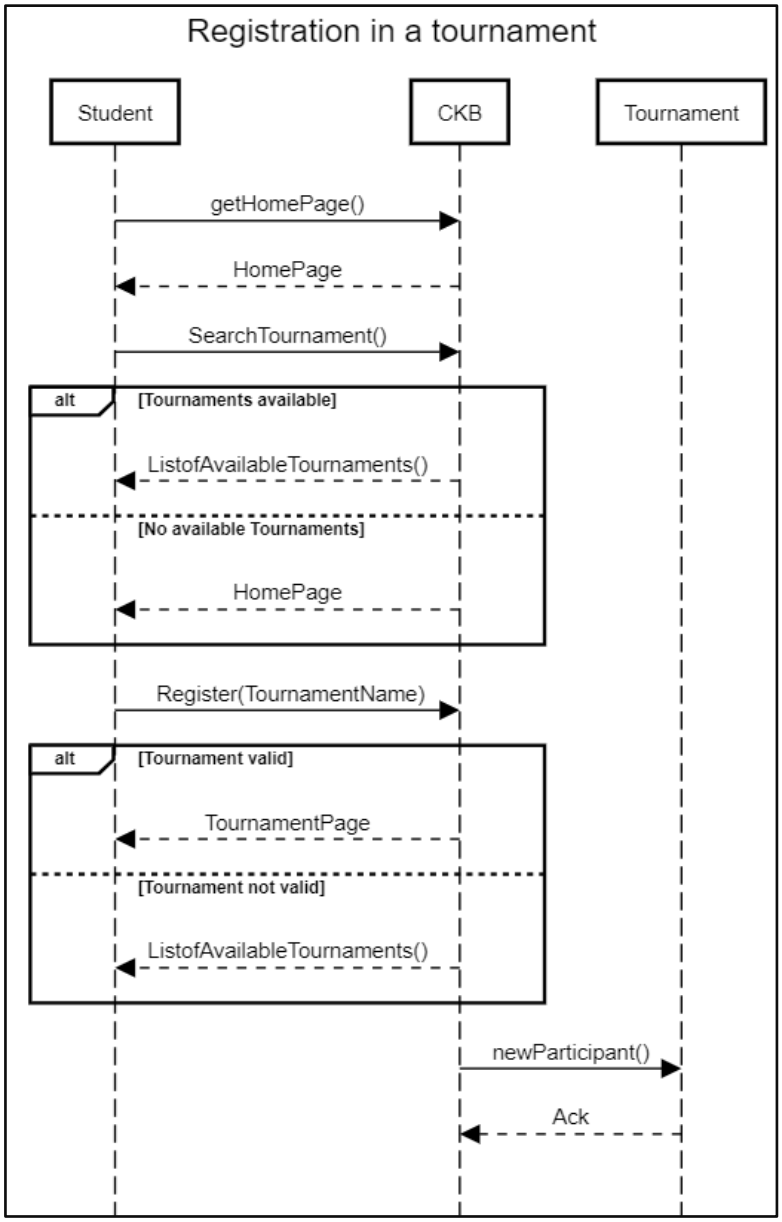
2. UC2 – Student Login

Name	Student Login
Actor	Student
Entry Condition	The user is registered on the platform, but he isn't logged in yet and is on the login view
Event Flow	1. The user selects the "Login" button. 2. The user inserts the username and password. 3. The system acquires these data and verifies whether they are correct or not. 4. The system shows to the user the home page.
Exit Condition	The user is logged into the system
Exceptions	1. The user doesn't insert the correct credentials. 2. The user's account doesn't exist. 3. The user is not registered yet. 4. The user doesn't insert his credentials before hitting the button to login.



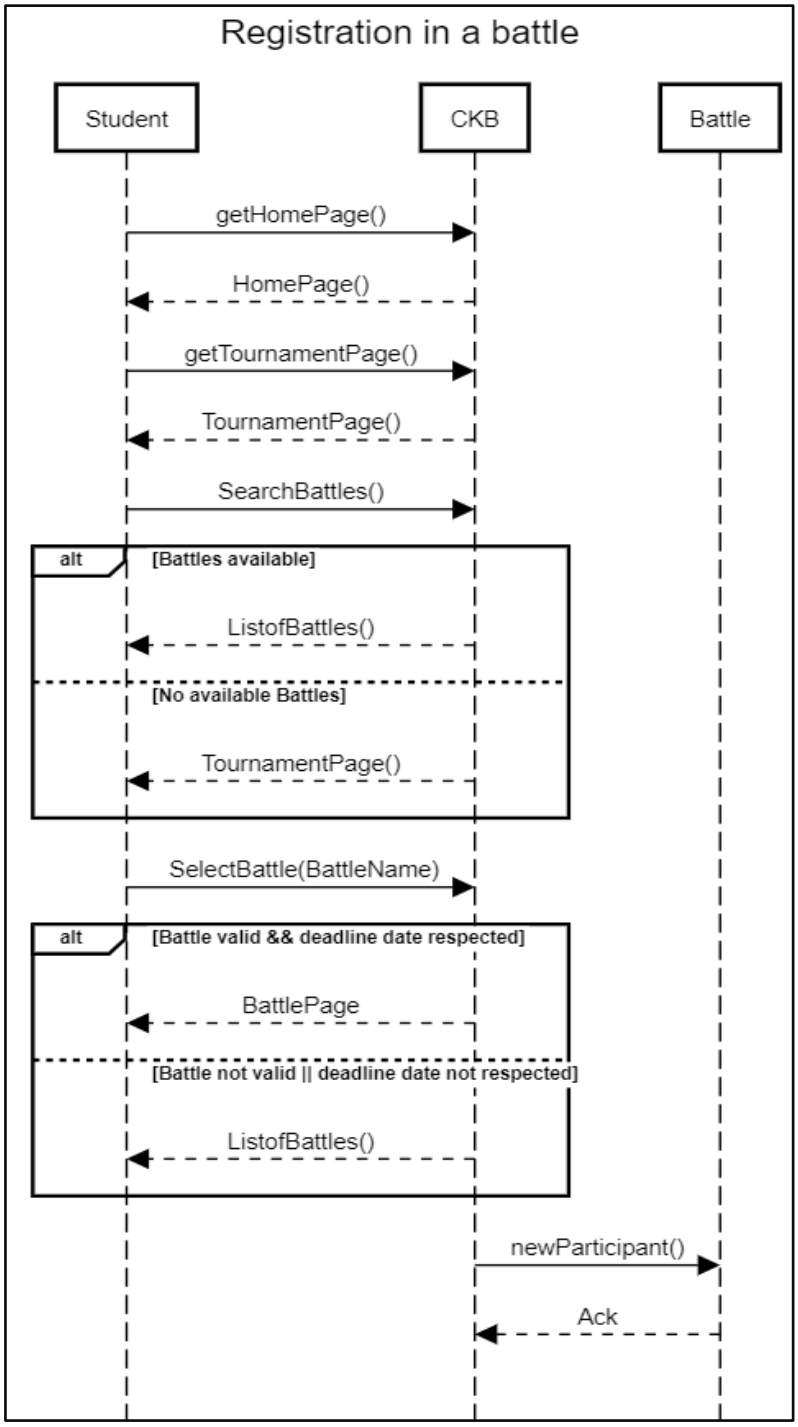
3. UC3 – Student Registers in a tournament

Name	Registration in a tournament
Actor	Student
Entry Condition	The user is logged in and wants to participate in a tournament
Event Flow	1. The student goes on the page in the platform shared by the educator. 2. The student clicks on the "Register" button on the page shared by the educator. 3. The system registers the new student to the tournament. 4. The system inserts the new student in the list of students that will receive the notification when a battle will be created. 5. The system shows the student the home page of the tournament.
Exit Condition	The student is successfully registered to the tournament
Exceptions	1. The student was already registered to the same tournament. 2. The student tried to register after or before the deadline's dates.



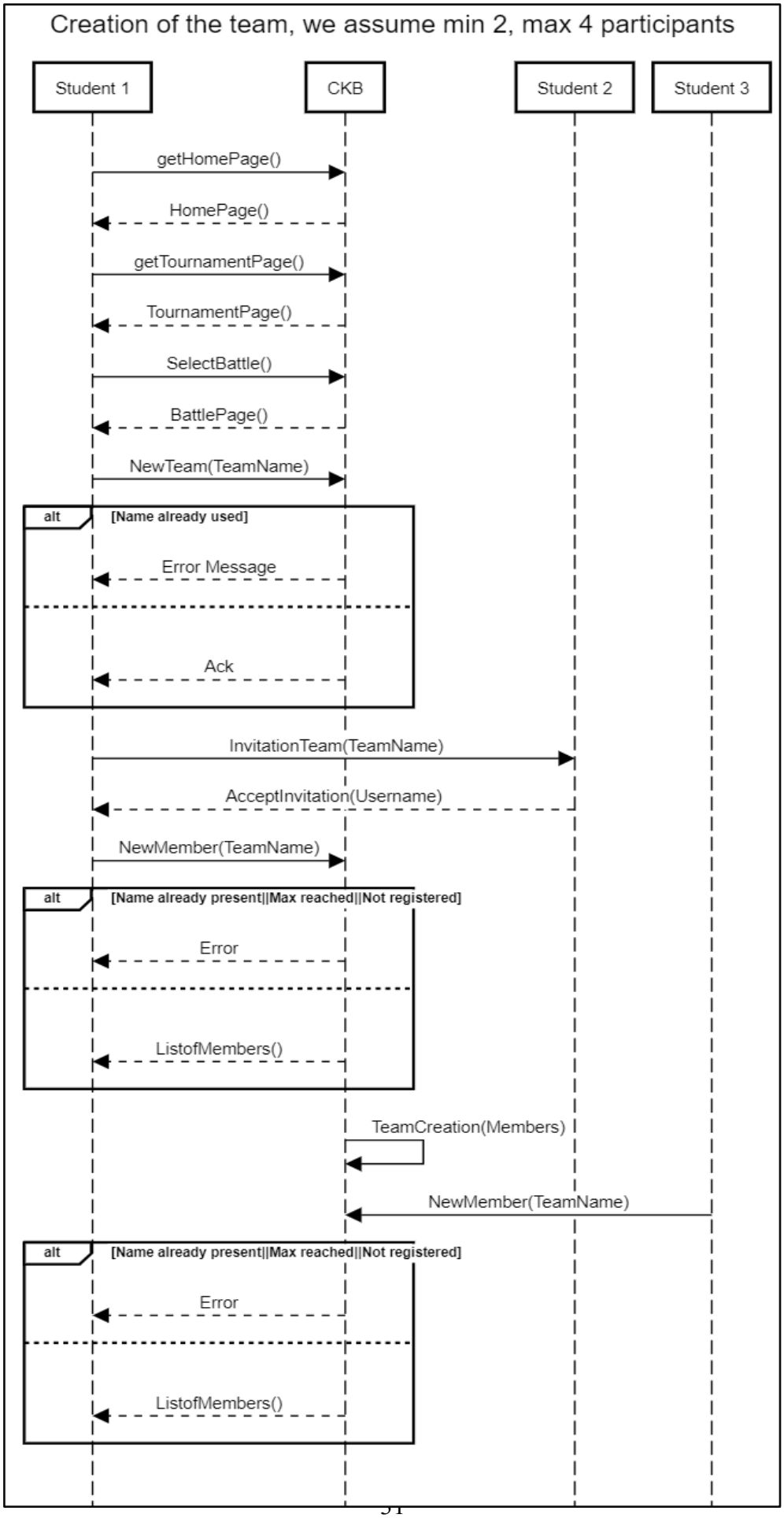
4. UC4 – Student Registers to a battle

Name	Registration in a battle
Actor	Student
Entry Condition	The student is registered in the tournament
Event Flow	1. After the registration the system will show to the student the list of battles available. 2. The student chooses one battle. 3. The system allows the student to create or participate in a team.
Exit Condition	The student is registered in the battle, but not as a team.
Exceptions	1. The student is not registered in the tournament. 2. There are no battles available. 3. The registration deadline has passed.


