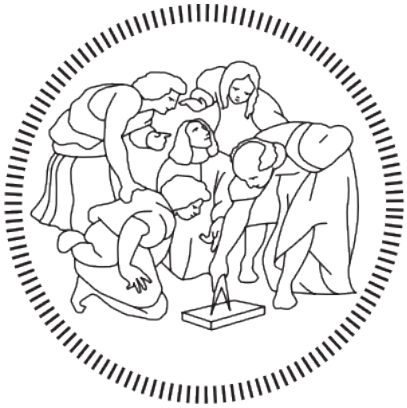

POLITECNICO
MILANO 1863

CodeKataBattle

Requirements Analysis and Specifications Document

Software Engineering 2 project
Academic year 2023 - 2024

20 October 2023
Version 0.0

Authorss:
Tommaso Pasini
Elia Pontiggia
Michelangelo Stasi

Professor:
Matteo Camilli

Revision History

Date	Revision	Notes
23/10/2023	v.0.0	Document creation
	v.1.0	First relese

Contents

1. Introduction

1.1 Purpose

Software development is one of the most sought-after skills globally, and its growing importance is setted to persist in 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 assist 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, competition, and evaluation, fostering a collaborative and skill-building atmosphere.

1.1.2 Goals

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

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 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 team and collaborating with the members.

Table 1.1: Goals

1.2 Scope

CodeKataBattle (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 their chosen language while adhering to the test-first approach.

Educators create a battle by uploading a programming exercise (also known as Code Kata), specifying the group size, registration and submission deadlines, and scoring parameters. Once created, students can form teams, a team colud be composed of just one person or at most the number specified by the group size. Each team can be changed between different battles in the same tournament, register for the battle, and receive access to a GitHub repository containing the code kata. They are required to fork the repository and automate their workflow through GitHub Actions.

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, in this way students and educators can keep track of the battle's ranking. Following the deadline for submission, educators perform an optional manual evaluation before sharing the final battle rank with participants.

All students on the CKB platform have a personal tournament score, which is the sum of their battle scores in that tournament. 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. Gamification badges, which educators can define when creating a tournament, reward students for their performance or achievement.

CodeKataBattle fosters collaborative coding challenges, helping students improve their software development skills while introducing competition and gamification elements to enhance engagement and recognition.

1.2.1 World Phenomena

ID	Description
WP1	Students partecipating in a tournament can decide whether doing a battle or not by subscribing or not to that battle.
WP2	Students choose for each battle if coding alone or forming a team.
WP3	The student who formed the team fork the GitHub repository of the code kata
WP4	Students set up an automatic workflow in the GitHub repository.
WP4	Student invites team member to its repository.
WP4	Students work and complete the code kata battle with their code.
WP4	The students push their work to the GitHub repository.
WP4	An educator creates a correct code kata, defining a coherent description of the project within its test cases and the configuration for automation scripts.
WP4	An educator checks the work done by students in order to evaluate them.

Table 1.3: World Phenomena

1.2.2 Shared Phenomena

ID	Description	Controlled by
SP1	A user registers its personal datas in CodeKataBattle system specifing if it is a student or an educator	
SP2	A registered user insert its credentials to get in CodeKataBattle environment	
SP3	An educator creates a tournament, defining all necessary details	
SP5	The educator who created a specific tournament grants other colleagues permission to create battles inside it	

SP4	An educator that has permission creates battles, defining all necessary details, within a tournament	
SP7	Students are notified of an upcoming tournament	
SP8	Students join a tournament	
SP9	Students are notified of an upcoming battle within a tournament they are subscribed to	
SP10	Student joins a battle	
SP13	Student invites other students, which are subscribed in the same tournament, to join its team respecting the boundaries imposed	
SP14	Student joins a team and therefore it gets enrolled to a battle via an invite by another student	
SP16	The platform sends the link of the GitHub repository to the students who created the team for the battle	
SP17	Students who recieved the GitHub repository link are asked to fork it and set up an automated workflow	
SP18	The forked repository's workflow notifies the platform of a new GitHub push action	
SP19	The platform updates the battle score of the students whenever their repository gets pushed	
SP20	Educator and student subscribed to the battle can monitor the battle ranking among other partecipants	
SP21	The educator uses the platform to go through the sources produced by each team	
SP6	The educator evaluates manually the work done by students in his own battle	
SP23	The platform notifies students of the end of a battle when the final battle rank becomes available	

SP25	An educator that has permission to create battles whitin a tournament, can also closes the tournament	
SP26	The platform notifies all students involved in the tournament about its end when the tournament ranking is available	
SP27	All users can see the list of ongoing and ended tournaments as well as the corresponding tournament rank at any time	
SP285	An educator creates a gamification badge within the CKB platform and asociates to it a rule and a variable	
SP29	An user checks the personal badges gained by anyone subscribed to the CKB platform	

Table 1.4: Shared Phenomena

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Term	Definition
User	Can be a Student or a User
Educator	Identifies a person who provides instruction or education, such as a teacher.
Student	Identifies a person who is studying at a 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 that is 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.

Table 1.5: Definitions

1.3.2 Acronyms

Acronyms	Term
CKB	CodeKataBattle
EDU	Educator
STU	Student

Table 1.6: 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 Dependencie
$\mathbf{R}_i$	i-th Requirement
$\mathbf{UC}_i$	i-th Use Case
i.e.	that is
e.g.	for example

Table 1.7: Abbreviations

1.4 Reference Documents - qui ci mettiamo i link

- Assignment RDD A.Y 2023-2024
- Course slides on WeeBeep
- RASD review by Prof. M. Camilli
- ISO/IEC/IEEE 29148 dated 2018, Systems and software engineering - Life cycle processes - Requirements engineerings

1.5 Document Structure

The structure of this RASD document follows six main sections:

1. **Introduction:** provides an overview of the problem at hand, the purpose of the document and 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 details about its main functions. The description is assisted with the help of UML diagrams, such as class, activity and state diagram. 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 cases diagrams, descriptions of each use case, and related sequence diagrams. Finally, it provides a mapping of the requirements to both goals and use cases.

4. **Formal Analysis Using Alloy:** contains Alloy models which are used for the description of the application domain and his properties, referring to the operations which the system has to provide and some critical aspects of the system.

5. **Effort Spent:** keeps track of the time spent to complete this document. The first table defines the amount of hours used by the whole team to get important decisions and to make reviews, the other tables contains the individual effort spent by each team member.

6. **References:** lists all the documents used and that were helpfull in drafting the RASD.

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. Students 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 students, so he adds th badge for the student who participated in the most tournaments, the one for the student who won the most battles and the one for the student that wrote the most lines of code. All students already subscribed to CKB were notified of the new tournament, and they could join it from the tournament page.

Creating a battle

In order to get the students familiar with the CKB platform and its features, Chip created an easy battle for his students to practice on, called "wordcheck", that basically requires the student to implement the game wordle, to be implemented in c language. He decides that the battle will last 2 weeks and that the students will be able to work in teams of 2 or 3 people; they will be able to join the battle until the last day of the battle. In addiction, he wants to give extra points to the code cleanup, so he will have to revise the code of each team at the end of the battle and assign extra points to the teams that wrote clean code. He sets all this information in the battle creation form and then he creates the battle.

Creating Game badges

Scrooge, a professor of Algorithm and Data Structures at Duckburg University, thinks that, in addiction to the badges that are already present in the CKB platform, it would

be nice to have also a badge for the student who reaches the perfect score in a battle and a badge for the student who reaches the perfect score in a tournament. Since the CKB platform allows educators to create new badges, he creates the two badges and from now on, all educators will be able to include such badges in their tournaments and battles.

Joining a battle

Huey and Dewey, two students of Chip's class, are notified of a new battle and decide to join it. Since the 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 and they are given the link to the GitHub repository of the battle to fork and set up the automated workflow in order to be able 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 a warrior of the "wordcheck" battle and he is working on the battle alone. After a fitst 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 leader group, that is composed by Huey, Dewey and Louie. Fortunately, the battle is still open 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 work to the GitHub repository. After checking his score again, he sees that he is now in the 1st position and moreover he obtained a badge for being the first to reach 100 points in the battle and now both students and professors can see this badge when they visit Donald's profile.

