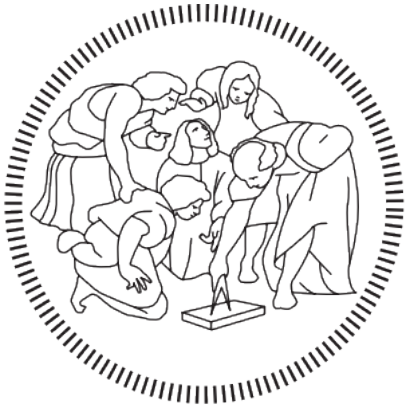

POLITECNICO
MILANO 1863

CodeKataBattle

Requirements Analysis and Specifications Document

Software Engineering 2 project
Academic year 2023 - 2024

20 October 2023
Version 1.0

Authors:
Tommaso Pasini
Elia Pontiggia
Michelangelo Stasi

Professor:
Matteo Camilli

Revision History

Date	Revision	Notes
23/10/2023	v.0.0	Document creation
18/12/2023	v.1.0	First release

Contents

1. Introduction

1.1 Purpose

Software development is one of the most sought-after skills globally, and its growing importance is set to persist during future years. The versatility of this ability allows solving many problems in different fields, including science, art, and entrepreneurship. The acquisition of software development skills improves both creativity and innovation, as well as fosters the development of fundamental skills such as problem-solving, logical thinking, and effective communication.

To understand and master software development, it is essential to have the ability to design, write, and test software programs. Dedication and patience are necessary for the gradual process of learning programming. Theory is a foundational element, but it is only the start of developing your skills. Consistent practice is necessary to master concepts and programming techniques.

1.1.1 Purpose of the product

The purpose of the product is to provide a solution to the problem previously highlighted. The **C**ode**K**ata**B**attle project, or CKB, is a platform that assists students in enhancing their software development abilities through battles. Students in teams participate in programming exercises where they complete software projects following a test-first approach. Creating battles, setting rules, and evaluating students' performance are tasks that educators do. CKB promotes skill development, encourages healthy competition, and facilitates assessment, fostering an environment that encourages collaboration and enhances competencies.

1.1.2 Goals

The following table provides an aggregate list of the specific goals that must be accomplished by CodeKataBattle system.

ID	Description
G1	Educators can create tournaments that involve coding battles to challenge students.
G2	Provides educators with the ability to track student software development knowledge.
G3	Students can improve their software development skills by taking part in coding tournaments and battles where they must write programs.
G4	Coding battles enable students to enhance their soft skills, such as communication, collaboration, and time management, by creating teams and collaborating with the members.

Table 1.1: Goals

1.2 Scope

CKB is a platform that allows students to improve their software development skills by participating in coding challenges. Educators create coding battles within specific tournaments, encouraging students to improve their programming proficiency. Students compete in coding battles to solve programming exercises in the defined programming language while following a test-first approach.

Educators create tournaments and battles by uploading a programming exercise (also known as Code Kata), defining group size limits by specifying both minimum and maximum participants, registration and submission deadlines, and scoring parameters. Once a battle is created, students can form teams, a team is a group of students that follows the size boundaries defined, if no minimum is established CKB accepts teams formed by only one person. Each team can be changed between different battles in the same tournament, and so also in different tournaments. When the registration deadline for the battle has expired, each team receives access to a GitHub repository containing the Code Kata. They are required to fork the repository and an automatic workflow through GitHub Actions, so from this point, students can work on their code.

The battle score ranges from 0 to 100 and is determined by both mandatory automated evaluation (test case pass rate, timeliness, source code quality) and optional manual evaluation (personal scores assigned by educators). The platform updates the battle score as students commit their code to GitHub, in this way students and educators can keep track of the battle's ranking. Following the deadline for submission, educators can perform an optional manual evaluation, if previously defined, before sharing the final battle rank with all the participants.

Each enrolled student in the tournament is assigned a personal score, calculated as the sum of their battle scores within it. This score is determined by CKB when

educators close a battle, and it is accessible to all platform users. When educators close a tournament, the final tournament rank remains available to all platform users.

Educator allows other colleagues to create battles within the context of a tournament. Educators can create gamification badges, which can be used in tournaments to reward students for their performance or achievement.

1.2.1 World Phenomena

ID	Description
WP1	Students participating in a tournament can decide whether to do a battle or not by subscribing or not to it.
WP2	Students choose for each battle if coding alone, when it is possible, or forming a team, respecting the limit imposed.
WP3	Only the student who formed the team forks the GitHub repository of the Code Kata.
WP4	Only the student who formed the team sets up an automatic workflow in the GitHub repository.
WP5	Student invites team members to its Code Kata repository.
WP6	Students work and compete in CKB with their code.
WP7	The students push their work to the GitHub repository.
WP8	An educator creates correct Code Kata, defining a coherent description of the project within its test cases and the configuration for automation scripts.
WP9	Educators check the work done by students in order to evaluate them, this is possible only if the manual evaluation option is chosen.

Table 1.2: World Phenomena

1.2.2 Shared Phenomena

ID	Description	Controlled by
SP1	A user registers their personal data in CKB system specifying if it is a student or an educator.	World
SP2	A registered user inserts its credentials to get into CKB environment.	World
SP3	An educator creates a tournament, defining all necessary details.	World
SP4	The educator who created a specific tournament grants other colleagues permission to create battles inside it.	World
SP5	An educator that has permission creates battles, defining all necessary details, within a tournament.	World
SP6	Students are notified of upcoming tournaments.	Machine
SP7	Students join tournaments.	World
SP8	Students are notified of upcoming battles within a tournament they are subscribed to.	Machine
SP9	Student joins battles.	World
SP10	Student invites other students, who are subscribed in the same tournament, to join its team respecting the boundaries imposed.	World
SP11	Student joins a team, it gets enrolled in a battle via an invite by another student if it was not already part of it.	World
SP12	CKB platform sends the link of the GitHub repository to the students who created the team for the battle.	Machine
SP13	Students who received the GitHub repository link are asked to fork it and set up an automated workflow.	Machine

SP14	The forked repository's workflow notifies the platform of a new GitHub push action, performed by students.	World
SP15	The platform updates the battle score of the students whenever their repository gets pushed.	Machine
SP16	Educator and student subscribed to the battle can monitor the battle ranking among other participants.	World
SP17	The educator uses the platform to go through the sources produced by each team.	World
SP18	The educator, in its own battles, manually evaluates the work done by students.	World
SP19	The platform notifies students of the end of a battle as soon as the final battle rank becomes available.	Machine
SP20	An educator that has permission to create battles within a tournament, can also closes that tournament.	World
SP21	The platform notifies all students involved in the tournament about its end when the tournament rank is available.	Machine
SP22	At any time, all users can see the list of ongoing and ended tournaments as well as the corresponding tournament rank.	World
SP23	An educator creates gamification badges inside CKB platform, defining a name, some variables, and a rule.	World
SP24	A user checks the profile of any student subscribed to the CKB platform.	World

Table 1.3: Shared Phenomena

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Term	**Definition**
User / Actor	A person who uses CKB platform, could be a Student or an Educator.
Educator	Identifies a person who provides instruction or education, such as a teacher.
Student	Identifies a person who is studying at school or college.
Kata	A training exercise system for karate where you repeat a form multiple times, making small improvements to each one.
Test-first approach	A software development process based on converting software requirements into test cases before creating the software, and then tracking the entire development process by repeatedly testing the software against those test cases.
Code Kata	Programming battles in which teams of students compete against each other.

Table 1.4: Definitions

1.3.2 Acronyms

Acronyms	**Term**
CKB	CodeKataBattle
EDU	Educator
STU	Student
IDE	Integrated Development Environment
ESP	Email Service Provider
API	Application Programming Interface
UML	Unified Modeling Language

Table 1.5: Acronyms

1.3.3 Abbreviations

Abbreviation	**Term**
$\mathbf{G}_i$	i-th goal
$\mathbf{WP}_i$	i-th World Phenomena
$\mathbf{SP}_i$	i-th Shared Phenomena
$\mathbf{DA}_i$	i-th Domain Assumption
$\mathbf{Dep}_i$	i-th Dependencies
$\mathbf{R}_i$	i-th Requirement
$\mathbf{UC}_i$	i-th Use Case
i.e.	in other words
e.g.	for example
iff	if and only if

Table 1.6: Abbreviations

1.4 Reference Documents

- Assignment RDD A.Y. 2023-2024[1]
- Course slides on WeeBeep[2]
- ISO/IEC/IEEE 29148 dated 2018,
  Systems and software engineering - Life cycle processes - Requirements engineering[3]

1.5 Document Structure

The structure of this RASD document follows five main sections:

1. **Introduction:** provides an overview of the problem at hand, the purpose of the project, the scope of the domain, and introduces the main goals of the system as a solution.
2. **Overall Description:** gives a general description of the system, going into more detail about its main functions. The description is assisted with the help of UML diagrams, such as class, activity, and state diagrams. The domain assumptions of the examined world are then explained along with any dependencies and constraints.
3. **Specific Requirements:** specifies the functional and non-functional requirements of a software system. It includes use case diagrams, descriptions of each use case, and related sequence diagrams. Finally, it provides a mapping of the requirements for both goals and use cases.
4. **Formal Analysis Using Alloy:** contains Alloy models which are used for the description of the application domain and its properties, referring to the operations that the system has to provide and some critical aspects of the system.
5. **Effort Spent:** keep track of the time spent to complete this document. The first table defines the amount of hours used by the whole team to make important decisions and to make reviews, the other tables contain the individual effort spent by each team member.

[1] `https://webeep.polimi.it/mod/folder/view.php?id=219353`
[2] `https://webeep.polimi.it/mod/folder/view.php?id=207692`
[3] `https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29148:ed-2:v1:en`

2. Overall Description

2.1 Product Perspective

2.1.1 Scenarios

Creating a tournament

Chip, a professor of Algorithm and Data Structures at Mouseton Institute of Technology prepared to teach the chapter on strings, launching the "Strings Operations" coding tournament on CKB. To expand participation, he allowed his colleague Dale to create challenges for his software engineering class. STUs across classes would compete in string manipulation tasks, ranging from basic concatenation to advanced text analysis, fostering collaboration and learning. To make the tournament more interesting, Chip decided to award badges to the best-performing STUs, so he added badges for the STUs who participate in most tournaments, one for the STUs who win most battles, and one for the STUs who write most lines of code. All STUs already subscribed to CKB were notified of the new tournament, and they could join it from the tournament page till a defined deadline.

Creating a battle

In order to familiarize STUs with the CKB platform and its features, Chip created an easy battle for his STUs to practice, called "Wordcheck". The task essentially required STUs to implement the game Wordle in the C language. He decided that the battle would last for two weeks, allowing STUs to work in teams of 1 up to 3 people. STUs would be able to join the battle until the deadline's last day.
In addition, he wanted to give extra points for code cleanup. Therefore, he had to review the code of each team at the end of the battle and assign extra points to the teams that wrote clean code. Chip set all this information in the battle creation form and then created the battle.

Joining a battle

Huey and Dewey, two STUs of Chip's class, are notified of an incoming battle and decide to join it. Since the more the merrier, they decide to invite their friend Louie to join

them in the battle. Louie receives the invitation mail and decides to join the battle in their team. After the registration deadline, they are notified that the battle is about to start. They get the link to the GitHub repository of the battle to fork it and then set up an automated workflow to link their GitHub account to the CKB platform.
After the automated workflow is set up, they are ready to start working on the battle.

Improving the score and obtaining a badge

Donald is another warrior of the "Wordcheck" battle and he is working on the battle alone. After the first commit, he logs in to the CKB to check his score. He sees that he is in the 3rd position and that he is 10 points behind the leading team, composed of Huey, Dewey, and Louie. Fortunately, the battle is still in progress and the CKB platform allows him to improve his score by pushing new commits to the GitHub repository, so he decides to work on the battle for a couple of days and then push his updated work to the GitHub repository. After checking his score again, he is now in the 1st position and moreover, he obtained a badge for being the first to reach 100 points in the battle. From now on, both STUs and professors can see this badge when they visit Donald's profile.

Closing a battle

When the deadline for the battle created by Chip is reached, all participants are notified that the battle is closed and that they can not push new commits to their GitHub repository. Chip can now evaluate the code of each team and assign extra points for the clarity of the comments and the code, as he decided when he created the battle. After the evaluation, the final rank of the battle is available to all participants, and the STUs are notified that they can now see the final rank of the battle.

Closing a tournament

Chip decides to close the "Strings Operations" tournament when all the battles end. To do so, he logs into the CKB platform and he closes the tournament. All participants are notified that the tournament is closed, in the end, the CKB platform makes the final rank of the tournament available to all participants.

Accessing the scores of the players

Huey wishes to enroll in the class Advanced Algorithms and Data Structures held by Professor Pippo, so he applies for the class. Pippo, who wants to make sure that Huey is a good STU, comes to know that Huey is a very active user of the CKB platform and he decides to check his profile. He sees that Huey has a very high score in the "Strings Operations" tournament and that he has a badge for being the most active user of the platform, he also notes that Huey is involved in more than one tournament simultaneously. Thanks to the CKB platform, Pippo now has a complete overview of Huey's skills and he can decide whether to accept his application or not.

Creating Game badges

Scrooge, a Software Engineering professor at Duckburg University, believes that, in addition to the existing badges in the CKB platform, it would be nice to introduce badges for STUs who achieve a perfect score in a battle and a tournament. Since the CKB platform allows EDUs to create new badges, he creates the two badges and from now on, all EDUs will have the option to include these badges in their tournaments and battles.

2.1.2 Class Diagram

Figure 2.1 represents a simplified view of the UML class diagram of the system. This is not a comprehensive view of all the classes needed, but rather a view to capture the general composition and the different relations between them. In particular, the most important details are:

- There are only two possible types of users: STU and EDU type.
- EDUs can create tournaments and battles, moreover, they can grant permission to create battles to other colleagues inside their own tournament.
- STUs can subscribe to tournament and battles, invite other STUs to form teams, and achieves badges, using both the CKB platform and GitHub. The most important detail is subscribing to a battle: STUs can join a battle by creating their own team (a team is composed respecting imposed boundaries) or by joining an existing team. In particular, the team class exists iff a battle exists, i.e. a team exists only in the battle scope.
- Tournament and Battle classes use the MailAPI to notify STUs about events.

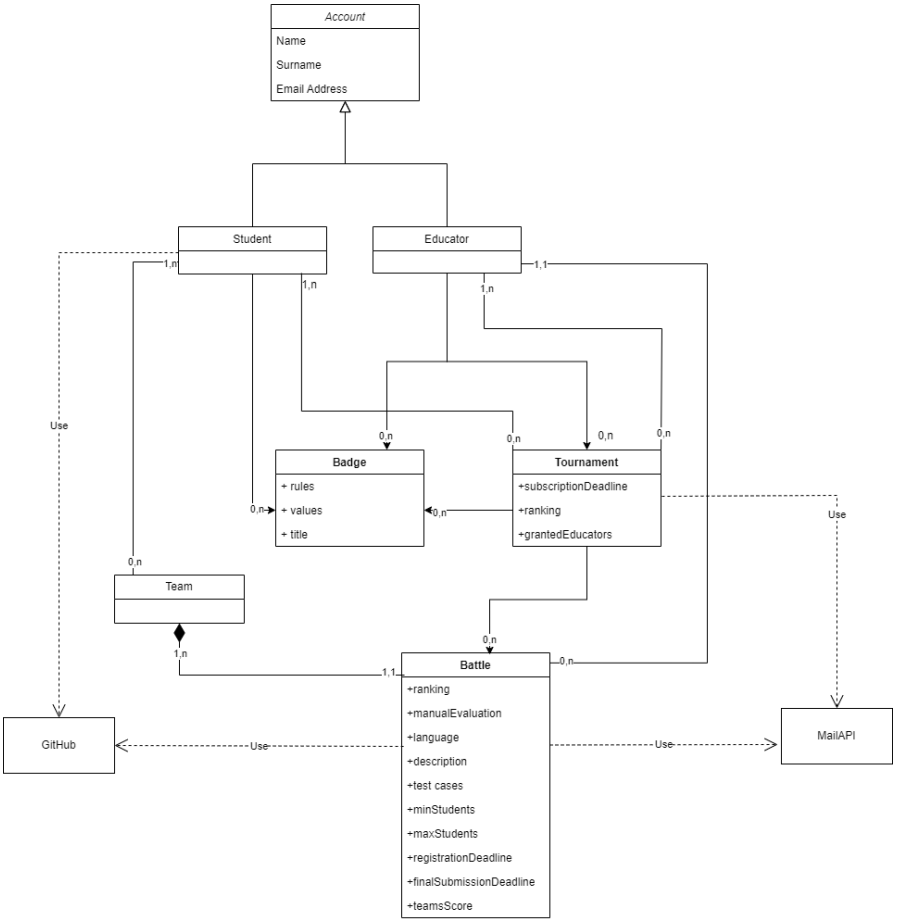

Figure 2.1: Class Diagram

2.1.3 State Diagrams

The following state diagrams describe the life cycle of the main entities of the system. Moreover, they specify the sequence of states that an object goes through during its lifetime in response to stimuli from the environment. We want to focus on the events that cause a transition from one state to another and the actions that result from a state change.

Tournament

After an EDU creates a tournament, it is both in the *registration open* and *tournament open* states.
In the *registration open* state, STUs can join the tournament, while in the *tournament open* state, EDUs with the right permissions can create battles within the tournament, and that leads the tournament to the *battling* state.
When the deadline for registration is reached, the tournament moves to the *registration closed* state and no more STUs can join it.
When the deadline for the registrations is reached, no more STUs can join the tournament and it moves permanently to the *registration closed* state.
During the *battling* state EDUs can start multiple parallel battles or can finally close the tournament, iff all battles are ended.
The diagram is shown in figure 2.2.

Battle

The battle evolves linearly, starting from the *registration open* immediately followed by the *registration closed* state.
After the registration deadline is reached, the GitHub repository of the battle is created and thus the battle moves to the *coding* state, allowing the STUs to fork the repository and start working on the battle.
When the deadline for the battle is reached, the EDUs can start evaluating the code of the STUs, if previously enabled (*consolidation* state).
After the evaluation is completed, the battle can be closed and the final rank is available to all participants.
The diagram is shown in figure 2.3.

Score evaluation

The score evaluation of a battle is a process that is triggered by the end of a battle and it is composed of multiple steps.
First, three aspects can be automatically evaluated: functional aspects (the higher the better, +), timeliness (the lower the better, -), and quality level of the sources, extracted through static analysis tools (+).
Finally, if the EDU enabled the manual evaluation, it can assign extra points.
The diagram is shown in figure 2.4.