Registration in a battle
Student
CKB
Battle
getHomePage()
HomePage()
getTournamentPage()
TournamentPage()
SearchBattles()
alt
[Battles available]
ListofBattles()
[No available Battles]
TournamentPage()
SelectBattle(BattleName)
alt
[Battle valid && deadline date respected]
BattlePage
[Battle not valid || deadline date not respected]
ListofBattles()
newParticipant()
Ack

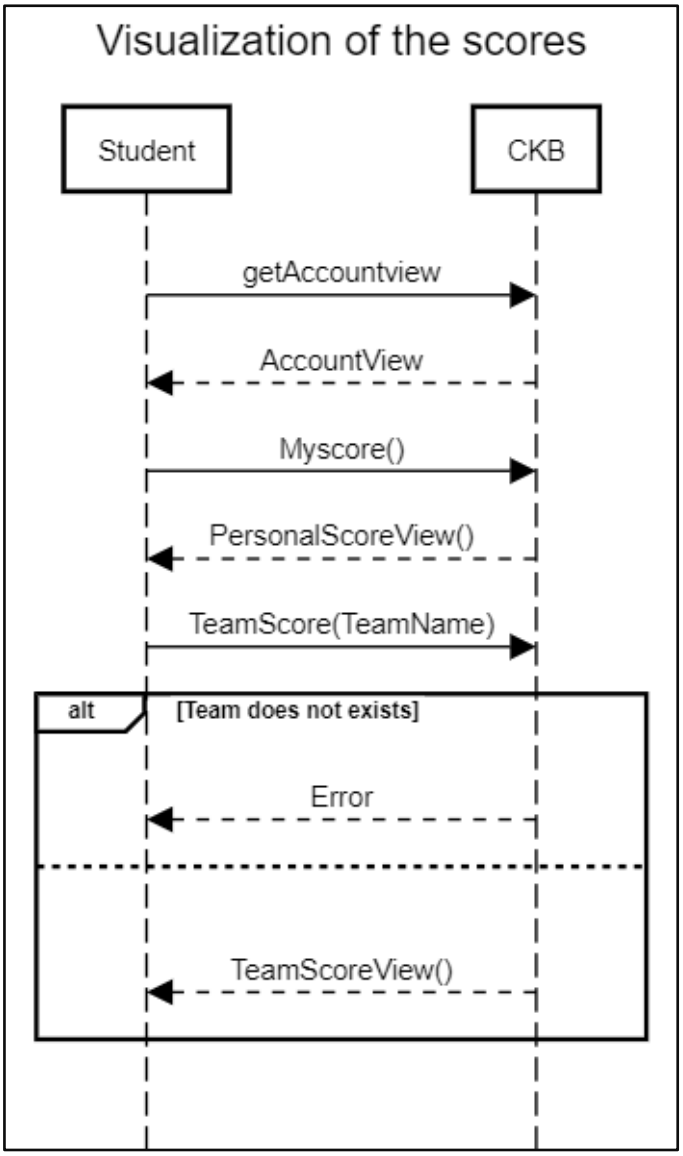
5. UC5 – Student Creates a team for the battle.

Name	Creation of the team
Actor	Students
Entry Condition	The battle has been created and the students subscribed to it.
Event Flow	1. The students choose which students to be in the team with. 2. The student can enter in a team by choice or by invite. 3. The student accepts the invite to join the team. 4. The student subscribes his team to the battle. 5. The system checks the validity and creates the team for that battle. 6. The system creates the Git repository for the battle. 7. The system sends the link to the repository to all the members of the teams.
Exit Condition	The battle has ended.
Exceptions	1. The student was already in another team. 2. The team is composed by the same student multiple times. 3. The team has more students than allowed. 4. The team has less students than allowed. 5. The student invited is not registered to the tournament.


Creation of the team, we assume min 2, max 4 participants
Student 1
CKB
Student 2
Student 3
getHomePage()
HomePage()
getTournamentPage()
TournamentPage()
SelectBattle()
BattlePage()
NewTeam(TeamName)
alt
[Name already used]
Error Message
Ack
InvitationTeam(TeamName)
AcceptInvitation(Username)
NewMember(TeamName)
alt
[Name already present||Max reached||Not registered]
Error
ListofMembers()
TeamCreation(Members)
NewMember(TeamName)
alt
[Name already present||Max reached||Not registered]
Error
ListofMembers()

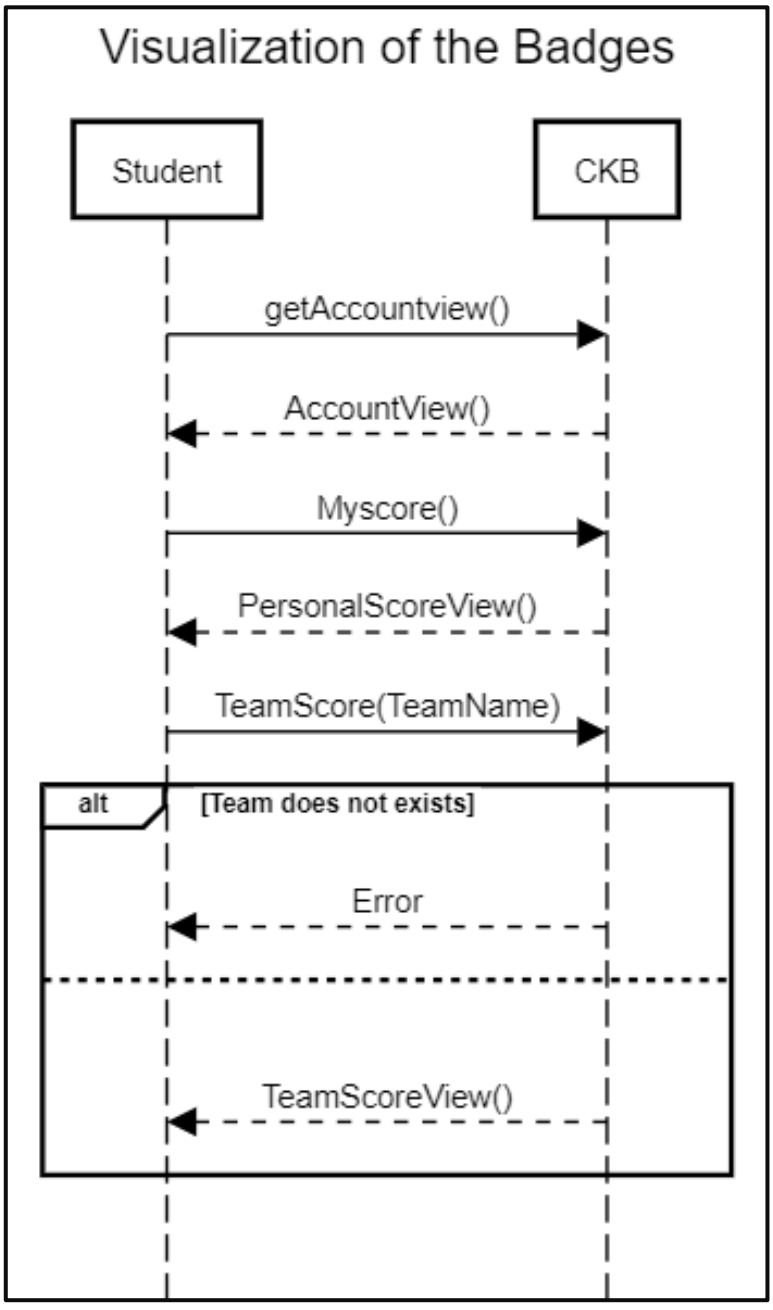
6. UC6 – Student Views his and his team's score.

Name	Visualization of the score
Actor	Students
Entry Condition	The student is logged in and is on his personal account page
Event Flow	1. The student clicks on the "view my score" on his personal page or "view the rankings" on the tournament page. 2. If he wants to see his personal score, the system will check his stored score related to his ID and will print it on the account page. 3. If he wants to see the team's score, the system will access to the rankings of the tournament, check which team he is part of and will print it on the tournament page.
Exit Condition	The student has seen the score and closes the page.
Exceptions	1. The student has never participated in a battle or tournament. 2. The team has not submitted any solutions yet, thus there is no evaluation score. 3. The score is being changed by the educator with a manual evaluation.



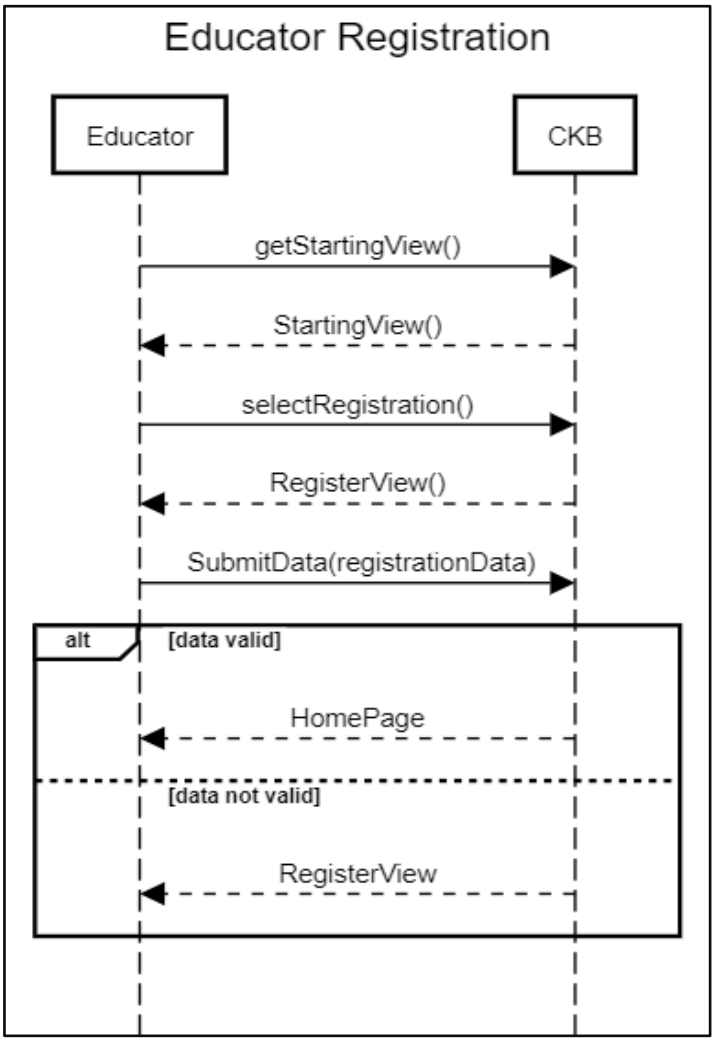
7. UC7 – Student Views his badges.

Name	Visualization of the badges
Actor	Students
Entry Condition	The student is logged in, and he's on his personal account page.
Event Flow	1. The student clicks on the "Badges" button 2. The system matches the information about the student, stored in the memory. 3. The system shows the badges won by the student.
Exit Condition	The student sees his badges and then closes the page.
Exceptions	1. The student hasn't won a badge. 2. The student hasn't succeeded all the requirements required for gaining a badge.