Closing a battle

When the deadline for the battle created by Scrooge is reached, all participants are notified that the battle is closed and that they can't push new commits to the GitHub repository. Scrooge is notified that the battle is closed and he 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 students are notified that they can now see the final rank of the battle.

Closing a tournament

Chip decides to close the "Strings Operations" tournament, even if the deadline is not reached yet. In order to do so, he logs in to the CKB platform and he closes the tournament. All participants are notified that the tournament is closed and that they can't join it anymore, 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 to the class of 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 student, 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 and he also notes that Huey is involved in more than one tournament simultaneously. Thanks to the CKB platform, Pippo has now a complete overview of Huey's skills and he can decide whether to accept his application or not.

2.1.2 Class Diagram

AGGIUNGERE classe testcase. TOGLIERE TeamScore, è uguale a ranking in battle. Togliere creates battle da educator, ridondante In figure 2.1 is represented the class diagram of the software. In particular, the most important details are :

- There are only two possible types of users : the Student type and the Educator type;
- The Educator can create tournaments and battles, he can grants other educators to his own tournament;
- The Student can invite other students, he can achieves some badges, he can subscribes to a tournament and he will use GitHub. But the most important detail is the one about subscribing to a battle : in fact, a student can subscribe to a battle by creating is own team (a team is composed by at least one person) or by joining an existing team. The team subscribes to a battle, BUT, if the battle does not exist, the team class does not exist. In other words, a team exists only in the battle scope;
- The Tournament and Battle classes use the MailAPI to notify students about events.

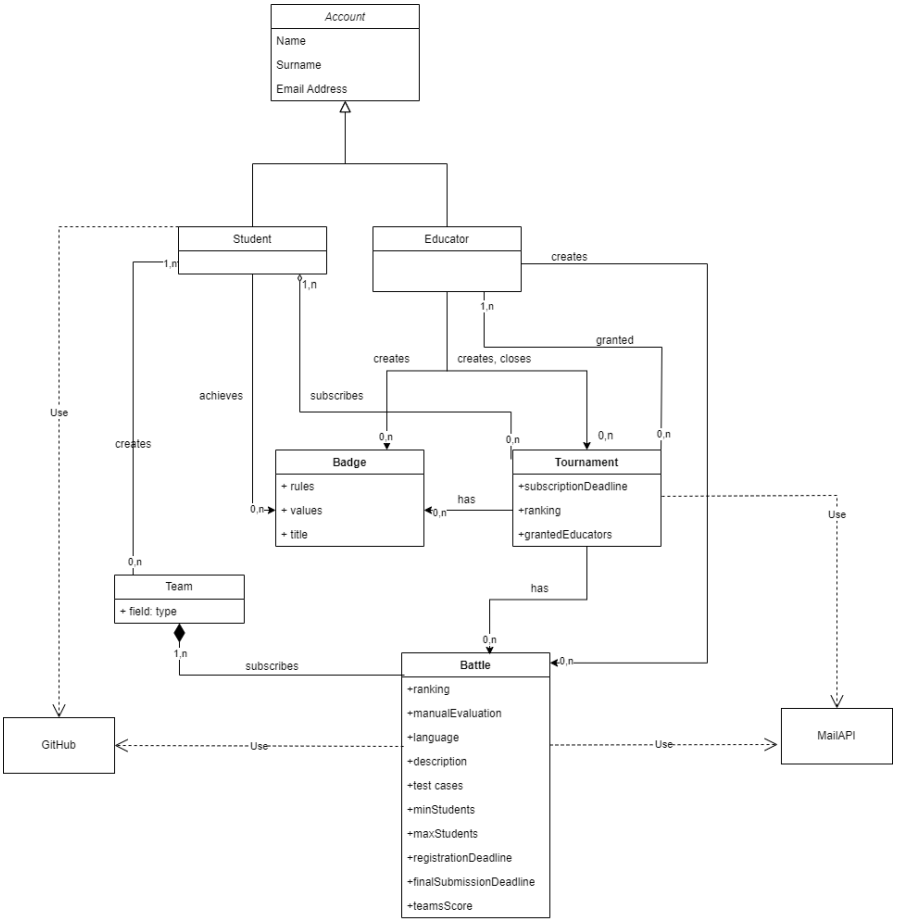

Figure 2.1: Software Class Diagram

2.1.3 State Diagrams

The following state diagrams describe the life cycle of the main entities of the system. Moreover, they also 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 educator creates a tournament, it is both in the *registration open* and *tournament open* states.
In the *registration open* state, students can join the tournament, while in the *tournament open* state, educators with the right permissions can create battles within the tournament, and that leads the tournament to the *battling* state.

When the deadline for the registration is reached, the tournament moves to the *registration closed* state and no more students can join it.
When the deadline for the registrations is reached, no more students can join the tournament and it moves permanently to the *registration clcsed* state.
During the *battling* state educatos can start multiple parallel battles and, if and only if all battles are ended, the educator can finally close the tournament.
The diagram is shown in figure 2.2.

Battle

The battle evolves in a linear way, starting from the *registration opem* 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 students to fork the repository and start working on the battle.
When the deadline for the battle is reached, the educators can start evaluating the code of the students, 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 by 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 educator enabled the manual evaluation, he can assign extra points.
The diagram is shown in figure 2.4.