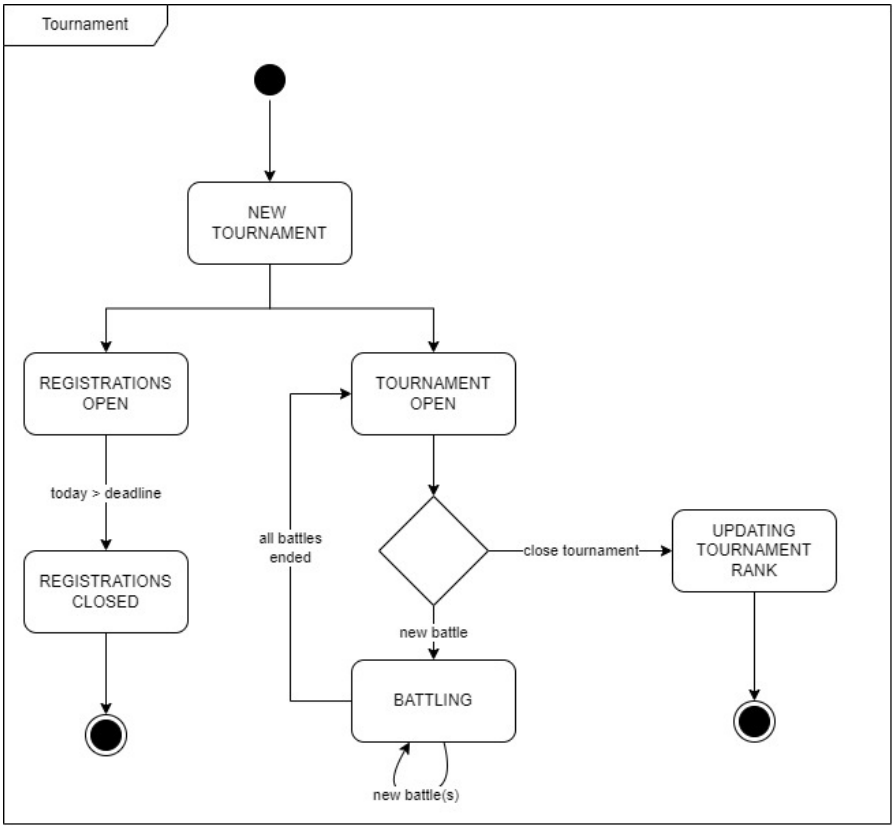

Figure 2.2: Tournament state diagram

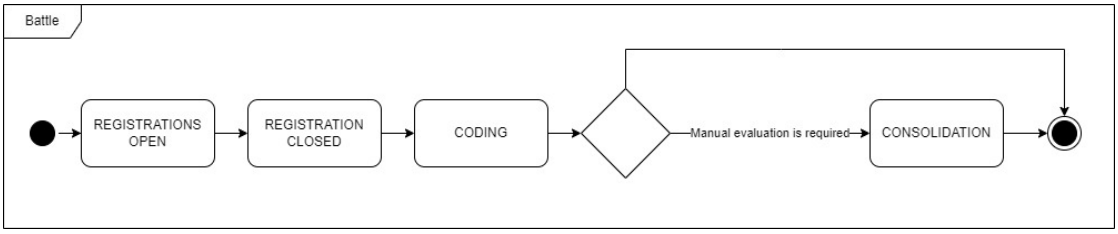

Figure 2.3: Battle state diagram

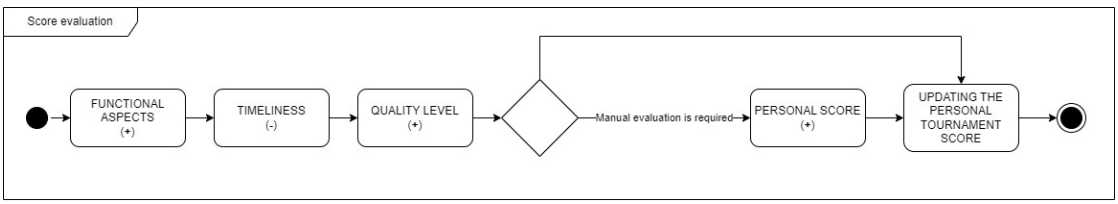

Figure 2.4: Score evaluation state diagram

2.2 Product Functions

2.2.1 Register function

A user approaching CKB for the first time can register on the platform. The information necessary for the system to keep track of users is personal data such as Name, Surname, Email, School they belong to, and what role they hold in the school environment (STU / EDU). This last information is very important as it guarantees two types of accounts with different rights and duties.

By expressing that it is an EDU, all rights related to the creation of tournaments, battles, and badges are guaranteed. By choosing a STU account, it can actively participate in tournaments and battles and earn badges.

2.2.2 Create tournament function

To create a tournament, an EDU registered in the CKB platform must define all the required information. The tournament needs:

- a name.
- a time window aimed at welcoming STU registrations.
- a list of badges that STUs can obtain during the whole tournament duration.

2.2.3 Creating battles function

In the context of an active tournament, any authorized EDU can decide to create a battle that will be managed by itself. New battles require:

- the programming language to solve the problem.
- the test cases that must be passed by each team's code, at the end of the battle.
- the build automation scripts correctly set.
- the specification of the problem to be solved including at least one example to achieve a test-first approach.
- the deadline for the battle registration period.
- the deadline for the conclusion of the battle.
- the type of evaluation method, in particular, set manual evaluation in addition to the automatic one performed by the platform.
- constraints for the maximum and minimum number of players required for each team.

2.2.4 Join tournament and battles function

STUs registered on the platform receive notification every time a new tournament is created and can decide whether to participate. Likewise, in the context of a tournament in which they applied, they are informed of new upcoming battles. Registration for a battle can be done in various ways as long as it is before the end of the registration window.

2.2.5 Gamification function

EDUs, at any time, can create a new badge by defining a title and a rule. Once a badge is created, it will be available on the CKB platform for every EDU who wishes to use it in a new tournament. When creating a tournament, the EDU specifies the list of included badges from those available. At the end of the tournament in which they are enrolled, STUs receive the badges for which the specified rule has been satisfied. The same badge can be assigned to more than one STU.

2.3 User characteristics

Users of the system fall into one of the following categories: STU or EDU.

STU

STUs participate actively in Code Kata tournaments and battles to enhance their software development skills. During a battle, STUs develop solutions following the "test-first" approach and use GitHub to manage their code. The CKB platform automatically evaluates STU progress based on the number of tests passed, timeliness, and quality of code, providing scores updated in real-time. STUs aim to achieve high scores and accumulate gamification badges defined by EDUs. In addition to participating in battles, STUs can view their tournaments' rank and receive notifications on final results. The STU's primary goal is to improve programming skills, obtain competitive scores, and earn badges through active participation in the collaborative context of the CKB platform.

EDU

EDUs take a central role in directing STUs toward improving software development skills through programming competitions. Their duties include creating tournaments and battles, finalizing challenge details, and managing STU registrations. EDUs assign specific tasks to STUs, and then they manually evaluate STUs' solutions, contributing to the automatic evaluation of projects performed by the CKB platform of functional, temporal, and code quality aspects. Furthermore, they can create gamification badges, and rewards based on rules established, to motivate STUs. The main objective is to improve STUs' skills, ensuring fair and efficient assessment and encouraging active involvement. The EDU plays a key role in educational innovation, exploring new possibilities through the

definition of personalized rules and badges that stimulate growth and collaboration. In summary, the EDU acts as a promoter of an engaging and competitive training challenge on CKB.

2.4 Assumptions, Dependencies and Constraints

2.4.1 Domain Assumptions

ID	Description
DA1	STUs code with the programming language set for the battle they are taking part.
DA2	EDUs upload the Code Kata with the correct description and software project, including test cases, and build automation scripts related to it.
DA3	STUs fork the GitHub repository of the Code Kata and set up an automated workflow through GitHub Actions that informs the CKB platform (through proper API calls) as soon as STUs push a new commit into the main branch of their repository.
DA4	EDUs manual evaluation ranges from 0 to 100[1].
DA5	The data inserted, by users, at registration time, are truthful.
DA6	GitHub and the tool for static analysis always work properly and they are reliable.
DA7	A team is composed of at least the minimum number of people up to the maximum number defined by EDUs, if no minimum is defined it will be 1 by default.
DA8	All users subscribed to the CKB platform have a GitHub account.

Table 2.1: Assumption

2.4.2 Dependencies

ID	Description
Dep1	The system requires an internet connection to interact with CKB and other users.
Dep2	The system integrates an external API to compile the code written by STUs.
Dep3	The system integrates a GitHub API to create a repository for each battle

Table 2.2: Dependencies

2.4.3 Constraints

- The software must follow local laws and rules, especially when it comes to handling user data, such as letting users access their data when they want.
- The software should only collect personal data it really needs, like just the user's name, email address, and the school it belongs to and few more.
- To keep users' important info safe, like passwords and personal data, it must be stored in SHA256 encoding in the database.
- When choosing external APIs, especially those that are crucial for it to work properly, we should pick the ones that are the most dependable and always available.

3. Specific Requirements

3.1 External Interface Requirements

This section presents all the functional requirements of the CKB platform. They outline how the system interacts with other components. It follows a description of software and hardware interfaces for the system.

3.1.1 User Interfaces

Here it is shown the mockup of the main web pages of the system.