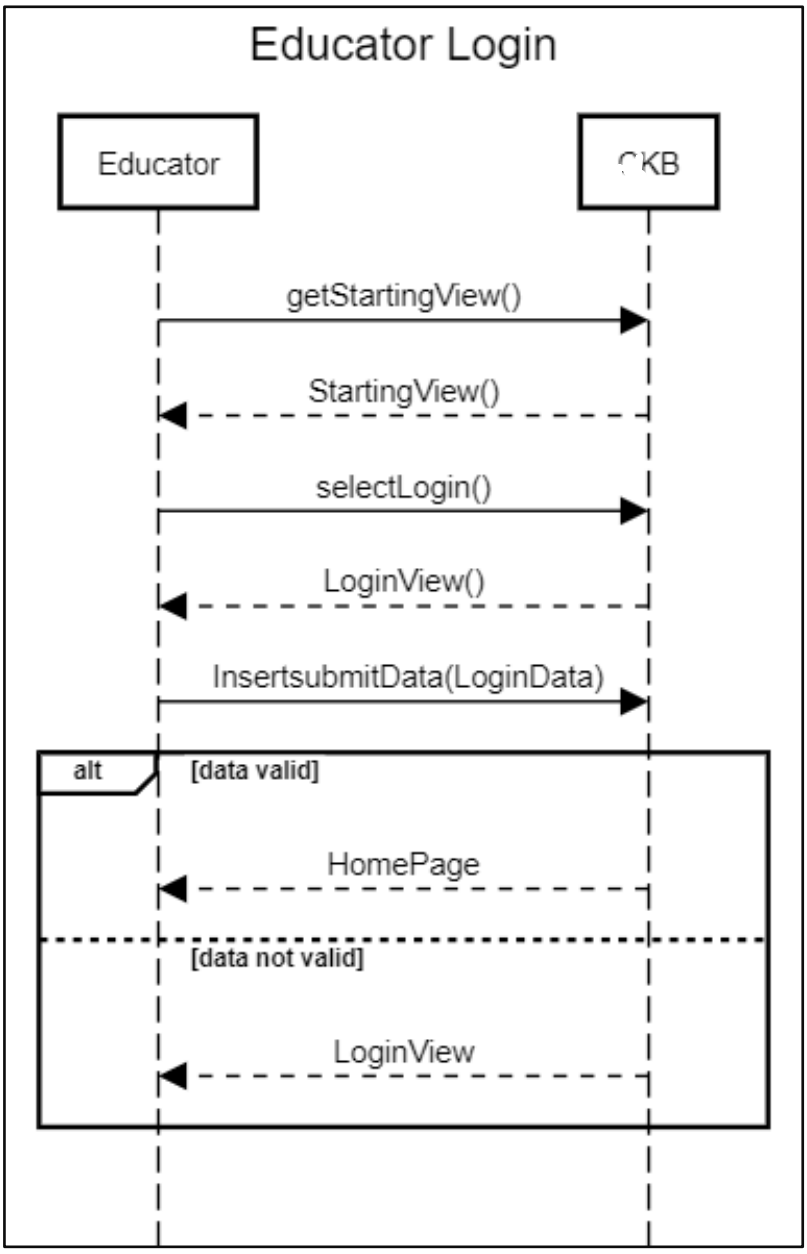
8. UC8 – Educator Signs up.

Name	Educator registration
Actor	Educator
Entry Condition	The educator is on the CKB page and wants to enter in the platform.
Event Flow	1. The educator is at the sign-up page. 2. The educator selects the "Sign up" button. 3. The educator inserts his credentials and his personal information, such as: name, surname, school, email address, password. 4. The system checks whether there is already the same user registered. 5. The system checks the validity of the credentials inserted. 6. The system sends a confirmation email. 7. The educator will confirm the registration. 8. The system will show the home page of CKB.
Exit Condition	The educator is registered.
Exceptions	1. The educator inserts non valid credentials. 2. The educator is already registered. 3. The educator has already used the same email address. 4. The Password inserted is different from the Confirm password one.



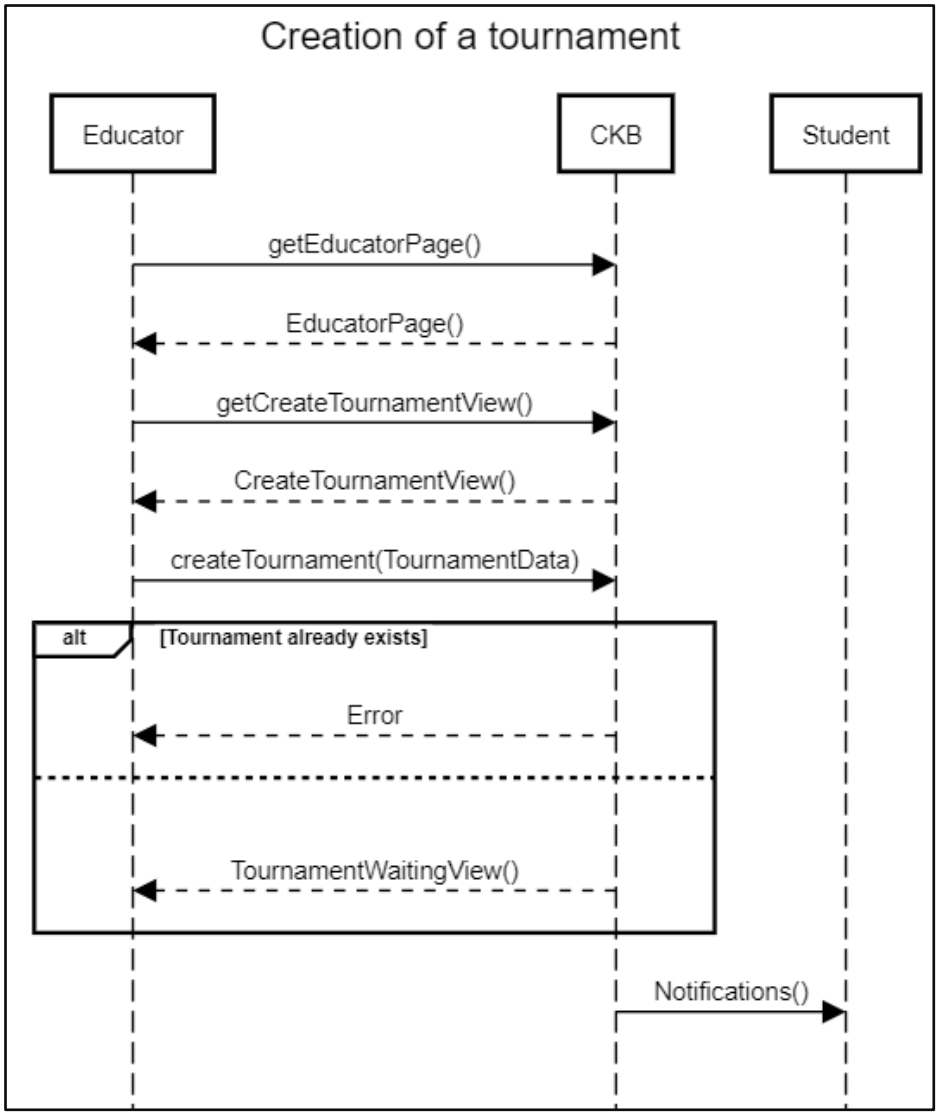
9. UC9 – Educator Login.

Name	Educator Login
Actor	Educator
Entry Condition	The educator is on the CKB page and wants to login.
Event Flow	1. The educator clicks on the “Login” button. 2. The educator fills the form, inserting his personal credentials: username and password. 3. The system receives the data and checks if the account exists. 4. The system checks if the credentials are correct. 5. The educator sees the home page once he’s logged in.
Exit Condition	The educator is logged in.
Exceptions	1. The educator has inserted wrong credentials. 2. The username doesn’t correspond to any account.



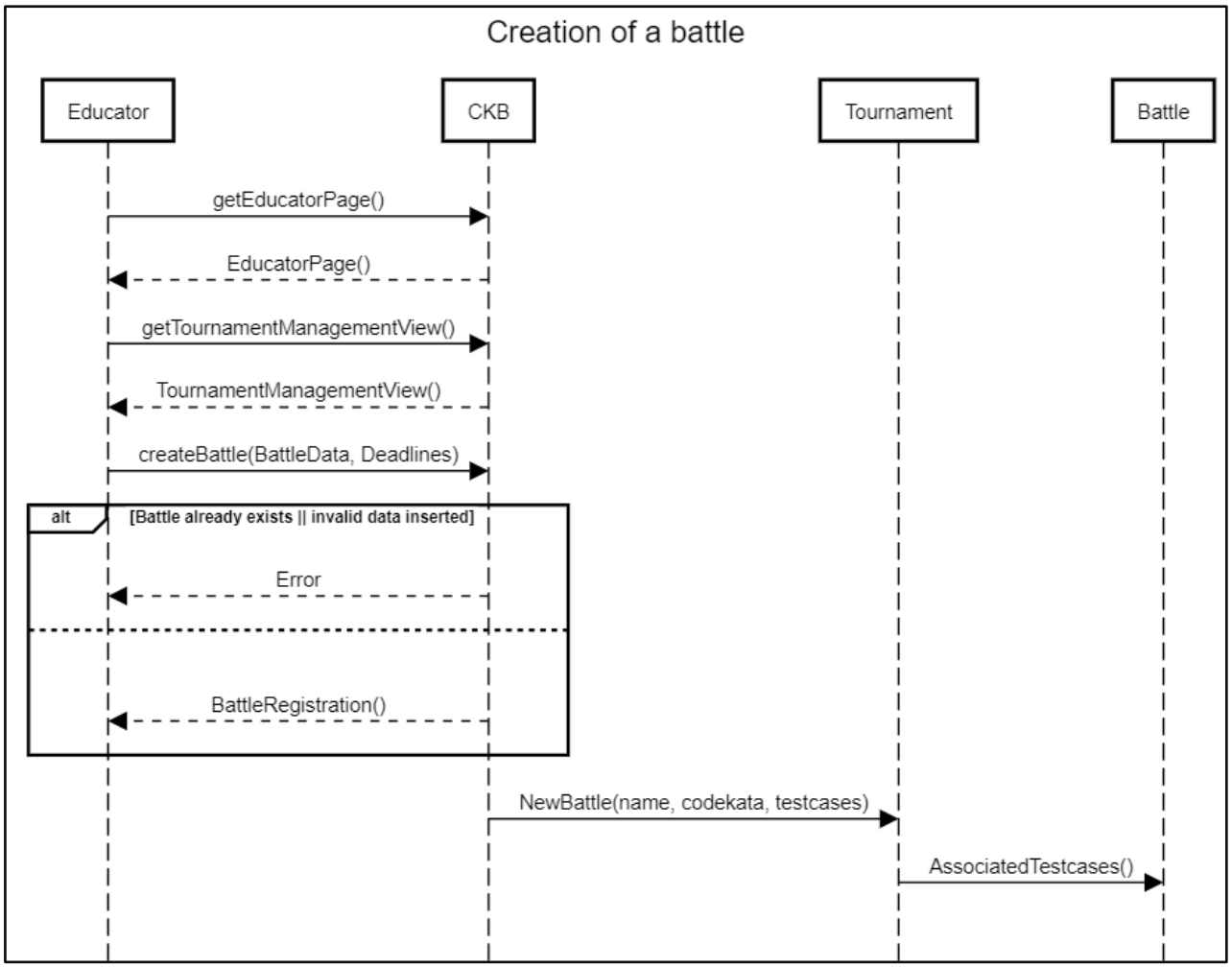
10. UC10 – Educator Creates a tournament.

Name	Creation of a tournament
Actor	Educator
Entry Condition	The educator is on the home page and selects the option for creating a new tournament.
Event Flow	1. The educator fills up the form's field with information related to the new tournament, such as: name, name of the educator responsible. 2. The educator creates a link to share the tournament with other students via other communications.
Exit Condition	The educator sets the tournament as visible and allows the students to register.
Exceptions	1. Another tournament with the same name exists. 2. The educator selects deadline dates invalid or inconsistent. 3. The educator chooses an invalid max and min number of student per team. 4. The educator doesn't save and publish the tournament.