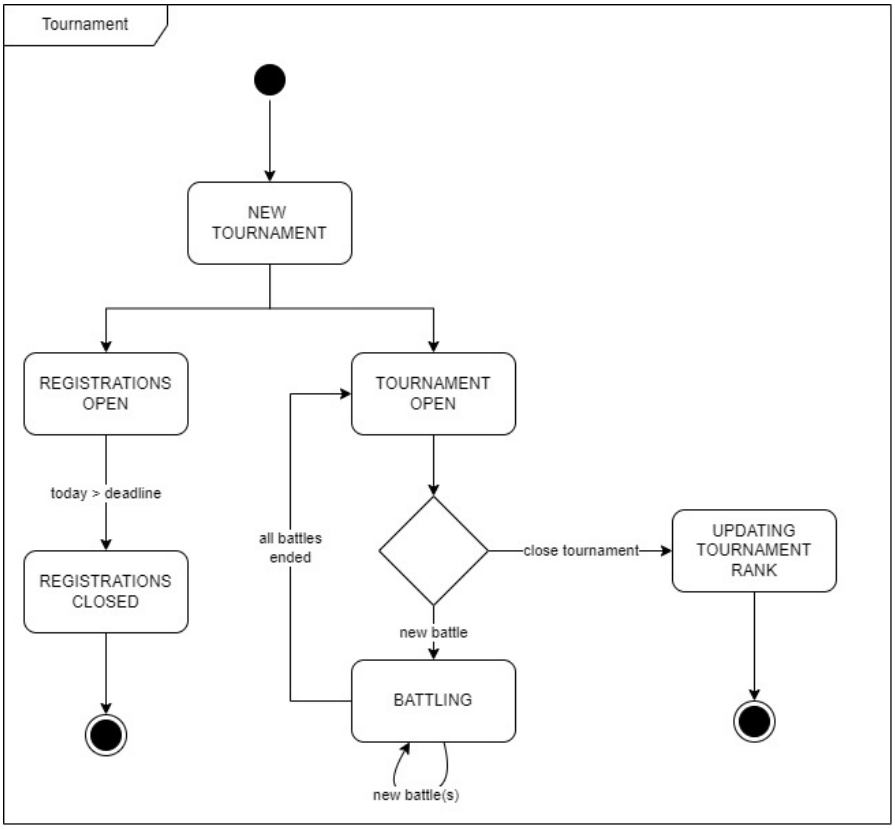

Figure 2.2: Tournament state diagram

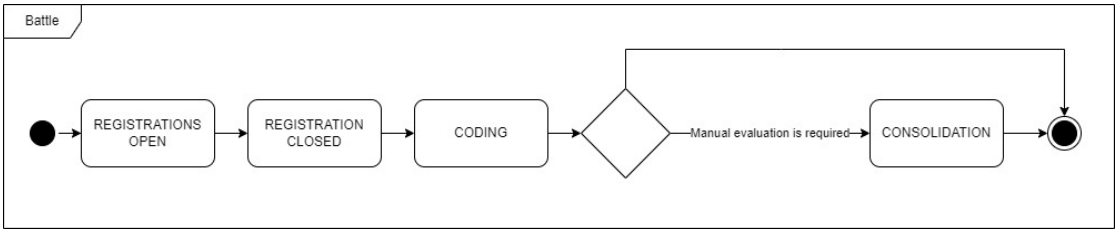

Figure 2.3: Battle state diagram

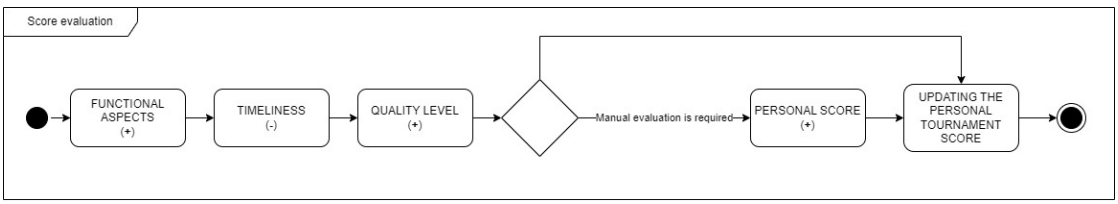

Figure 2.4: Score evaluation state diagram

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 are personal data such as Name, Surname, Email, School they belong to, and what role they hold in the school environment. 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 will be guaranteed. By choosing a STU account, you will be able to actively participate in tournaments and battles by earning badges.

2.2.2 Create tournament function

To create a tournament, an educator registered in CKB platform must define all the information necessary for its generation. The tournament requires:

- a name
- a time window aimed at welcoming student registrations
- a list of badges that students 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. New battles require:

- The programming language required to solve the problem
- The test cases that must be passed at the end of the battle by each team's code
- The build automation scripts correctly set
- The specification of the problem to be solved including at least one example in order to achieve test-first approach
- The deadline for the battle registration period
- The deadline for the conclusion of the battle
- The choice of the evaluation method, in particular, if it can choose 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

Students 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 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

Educators, 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 educator who wishes to use it in a new tournament. When creating a tournament, the educator specifies the list of included badges from those available. At the end of the tournament in which they are enrolled, students receive all the badges for which the specified rule has been satisfied.

2.3 User characteristics

Users of the system fall into one of the following categories: student or educator.

Student

Student participates actively in code kata tournaments and battles to enhance their software development skills. During a battle, students develop solutions following the "test-first" approach and use GitHub to manage the code. The CKB platform automatically evaluates student progress based on the number of tests passed, timeliness, quality of code, providing scores updated in real-time. Students aim to achieve high scores and accumulate gamification badges defined by educators. In addition to participating in battles, students can view their tournaments' rank and receive notifications on final results. The student'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.

Educator

The educator takes a central role in directing students toward improving software development skills through programming battle competitions. Its duties include creating tournaments and competitions, finalizing challenge details, and managing student registrations. The educator assigns specific tasks, manually evaluates students' solutions, and contributes to the automatic evaluation of projects, considering functional, temporal, and code quality aspects. Furthermore, it can create gamification badges, motivating students with personalized rewards based on rules it establishes. The main objective is to improve students' skills, ensuring fair and efficient assessment and encouraging active involvement. The educator 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 educator acts as a promoter of an engaging and competitive training challenge on CKB.

2.4 Assumptions, Dependencies and Constraints

2.4.1 Domain Assumptions

D1. STUs code with the programming language set for the battle to which they are partecipating

D2. EDUs upload the code kata with the description and the correct software project, including test cases and build automation scripts relate to it

D3. 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

D4. EDUs manual evaluation range from 0 to 100[1]

D5. The information inserted at registration moment of all users are truthful

D6. GitHub and the tool for static analysis work properly

D7. A team is formed by at least one person up to the maximum number defined by EDUs.

D8. All users subscribed to the CKB platform have a GitHub account

2.4.2 Dependencies

Dep1	The system will require internet connection to interact with CKB and other users
Dep2	The system will integrate an external API in order to compile the code written by students
Dep3	The system will integrate a GitHub API in order create repository for each battle

Table 2.1: Dependencies

2.4.3 Constraints

- The software must follow local laws and rules, especially when it comes to handling user data, like letting users access their data when they want.

[1]The full evaluation will be given by the average of all the four aspects evaluated by the CKB platform, the three automatic ones and the manual one

- The software should only collect the data it really needs, like just the user's email address.
- 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

In the following, the interfaces, both hardware and software, of the system are described.

3.1.1 User Interfaces

In the following it is shown the mockup of the main pages of the system.

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, he system relies on various external interfaces accessible via uniform web API. These services are:

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

3.2 Functional Requirements

In order to work properly, the software must fulfill the following functional requirements, which are written in hierarchical order, starting from the requirements of EDUs agents and then STUs agents...