Common views

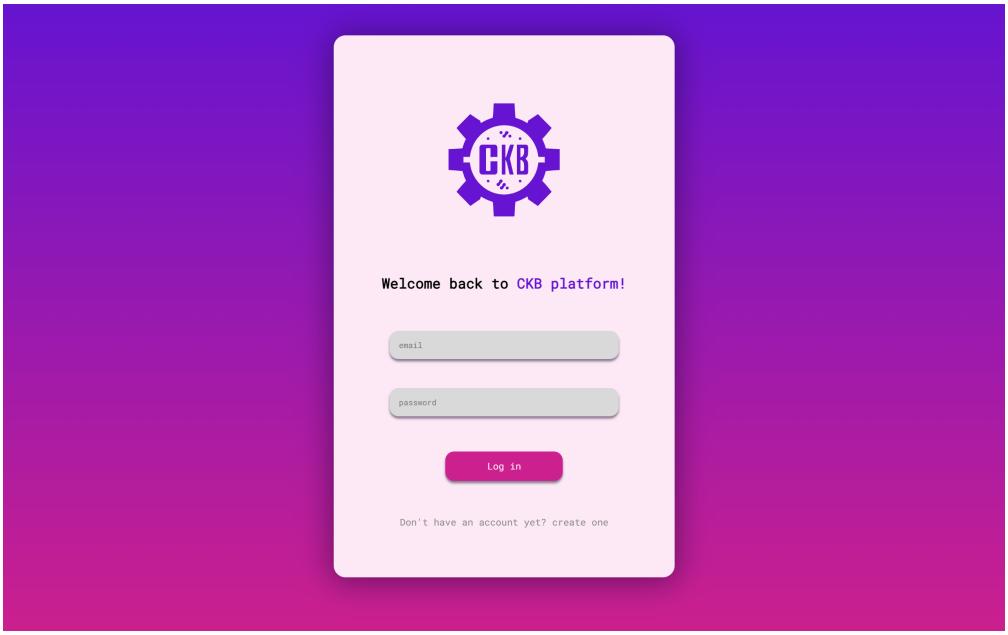

Figure 3.1: Log in page

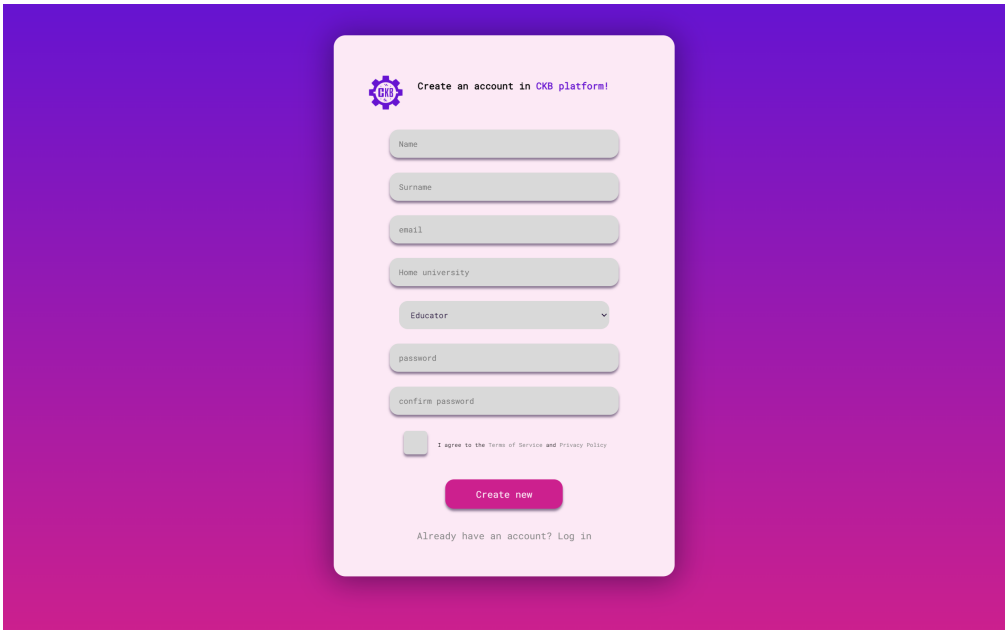

Figure 3.2: Sign up page

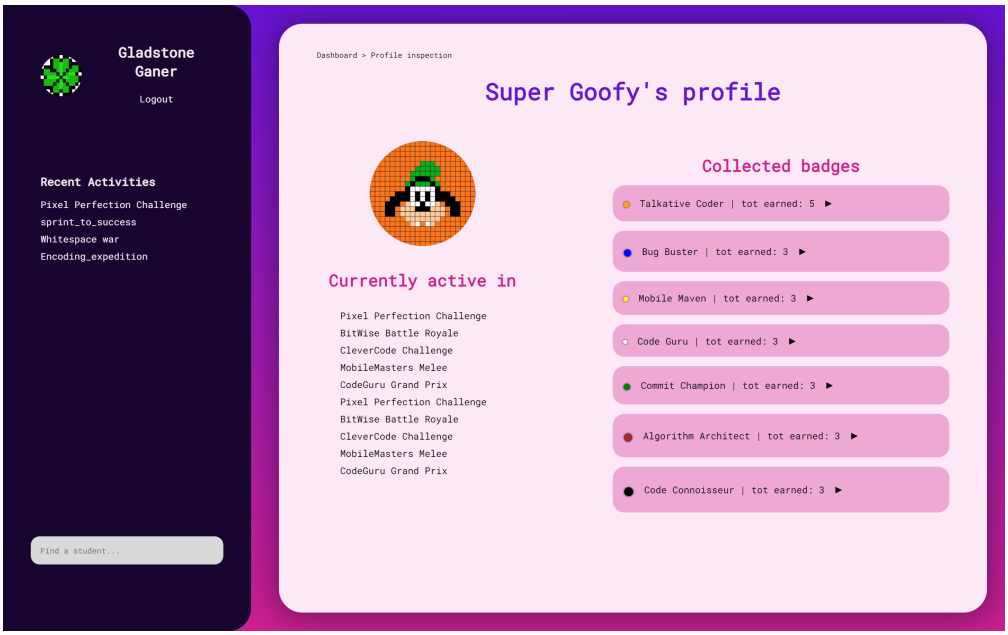

Figure 3.3: Page that shows badges acquired by a STU

EDU's views

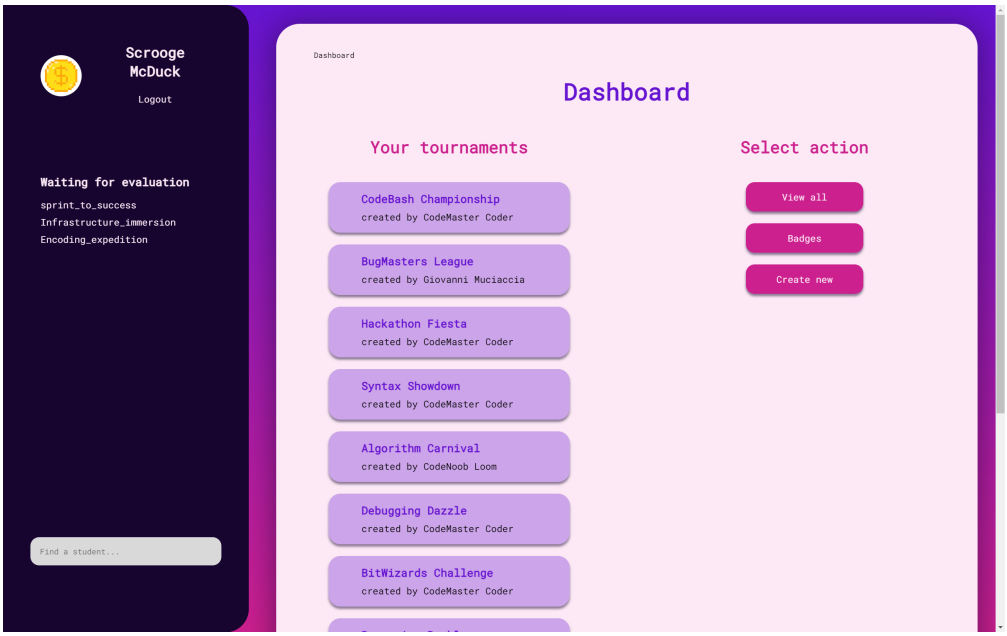

Figure 3.4: Dashboard

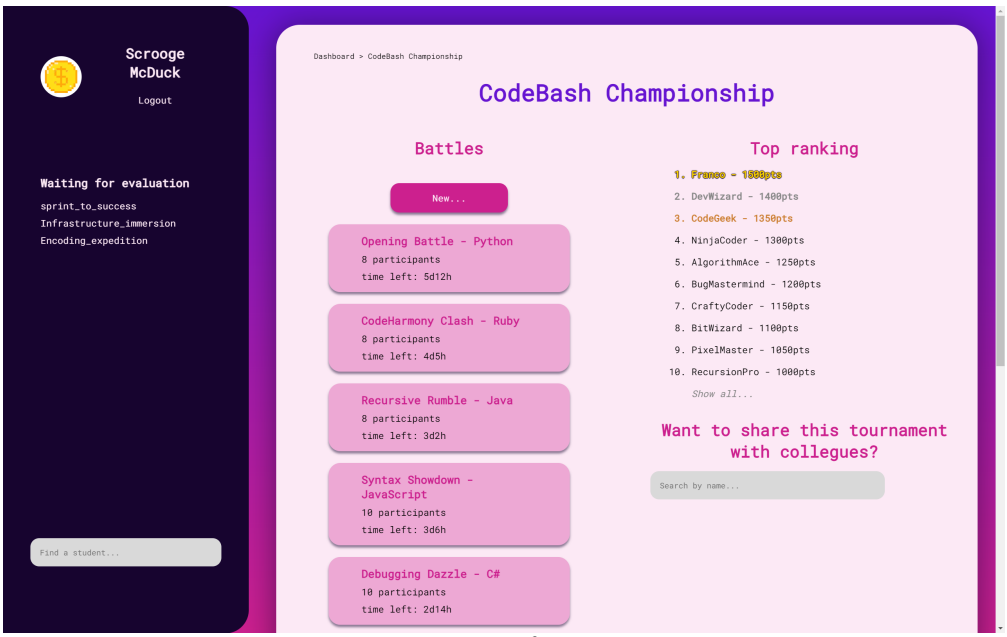

Figure 3.5: View of a tournament

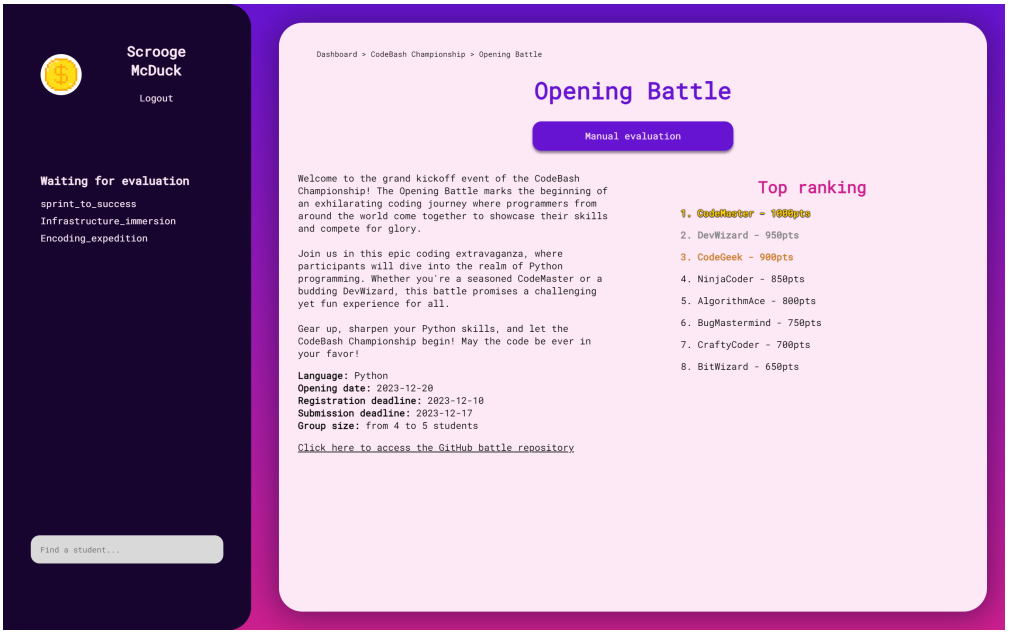

Figure 3.6: View of a battle

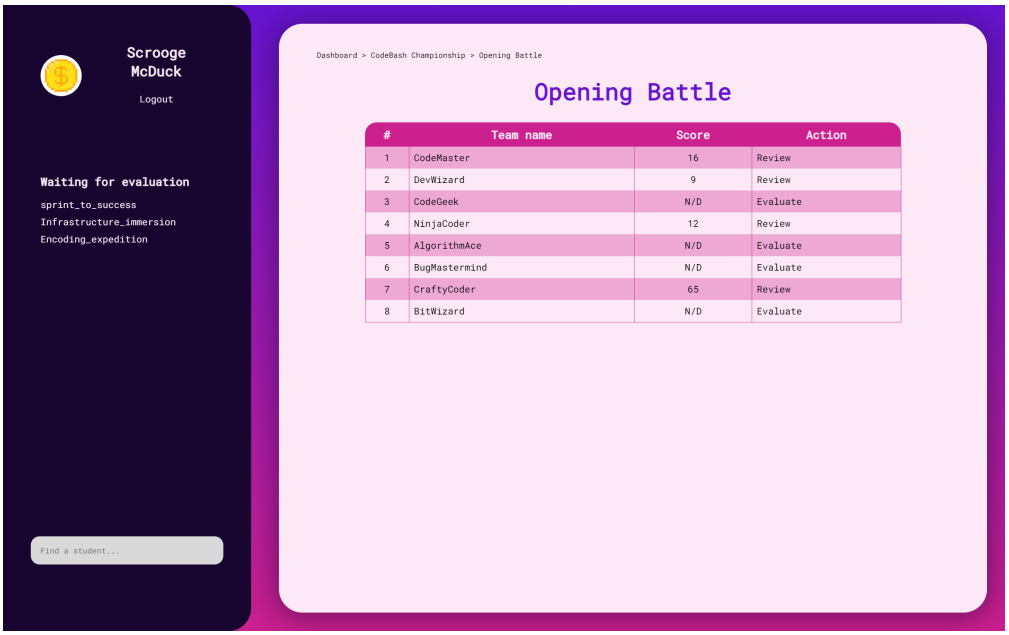

Figure 3.7: Manual evaluation table

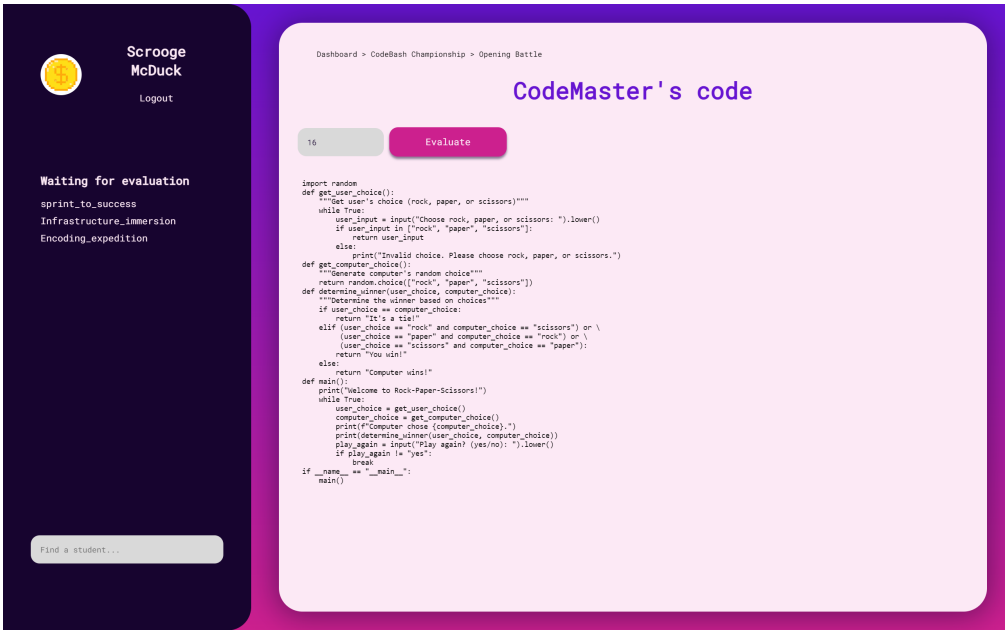

Figure 3.8: Manual evaluation of a team's code

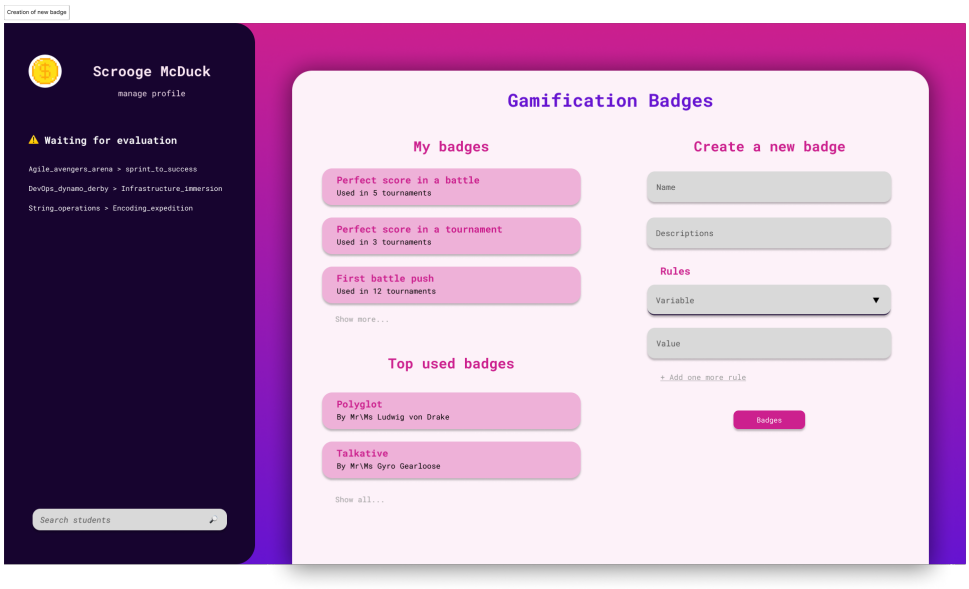

Figure 3.9: Page to create a new badge

STU's views

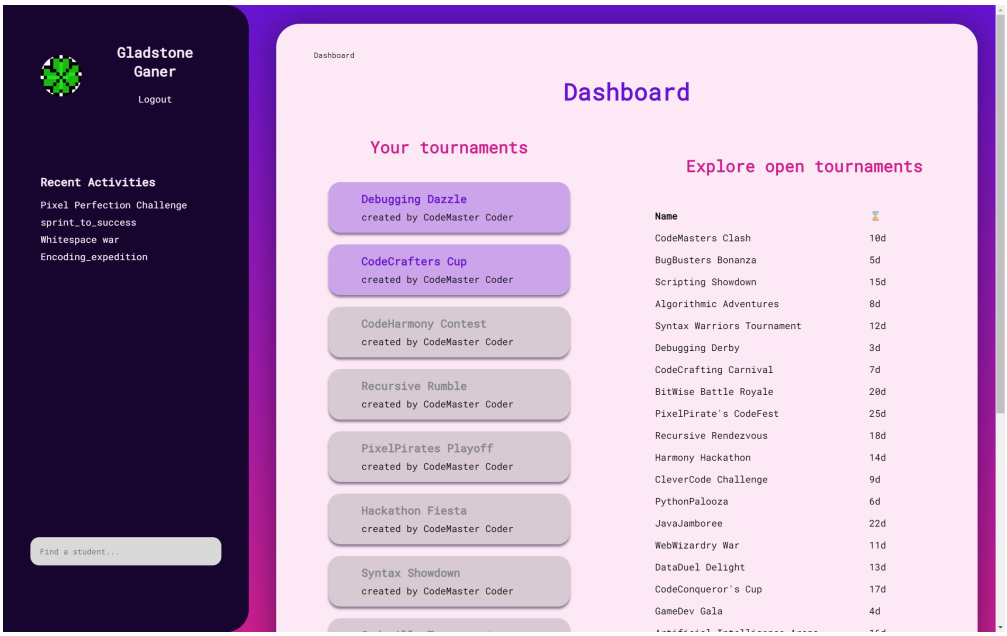

Figure 3.10: Dashboard

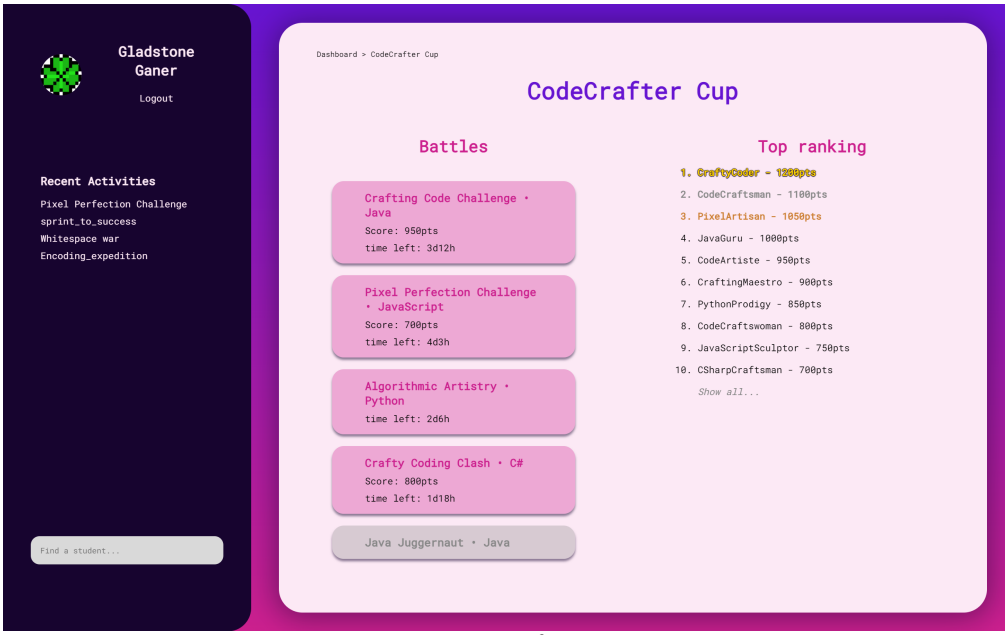

Figure 3.11: View of a tournament

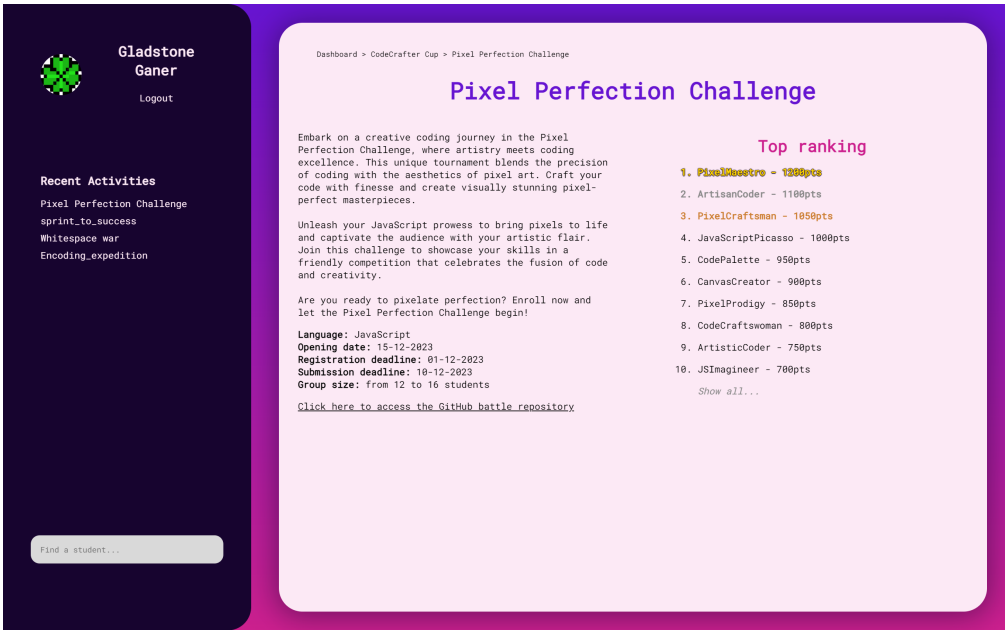

Figure 3.12: View of a battle

3.1.2 Hardware Interfaces

To use the CKB platform, both the Educators and the Students need an electronic device connected to the Internet, like a computer, a tablet, or a smartphone.
As the platform's primary functionality is closely tied to coding activities, it is expected that users will predominantly employ personal computers to access an Integrated Development Environment (IDE). Consequently, the platform's interfaces have been optimized for use on computer screens.

3.1.3 Software Interfaces

Since the platform is web-based, it is compatible with all the major operating systems, as long as they have a modern browser installed.

3.1.4 Communication Interfaces

The system requires a stable internet connection to work properly. The backend of the system will expose a unified RESTful API to communicate with all clients.
Furthermore, the system relies on various external interfaces accessible via uniform web API. These services are:

- **GitHub API:** to create and manage repositories and to retrieve students' code.
- **Mail API:** to send emails to the users to notify them about events.

3.2 Functional Requirements

To work properly, the software must fulfill the following functional requirements, which are written in hierarchical order, starting from those about EDUs and then STUs.