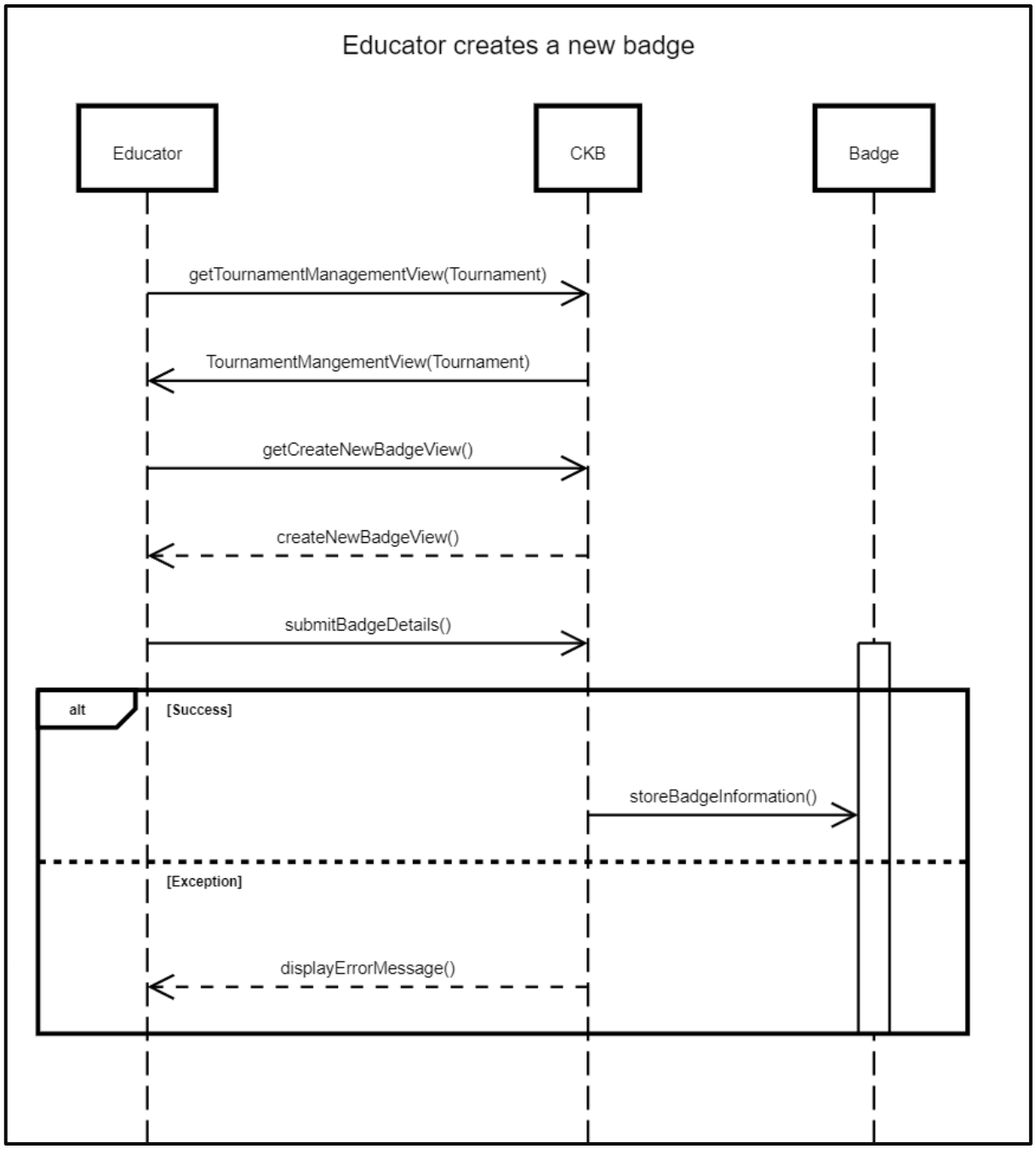
11. UC11 – Educator Creates a battle.

Name	Creation of a battle
Actor	Educator
Entry Condition	The educator is on the home page of one tournament and clicks on the "New battle" button.
Event Flow	1. The educator inserts the name of the battle. 2. The educator sets the registrations deadline and the final submissions deadline of the solutions. 3. The educator sets the minimum and maximum number of students per team. 4. The educator decides whether to enable manual evaluation or not. 5. The educator uploads the code kata, that means the description and software project, test cases and basic automation scripts. 6. The educator clicks on the button to confirm the creation of the battle with the setting he has decided.
Exit Condition	The battle has been created and students can register in it.
Exceptions	1. The educator selects deadline dates invalid or inconsistent. 2. The educator chooses an invalid max and min number of student per team. 3. Another battle has the same description. 4. Another battle has the same name.



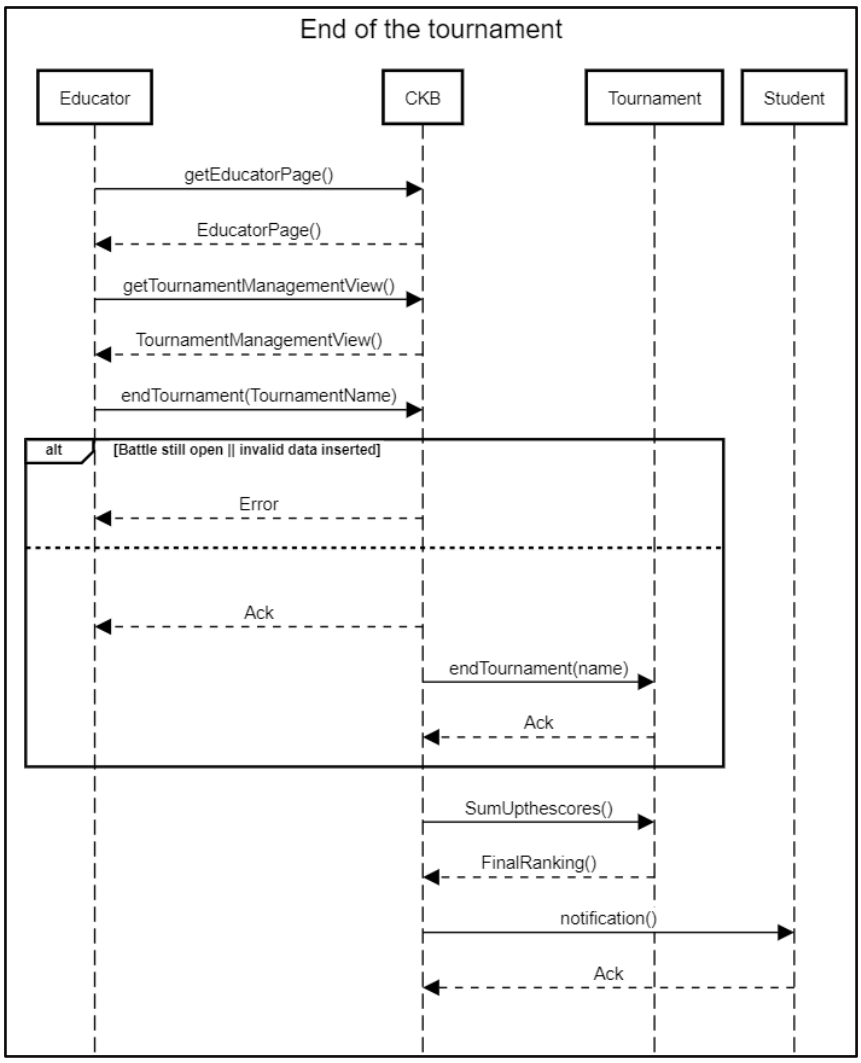
12. UC12 – Educator creates a new Badge.

Name	Educator creates a new Badge
Actor	Educator
Entry Condition	• The educator is logged into the CodeKataBattle (CKB) platform. • The educator has created a tournament and intends to define a new badge.
Event Flowgr	1. The educator navigates to the tournament management section of the tournament he wants to create the Badge for, from the CKB platform. 2. Within the tournament management view, the educator clicks on the “Create new badge” button to create a new badge. 3. The platform prompts the educator to provide a title for the badge (e.g., "Top Performer"). 4. The educator sets the rules for the badge. This may include criteria such as the number of battles attended, the average score, or any other relevant performance metrics. 5. Additionally, the educator may define specific test cases that students need to pass to be eligible for the badge. 6. The educator submits the badge details, and the platform stores the information for the new badge.
Exit Condition	The new badge is successfully created and stored in the CKB platform.
Exceptions	• If the educator encounters technical issues or there are errors in defining the badge rules or test cases, the platform provides appropriate error messages. • If the educator decides to cancel the badge creation process, the platform returns to the tournament management view without saving the badge.


Educator creates a new badge
Educator
CKB
Badge
getTournamentManagementView(Tournament)
TournamentMangementView(Tournament)
getCreateNewBadgeView()
createNewBadgeView()
submitBadgeDetails()
alt
[Success]
storeBadgeInformation()
[Exception]
displayErrorMessage()

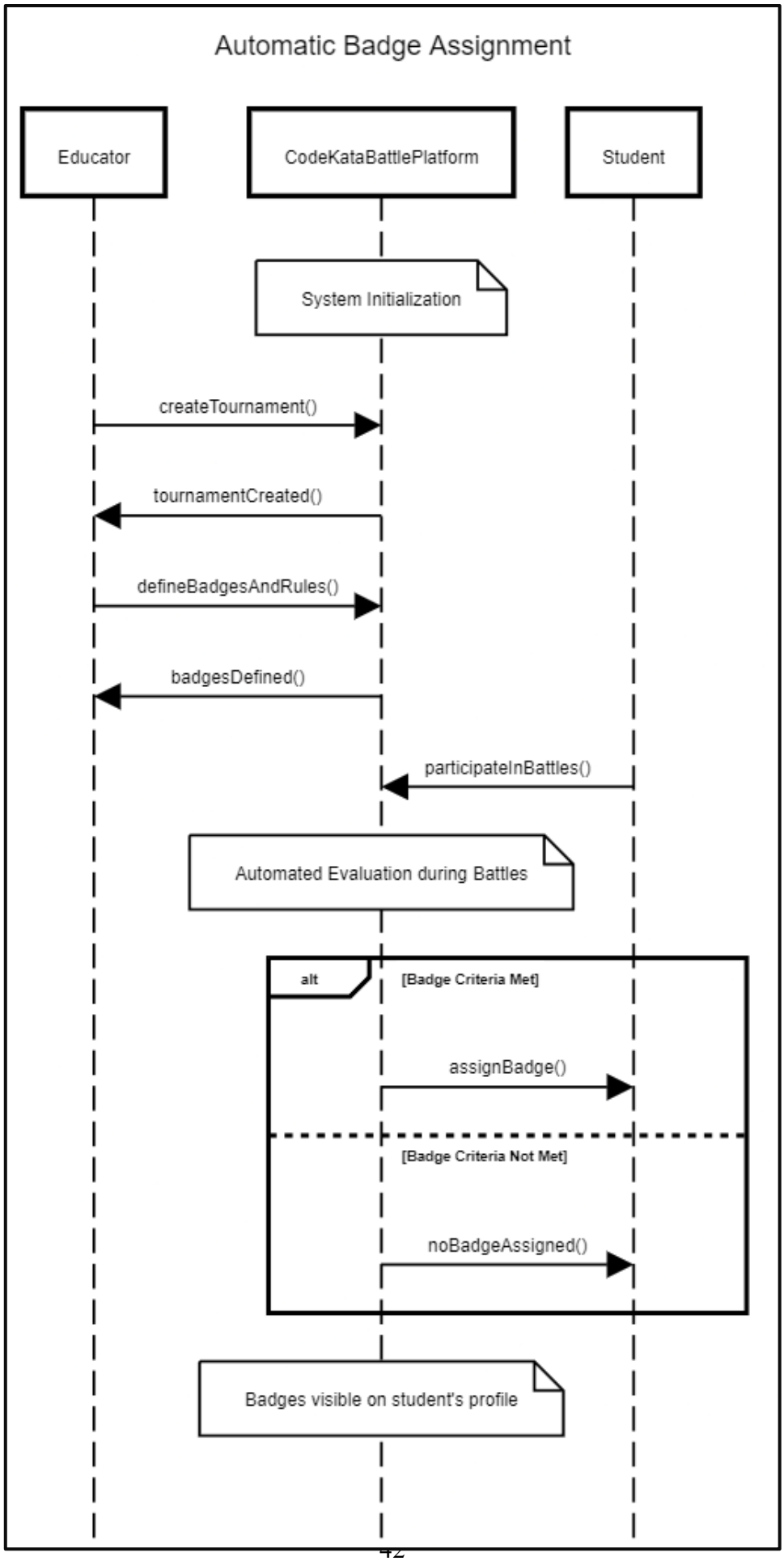
13. UC13 – Educator Ends the tournament.