3.2.1 Use cases Diagrams

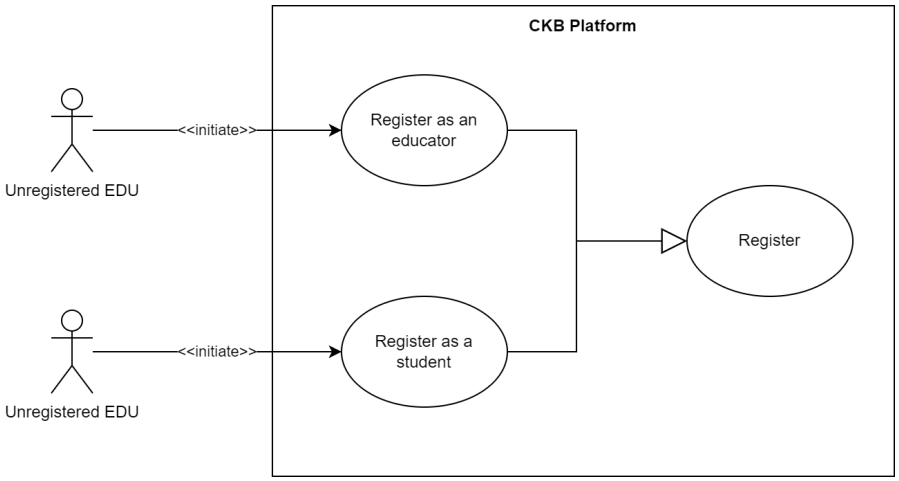

Figure 3.1: Registration Use Case Diagram

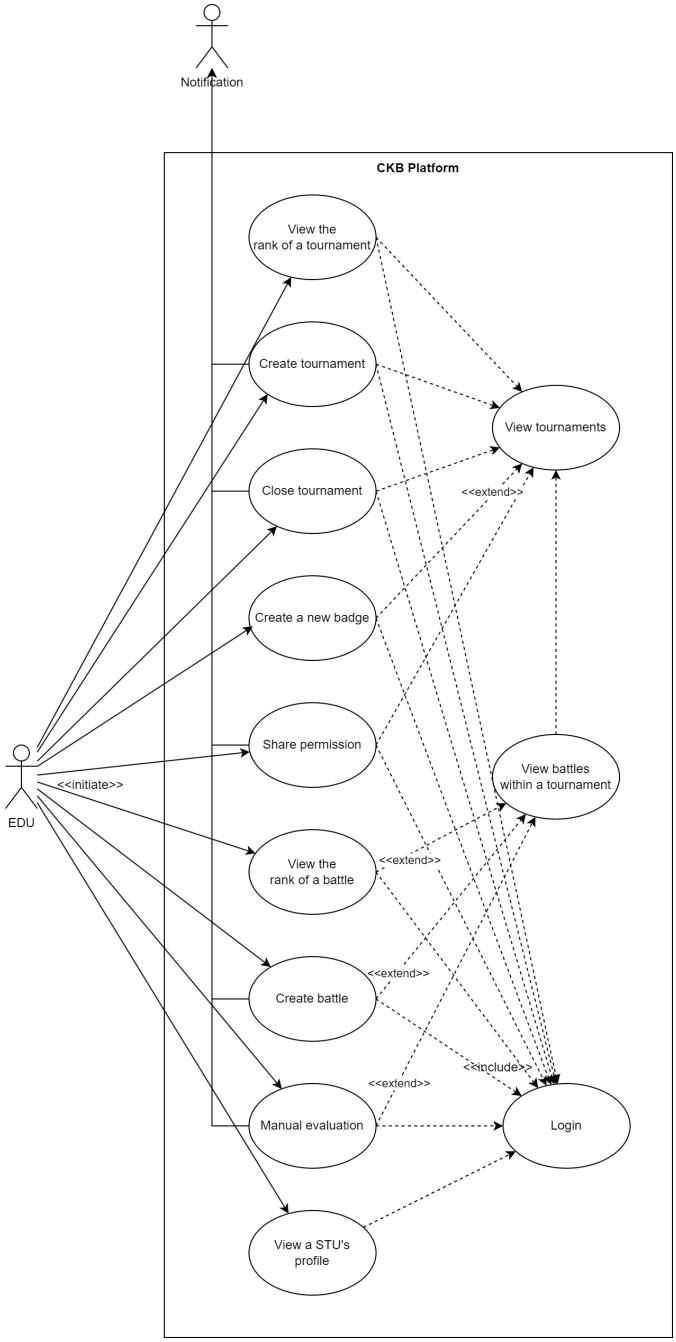

Figure 3.2: EDU Use Case Diagram

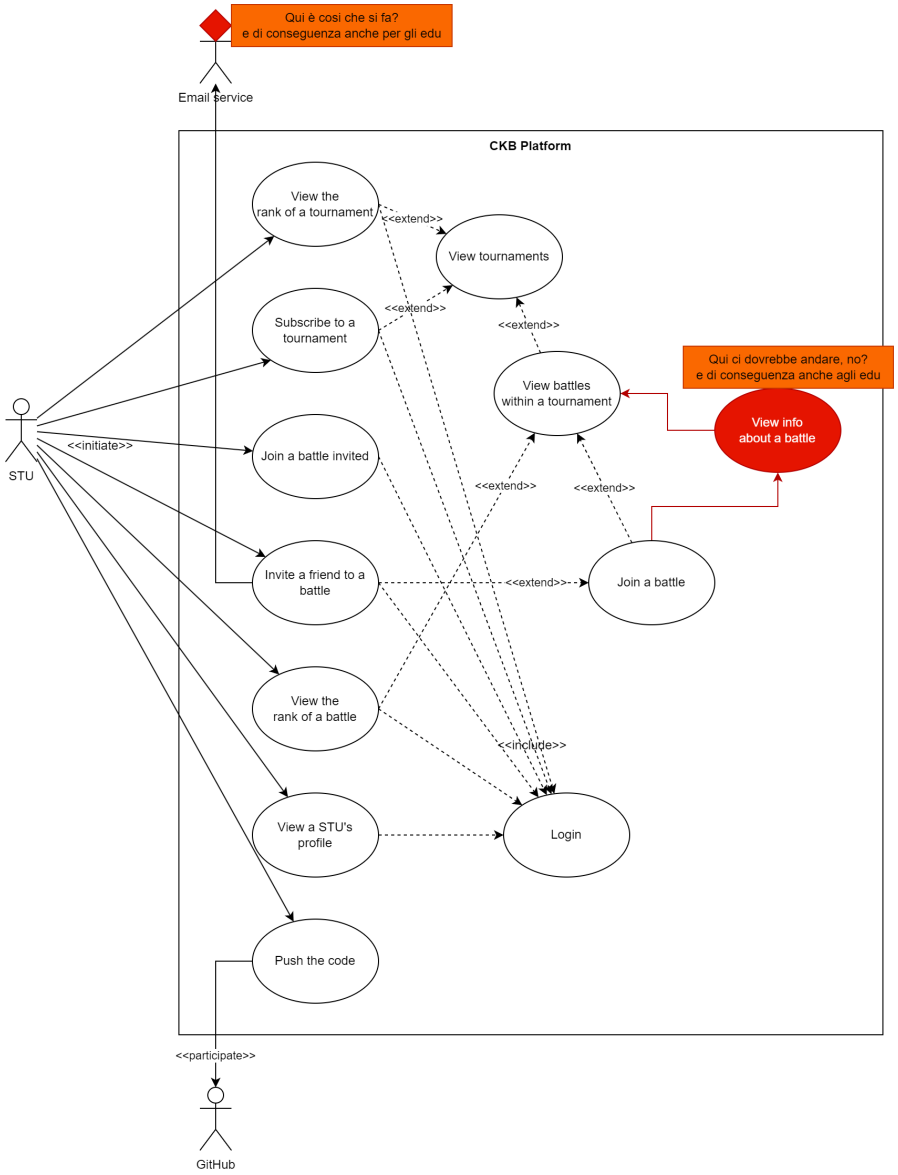

Figure 3.3: STU Use Case Diagram

3.2.2 Use cases Description

Name	Register
ID	UC1 (Figure 3.4)
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 it is an educator or a student 4. The actor fills the form with truthful informations 5. The actor confirms the account creation 6. The system creates an account with respect to the informations inserted by the actor
Exit conditions	The system successfully creates the account
Exceptions	• The actor inserts informations (i.e. email) about an account which already exists: a human-readable message is displayed • The actor inserts data in the wrong format: a human-readable message is displayed

Name	Log in
ID	UC2
Actors	EDU or STU
Entry conditions	The actor is already subscribed to the CKB platform
Event flow	1. The actor enters the login 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 it has access to the home page
Exceptions	The actor inserts a wrong combination of username - password: a human-readable message is displayed

Name	Tournament creation (Figure **??**)
ID	UC3
Actors	EDU, Mail provider
Entry conditions	The user is an authenticated EDU in "Dashboard" page
Event flow	1. User clicks on "Create new" button 2. The system show "Create tournament" tab with a form to be compiled 3. User fills the form with the required information as Tournament name, registration window for STUs, and the optional list of badges 4. User clicks on "Create" button 5. The systems processes the information, if everything is correct it create the new tournament 6. User is redirected to the "Dashboard" page showing the new tournament created
Exit conditions	The system notify all the registered STU about the creation of a new tournament, by sending an email
Exceptions	• At least one of the inserted data is not valid: the system display an error based on the invalid data inserted • A crash happens while creating the tournament: the system must ensure execution atomicity

Name	Battle creation (Figure 3.5)
ID	UC4
Actors	EDU
Entry conditions	The actor is already subscribed to the CKB platform and he is an educator already logged in the platform
Event flow	1. The actor opens the page correlated to the tournament in which he wants to create a new battle 2. The system shows the tournament page 3. If the actor has enough permissions, the system shows the "create a new battle" button 4. The actor opens the battle creation page 5. The system shows the battle creation form 6. The actor fills the form with the details about the battle (e.g. code language, deadlines) 7. The actor uploads the code kata 8. The system checks if all fields are correctly filled 9. The system creates the battle
Exit conditions	The system creates successfully the new battle Maybe we should say that the system send to all the STU in the tournament a notification about the new battle
Exceptions	• The actor has not enough permissions for the selected tournament: the system doesn't show the "create a new battle" button • The actor uploads files not allowed or it inserts data in the wrong format: a human-readable message is displayed

Name	Sharing permission
ID	UC5
Actors	EDU
Entry conditions	A tournament is in progress **and** the actor is logged in **and** it is on the home page
Event flow	1. The EDU selects a tournament it has created 2. The system shows the tournament 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
Exit conditions	The other EDU(s) can now start battles within the selected tournament
Exceptions	*None*
Note	In order to make the research of a specific EDU easier, the system provides a search bar

Name	Join a tournament (Figure **??**)
ID	UC6
Actors	STU
Entry conditions	STU is registered in the system (and) has received the e-mail from the system about the new tournament created
Event flow	1. User open the e-mail and clicks on the link attached to be redirect to the tournament 2. Should we talk about the log-in here? 3. The system show the "Tournament view" page 4. User clicks on "Join tournament" button 5. The system process the request
Exit conditions	STU is now enrolled to that tournament
Exceptions	• The tournament registration window is expired: the "Join tournament" button is disabled • The STU is already part of the tournament: the system display an error