3.2.1 Use cases Diagrams

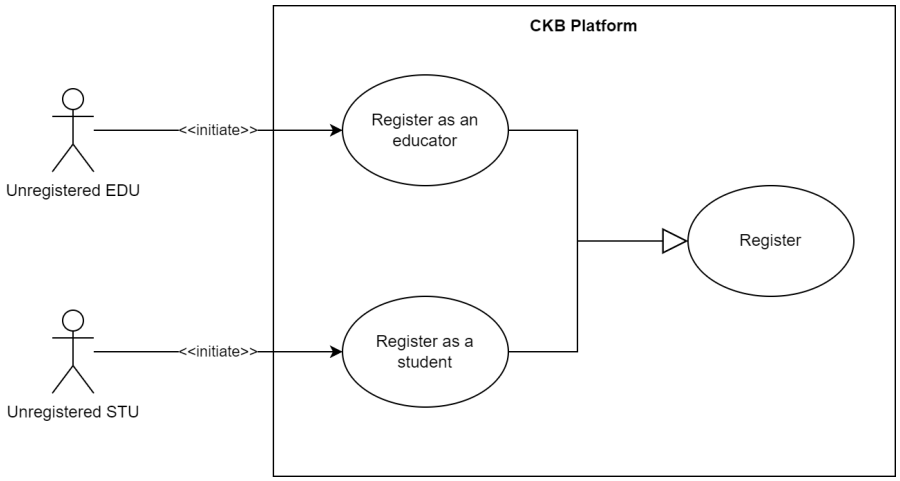

Figure 3.13: Registration Use Case Diagram

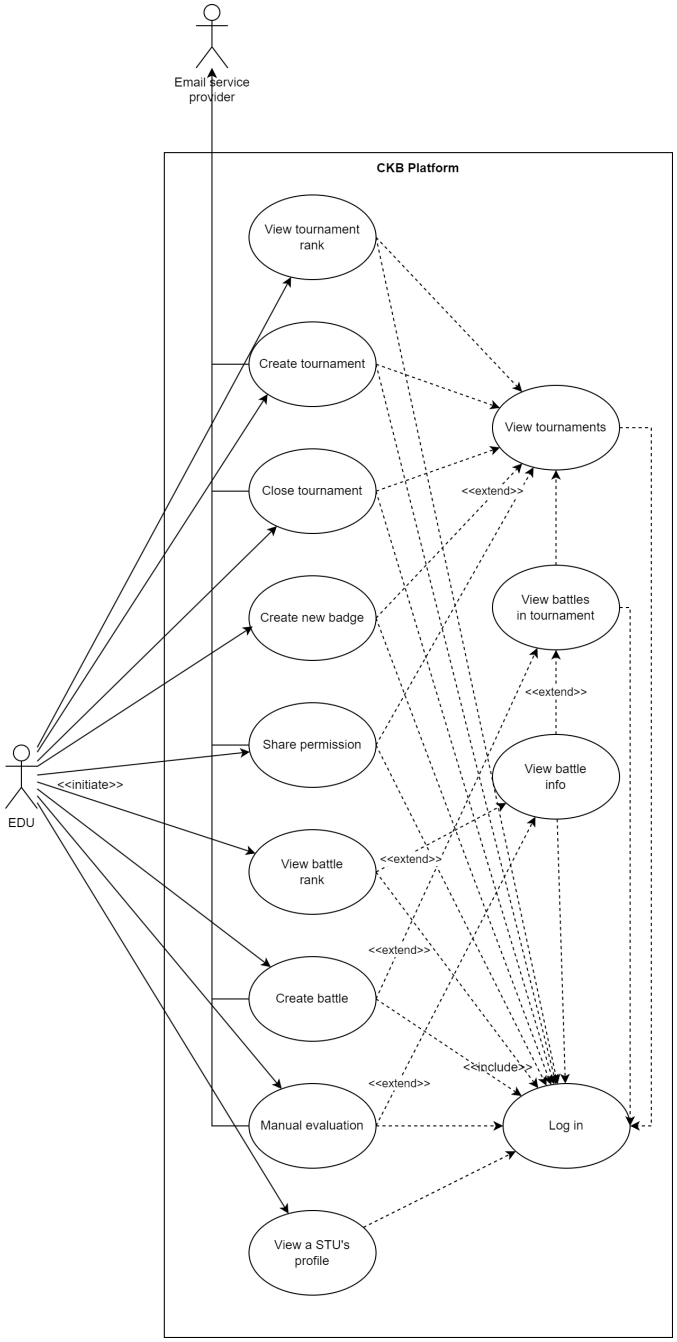

Figure 3.14: EDU Use Case Diagram

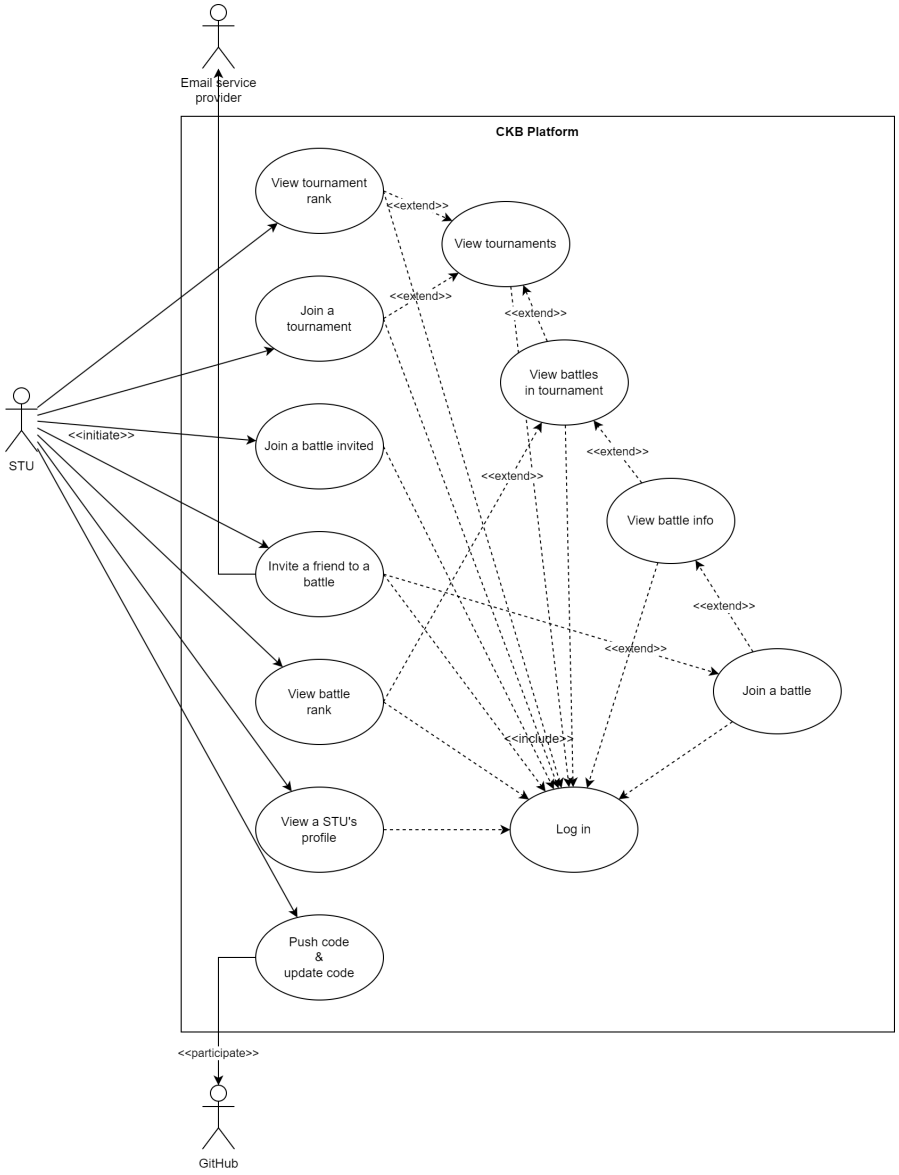

Figure 3.15: STU Use Case Diagram

3.2.2 Use cases Description

The following use cases within the exception section do not contain, for the sake of enhancing readability and simplifying the sequence diagrams, these anomalies:

1. Verification of the existence of the actor in the CKB system each time said actor undertakes an action.
2. Verification of the possession of all requisite permissions by an actor before the execution of an action.
3. Validation of submitted forms, to ensure the absence of data with illegal formats or empty fields.
4. Assurance of the atomicity of each transaction, signifying its completion or abort.

It is imperative to note that in the event of any anomalies related to these scenarios, appropriate error messages will be displayed on the user interface.

Name	Register
ID	UC1 (Figure 3.16)
Actors	EDU or STU
Entry conditions	The actor is not registered and wants to create an account
Event flow	1. The actor enters the registration page 2. The system shows the registration form 3. The actor specifies whether is an educator or a student 4. The actor fills out the form with truthful information 5. The actor confirms the account creation 6. The system creates an account concerning the information inserted by the actor
Exit conditions	The system successfully creates the account and shows the "Log in" page
Exceptions	• The actor inserts information (e.g. email) about an account that already exists: a human-readable message is displayed

Name	Log in
ID	UC2 (Figure 3.17)
Actors	EDU or STU
Entry conditions	The actor is already subscribed to the CKB platform
Event flow	1. The actor enters the "Log In" page 2. The system shows the login form 3. The actor fills the form with its credentials and submits it 4. The system checks the credentials and logs the actor in
Exit conditions	The actor is logged in and the "Dashboard" page is displayed
Exceptions	• The actor inserts a wrong combination of username - password: a human-readable message is displayed • The actor is not registered so the username does not exist: a human-readable message is displayed

Name	Create tournament
ID	UC3 (Figure 3.18)
Actors	EDU and ESP
Entry conditions	The actor is logged in as an EDU **and** it is in the "Dashboard" page
Event flow	1. EDU clicks on the "Create new" button 2. The system shows the "Create tournament" page with a form to be compiled 3. EDU fills the form with the required information as Tournament name, STUs' registration window, and the optional list of badges 4. EDU clicks on the "Create" button 5. The system notifies all the registered STUs about the creation of a new tournament, by sending an email through ESP 6. EDU is redirected to the "Dashboard" page showing the new tournament created
Exit conditions	The system has created the new tournament
Exceptions	• The EDU inserts a Tournament name already used for another tournament: a human-readable message is displayed • The EDU specifies a deadline in the past: a human-readable message is displayed

Name	Create Battle
ID	UC4 (Figure 3.19)
Actors	EDU and ESP
Entry conditions	The actor is logged in as an EDU **and** it is in the "Dashboard" page **and** it has the permission to create battles inside a specific tournament
Event flow	1. The EDU clicks on the tournament in which it wants to create a new battle 2. The system shows the "Tournament View" page with the "Create new battle" button 3. The EDU clicks the "Create new battle" button 4. The system shows the "Battle creation" page with a form to be compiled 5. The EDU fills the form with the details about the battle (e.g. battle name, code language, deadlines) and uploads the Code Kata 6. The system checks if all fields are correctly filled and then creates the battle
Exit conditions	All STU subscribed to the tournament in which the battle is created get notified through ESP, that a new battle is available
Exceptions	• The EDU does not have enough permissions for the selected tournament: the system does not show the "Create new battle" button

Name	Share permission
ID	UC5 (Figure 3.20)
Actors	EDU
Entry conditions	The actor is logged in as an EDU **and** it is in the "Dashboard" page **and** a tournament is in progress
Event flow	1. The EDU selects a tournament it has created 2. The system shows the "Tournament View" page 3. The EDU clicks on the "Share permission" button 4. The system shows the list of all the EDUs subscribed to the CKB platform 5. The EDU searches the EDU(s) to which it wants to share the permission 6. The user adds an EDU to the tournament by clicking the "Add" button
Exit conditions	The other EDU(s) can now create battles within that tournament
Exceptions	• The actor opens a tournament that does not exist: a human-readable message is displayed • The actor searches for an EDU that does not exist in the system: a human-readable message is displayed • The actor adds an EDU already part of the tournament: a human-readable message is displayed
Note	To make the research of a specific EDU easier, the system provides a search bar

Name	Join a tournament
ID	UC6 (Figure 3.21)
Actors	STU
Entry conditions	The actor is logged in as a STU **and** it has received an e-mail from about a new tournament created
Event flow	1. STU opens the e-mail and clicks on the link attached to be redirected to the tournament 2. The system shows the "Tournament View" page 3. STU clicks on the "Join Tournament" button 4. The system processes the request and confirms the enrollment
Exit conditions	STU is now part of that tournament
Exceptions	• The actor opens a tournament that does not exist: a human-readable message is displayed • The tournament registration window is expired: a human-readable message is displayed • The STU is already part of the tournament: a human-readable message is displayed

Name	Join a Battle
ID	UC7 (Figure 3.22)
Actors	STU and ESP
Entry conditions	The actor is logged in as a STU **and** a tournament is in progress **and** the STU is subscribed to that tournament
Event flow	1. The STU clicks on the tournament in which it wants to battle 2. The system shows the "Tournament View" page with a list of battles 3. The STU clicks on the battle it wants to join 4. The system shows the "Battle View" page with all the information about it, if the subscribing deadline has not expired yet, the system shows the "Join Battle" button 5. The STU clicks on the "Join Battle" button to enter the battle 6. The system displays a form to create a team and invite other members 7. The STU, if wants or must invite friends to be part of its team, inserts their emails and then presses the "Join" button 8. The system sends an email to each invited member
Exit conditions	The system subscribes the STU to the battle that now waits for the reply of other STUs or can invite other mates to the team
Exceptions	• The STU wants to invite a number of mates greater or less than the boundaries imposed: a human-readable message is displayed • The deadline expires and the team does not satisfy the constraint for the minimum team participants: the team is not enrolled in the battle • The emails inserted are not subscribed to the tournament: a human-readable message is displayed
Note	• To make the research of a specific STU easier, the system provides a search bar • The system offers the possibility to invite new members to the team and let other STUs join it, iff boundaries constraint are respected

Name	Join a team - when already in the battle
ID	UC8.1 (Figure 3.23)
Actors	STU and ESP
Entry conditions	STU is registered in CKB platform **and** is enrolled in a tournament **and** has received an email notification from the system inviting it to join a team for an upcoming battle within the context of the enrolled tournament **and** it has already joined that battle alone
Event flow	1. User opens the e-mail and clicks on the link attached to be redirected to the system 2. The system shows the "Dashboard" page with the invitation if the join window for the battle is still open 3. User decides to Accept or Decline the invite • User clicks the "Accept" button (a) The system adds the STU to the team (b) User now is part of that team in the battle • User clicks the "Decline" button (a) The system deletes the notification from its "Dashboard" page (b) The system notifies the STU that sent the invite of the decision and allows it to invite a new STU
Exit conditions	STU participates in the battle, depending on its choice, alone or in a team of more than one STU
Exceptions	• STU ignores the invite received: STU participates alone and at the end of the registration window, for that specific battle, all the invites related are canceled from the "Dashboard" page • STU received multiple invitations for the same battle, but from different teams: after one invite is accepted all the others are canceled in its "Dashboard" page • STUs already part of a team can not be selected to be in a different team

Name	Join a team - when not in the battle yet
ID	UC8.2 (Figure 3.23)
Actors	STU and ESP
Entry conditions	STU is registered in CKB platform **and** is enrolled in a tournament **and** has received an email notification from the system inviting it to join a team for an upcoming battle within the context of the enrolled tournament
Event flow	1. User opens the e-mail and clicks on the link attached to be redirected to the system 2. The system shows the "Dashboard" page with the invitation if the join windows for the battle is still open 3. User decides to Accept or Decline the invite • User click the "Accept" button (a) The system enrolls the STU to the battle and then adds it to the team that sent the invite (b) User is part of the battle and it competes in it with a team • User clicks the "Decline" button (a) The system deletes the notification from its "Dashboard" page (b) The system notifies the STU that sent the invite of the decision and allows it to invite a new STU
Exit conditions	STU is part of the battle and the team if it accepts the invite, else is not part of none of that
Exceptions	• STU ignores the invite received: STU is neither part of the battle nor the team and at the end of the registration window, for that specific battle, all the related invites are canceled from the "Dashboard" page • STU received multiple invitations for the same battle, but from different teams: after one invite is accepted all the others are canceled in its "Dashboard" page • STUs already part of a team can not be selected to be in a different team

Name	Manual evaluation
ID	UC9 (Figure 3.24)
Actors	EDU ans ESP
Entry conditions	The actor is logged in as an EDU **and** it is on the "Tournament View" where at least a battle's deadline has expired **and** that battle has the "manual evaluation" enabled
Event flow	1. The EDU selects a battle it has created 2. The system shows the "Battle View" page, which has a list of all the teams that have submitted their work within the deadline 3. One by one, the EDU selects a team and the system shows the team's code 4. The EDU evaluates the code and assigns a score in the range [0, 100] 5. The EDU clicks on the "Submit" button 6. The system updates the battle score of the team 7. When all the teams are evaluated, the EDU clicks on the "Close battle" button
Exit conditions	All participants in the battle get notified through ESP, that the final score is available
Exceptions	• The EDU does not evaluate the code of a team and clicks on the "Submit" button: a human-readable message is displayed • The EDU assigns a score that is not in the range [0, 100]: a human-readable message is displayed • The EDU wants to change the score assigned to a team: the system lets the EDU change the score and then it updates the battle score of the team
Note	The system offers the possibility to stop the evaluation process, it saves the scores assigned to the teams and the evaluation process can be resumed later

Name	Create new badge
ID	UC10 (Figure 3.25)
Actors	EDU
Entry conditions	The user is logged in as EDU **and** it is in the "Dashboard" page
Event flow	1. EDU clicks on the "Badges" button 2. The system shows the "Create badge" page with a form to be compiled 3. EDU fills the form with the required information as a title and rule that STU must fulfill to obtain that specific badge 4. EDU clicks on the "Create" button 5. The system redirects the EDU to the "Dashboard" page
Exit conditions	The systems create the new badge and now available whenever EDUs in CKB create tournaments
Exceptions	*None*

Name	Close tournament
ID	UC11 (Figure 3.26)
Actors	EDU and ESP
Entry conditions	The actor is logged in as an EDU **and** it has all the authorization in the tournament **and** it is in the "Tournament View" page
Event flow	1. The EDU clicks on the "Close tournament" button 2. The system shows a confirmation message 3. The EDU clicks on the "Confirm" button 4. The system closes the tournament and elaborates the final tournament rank
Exit conditions	All the STUs enrolled in the tournament get notified through ESP and can check the tournament final ranking
Exceptions	• The EDU clicks on the "Cancel" button: the system closes the message and shows normally the tournament page • The tournament is already closed: a human-readable message is displayed

Name	View battle rank
ID	UC12 (Figure 3.27)
Actors	EDU and STU
Entry conditions	The actor is logged in as an EDU or a STU **and** it is in the "Tournament View" page of the tournament containing the battle is looking forking
Event flow	1. The actor searches through battles 2. The actor clicks on the chosen battle 3. The system shows the rank of that battle next to the battle list
Exit conditions	The actor can see the rank of the battle it was looking for
Exceptions	*None*
Note	If the battle is not started yet, the system will show an empty rank

Name	View tournament rank
ID	UC13 (Figure 3.28)
Actors	EDU or STU
Entry conditions	The actor is logged in as an EDU or a STU **and** it is in the "Dashboard" page
Event flow	1. The system shows the list of all the tournaments 2. The actor selects a tournament, either ongoing or closed 3. The system shows the "Tournament View" page
Exit conditions	The actor can now see the tournament rank
Exceptions	*None*
Notes	The "Tournament View" page shows both the rank and the list of battles within it, ongoing or ended

Name	View STU's profile
ID	UC14 (Figure 3.29)
Actors	EDU or STU
Entry conditions	The actor is logged in as an EDU or a STU
Event flow	1. The actor clicks on a STU username 2. The system redirects the actor to the "Profile View" page of the STU with that username
Exit conditions	The user sees the profile of the STU, it can visualize gained badges and the list of the tournament it is enrolled in
Exceptions	1. There is no STU with the username specified

Name	View tournaments
ID	UC15.1 (Figure 3.30)
Actors	EDU
Entry conditions	The actor is logged in as an EDU **and** it is in the "Dashboard" page
Event flow	1. The system, in the "Dashboard" page, shows the list of all tournaments created by the EDU 2. The EDU can select a tournament, either ongoing or closed 3. The system shows the "Tournament View" page with the list of its ongoing or closed battles **Alternatively:** 1. The EDU, in the "Dashboard" page, clicks on the "Show all" button 2. The EDU type in the search bar the name of a tournament it is looking for 3. The system shows the matching result 4. The EDU can select the resulting tournament, either ongoing or closed 5. The system shows the "Tournament View" page with the list of its ongoing or closed battles
Exit conditions	The EDU see the "Tournament View" page and can take actions in it
Exceptions	1. The EDU type, in the search bar, a tournament name that does not exist: a human-readable message is displayed