Name	End of the tournament
Actor	Educator
Entry Condition	All the battles have ended and have been evaluated.
Event Flow	1. The educator is on the web page to close the tournament. 2. The educator decides to close the tournament by selecting the button “Close tournament”. 3. All the battle scores received in a tournament are summed up, by teams and by personal achievements. 4. Once the tournament is closed, the final tournament rank is shown. 5. The platform notifies all the students registered in the tournament, that the final ranking is available, and the tournament is closed
Exit Condition	The tournament is officially closed.
Exceptions	1. The tournament name is not right. 2. The tournament has some battles still opened. 3. Some battles have to be evaluated yet.



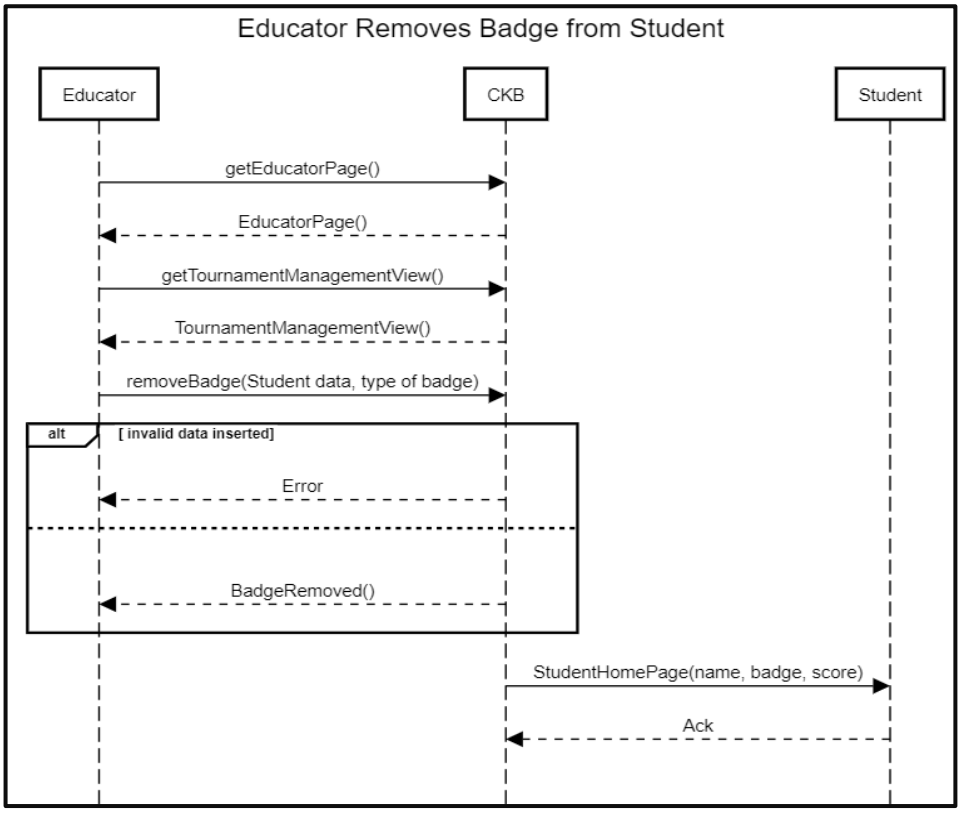
14. UC14 - Automatic Badge Assignment to Students

Name	Automatic Badge Assignment
Actor	CodeKataBattle Platform
Entry Condition	• The system is operational, and a tournament with predefined badges and rules has been closed.
Event Flow	1. **Educator Creates Tournament:** An educator creates a tournament on the CodeKataBattle platform. 2. **Define Badges and Rules:** The educator defines badges for the tournament, specifying titles, rules, and associated variables. 3. **Students Participate in Battles:** Students participate in battles within the tournament, working on code kata challenges. 4. **Automated Evaluation:** As battles progress, the platform automatically evaluates students' performance based their performances. **5. The Educator Closes The Tournament:** When the educator decides to close the competition, if all the battles are terminated, he clicks on the button to close the ongoing Tournament. 6. **Badge Criteria Met:** When a student's performance meets the criteria defined for a specific badge, the system automatically identifies the achievement. 7. **Badge Assignment:** The system automatically assigns the relevant badge to the student without requiring manual intervention. 8. **Badge Visibility:** The badge becomes visible on the student's profile, indicating the achievement.
Exit Condition	Badges are automatically assigned to eligible students based on the predefined rules during the tournament.
Exceptions	• *Invalid Badge Definition:* If an educator defines a badge with invalid or conflicting rules, the system generates an error, and the badge is not considered for automatic assignment. • *Technical Issues:* In case of technical issues during automated evaluation, the system generates an error, and badge assignment may be delayed.


Automatic Badge Assignment
Educator
CodeKataBattlePlatform
Student
System Initialization
createTournament()
tournamentCreated()
defineBadgesAndRules()
badgesDefined()
participateInBattles()
Automated Evaluation during Battles
alt
[Badge Criteria Met]
assignBadge()
[Badge Criteria Not Met]
noBadgeAssigned()
Badges visible on student's profile

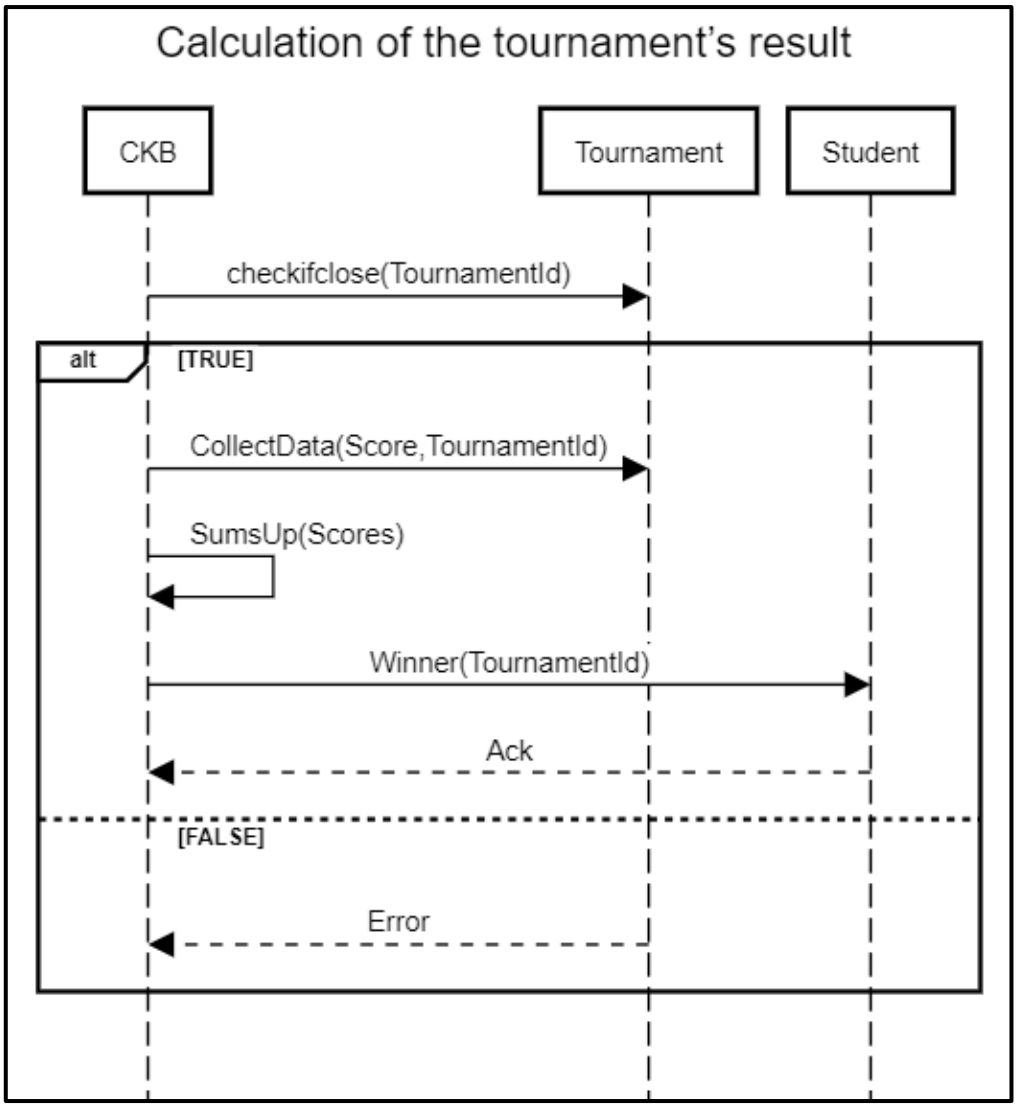
15. UC15 - Educator Removes Badge from Student

Name	Educator Removes Badge from Student
Actor	Educator
Entry Condition	The educator is logged in and has the necessary privileges to manage badges
Event Flow	1. The Educator navigates to the Tournament Management view on the CodeKataBattle (CKB) platform. 2. The Educator selects the tournament for which they want to remove a badge from a student. 3. The Educator accesses the list of participating students in the selected tournament. 4. The Educator identifies the student from whom they want to remove the badge. 5. The Educator initiates the badge removal process for the selected student. 6. The CKB platform validates the Educator's request and confirms the removal action. 7. The selected badge is removed from the student's profile. 8. The CKB platform updates the student's profile to reflect the badge removal.
Exit Condition	The selected badge is successfully removed from the student, and the Educator receives confirmation.
Exceptions	• If the Educator does not have the necessary privileges to manage badges, the system displays an error message. • If there is an issue with the badge removal process (e.g., technical error), the system provides an error notification.