Name	Join a Battle (Figure 3.6)
ID	UC7
Actors	STU, Mail Provider
Entry conditions	The actor is an authenticated STU **and** a tournament is in progress **and** the STU is subscribed to the tournament
Event flow	1. The actor opens the tournament page in which he is subscribed 2. The system shows the battles of the tournament 3. The actor opens the kata battle page 4. If the subscribing deadline is not expired yet, the system shows the kata battle page with the function "join and create a team" 5. The actor clicks on "join and create a team" 6. The system displays a form to invite other members 7. The actor inserts emails of his mates 8. If the invited members are subscribed to that tournament, the system sends an email to each invited member 9. The system subscribes the actor to the battle
Exit conditions	The actor can now invite other STUs to his team or he can wait the start of the battle and partecipating alone
Exceptions	• The deadline expires and the team does not satisfy the constraint for the minimum team participants: the team is not enrolled to the battle • The emails inserted are not subscribed to the tournament : the system displays a human-readable error message
Note	The system offers always the possibility to invite new members and sends via email the link to let other STUs join the team created by the actor

Join battle da mail di CKB: - Da solo -> Team a 1 - Invitando altri giocatori -> Team in base a quanti accettano

Join a team: - Iscritto già alla battaglia -> Accetto e mi unisco al team Declino o

ignoro resto team unico - Non inscritto alla battaglia -> Accetto mi iscrivo alla battaglia e unisco al team Declino o ignoro non mi iscrivo nè al team nè alla battaglia

Name	Join a team when already in the battle (Figure **??**)
ID	UC8.1
Actors	STU
Entry conditions	STU is registered in the system (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 open the e-mail and clicks on the link attached to be redirect to the system 2. Should we talk about the log-in here? 3. The system show the "Dashboard" page with the invitation, if the join windows for the battle is still open 4. User decide to Accept, Decline or Ignore the invite • User click "Accept" button (a) The system process the request, enroll the STU to the battle and to the team (b) User now is part of the team in t he battle • User click "Decline" button (a) The system process the request, and allow the STU that sent the invite to invite a new STU
Exit conditions	...
Exceptions	•

Name	Join a team when not in a battle (Figure **??**)
ID	UC8.2
Actors	STU
Entry conditions	STU is registered in the system (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 open the e-mail and clicks on the link attached to be redirect to the system 2. Should we talk about the log-in here? 3. 4.
Exit conditions	...
Exceptions	•

Name	Manual evaluation
ID	UC9
Actors	EDU
Entry conditions	The deadline for submission a certain battle has expired **and** the battle has the "manual evaluation" enabled **and** the EDU is logged in **and** it is on the tournament page referred to that battle
Event flow	1. The EDU selects a battle he has created 2. The system shows the battle page, that has a list of all the teams that have submitted their work 3. One by one, the EDU selects a team and the system shows the team code 4. The EDU evaluates the code and assigns a score to the team (in the range [0, 100]) 5. The EDU clicks on the "Submit" button 6. The system updates the battle score of the team
Exit conditions	For every team the final score is available, the battle has a final rank and all the participants to the battle get notified
Exceptions	• The EDU doesn't 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 stops the evaluation process: the system saves the scores assigned to the teams and the evaluation process can be resumed later from the first team that hasn't been evaluated yet • 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

Name	Create a new badge (Figure **??**)
ID	UC11
Actors	...
Entry conditions	...
Event flow	1. 2.
Exit conditions	...
Exceptions	•

Name	Closing a tournament
ID	UC11
Actors	EDU
Entry conditions	The actor has the authorization to create battles within the tournament **and** it is on the tournament page **and** the actor is logged in
Event flow	1. The actor clicks on the "Close tournament" button 2. The system shows a confirmation message 3. The actor clicks on the "Confirm" button 4. The system closes the tournament and elaborates the final tournament rank
Exit conditions	All the STUs enrolled to the tournament gets notified and can access to the tournament final ranking
Exceptions	The EDU clicks on the "Cancel" button: the system closes the message and shows normally the tournament page

Name	Visualizing battle ranking (Figure 3.7)
ID	UC12
Actors	EDU and STU
Entry conditions	The actor is in the page of the tournament it is looking for
Event flow	1. The actor searches the battle 2. The actor clicks on the button of the battle 3. The system shows the battle page
Exit conditions	The actor is able to see the ranking of the battle it was looking for
Exceptions	None
Note	If the battle is not started yet, the system will show the battle page and the score will be zero.

Name	Visualizing tournament ranking
ID	UC13
Actors	EDU or STU
Entry conditions	The actor is logged **and** it is on the home 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 page
Exit conditions	The actor can see the tournament rank
Exceptions	*None*
Notes	The tournament page shows both the rank and the list of battles within it, ongoing or ended

Name	Visualize user profile (Figure **??**)
ID	UC14
Actors	...
Entry conditions	...
Event flow	1. 2.
Exit conditions	...
Exceptions	•

Name	Visualizing tournaments (Figure 3.9)
ID	15.1
Actors	EDU
Entry conditions	The actor is logged **and** he is in his dashboard
Event flow	1. The system shows the list of all the tournaments the actor has created 2. The actor selects a tournament, either ongoing or closed 3. The system shows the tournament page Alternatively: 1. The actor clicks on "Show all" button 2. The actor clicks the search bar and insert the name of a tournament he is looking for 3. The system shows the tournament page corresponding to the search
Exit conditions	The actor can see the tournament page
Exceptions	*None*

Name	Visualizing tournaments (Figure 3.9)
ID	15.2
Actors	STU
Entry conditions	The actor is logged **and** he is in his dashboard
Event flow	1. The system shows the list of all the tournaments to which the actor has subscribed 2. The actor selects a tournament, either ongoing or closed 3. The system shows the tournament page Alternatively: 1. The actor clicks on "Show all" button 2. The actor clicks the search bar and insert the name of a tournament he is looking for 3. The system shows the tournament page corresponding to the search
Exit conditions	The actor can see the tournament he has selected
Exceptions	If the tournament does not exists, the system suggests other tournaments showing their names and the subscribing deadlines
Notes	In the dashboard are shwon both the suggested torunaments and the torunaments to which the actor is subscribed

Name	Visualizing battles in a tournament
ID	UC16
Actors	EDU or STU
Entry conditions	The actor is logged **and** it is on the home page
Event flow	1. The actor selects a tournament 2. The system shows the tournament page
Exit conditions	The actor can see the list of all battles within the 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 (Figure 3.10)
ID	UC18
Actors	CKB platform, GitHub, STU
Entry conditions	The registration deadline for a battle has expired **and** STU is subscribed to the battle **and** the battle is not ended yet
Event flow	1. STU pushes his changes to the GitHub repository 2. GitHub actions use the CKB platform 's API to inform of the new changes 3. CKB platform pulls the latest sources and analyzes them 4. CKB platform runs the tests on the corresponding executables 5. CKB platform 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 have assumed that the STU setted properly the GitHub repository and the GitHub actions in the domain assumptions, so, by the side of the platform, nothing could go wrong

Name	Repository creation
ID	UC19
Actors	CKB platform, GitHub, STU
Entry conditions	The registration deadline for a battle has expired **and** STU is subscribed to the battle
Event flow	1. The CKB platform creates a GitHub repository for the battle through the GitHub API 2. The CKB platform 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 4. The STU forks the repository
Exit conditions	The STU and its team can start working on the coding battle
Exceptions	*None*
Notes	The correct functioning are assumed, so, by the side of the platform, nothing could go wrong