Name	View tournaments
ID	UC15.2 (Figure 3.31)
Actors	STU
Entry conditions	The actor is logged in as a STU **and** it is in the "Dashboard" page
Event flow	1. The system, in the "Dashboard" page, shows the list of all the tournaments to which the STU is subscribed 2. The STU can select a tournament, either ongoing or closed 3. The system shows the "Tournament View" page with the list of its ongoing or closed battles **Alternatively:** 1. The STU, in the "Dashboard" page, clicks on the "Show all" button 2. The STU type in the search bar the name of a tournament it is looking for 3. The system shows the matching result 4. The STU can select the resulting tournament, either ongoing or closed 5. The system shows the "Tournament View" page with the list of its ongoing or closed battles
Exit conditions	The STU see the "Tournament View" page and can take actions in it
Exceptions	1. The STU type, in the search bar, a tournament name that does not exist: the system suggests other tournaments showing their names and the subscribing deadlines
Notes	In the "Dashboard" page are shown both suggested tournaments and tournaments to which the STU is subscribed

Name	View battles in tournament
ID	UC16 (Figure 3.32)
Actors	EDU or STU
Entry conditions	The actor is logged as an EDU or a STU **and** it is in the "Dashboard" page
Event flow	1. The actor selects a tournament 2. The system shows the "Tournament View" page
Exit conditions	The actor can see the list of all battles within that tournament
Exceptions	*None*
Notes	The tournament page shows both the rank and the list of battles within it, ongoing or ended

Name	Push commit and score updating
ID	UC17 (Figure 3.33)
Actors	STU and GitHub API
Entry conditions	STU is subscribed to the battle **and** the registration deadline for a battle has expired **and** the battle is in progress and not ended yet
Event flow	1. STU pushes changes to its GitHub repository 2. GitHub actions inform the CKB platform's API of a new push has been performed by STU 3. The system pulls the latest sources and analyzes them 4. The system runs the tests on the corresponding executables 5. The system computes and updates the battle score of the team corresponding to STU
Exit conditions	The STU's team can see the updated score
Exceptions	*None*
Notes	We assumed, in the Domain Assumptions section, that the STU set properly the GitHub repository and actions, so, by the side of the platform, nothing could go wrong

Name	Repository creation
ID	UC18 (Figure 3.34)
Actors	STU, GitHub API and ESP
Entry conditions	The actor is logged in as a STU **and** it is subscribed to a battle **and** the registration deadline for that battle is expired
Event flow	1. The system creates a GitHub repository for the battle through the GitHub API 2. The system sends an email to all STUs subscribed to the battle with the link to the GitHub repository 3. The STU clicks on the link and is redirected to the GitHub repository
Exit conditions	The STU its team can start working on the Code Kata, just after forking the repository and setting up an automated workflow
Exceptions	*None*
Notes	Only one student per team receives the GitHub repository link via mail, moreover, the correct functioning of the API is assumed