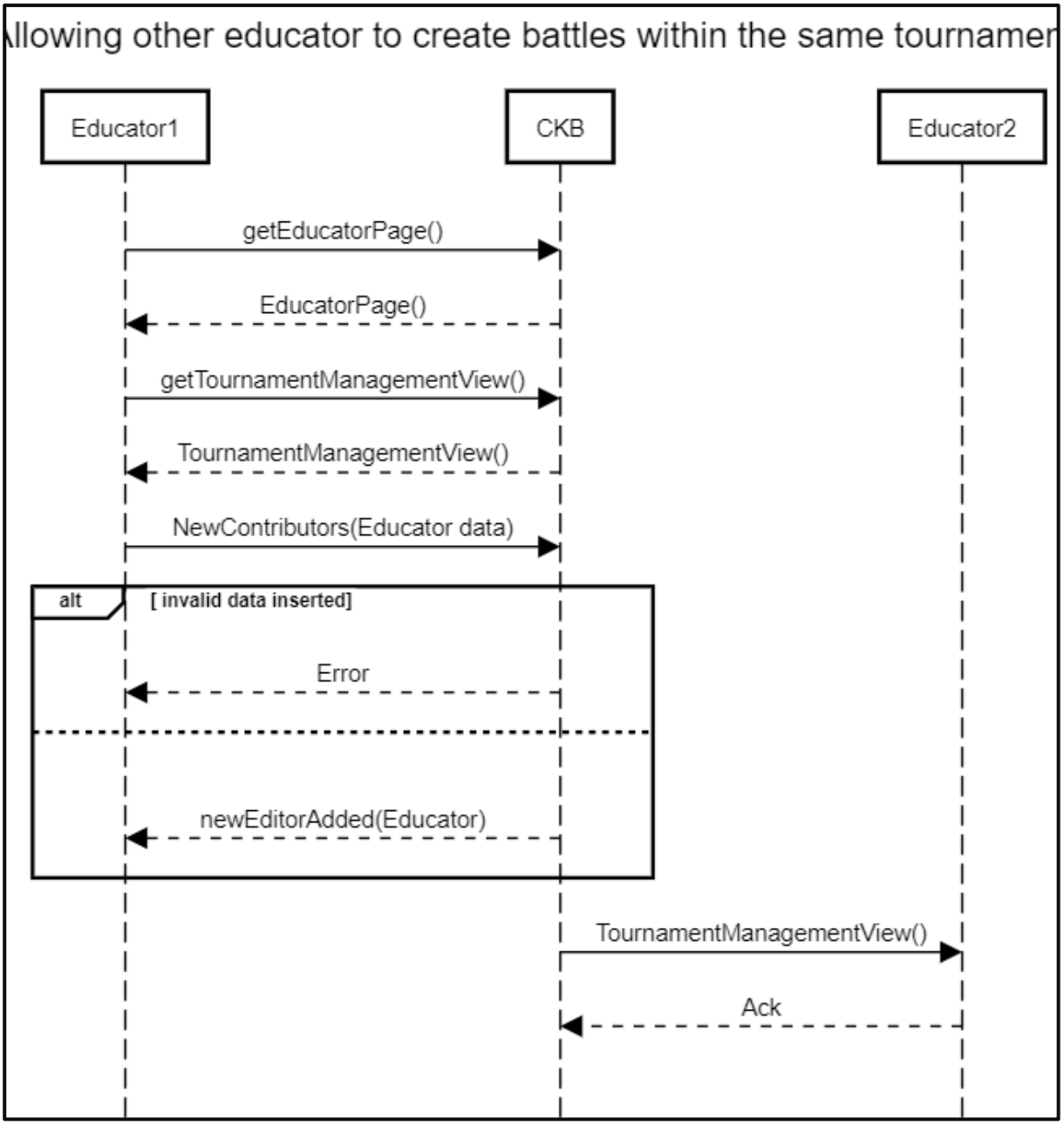
16. UC16 - Tournament Result Calculation

Name	Calculation of the tournament's result
Actor	CodeKataBattle, Educator
Entry Condition	All the battles have ended, and the Tournament has been closed
Event Flow	1. The Educators closes the Tournament by clicking on the apposite button. 2. The platform collects all the scores from all the battles that occurred in the tournament. 3. The platform sums up all the scores per student creating the tournament score. 4. The student with the higher tournament score is the winner of the tournament 5. The tournament score is added to the personal score of the student.
Exit Condition	The platform proclaims the student winner of the tournament.
Exceptions	1. The scores are not available. 2. The student winner is not registered in the tournament. 3. All the students have the same final score.



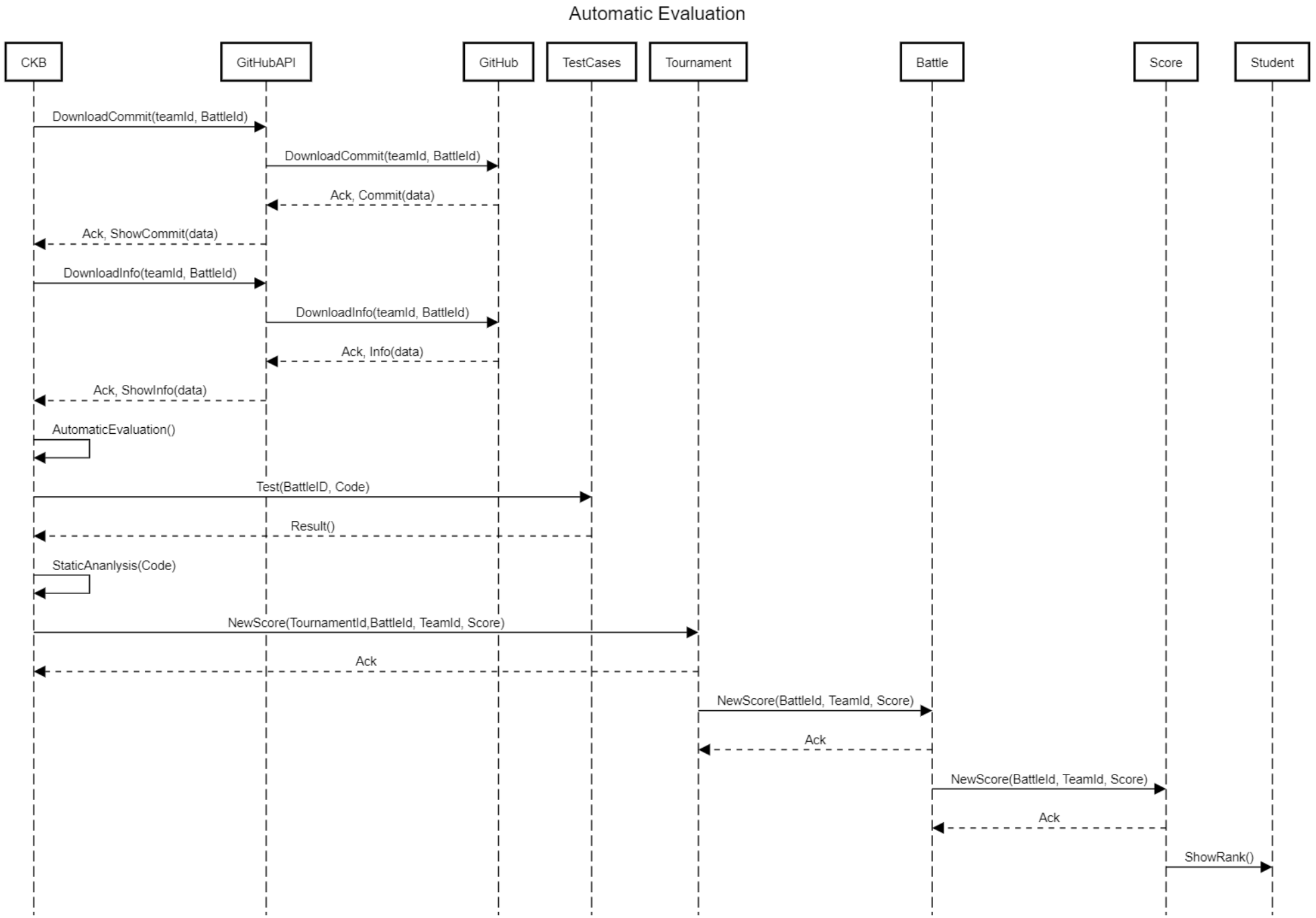
17. UC17 - Educator Grants Battle Creation Permissions

Name	Allowing other educator to create battles within the same tournament
Actor	Educator
Entry Condition	The educator is on the tournament home page and clicks on “New contributors” button.
Event Flow	1. The educator fills the form where he inserts the other educator’s name and surname. 2. The educator receives the invite to participate as editor in the same tournament. 3. A flag is set which states that the tournament can be edited by more educators.
Exit Condition	The second educator has now the privileges to create battles.
Exceptions	1. The second educator is not valid.



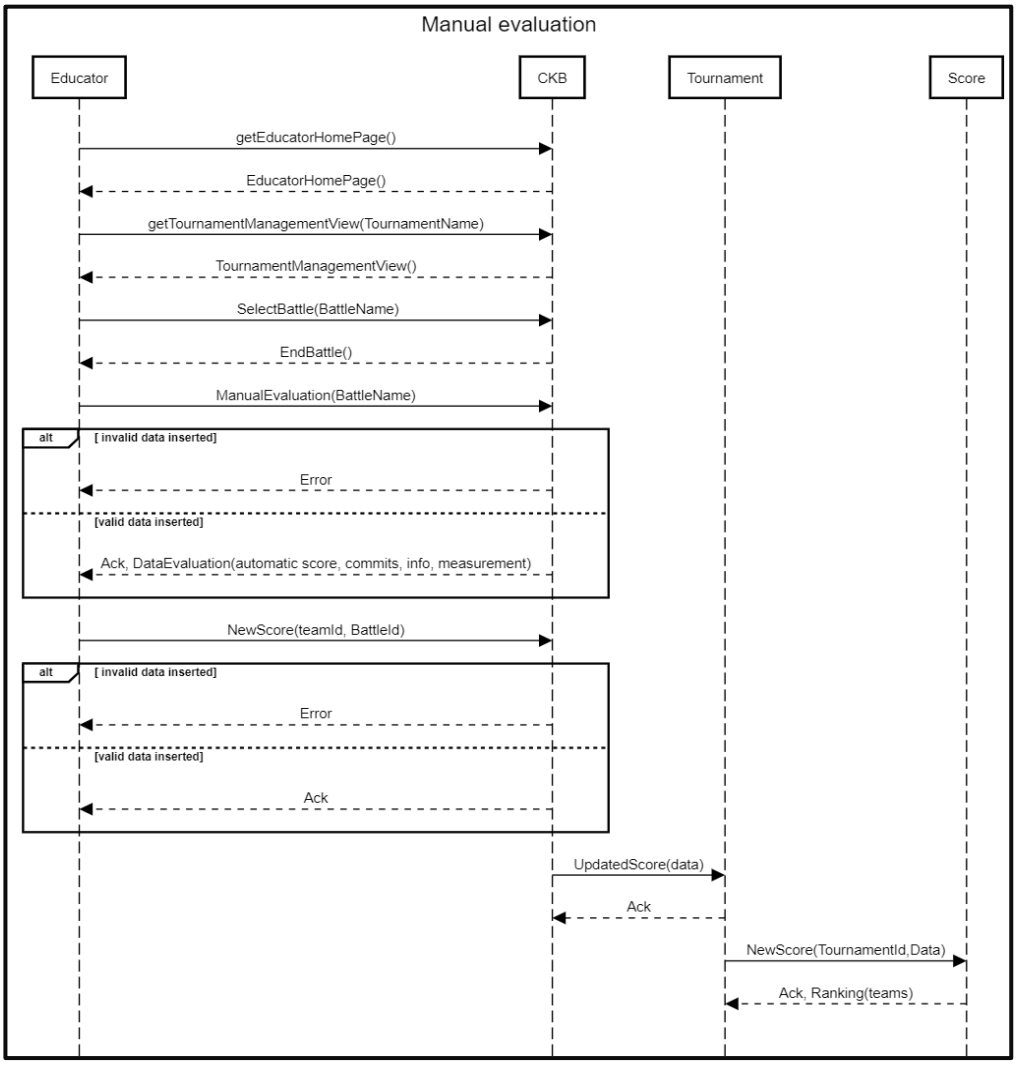
18. UC18 - Automatic Evaluation

Name	Automatic Evaluation
Actor	CodeKataBattle, TestCases.
Entry Condition	The students have done a commit.
Event Flow	1. The platform retrieves the ultimate solution published by the teams. 2. The platform performs all the tests cases on every single solution. 3. The platform collects the result data from each test done. 4. Collects the timestamp of each operation, from the registration and the last commit. 5. The platform computes the time passed from the registration and the last commit. 6. The platform performs static analysis on the solution and calculates the levels of security, reliability, maintainability. 7. Each of the 3 factors considered has a different weight on the computation of the final score. 8. The platform sums up all the factor weighted and calculates the battle score.
Exit Condition	The platform assigns to each team their computed score and thus to each member of the team.
Exceptions	1. The data collected are not valid. 2. The computed score is not a valid number. 3. The computed score is less than 0 or higher than 100. 4. The educator asks for a manual evaluation.