3.2.3 Use cases Sequence Diagrams

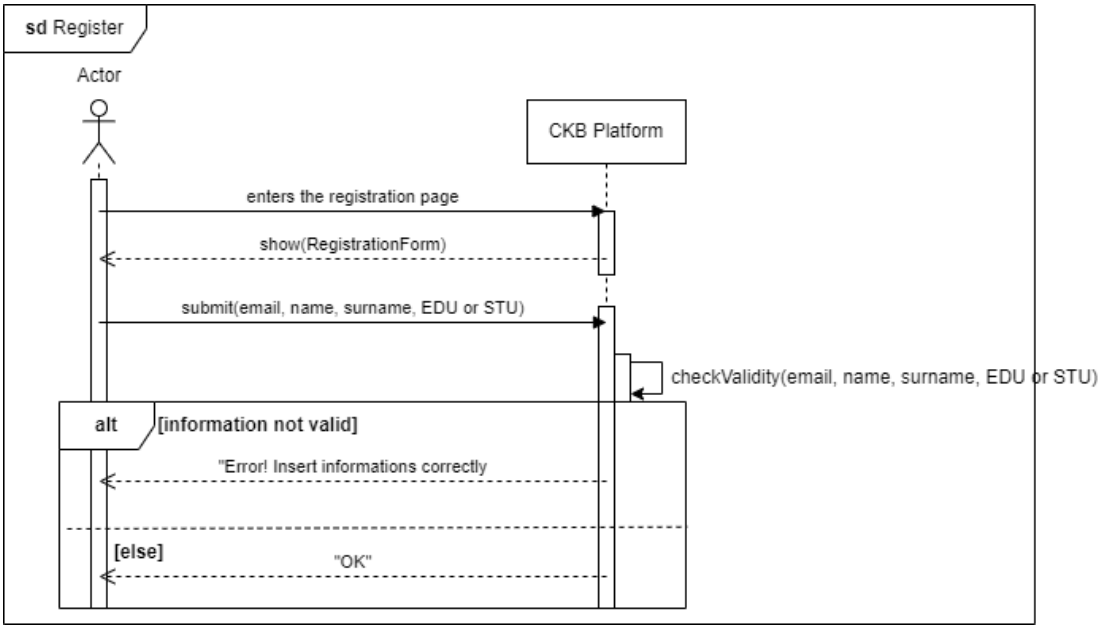

Figure 3.4: Register Use Case

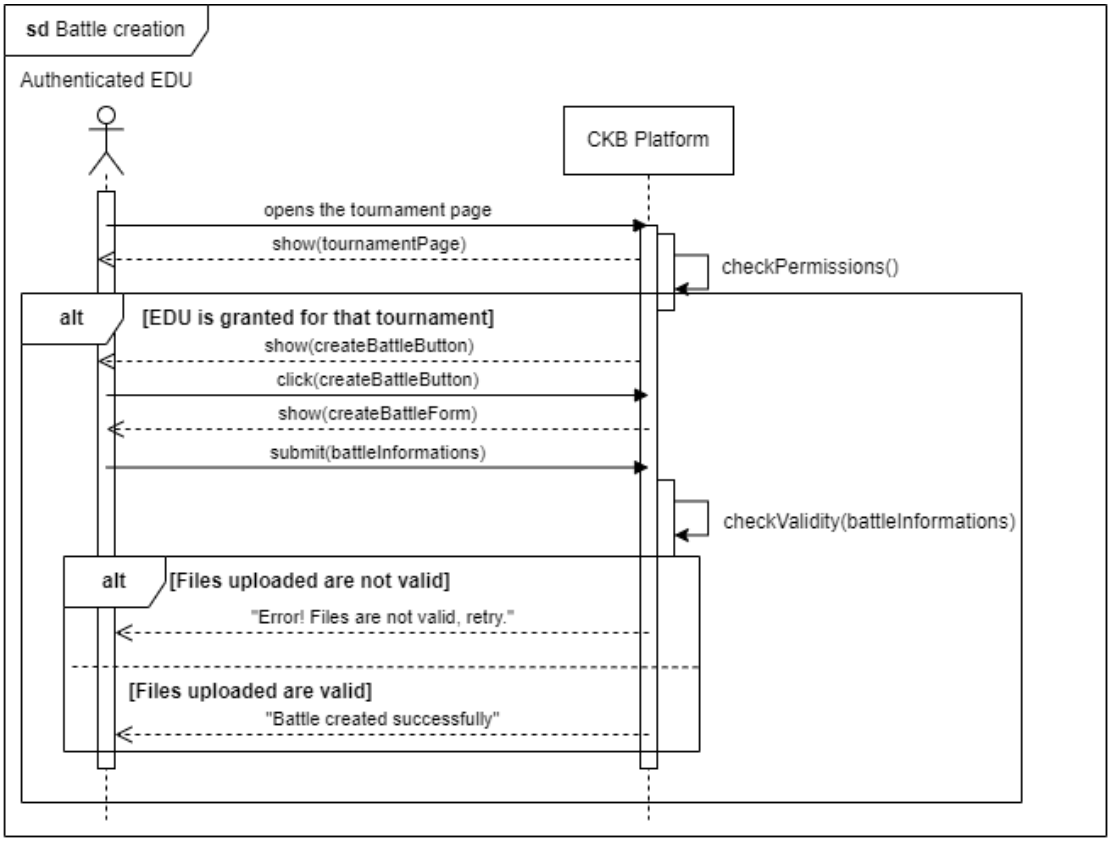

Figure 3.5: Battle creation Use Case

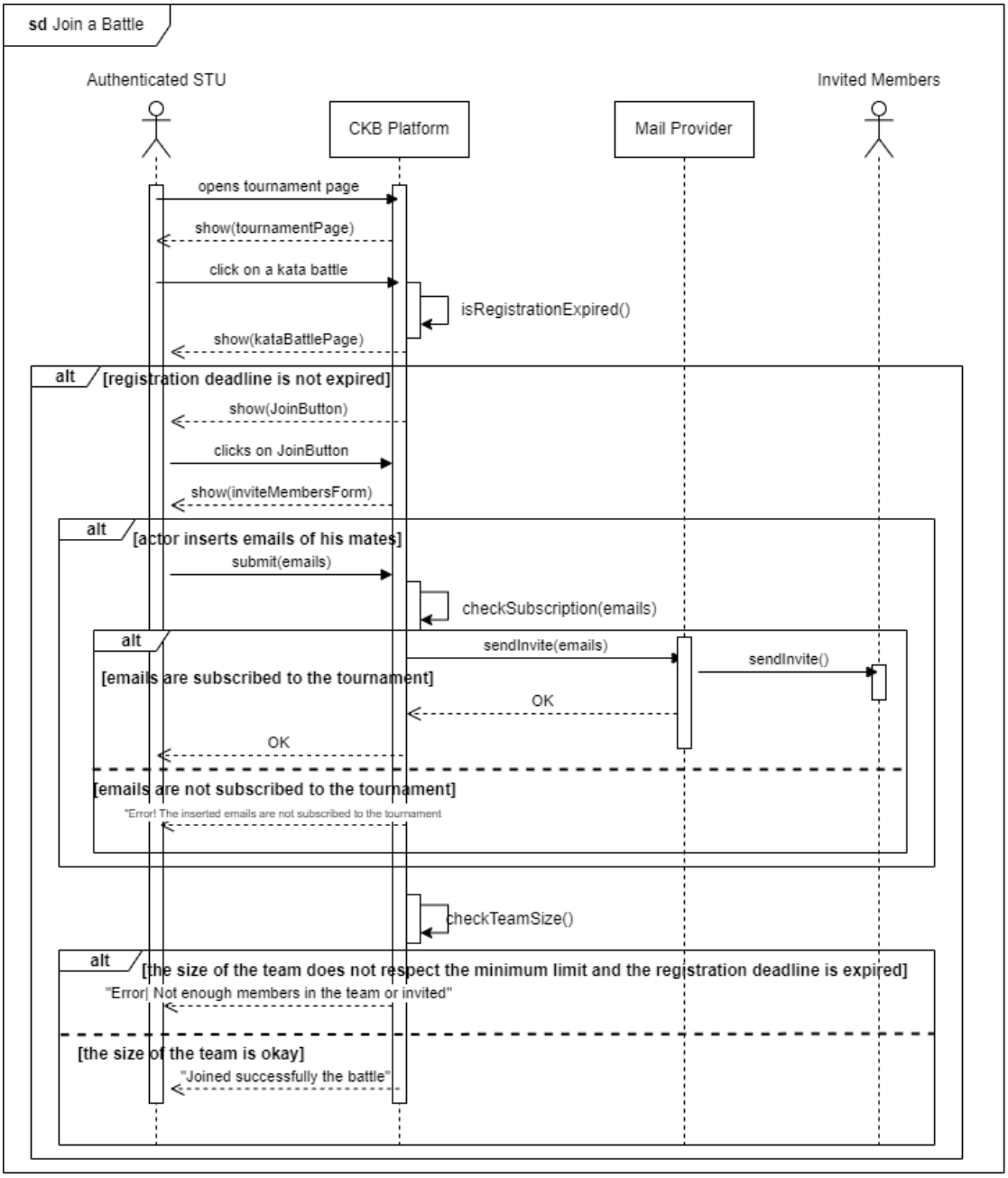

Figure 3.6: Join a Battle Use Case

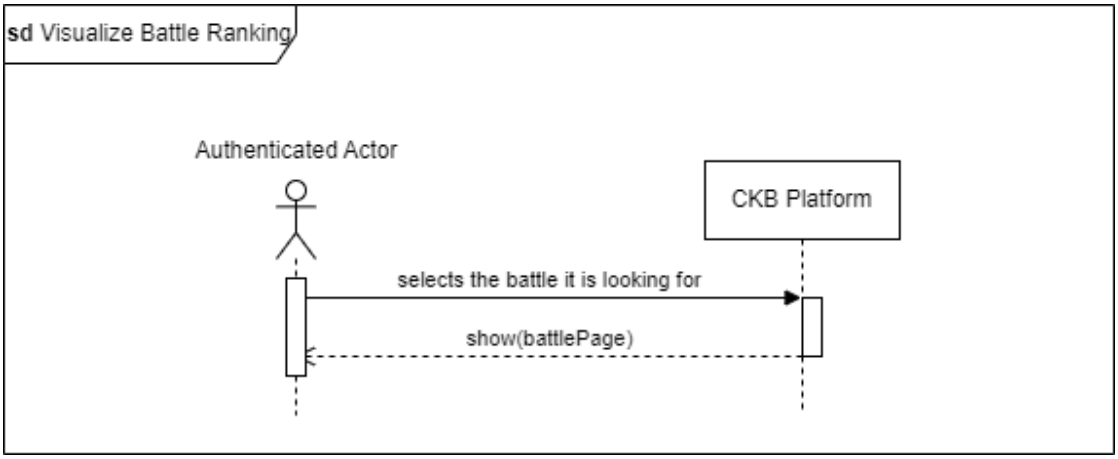

Figure 3.7: Visualizing battle ranking Use Case

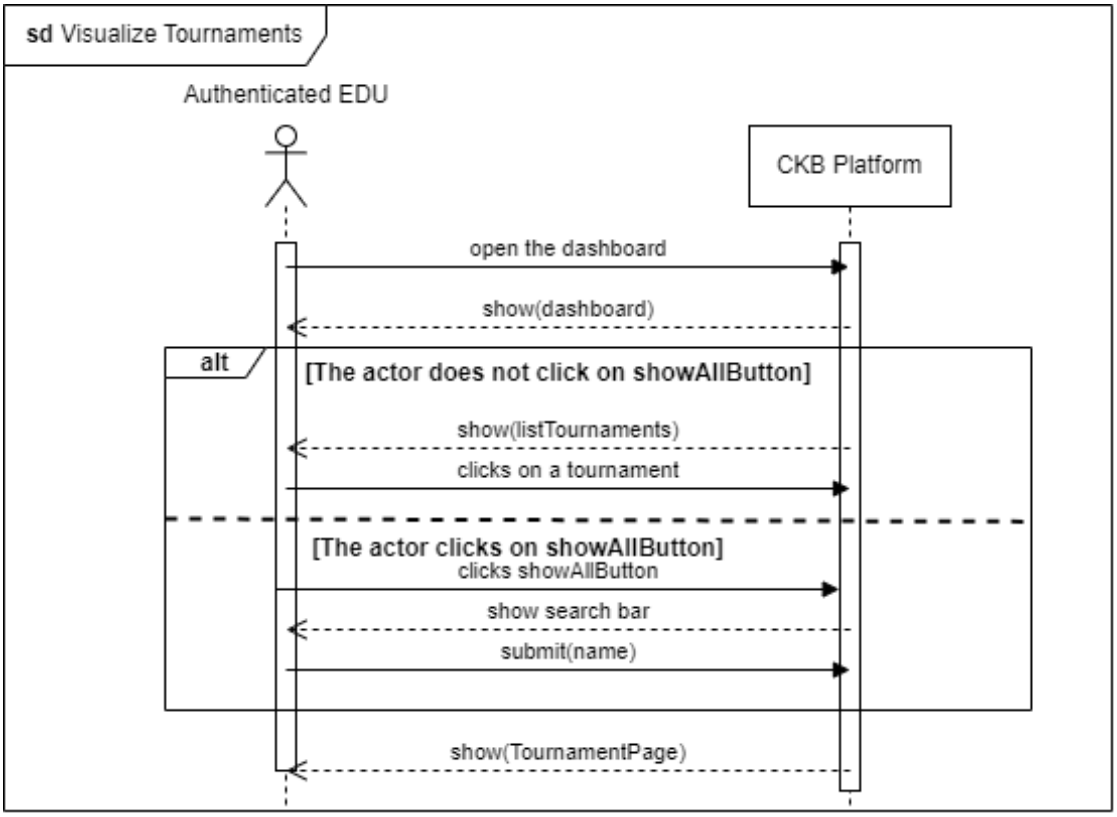

Figure 3.8: Visualizing tournaments Use Case

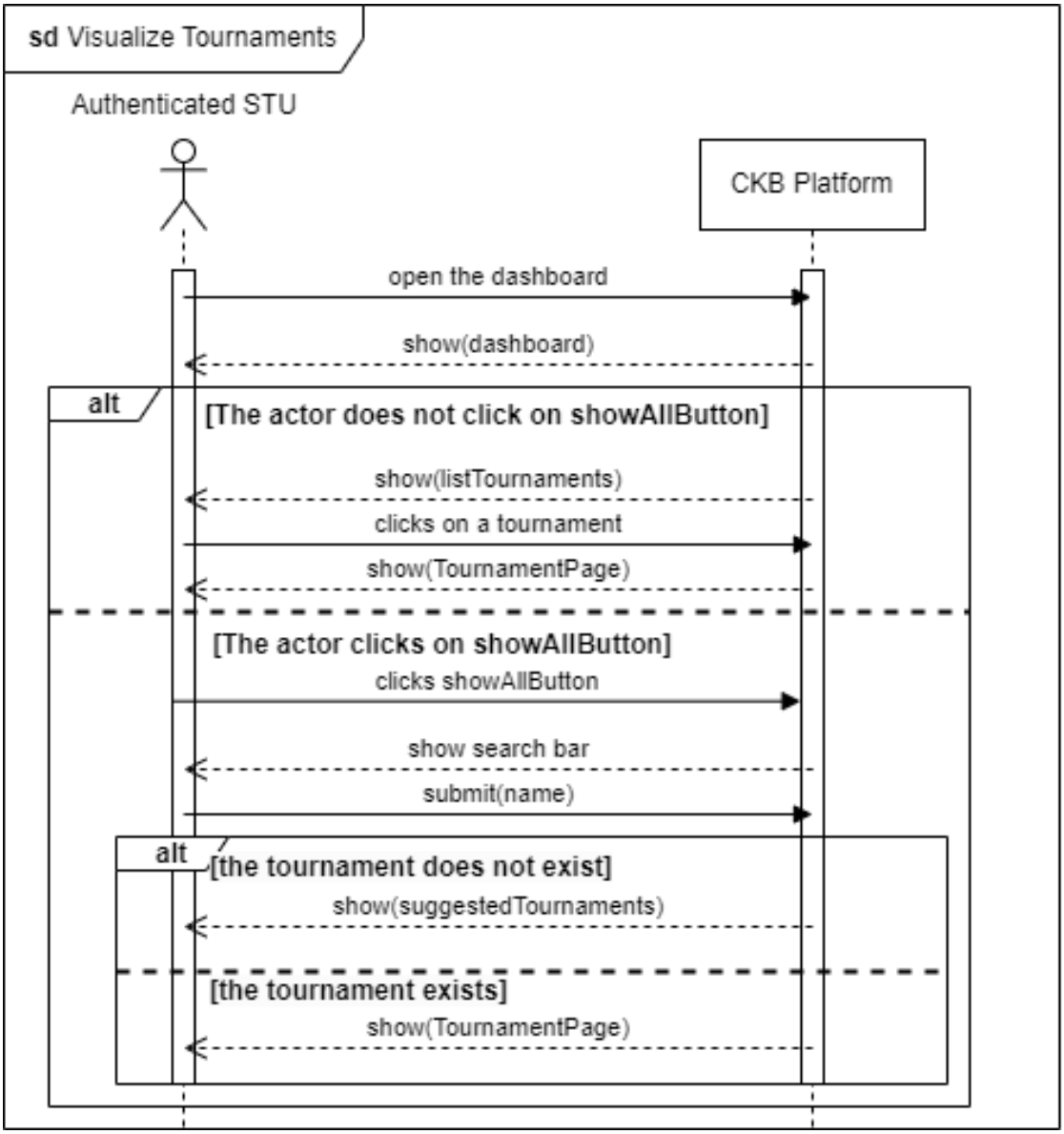

Figure 3.9: Visualizing tournaments Use Case

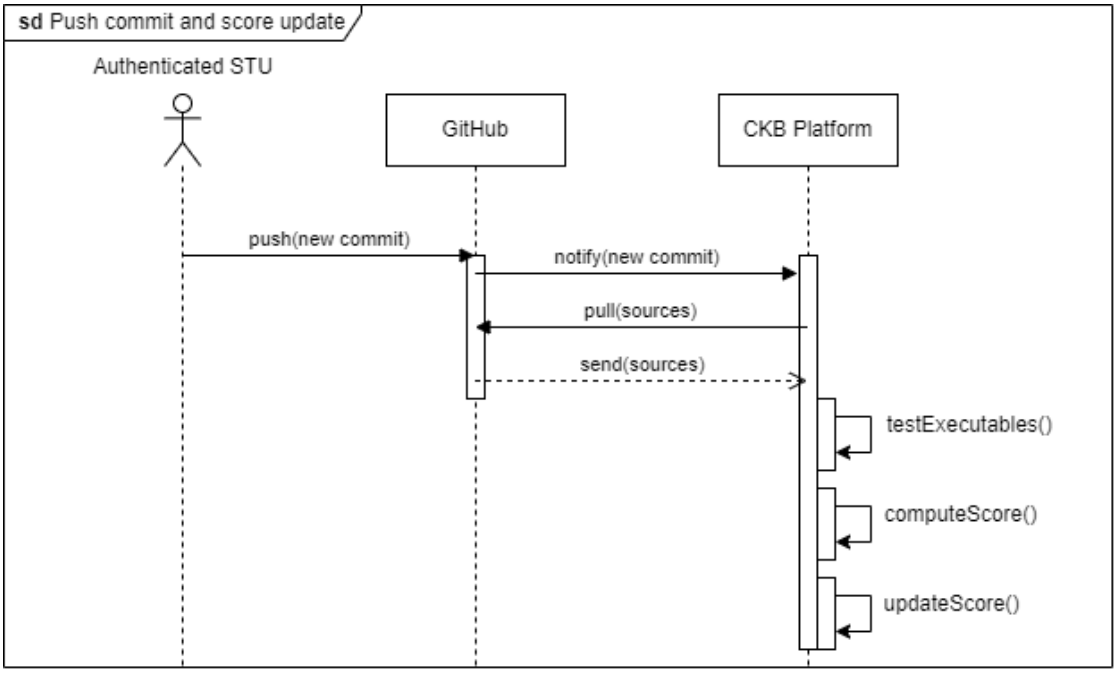

Figure 3.10: Push commit and score updating Use Case

3.2.4 List of functional requirements

EDU functional requirements

R1. The software shall allow the unregistered EDUs to create an account

R2. The software shall allow the registered EDUs to login

R3. The software shall allow the authenticated EDUs to create new tournaments

R4. The software shall allow an authenticated EDU to grant to other authenticated colleagues the permission to create battles in his own tournament

R5. The software shall allow the authenticated EDUs to create coding kata battles in their own tournaments by letting them uploading the code kata, setting minimum and maximum number of STUs per group, setting a registration and a final submission deadlines, setting score configurations

R6. The software shall allow the authenticated EDUs to manually evaluate the work done by STUs subscribed to its own kata battle

R7. The software shall allow the authenticated EDUs to see the sources produced by each team partecipating to their own torunaments

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

R12. The software shall allow an authenticated EDU to close a tournament if and only if 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 to all authenticated EDUs to visualize the badges: in particular, EDUs that visualize the profile of a STU can see its collected badges

STU functional requirements

R16. The software shall allow the unregistered STUs to create an account

R17. The software shall allow the registered STUs to login

R18. 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

R19. The software shall allow the authenticated STUs to join a team by an invite

R20. The software shall allow the authenticated STUs to subscribe to a tournament until a certain deadline

R21. The software shall allow the authenticated STUs subscribed to a kata battle to upload their work until the final submission deadline of that kata battle