3.2.3 Use cases Sequence Diagrams

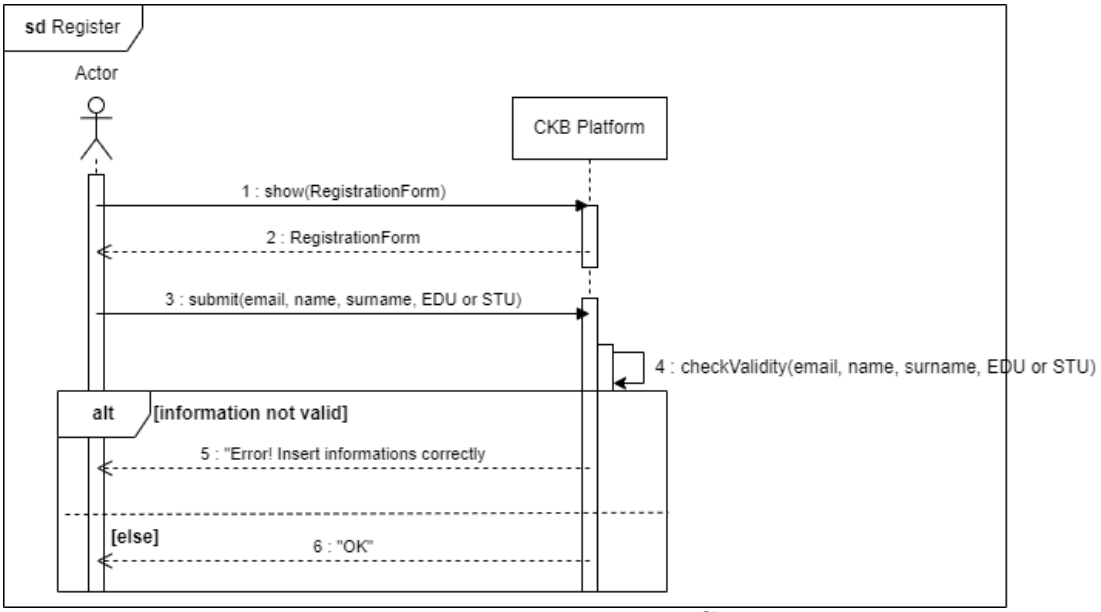

Figure 3.16: Register Use Case

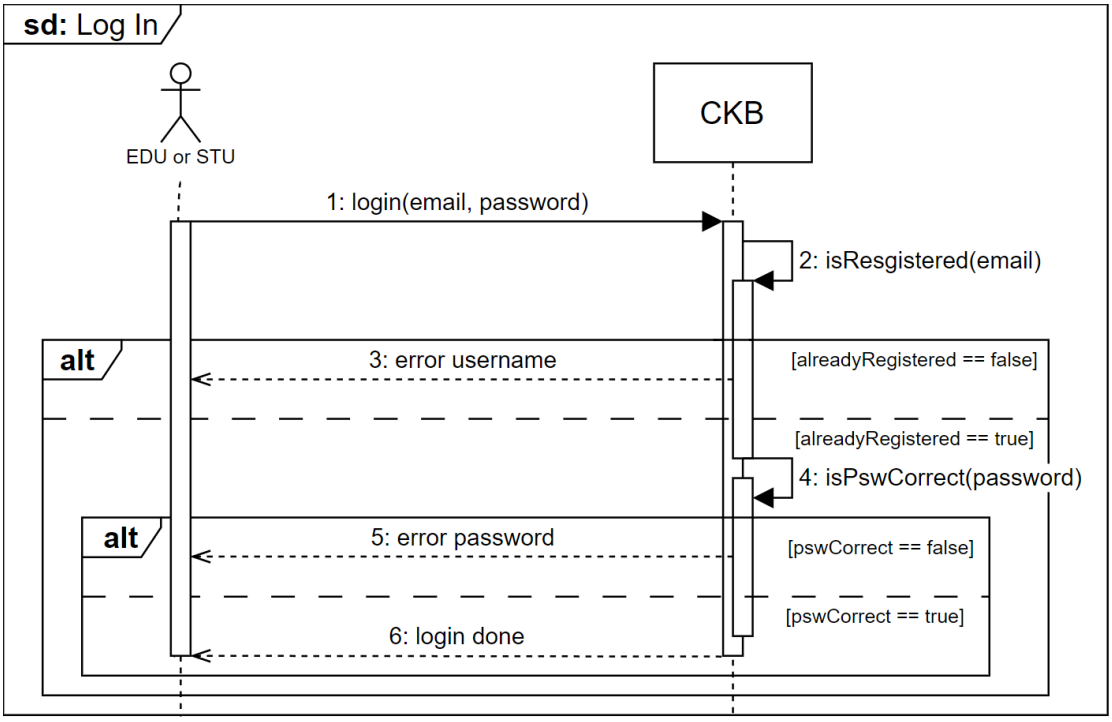

Figure 3.17: Log In Use Case

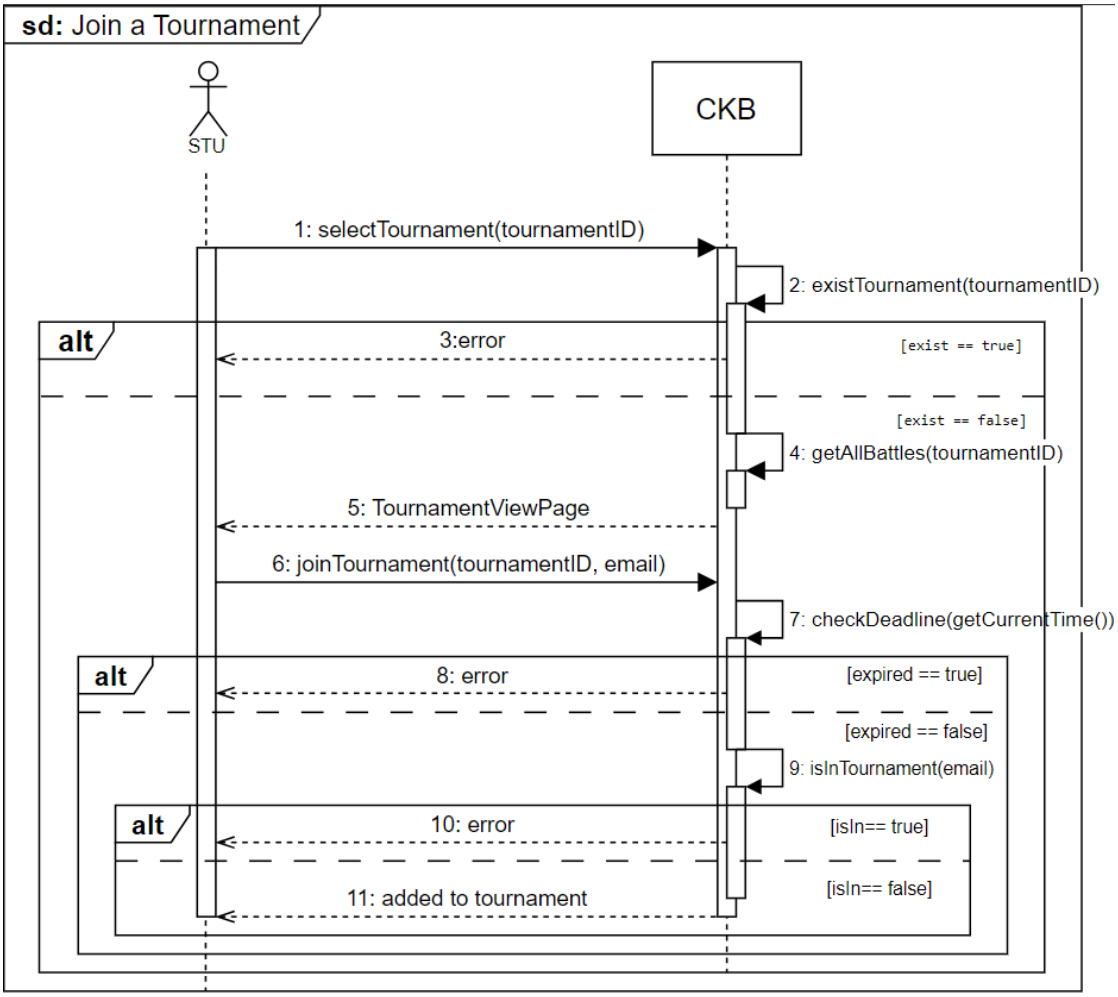

Figure 3.18: Create a Tournament Use Case

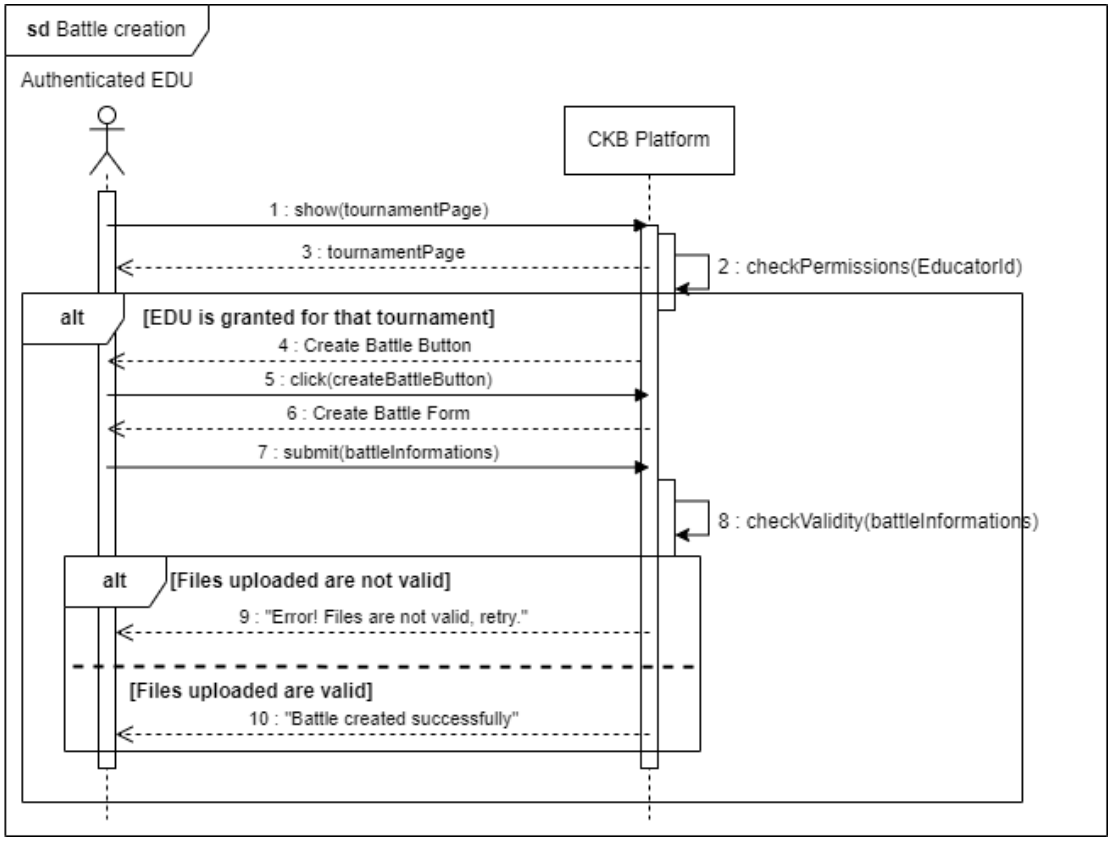

Figure 3.19: Battle creation Use Case

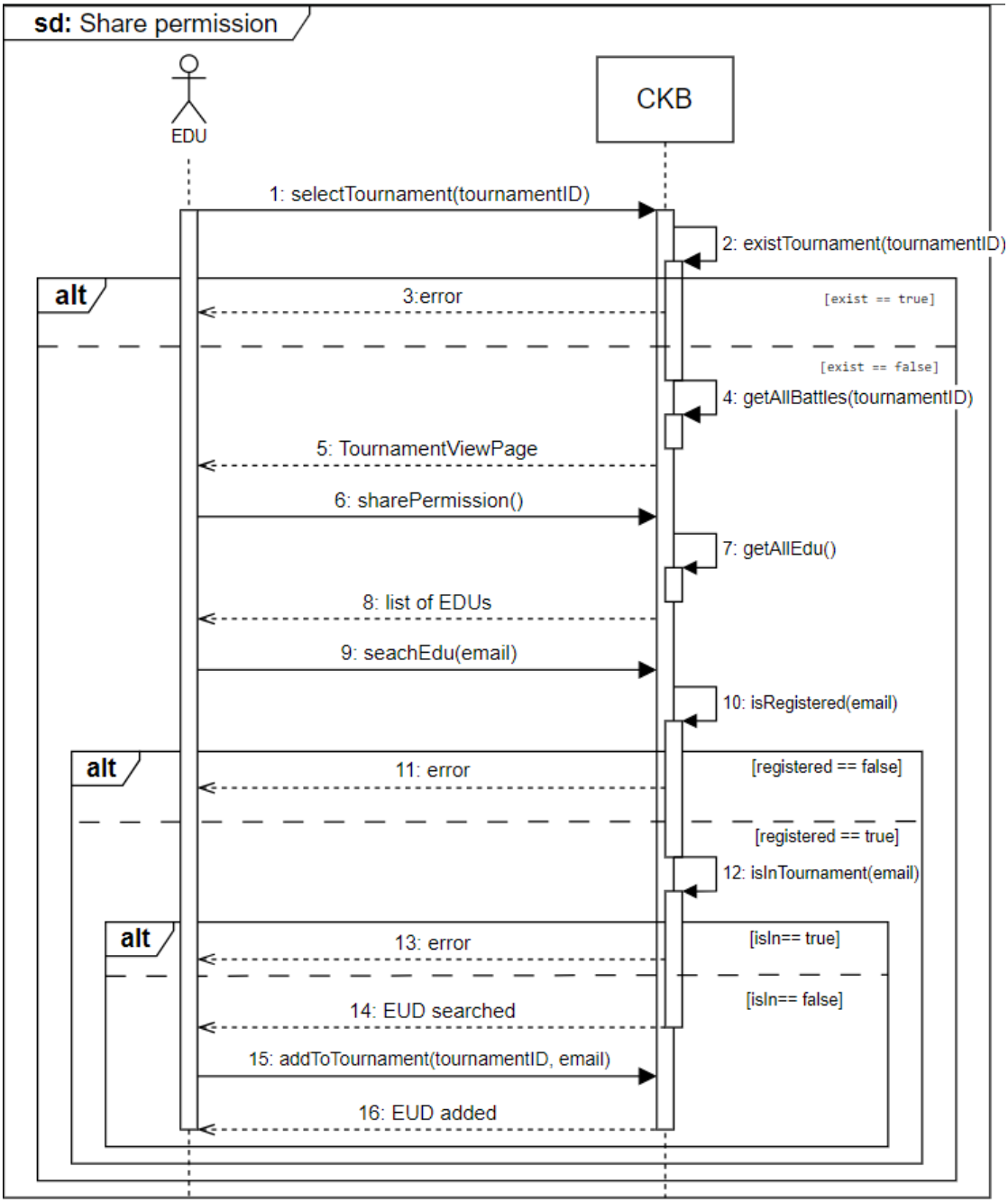

Figure 3.20: Share permission Use Case

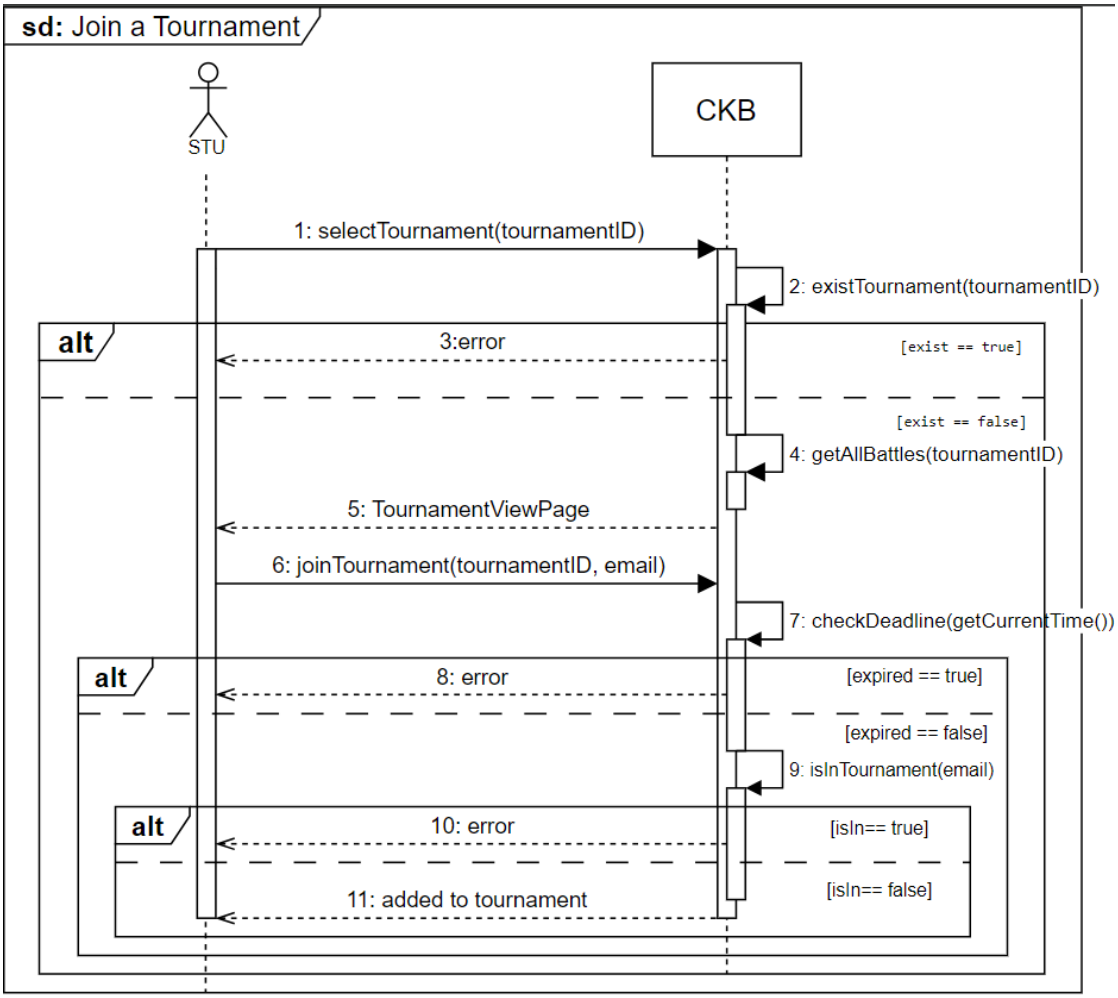

Figure 3.21: Join a tournament Use Case

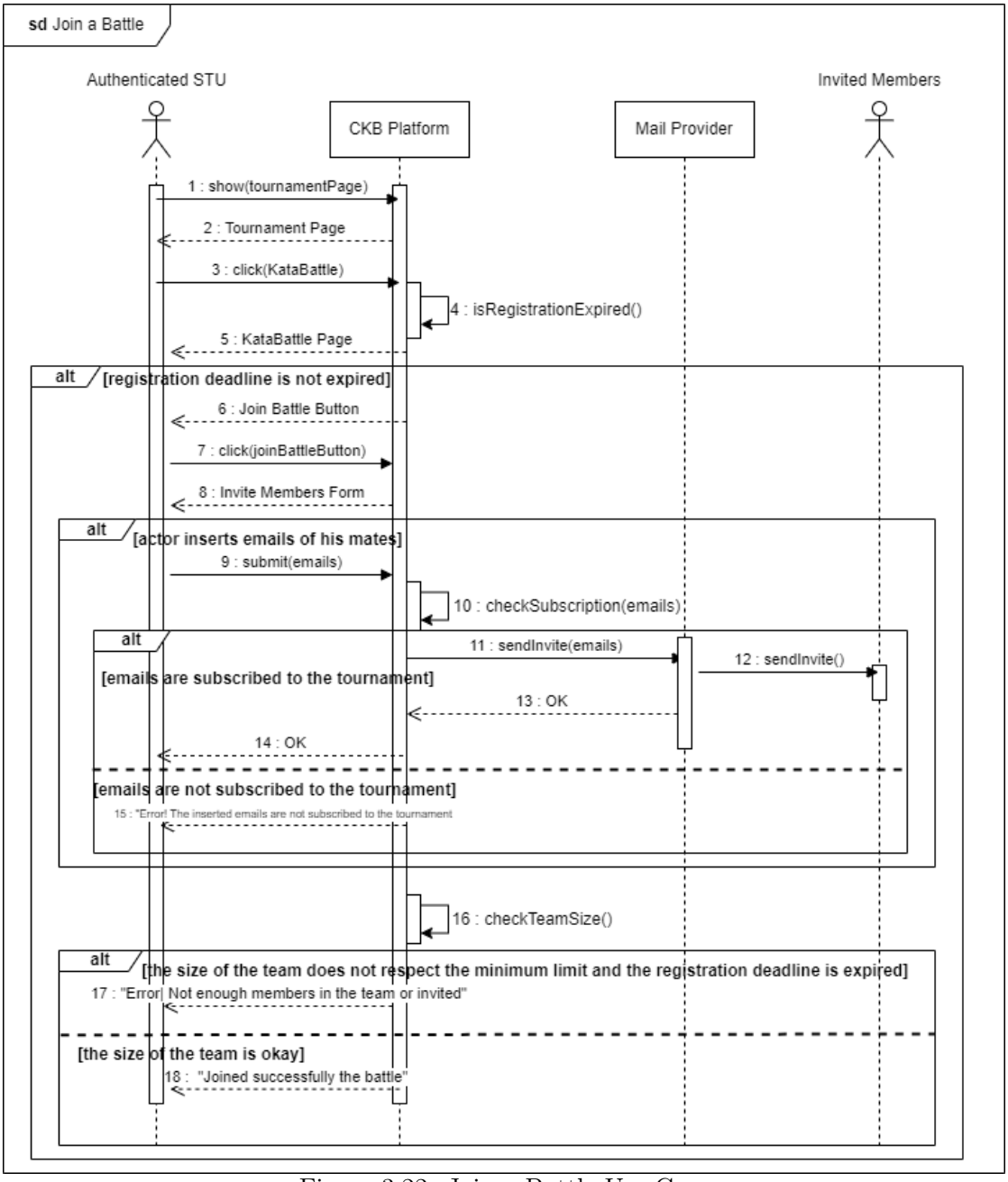

Figure 3.22: Join a Battle Use Case

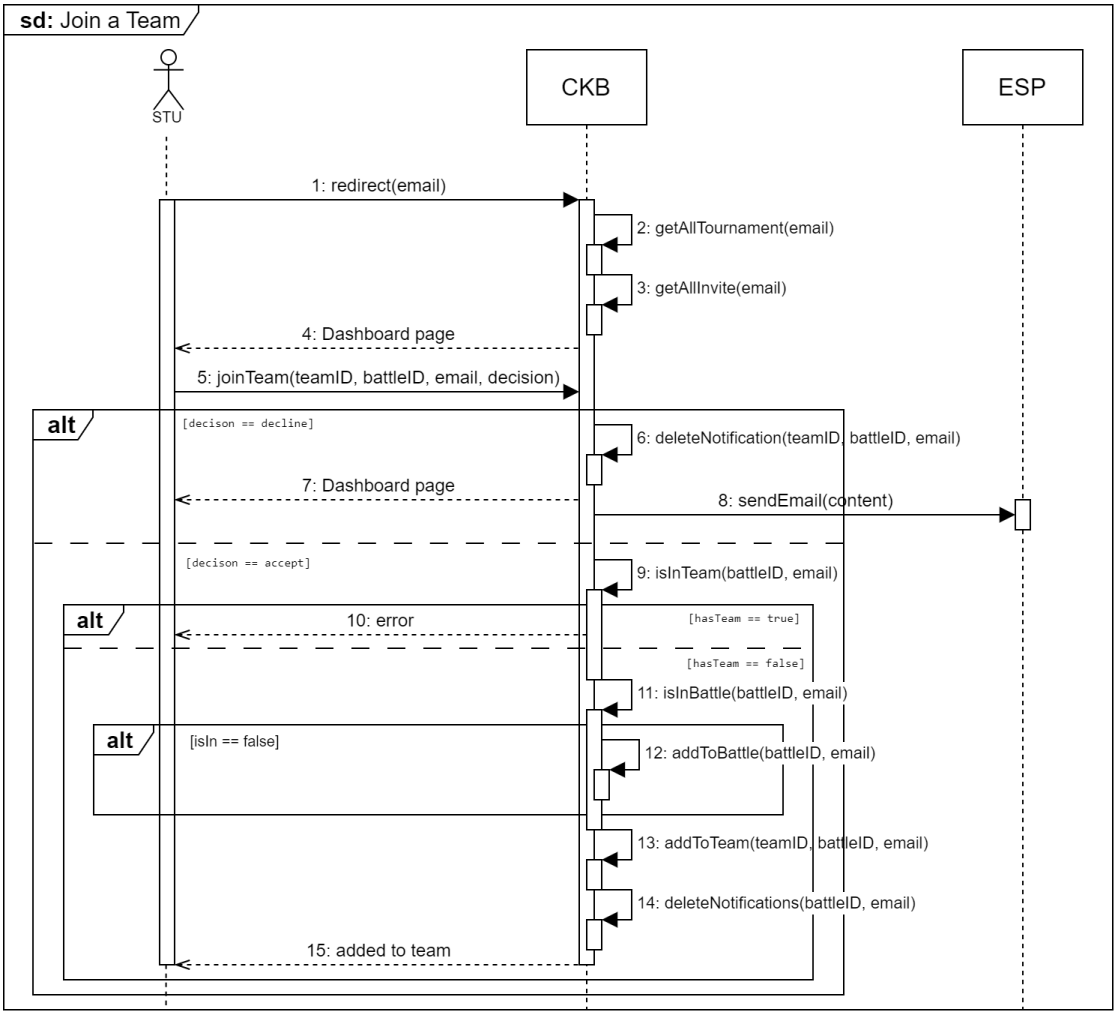

Figure 3.23: Join team Use Case

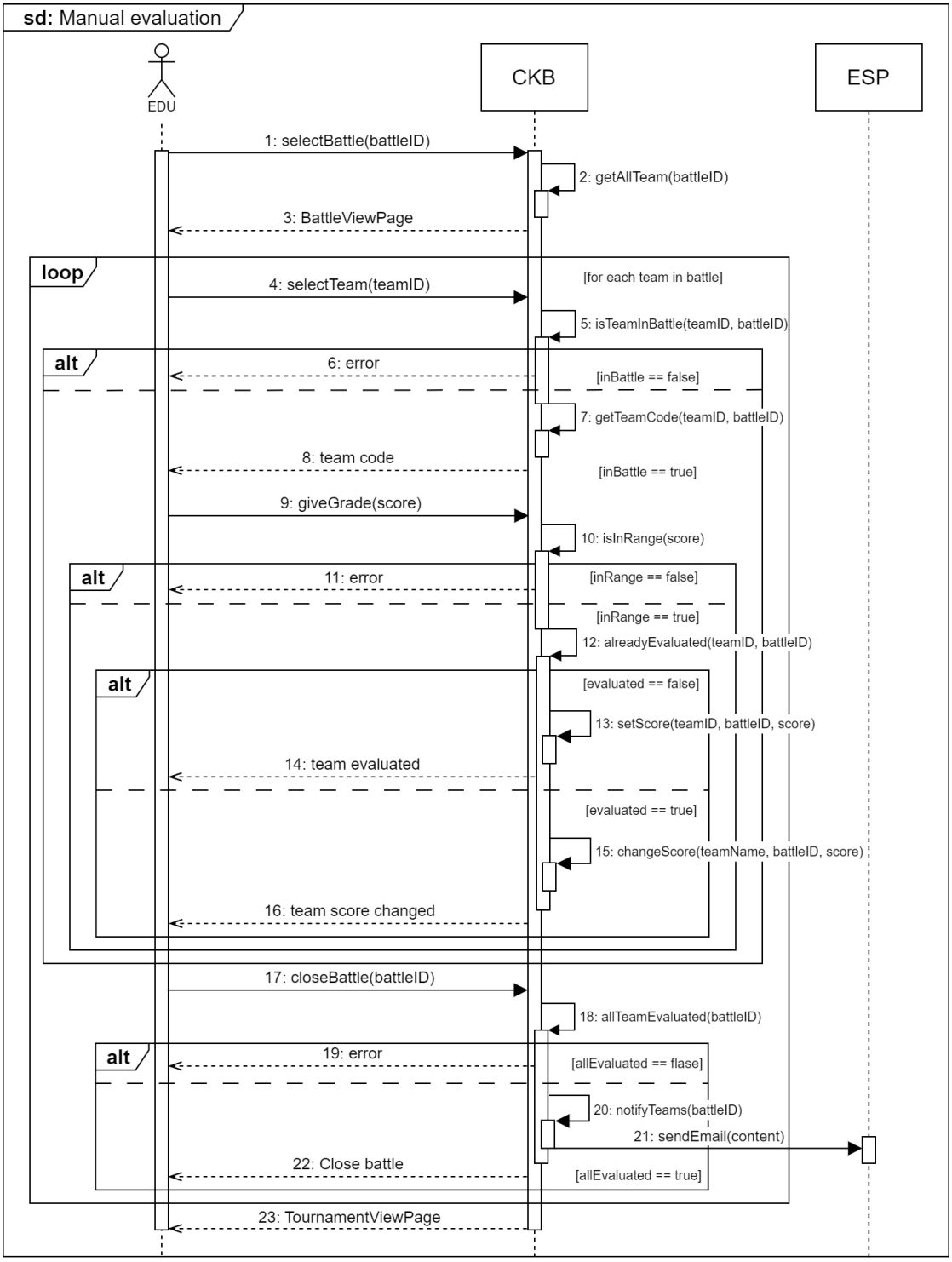

Figure 3.24: Manual evaluation Use Case

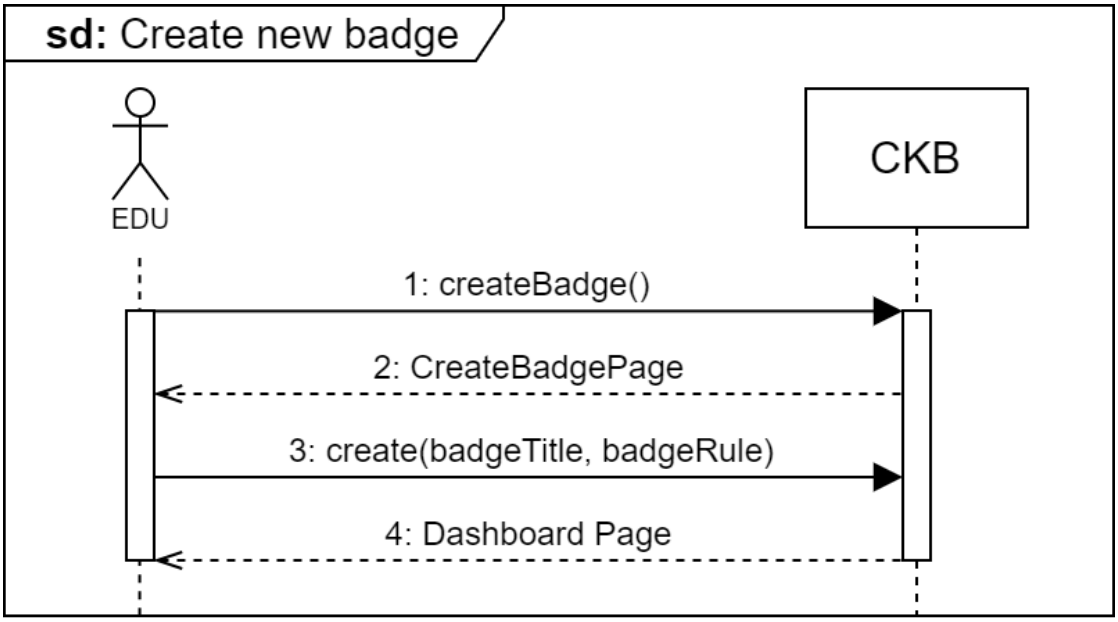

Figure 3.25: Create new badge Use Case

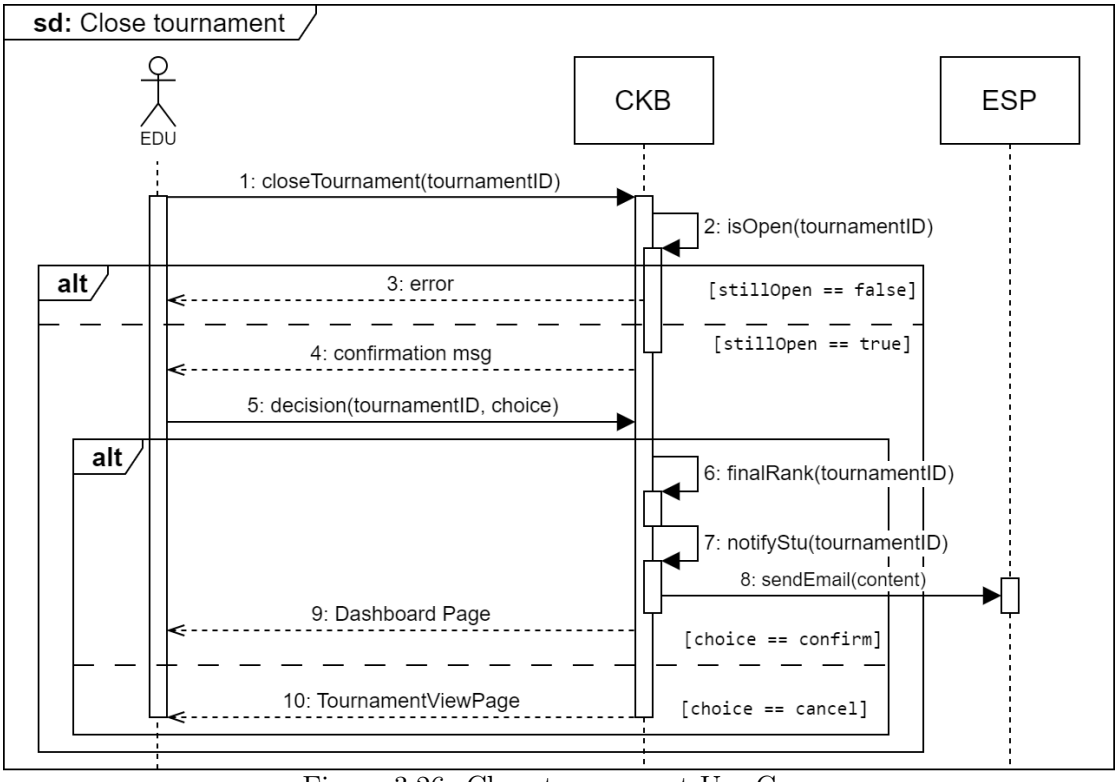

Figure 3.26: Close tournament Use Case

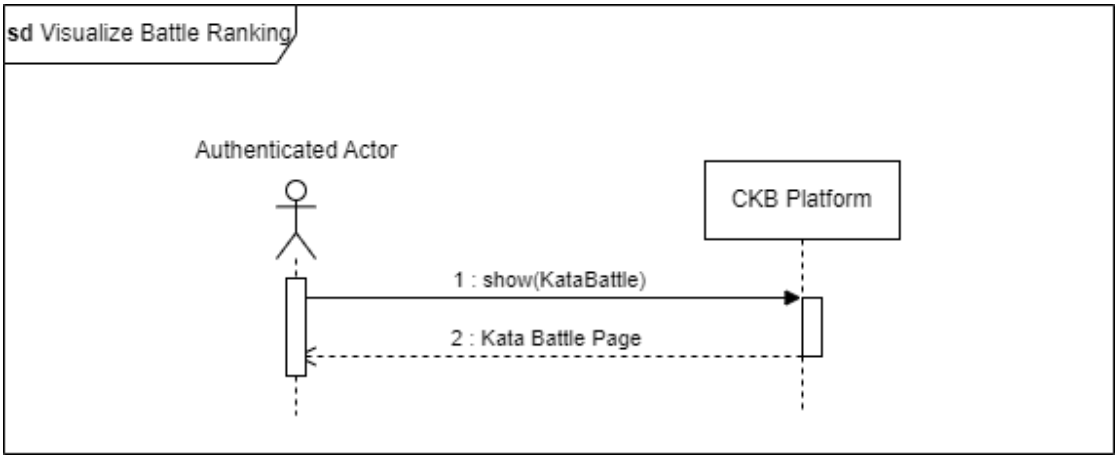

Figure 3.27: View battle rank Use Case

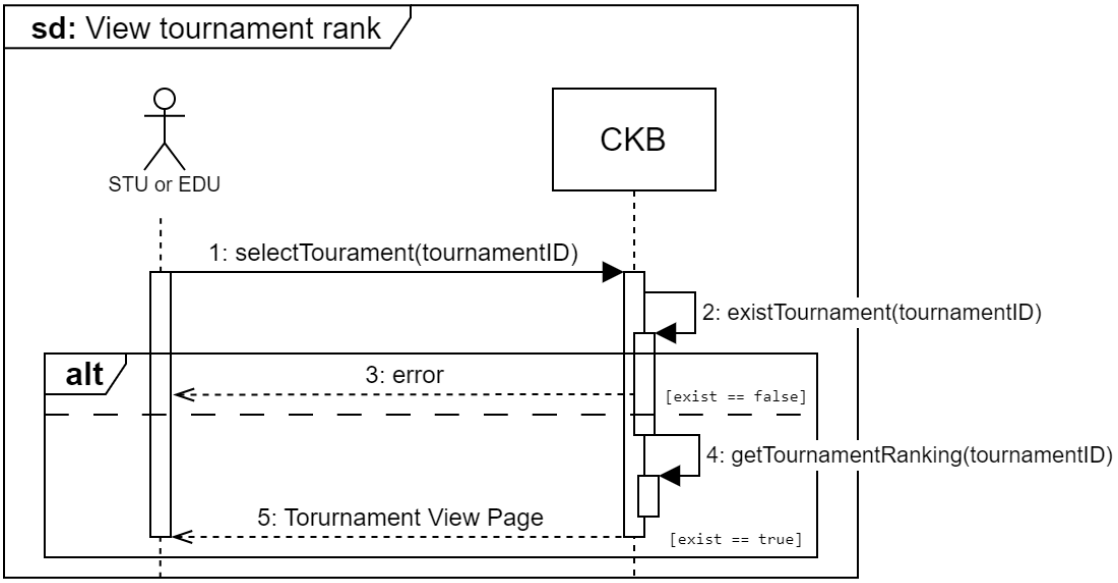

Figure 3.28: View tournament rank Use Case

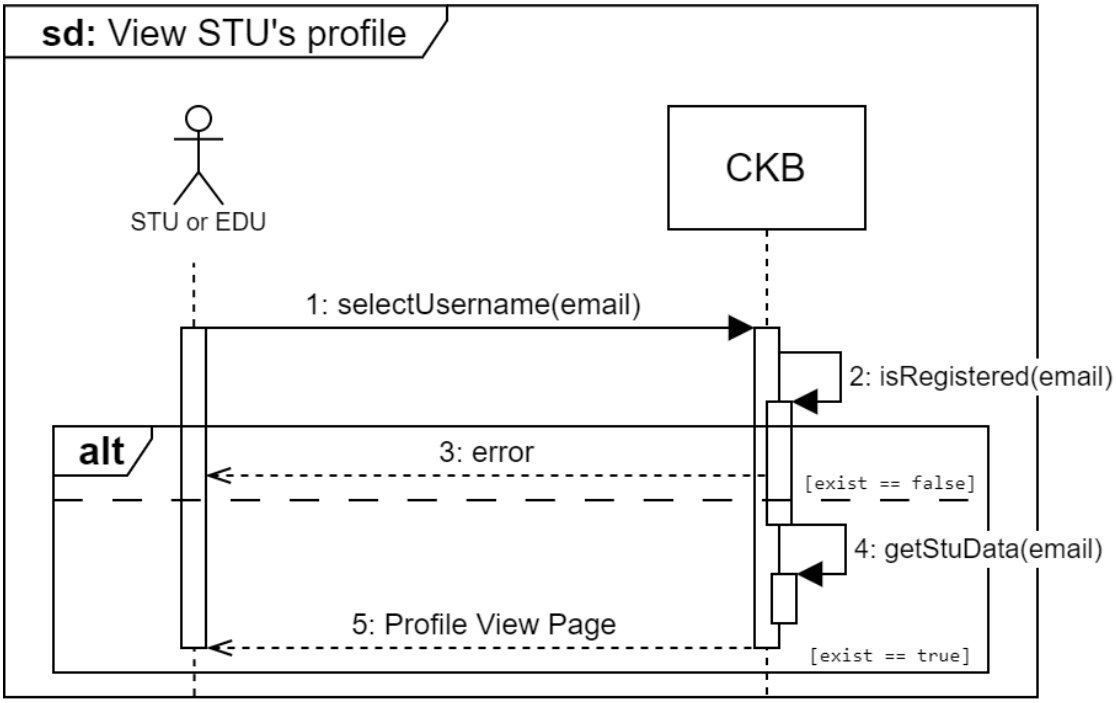

Figure 3.29: View STU's profile Use Case

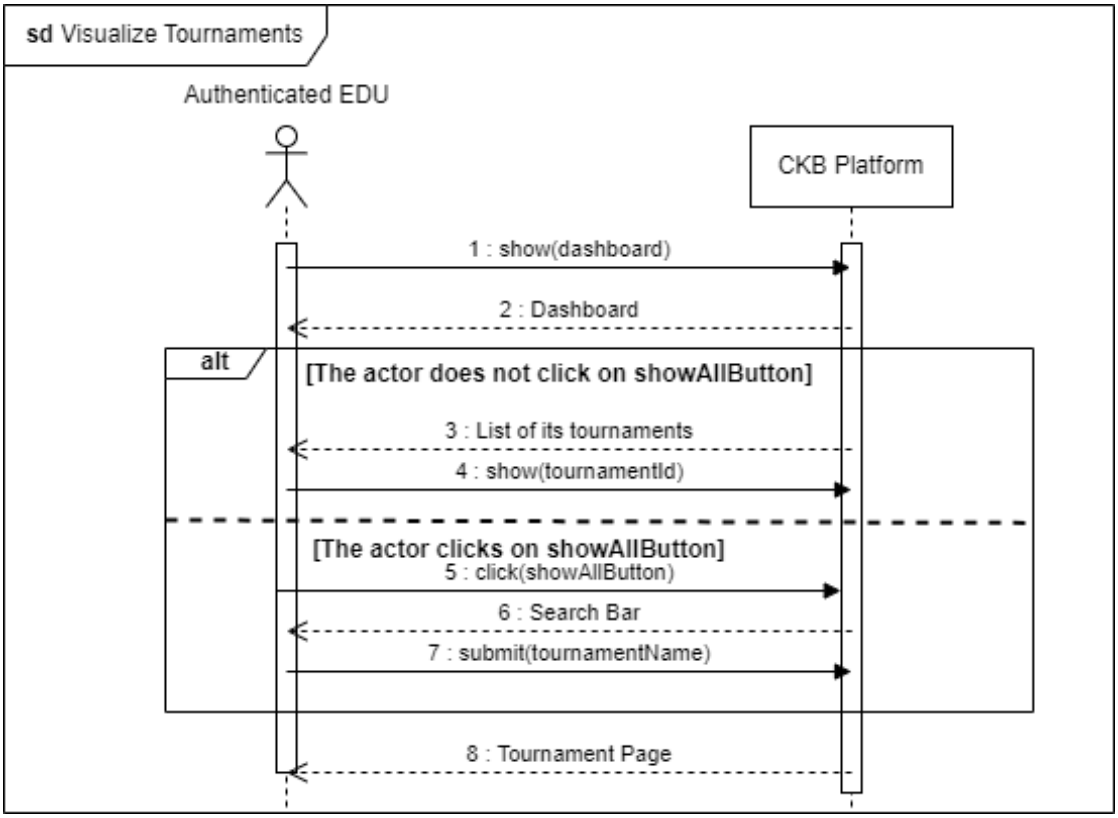

Figure 3.30: View tournaments Use Case

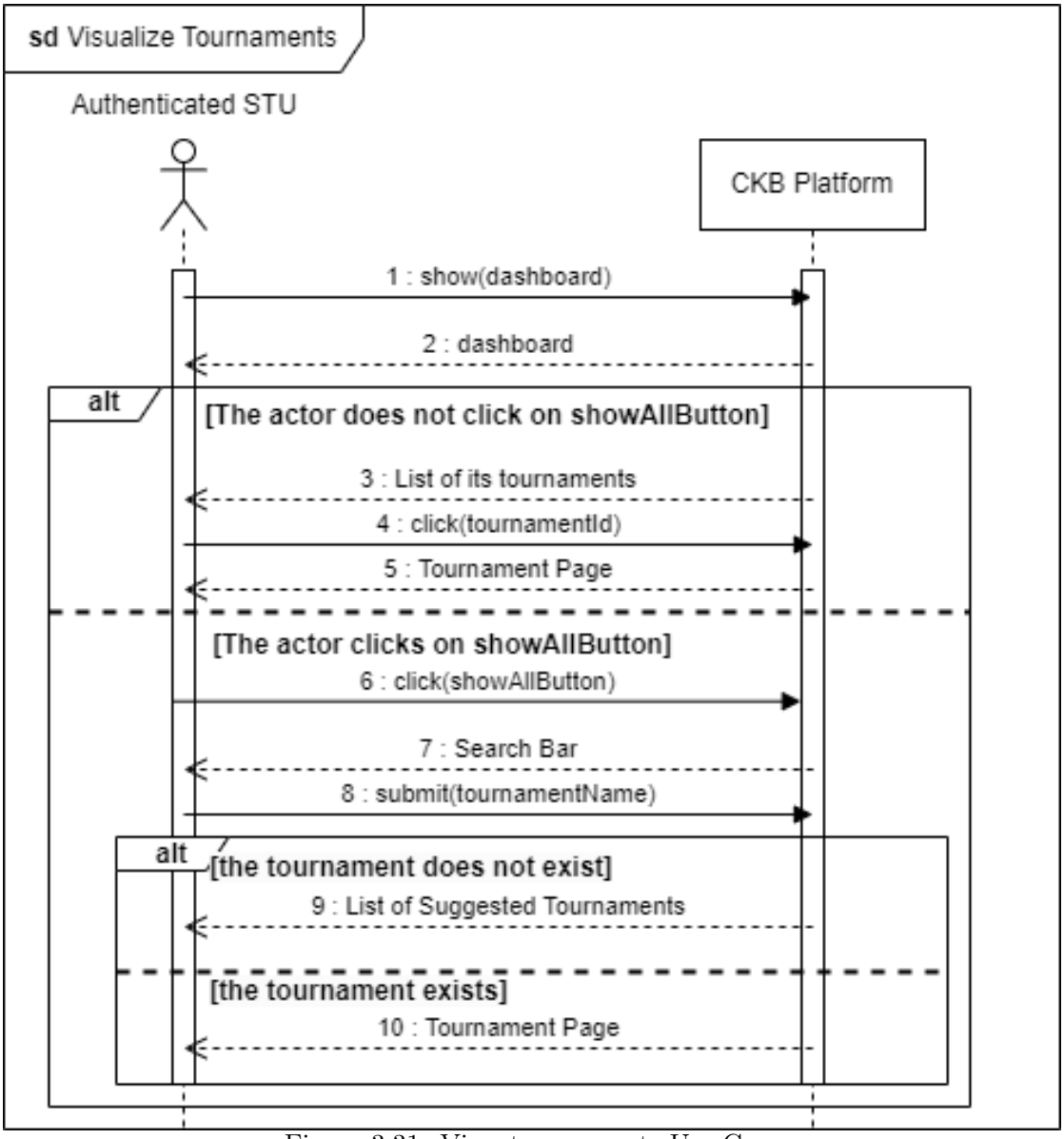

Figure 3.31: View tournaments Use Case

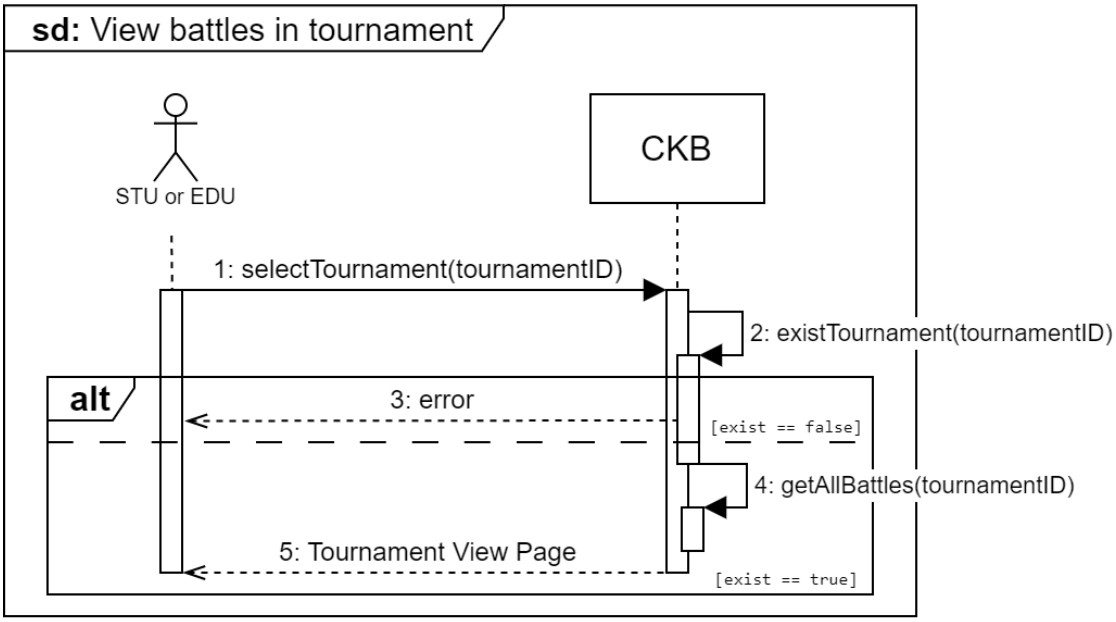

Figure 3.32: View battles in tournament Use Case

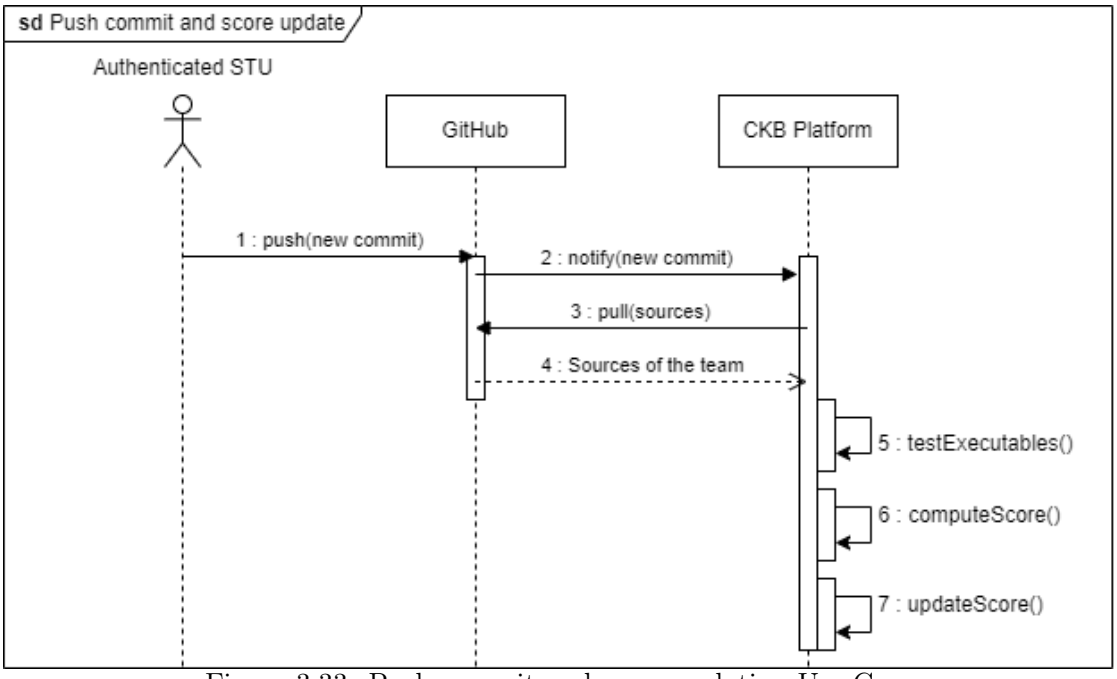

Figure 3.33: Push commit and score updating Use Case

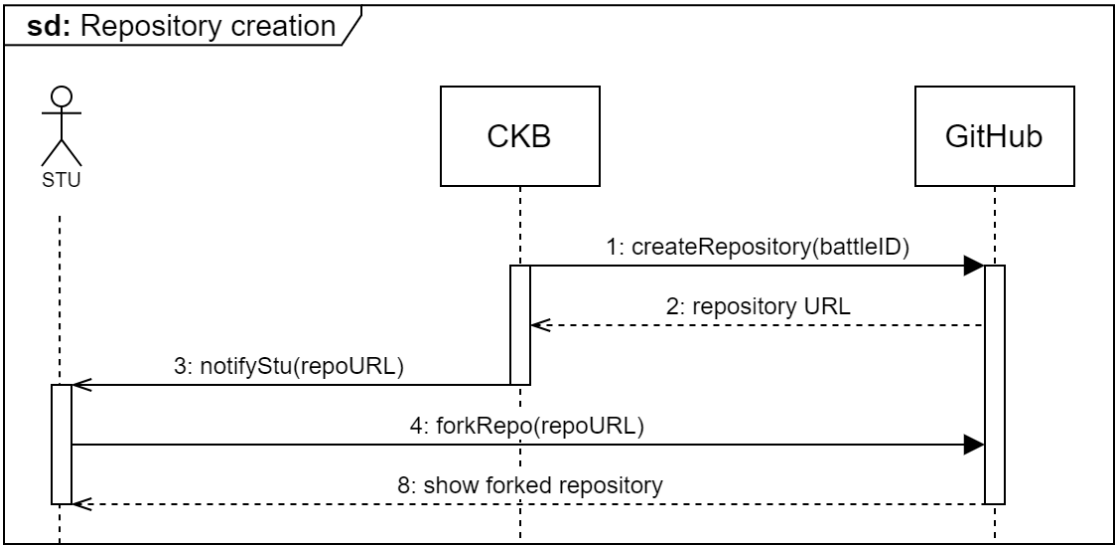

Figure 3.34: Repository creation Use Case

3.2.4 List of functional requirements

EDU functional requirements

ID	Description