Automatic Evaluation
CKB
GitHubAPI
GitHub
TestCases
Tournament
Battle
Score
Student
DownloadCommit(teamId, BattleId)
DownloadCommit(teamId, BattleId)
Ack, Commit(data)
Ack, ShowCommit(data)
DownloadInfo(teamId, BattleId)
DownloadInfo(teamId, BattleId)
Ack, Info(data)
Ack, ShowInfo(data)
AutomaticEvaluation()
Test(BattleID, Code)
Result()
StaticAnanlysis(Code)
NewScore(TournamentId,BattleId, TeamId, Score)
Ack
NewScore(BattleId, TeamId, Score)
Ack
NewScore(BattleId, TeamId, Score)
Ack
ShowRank()

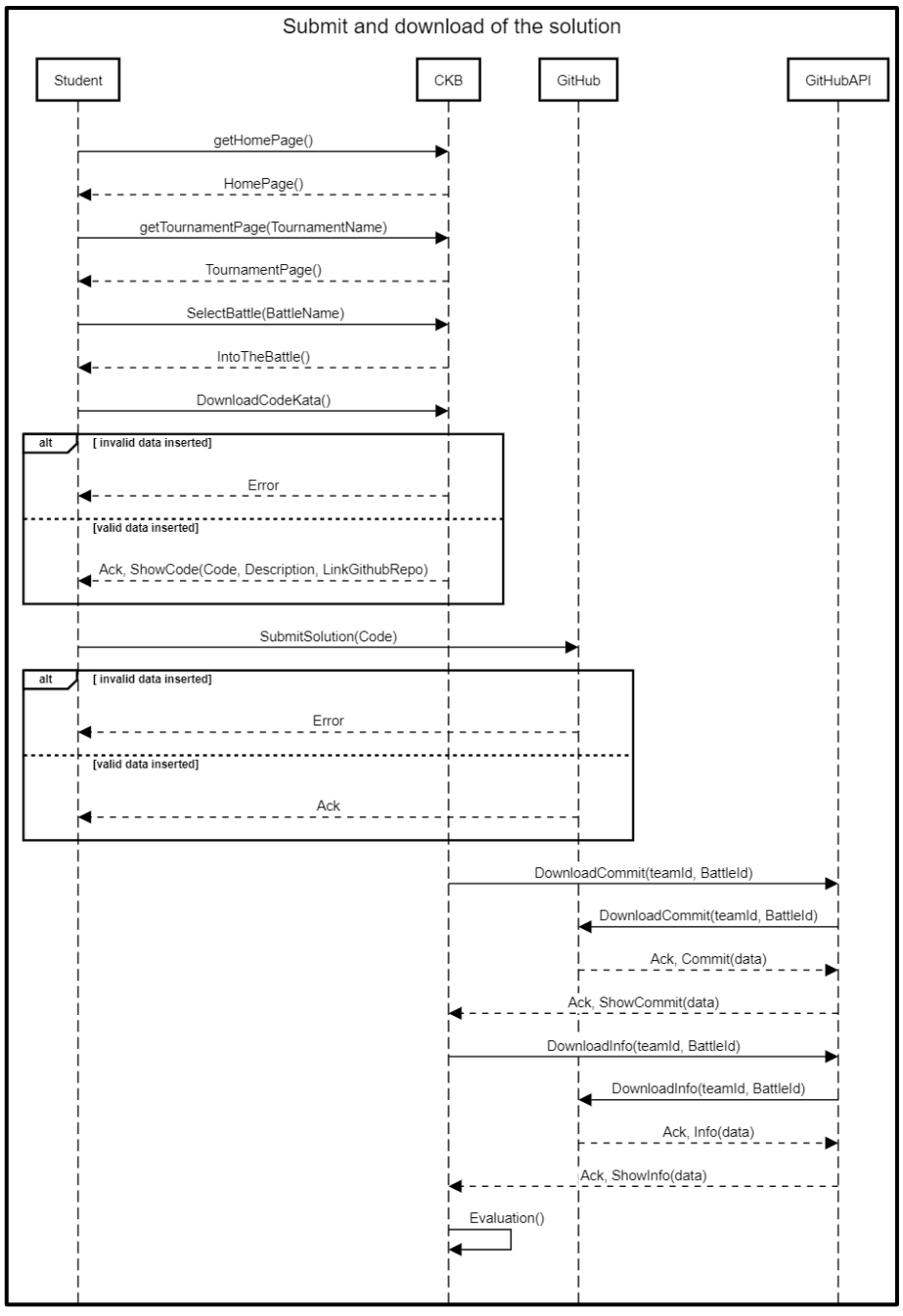
19. UC19 - Educator manually evaluates a Student

Name	Manual evaluation
Actor	Educator
Entry Condition	The educator has enabled the manual evaluation setting during the creation of the battle.
Event Flow	1. The submission deadline of the battle expires and there is the consolidation stage. 2. The educator receives all the scores data collected by the platform, such as the time passed from the registration to the submit of the ultimate solution. 3. The educator looks at the automatic evaluation as reference, and then proceeds to manually evaluate the solution based on his personal criteria.
Exit Condition	The educator sets the new score calculated and publishes it on the final rank.
Exceptions	1. The educator sets an invalid score. 2. The tournament was already closed. 3. Any other errors occurred in the evaluation.



20. UC20 – Student Submits the team's solution.

Name	Submit and download of the solution
Actor	Students and CodeKataBattle
Entry Condition	The student, who participates in the tournament and is playing in a battle, commits on GitHub his team solution
Event Flow	1. The students commit on GitHub. 2. The platform periodically downloads the file published on the repository created for each battle. 3. The platform collects even the data for the measurement like time and numbers of commits.
Exit Condition	This event triggers the automatic evaluation Use Case
Exceptions	1. The commit was non valid. 2. Incongruency errors.



3.2.3 Mapping requirements

G1. Help students to improve their software skills	
• R1, R2, R3, R4	• D4

G2. Allow students to subscribe into a specific tournament	
• R9, R10	• D1, D2, D8, D9

G3. Allow students to see the corresponding tournament rank	
• R24	• D8

G4. Allow students to be evaluated for the work they have done	
• R19, R21, R22, R32	• D3

G5. Help students to make a coworking environment by creating and joining teams.	
• R12	• D10

G6. Allow teams to participate to the battles.	
• R11, R13, R14, R15, R16, R17, R18, R29, R30, R31	• D5, D6, D10, D11

G7. Allow the educator to create CKB tournaments and battles	
• R5, R6, R7, R8	• D7

G8. Allow the educator to setup rules for the battle	
• R6, R32	• D11, D12

G9. Allow manually evaluation by the educator that check and evaluates the work done by the students	
• R20, R23	• D12

G10. Allow the educator to define badges and rules to reward students based on their performance	
• R25, R26, R27	• D12

3.4 Performance Requirements

The CKB platform must be able to handle a high number of concurrent requests. Therefore, the CKB platform should support a certain level of concurrent user activity, allowing multiple users to participate in battle and tournaments simultaneously.

In addition, the CKB platform should demonstrate minimal latency in responding to user interactions, ensuring a smooth and timely experience. This encompasses rapid execution of commands and swift loading of web pages.

Finally, given the platform's reliance on communication between users and the GitHub repository, careful attention must be paid to network performance considerations. The requirements should explicitly cover aspects such as data transfer speed and reliability of network connections. This is paramount for ensuring effective and consistent platform operations.

3.5 Design Constraints

3.5.1 Standards Compliance

All users' data should be treated in compliance with GDPR.

The platform should be available on all the main web browsers operating systems and the regulations of the external APIs used must be followed properly.

3.5.2 Hardware Limitations

Since the system will be made available as a web application, the requirement needed on the hardware is mobile device with an internet connection so it can access the internet through one of the main web browsers.

3.6 Software System Attributes

3.6.1 Reliability

The system must be able to run continuously without any interruptions. To achieve this, the platform must be reliable for users. This involves minimize downtime and ensured that the system is fault tolerance.

3.6.2 Availability

Users can use the system throughout the week at any time during the day. So, the system should offer an availability equal to 99% of the time. This means that, the MTTR (or downtime) should be contained around 3.65 days per year.

3.6.3 Security

All communications between different entities must be on a secure channel and must be encrypted in order to avoid packet sniffing and spoofing. User credential should be confidentially stored.

3.6.4 Maintainability

Maintainability in the CKB platform refers to the ease with which the system can be updated, modified, and sustained over time. This means that, the system should be easily extended with an additional feature with minimum effort. Also, a testing routine must be provided, and it must cover a good percentage (85%) of the entire codebase.

3.6.5 Portability

The system should be designed and developed to ensure compatibility with popular browsers such us Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. So, the platform must run on Windows and iOS operating systems.