R22. When the registration deadline of a kata battle expires, the software shall create a GitHub repository containing the code kata

R23. The software shall send to all authenticated STUs who are members of subscribed teams to a kata battle the link to a GitHub repository containing the code kata

R24. The software shall run the tests on executables pushed by a team and it shall also calculate and update the battle score of the corresponding team

R25. At the end of the consolidation stage of a specific battle b, the software shall send a notification to all authenticated STUs partecipating to b when the final battle rank becomes available

R26. The software shall allow the authenticated STUs to see the list of ongoing and finished battles as well as the corresponding battle rank

R27. The software shall allow to all authenticated STUs to see the personal tournament score of each STU (which is the sum of all battle scores received in that tournament)

R28. The software shall allow to all authenticated STUs to see a rank that measures how a STU's performance compares to other STUs in the context of that tournament.

R29. The software shall allow to all authenticated STUs to see the list of ongoing and finished tournaments as well as the corresponding tournament rank

R30. The software shall notify all authenticated STUs involved in a closed tournament when the final tournament rank becomes available

R31. The software shall allow to all authenticated STUs to visualize the badges: in particular, STUs can see collected badges when they visualize the profile of a STU

3.2.5 Traeciability matrices

Mapping of functional requirements on use cases

Use case	**Functional requirements**
UC1	R1, R16
UC2	R2, R17
UC3	
UC4	R5
UC5	R4
UC6	
UC7	R18, R19
UC8	
UC9	R6, R7
UC10	
UC11	R12

UC12	R11, R26
UC13	R8, R9, R10, R27, R28, R29
UC14	R10, R29
UC15	
UC16	R11, R26
UC17	
UC18	R7, R21, R24
UC19	R21, R23

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, R16, R17, R20, R21, R22, R23
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, R16, R17, R20, R21, R23, R24
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
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, R16, R17, R18, R19

3.3 Performance Requirements

In order 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 isn't great, so users have a smooth experience.
- Send email notifications really quickly, so users don't even notice the delay.