R1	The software shall allow the unregistered EDUs to create an account.
R2	The software shall allow the registered EDUs to log in.
R3	The software shall allow the authenticated EDUs to create new tournaments.
R4	The software shall allow an authenticated EDU to permit other authenticated colleagues to create battles in its tournament, those become owners of the tournament as much as who added them.
R5	The software shall allow the authenticated EDUs to create coding battles in their tournaments by letting them upload the Code Kata, setting the minimum and maximum number of STUs per group, registration and final submission deadlines, and the score configurations.
R6	The software shall allow the authenticated EDUs to manually evaluate the work done by STUs subscribed to their own Code Kata battle.
R7	The software shall allow the authenticated EDUs to see the sources produced by each team participating in their tournaments.
R8	The software shall allow the authenticated EDUs to see the personal tournament score of each STU (which is the sum of all battle scores received in that tournament).
R9	The software shall allow the authenticated EDUs to see a rank that measures how a STU's performance compares to other STUs in the context of that tournament.
R10	The software shall allow the authenticated EDUs to see the list of ongoing and finished tournaments as well as the corresponding tournament rank.
R11	The software shall allow the authenticated EDUs to see the list of ongoing and finished battles as well as the corresponding battle rank within a tournament.
R12	The software shall allow an authenticated EDU to close a tournament iff it is one of the owners of that tournament.