4.Alloy

```
/*SIGNATURES*/

abstract sig Boolean{}
one sig TRUE extends Boolean{}
one sig FALSE extends Boolean{}

sig Username{}
sig Password{}
sig EmailAddress{}

sig CodeKataBattle{
        tournament_available: some Tournament,
}

sig Student{
        username: one Username,
        password: one Password,
        email: one EmailAddress,
        personal_score: one Score,
        visualize_badge: set Badge,
}

sig Educator{
        username: one Username,
        password: one Password,
        email: one EmailAddress,
        assignScore: some Score,
        assignBadge: some Badge,
}
```

```
sig Tournament{
        tournamentId: one Int,
        battles_available: set Battles,
        participants: set Student,
        educator: some Educator,
        setted_by_more: one Boolean,
        registrationDeadline: one DateTime,
        startTime: one DateTime,
        endTime: one DateTime,
        Openstatus: one Boolean,
        Closedstatus: one Boolean,
}{
        tournamentId > 0 and
        startTime.timestamp < endTime.timestamp
        startTime.timestamp >=0 and Openstatus != Closedstatus and
        startTime.timestamp > registrationDeadline.timestamp and  startTime.timestamp <
endTime.timestamp
}



sig Battles{
        battle_Id: one Int,
        teams_in_battle: some Team,
        battle_rule: one BattleRules,
        link_repo: one GitHubRepo,
        test_for_battle: one TestCase,
        Startstatus: one Boolean,
        EndStatus: one Boolean,
        deadlineRegistration: one DateTime,
        startTimeBattle: one DateTime,
        endTimeBattle: one DateTime
```

```
}{
        battle_Id > 0 and EndStatus != Startstatus
        and startTimeBattle != endTimeBattle

}

sig Team{
        teamId: one Int,
        student_in_team: some Student,
        team_score: one Score,
        minParticipant: one Int,
        maxParticipant: one Int,
}{
        minParticipant < maxParticipant and
        minParticipant > 0 and teamId>0
}

sig Score{
        score: one Int
}{
 score >= 0 and score <= 100
}

sig TestCase{
  test: "test for the battle",
  testStatus: one Boolean,
}

sig Badge{
        variables: some Variables
}
```

```
sig Variables{
        constraint: "String Rules for the badge",
        setted_by: one Educator
}

sig BattleRules{
        rule: "String rules for the battle",
        setted_by: one Educator,
}

sig Notification {
   receiver: set Student,
   content:"news"
}

sig DateTime{
        timestamp: one Int
}{
        timestamp > 0
}

sig GitHubRepo{
 link: "String link"
}
```

```
/*FACTS OF THE PERSONAL DATA*/
fact noDuplicatedEmailStudent{
 all s1, s2: Student | s1 != s2 implies s1.email != s2.email
}

fact nosameEmailandUsername{
all s: Student, e: Educator|s.email !=e.email and s.username!= e.username
}

fact noDuplicatedUsernameStudent{
 all s1,  s2: Student | s1 != s2 implies s1.username != s2.username
}

fact noDuplicatedEmailEducator{
 all e1, e2: Educator | e1 != e2 implies e1.email != e2.email
}

fact noDuplicatedUsernameEducator{
 all e1,  e2: Educator | e1 != e2 implies e1.username != e2.username
}

//All the teams in the same battle mustn't have the same students in it
fact NoCommonStudentsInSameBattle {
  all b: Battles, t1, t2: Team |
    t1 in b.teams_in_battle and t2 in b.teams_in_battle and t1 != t2 implies no s: Student
    | s in t1.student_in_team and s in t2.student_in_team
}
```

```
//The number of the students in the platform must be greater or equal than the number of the tournament participants
fact RegisteredStudentsGreaterOrEqualParticipants {
   all t: Tournament | #t.participants <= # Student
}


/*CONSTRAINT OF THE BATTLE*/
/*name of the Battle must be unique*/
fact UniqueBattleName{
 all b1,b2:Battles | (b1!=b2)implies b1.battle_Id != b2.battle_Id
}

//The start time of the battle has to happen after the deadline of the registration in the tournament
fact TimestampBattle{
 all b: Battles |
        b.deadlineRegistration.timestamp < b.startTimeBattle.timestamp and
        b.startTimeBattle.timestamp >= 0
}

//One battle has to be connected only to one tournamert
fact BattlesConnectedToTournament {
  all b: Battles | one t: Tournament | b in t.battles_available
}

fact TeamsConnectedToOneBattle{
  all t: Team | one b: Battles | t in b.teams_in_battle
}
```

```
/*CONSTRAINT OF THE TOURNAMENT*/
fact UniqueTournamentName{
 all t1,t2: Tournament | (t1!=t2) implies t1.tournamentId != t2.tournamentId
}

fact TournamentsConnectedToCodeKataBattle {
  all t: Tournament, cb: CodeKataBattle | t in  cb.tournament_available
}

fact UniqueTeamId{
        all t1,t2: Team | t1!=t2 implies t1.teamId != t2.teamId
}

fact UniqueLinkGitHubRepo{
        all b1,b2: Battles | b1!=b2 implies b1.link_repo != b2.link_repo
}

fact TeamHasAtLeastOneStudent{
        all t:Team | #t.student_in_team > t.minParticipant and t.maxParticipant > #t.student_in_team
}
//The test cases for one battle are unique, based on the battle goal
fact testcaseinBattle{
        some  t1,t2: TestCase| all b: Battles|(t1 in b.test_for_battle and t2 in b.test_for_battle) implies
t1!= t2
}
//The start time of the Tournament must happen before the start time of the Battle
fact FirstTournamentThenBattle{
  all t:Tournament, b:Battles | (b in t.battles_available and b.Startstatus=TRUE)
  implies  t.startTime.timestamp<b.startTimeBattle.timestamp
}
```

```
//students can registrate in the tournament until the registration deadline
fact SubscriptionDeadline {
  all t: Tournament, s: Student , d: DateTime |
    s in t.participants implies d.timestamp <= t.registrationDeadline.timestamp
}

//The end time of the battle must happen before the end time of the toutnament
fact FirstBattleThenTournament{
  all t:Tournament, b:Battles | (b in t.battles_available and b.EndStatus = TRUE)
  implies  t.endTime.timestamp > b.endTimeBattle.timestamp
}

fact TestCaseRequiresBattle {
  all t: TestCase | some b: Battles | t in b.test_for_battle
}

fact endstatus{
all t:Tournament, b:Battles|(t.Closedstatus = TRUE ) implies b.EndStatus = TRUE  }

fact startstatus{
all t:Tournament, b:Battles| (t.Openstatus = TRUE ) implies b.Startstatus = TRUE}

/*CONSTRAINT FOR THE BADGE*/
//the same badge can be assigned to the same student only once
fact UniqueBadgeAssignment {
  all s: Student, b1, b2: Badge | (b1 in s.visualize_badge and b2 in s.visualize_badge) implies (b1 !=
b2)
}
fact BadgeVisibleAfterTournament {
  all t: Tournament, s: Student, b: Badge |
    (s in t.participants and b in s.visualize_badge) implies t.Closedstatus = TRUE
}
```

```
/* badges are assigned only when the battle is over*/
fact BadgeAssignmentAfterBattleCompletion {
  all s: Student, b: Badge, battle: Battles |
    (b in s.visualize_badge) implies
      battle.EndStatus = TRUE and battle.Startstatus=FALSE
}

fact oneBattleRulesForABattle{
          all br: BattleRules | one b: Battles | br in b.battle_rule
}

fact PersonalScoreVisibleAfterTournament {
  all t: Tournament, s: Student, sc: Score |
    (sc in s.personal_score) implies  t.Closedstatus = TRUE
}

fact BadgeVisibleAfterTournament {
  all t: Tournament, s: Student, b: Badge |
    (b in s.visualize_badge) implies t.Closedstatus = TRUE
}

//If one flag is TRUE, more educator can create new battles in the same tournament
fact moreEducatorforOneTournament{
 all e1,e2:Educator, t:Tournament |(e1!=e2 and t.setted_by_more = TRUE and e1 in t.educator)
implies e2 in t.educator
}
```

```
//Since the tournament is created, the user receives a notification
fact NotifyStudentsAfterBattleCreation {
  all t: Tournament, b: Battles, s: Student |
    b in t.battles_available and b.EndStatus=FALSE and s in t.participants implies
    some n: Notification |
      n.receiver = s
}

//Function to calculate the score
fun CalculateScore[s : Student]: one Int{
 let
          teams = {t : Team | s in t.student_in_team},
          team_score = teams.team_score,
          personalScore = team_score.score|
          sum personalScore
}

//ASSERT

// Check if the badge are uniquely assigned to each student
assert UniqueBadgeAssignment {
  all s: Student, b1, b2: Badge | (b1 in s.visualize_badge and b2 in s.visualize_badge) implies (b1 !=
b2)
}



// Check if the github repo link is unique for each battle
assert UniqueGitHubRepoLink {
  all b1, b2: Battles | (b1 != b2) implies b1.link_repo != b2.link_repo
}
```

```
// Check if the badge is visible only after the end of the torunament
assert BadgeVisibleAfterTournament {
  all t: Tournament, s: Student, b: Badge |
    (s in t.participants and b in s.visualize_badge) implies t.Closedstatus = TRUE
}

// Check if the personal score il visible after the Tournament is closed
assert PersonalScoreVisibleAfterTournament {
  all t: Tournament, s: Student, sc: Score |
    (sc in s.personal_score) implies t.Closedstatus = TRUE
}



check UniqueBadgeAssignment for 5
check UniqueGitHubRepoLink for 5
check BadgeVisibleAfterTournament for 5
check PersonalScoreVisibleAfterTournament for 5

pred show{
        #CodeKataBattle = 1
        #Tournament > 0
        #Student>0
        #Educator > 0

}

run show
```

```
Executing "Run show"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
  14713 vars. 783 primary vars. 37138 clauses. 168ms.
  Instance found. Predicate is consistent. 203ms.
```

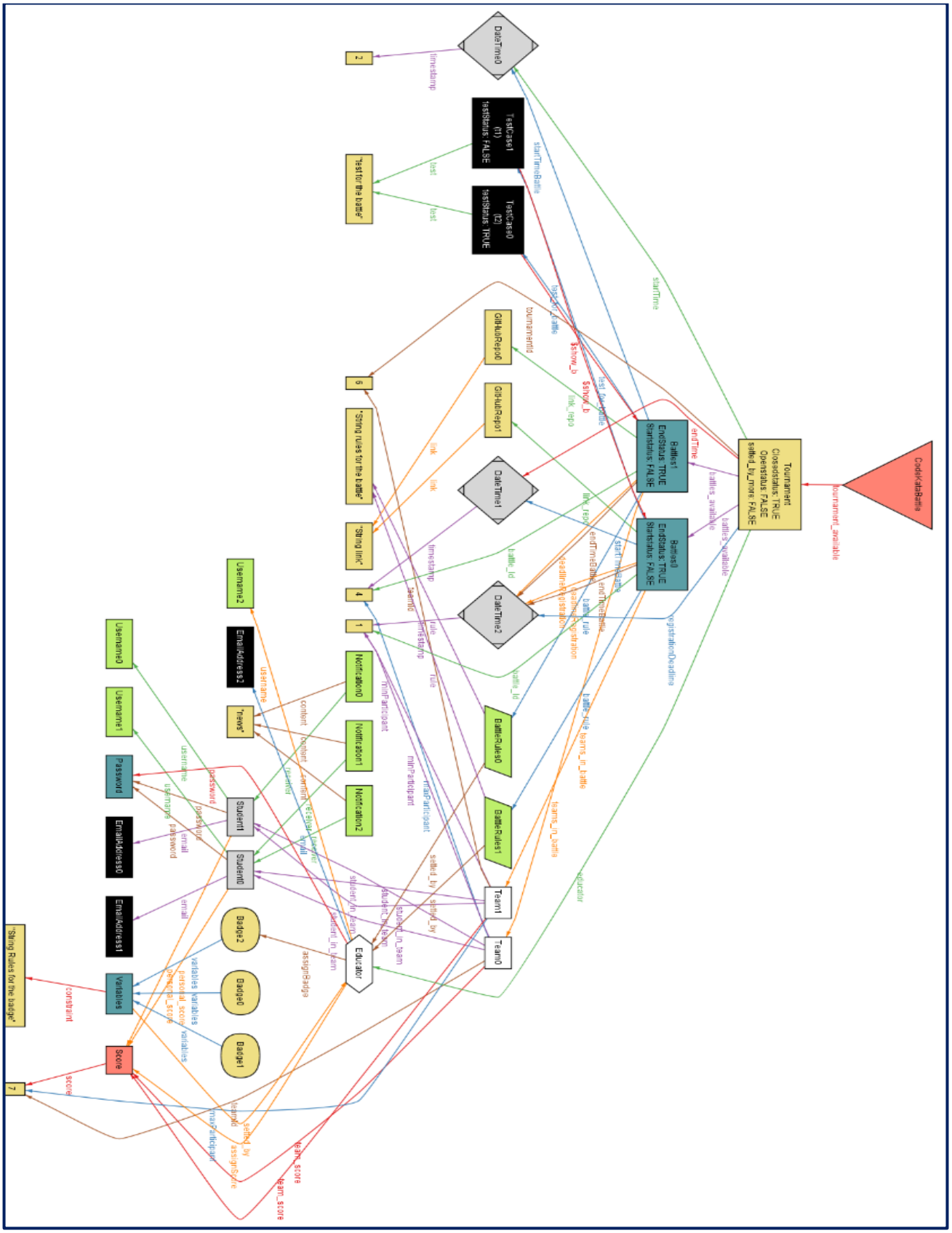


Figure 9 – Alloy World

5. Effort spent

The table in this section show the number of hours spent by each member of the group on each part of the project.

Task	Currò	Lombardo	Mondo
Introduction	8	8	8
Overall Description	15	15	15
Specific Requirements	20	20	25
Formal Analysis	15	15	10