R13	When the authenticated EDU creates a tournament, the software shall allow it to define gamification badges concerning that specific tournament.
R14	The software shall allow the authenticated EDU to create new badges and define new rules as well as new variables associated with them.
R15	The software shall allow all authenticated EDUs to visualize the badges created in CKB by other EDUs.
R16	The software shall allow all authenticated EDUs to visualize STUs' profile where they can see their collected badges, and some personal information.

Table 3.1: EDU's requirements

STU functional requirements

ID	Description
R17	The software shall allow the unregistered STUs to create an account.
R18	The software shall allow the registered STUs to log in.
R19	The software shall allow the authenticated STUs to form teams by inviting other STUs respecting the minimum and maximum number of STUs per group set for that battle.
R20	The software shall allow the authenticated STUs to join a team by an invite.
R21	The software shall allow the authenticated STUs to subscribe to a tournament until a certain deadline.
R22	The software shall allow the authenticated STUs subscribed to a coding battle to upload their work until the final submission deadline of that Code Kata battle.
R23	When the registration deadline of a battle expires, the software shall create a GitHub repository containing the Code Kata.
R24	The software shall send to all authenticated STUs who are members of subscribed teams to a battle the link to a GitHub repository containing the Code Kata.

R25	The software shall run the tests on executables pushed by a team, it shall also calculate and update the battle score of the corresponding team.
R26	At the end of the consolidation stage of a specific battle *'b'*, the software shall send a notification to all authenticated STUs participating to *'b'* when the final battle rank becomes available.
R27	The software shall allow the authenticated STUs to see the list of ongoing and finished battles as well as the corresponding battle rank, iff they are part of the tournament.
R28	The software shall allow all authenticated STUs to see the personal tournament score of each STU (which is the sum of all battle scores received in that tournament).
R29	The software shall allow all authenticated STUs to see a rank that measures how a STU's performance compares to other STUs in the context of that tournament.
R30	The software shall allow all authenticated STUs to see the list of ongoing and finished tournaments as well as the corresponding tournament rank.
R31	The software shall notify all authenticated STUs involved in a closed tournament when the final tournament rank becomes available.
R32	The software shall allow all authenticated STUs to visualize other STUs' profile where they can see their collected badges, and some personal information.

Table 3.2: STU's requirements

3.2.5 Traceability matrices

Mapping of functional requirements on use cases

Use case	**Functional requirements**
UC1	R1, R17
UC2	R2, R18
UC3	R3, R13
UC4	R5
UC5	R4
UC6	R21
UC7	R19, R20
UC8	R20
UC9	R6, R7, R26
UC10	R13, R14, R15
UC11	R12, R31
UC12	R11, R27
UC13	R8, R9, R10, R28, R29, R30
UC14	R16, R32
UC15	R10, R30
UC16	R11, R27
UC17	R7, R22, R25
UC18	R22, R23, R24

Mapping $D \wedge R \vDash G$

G1: Educators can create tournaments that involve coding battles to challenge students.	
D2, D5, D6	R1, R2, R3, R4, R5, R6, R12, R13, R17, R18, R21, R22, R23, R24
G2: Provides educators with the ability to track student software development knowledge.	
D2, D3, D4, D5, D6, D8	R1, R2, R6, R7, R8, R9, R10, R11, R13, R14, R15, R17, R18, R21, R22, R24, R25
G3: Students can improve software development skills by taking part in coding tournaments and battles where they must write programs.	
D1, D2, D3, D4,D5, D6, D7, D8	R1, R2, R3, R4, R5, R6, R7, R13, R14, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31, R32
G4: Coding battles enable students to enhance their soft skills, such as communication, collaboration, and time management, by creating a team and collaborating with the members.	
D3, D5, D7, D8	R5, R17, R18, R19, R20

3.3 Performance Requirements

To guarantee a good user experience, the system must:

- Make sure the backend can grow as needed, respond quickly to changes, and balance the workload effectively.
- Be protected against DDoS attacks to keep the system safe and stable.
- Create a user-friendly, responsive front end. It should handle well even when the internet is not great, so users have a smooth experience.
- Send email notifications quickly, so users do not even notice the delay.

3.4 Design Constraints

Standards Compliance

Since there are a lot of interactions between the various components of the system, it is important to follow some communication standards: in particular, the system uses the REST architectural style to communicate between the frontend and the backend, and the data will be exchanged in JSON format.
Furthermore, the Source code of the application must be commented on and documented adequately.

Hardware Limitations

The system is designed to be used on any device with a modern browser installed, so the only hardware limitation is the presence of a stable internet connection.

3.5 Software System Attributes

3.5.1 Reliability

Since some functionality of the system relies on external APIs, the system should not completely fail because of failure in one of those.
It is also important to avoid data loss through redundant storage methods.

3.5.2 Availability

In the event of an unplanned system downtime, all features should be restored as quickly as possible to minimize any inconvenience. To prevent such occurrences, the CKB platform must have a reliable infrastructure, including redundant servers, to ensure continuous operation. The aimed availability of the system is 3-nines availability (99.9%), which means that the system can be down for less than 9 hours per year.
The system should also be able to handle a large number of concurrent users.

3.5.3 Security

Users of the system have distinct privileges according to their roles (student / educator), determined during the login process.
All data and information transferred and stored within the system are secured through robust encryption methods, such as HTTPS, ensuring data privacy and security.

3.5.4 Maintainability

The source code and associated documentation must include clear comments and should be consistently maintained. During the design and development phases, emphasis should be placed on achieving modularity, minimizing coupling, and ensuring high cohesion between components. This is especially crucial for both the front-end and back-end, allowing developers to make updates to the back-end seamlessly without causing any disruptions or noticeable changes for users.
To avoid inconvenience in solving any type of problem (e.g. server downtime), maintenance services are notified to all users with an advance notice of at least 36 hours.

3.5.5 Portability

Due to the fact that the CKB platform is a distributed system, and it does not rely on specific hardware or software, it can be used / accessed in multiple ways.

4. Formal Analysis Using Alloy

In this chapter the Alloy model of the CKB system is implemented by describing the main constraints. The aim of the generated world is to underline the constraint about the team size of each battle and the fact that a student can not partecipate to the same battle with two different teams.

```
open util/relation
//Signatures
//DateTime is used to represent a couple <date, time>
sig DateTime{}

abstract sig Bool {}
one sig True, False extends Bool {}

/*TestCase represents what the educator will upload when
   creating a battle in order to test the code of the
   students*/
sig TestCase{}
sig Name{}
sig Surname{}
sig Email{}
sig Password{}
sig Language{}
sig Description{}
sig Rule{}
sig Title{}
sig Score{}
sig RankingTeam{}
sig RankingStudent{}

/*User is an abstract entity containing all the attributes
   that each user will have*/
abstract sig User {
```

```
    name : disj one Name,
    surname: disj one Surname,
    email: disj one Email,
    password : disj one Password,
}

//Student represents the STU of the system
sig Student extends User{
    achievedBadges : set Badge,
    tournaments : set Tournament,
    battles : set Battle
}

//Educator represents the EDU of the system
sig Educator extends User{
    ownedTournaments : set Tournament,
    closedTournaments : set Tournament,
    createdBattles : set Battle
}

/*Tournament entity represents the tournament created by an
    EDU. In particular, grantedEducators will have all the
   EDUs who have the same permissions of the creator EDU. "
   ranking" attribute will contain a map in which the keys
   will be the STUs and to each key will be assigned a
   value which is the sum of scores in the battles
   concerning that tournament */
sig Tournament {
    id : disj one Int,
    subscriptionDeadline : one DateTime,
    ranking : set RankingStudent,
    grantedEducators: some Educator,
    battles: disj set Battle,
    studentsSubscribed : set Student,
    badges : set Badge,
}{
    #studentsSubscribed = #ranking
}

/*Battle entity represents the battle created by EDUs in
   tournaments.
In particular, the ranking attribute will contain a map in
   which the keys will be the teams and to each key will be
```

```
    assigned a value which is the score of that team in
   this battle.
We use the subscribedTeams attribute instead of
   subscribedStudents because each team is composed by at
   least one person and all the components of the team will
    be subscribed to the battle.
As a consequence, we can derive all subscribed STUs by
   looking at the STUs who appear in the subscribedTeams
   attribute. */
sig Battle {
    id : disj one Int,
    creator : one Educator,
    closed : one Bool,
    rankingTeams : disj set RankingTeam,
    manualEvaluation: one Bool,
    language: one Language,
    description:disj one Description,
    testCases: disj some TestCase,
    minStudents : one Int,
    maxStudents : one Int,
    registrationDeadline : one DateTime,
    finalSubmissionDeadline : one DateTime,
    subscribedTeams:disj set Team,
    tournament : one Tournament,
}{
    #rankingTeams = #subscribedTeams

    /*maxStudents and minStudents can't have negative
       values by definition. */
    maxStudents>0
    minStudents>0

    /*minStudents as a minimum value will be less than or
       equal to maxStudents by definition */
    minStudents <= maxStudents

    /*the registrationDeadline must be earlier than the
       finalSubmissionDeadline by definition, otherwise it
       would not be possible to upload code after the
       registrationDeadline in some cases. */
    registrationDeadline != finalSubmissionDeadline
}
```

```
/*Team represents the team ( composed by at least one
   student by definition) created by a student when he
   subscribes to a battle*/
sig Team{
    battle : one Battle,
    students: some Student,
}

/*The entity Badge represents the badges which can be
   created by EDUs at any moment and can be associated to
   different tournaments at tournament creation time. */
sig Badge {
    rules : disj some Rule,
    values : disj some Int,
    title : disj one Title,
}

//Bool
pred isTrue[b: Bool] { b in True }

pred isFalse[b: Bool] { b in False }

// Facts

//Battle
//Subscription to a Battle by a Team
/*All students subscribed to a battle must be subscribed to
    the corresponding torunament too. */
fact subscribedTeamsAreSubscribedToTournament{
    all t: Tournament | all b: Battle | all te : Team | no
       s: Student | b in t.battles and te in b.
       subscribedTeams and s in te.students and s not in t.
       studentsSubscribed
}

fact teamIsSubscribed{
    all t: Team| all b : Battle | t in b.subscribedTeams
       <=> b in t.battle
}
/*All team subscribed to a battle must satisfy team size
   constraint of that battle, if the battle is started. */
fact teamSizeInBoundaries{
```

```
    all b: Battle| all t : Team | t in b.subscribedTeams =>
        (#t.students >= b.minStudents and #t.students <= b.
       maxStudents)
}

/*A STU can not participate to the same battle with two
   different teams.*/
fact noStudentInTwoTeams{
    all b: Battle, t1 : Team, t2 : Team | no s: Student |
       t1 in b.subscribedTeams and t2 in b.subscribedTeams
       and (s in t1.students and s in t2.students) and t1
       != t2
}

/*An Educator has the same privileges of the owner of a
   tournament if and only if it is a granted EDU for that
   tournament*/
fact ownerIsGranted{
    all t: Tournament, e : Educator | t in e.
       ownedTournaments <=> e in t.grantedEducators
}

/*A tournament is closed if and only if all its battles are
    ended*/
fact closedTournamentclosedBattles{
    all t :Tournament | all b : Battle | t in Educator.
       closedTournaments <=> (b in t.battles and b.closed =
        True)
}

/*A STU is participating to a battle if and only if it is
   in a subscribed team of that battle*/
fact inBattleIfInTeam{
    all s : Student | all t: Team | t.battle in s.battles
}

fact battleCreator{
    all b : Battle | all e : Educator | b.creator = e <=> b
        in e.createdBattles
}

fact battlesCreatedInOwnedTournaments{
```

```
    all b : Battle | all e : Educator | b in e.
       createdBattles <=> b.tournament in e.
       ownedTournaments
}

pred show{
    #Battle = 2
    #Badge = 1
    one b : Battle | b.minStudents = 1 and b.maxStudents =
       1 and #b.subscribedTeams >1
    one t1 : Team | #t1.students >1
}
run show
```

4.1 World

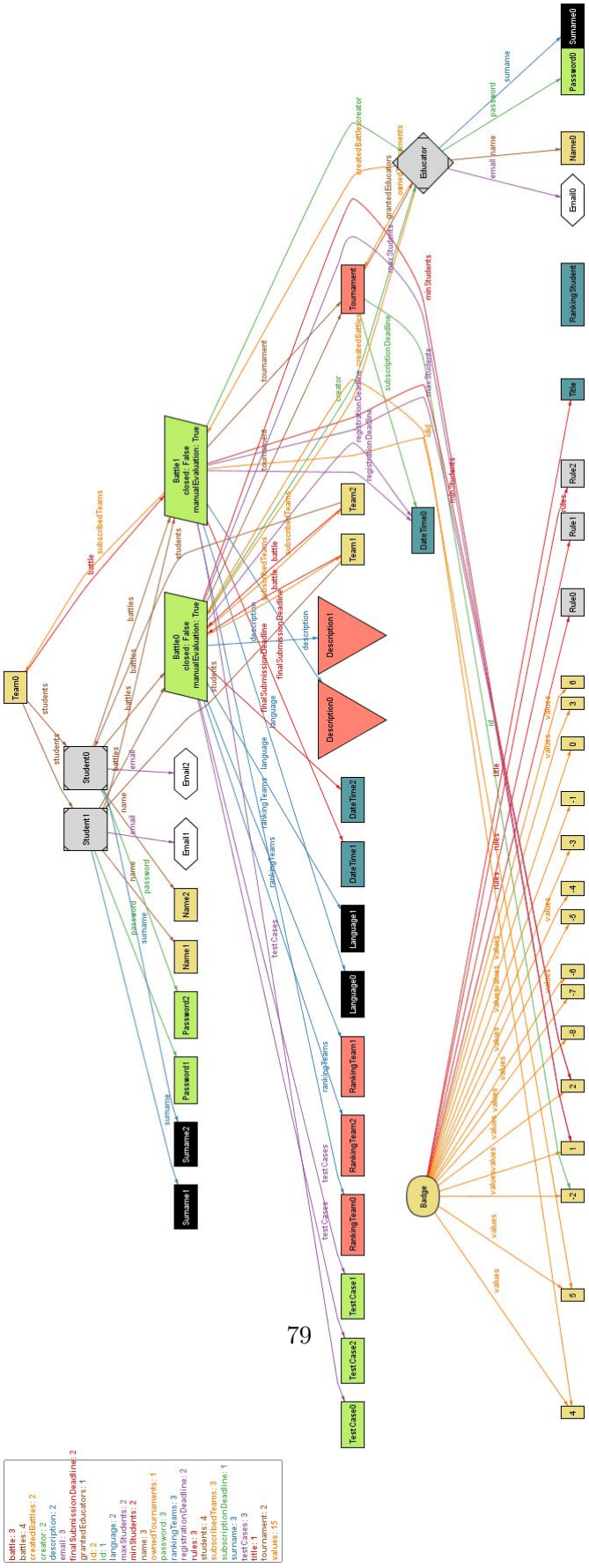

Figure 4.1: World

5. Effort Spent

Team

Topic	Time
Division of work	2h
Revision of chapters 1 and 2	2h
Definition and division of use cases	2h30m
Revision of chapter 3	1h
Last revision of the document	1h30m

Table 5.1: Effort Spent during team meetings

Tommaso Pasini

Topic	Time
Document structure organization	1h30m
Completion chapter 1	3h
Product Function and User characteristics	2h
Use Cases	2h
Sequence diagrams	3h30m
Check document consistency and grammar	5h

Table 5.2: Effort Spent by Tommaso Pasini

Elia Pontiggia

Topic	**Time**
Scenarios	1h
State diagrams	2h
Specific requirements	2h30m
Use cases	2h
User interface mockups	5h
LaTeX document setup and configuration	2h

Table 5.3: Effort Spent by Elia Pontiggia

Michelangelo Stasi

Topic	**Time**
Functional Requirements	2h
Domain Assumptions	1h30m
Use Cases	5h
Traceability Matrix	30m
Class Diagram	1h15m
Alloy	11h

Table 5.4: Effort Spent by Michelangelo Stasi