3.4 Design Constraints

Standards Compliance

Since there are a lot of interactions betweeen the various components of the system, it is important to follow some communic standards: in particular, the system will use the REST architectural style to communicate between the frontend and the backend, and the data will be exchanged in JSON format.
Furthermore, 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, though the system should not completely fail because of failure in one of those.
It's 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, it is crucial for the CKB platform to 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 a 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 doesn't rely on a specific hardware or software, it can be used / accessed in multiples way.

4. Formal Analysis Using Alloy

4.1 Signatures

```
    open util/relation
//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 : one Name,
```

```
        surname: one Surname,
        email: one Email,
        password : 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,
    }

/*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 : one Int,
        subscriptionDeadline : one DateTime,
        ranking : set RankingStudent,
        grantedEducators: some Educator,
        battles: 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 : one Int,
closed : one Bool,
      rankingTeams : set RankingTeam,
      manualEvaluation: one Bool,
      language: one Language,
      description: one Description,
      testCases: some TestCase,
      minStudents : one Int,
      maxStudents : one Int,
      registrationDeadline : one DateTime,
      finalSubmissionDeadline : one DateTime,
      subscribedTeams: 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 : some Rule,
      values : some Int,
      title : one Title,
  }

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

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



-- Facts
-- User
fact emailsAreUnique{no disj u1, u2: User  | u1.email=u2.
   email}
fact nameExistsOnlyWithUser{
  all un : Name | one u : User | un in u.name
}
fact surnameExistsOnlyWithUser{
  all un : Surname | one u : User | un in u.surname
}
fact passwordExistsOnlyWithUser{
  all un : Password | one u : User | un in u.password
}
fact emailExistsOnlyWithUser{
  all un : Email | one u : User | un in u.email
}

--TestCase
fact TestCaseExistsOnlyWithBattle{
  all tc: TestCase | one b: Battle | tc in b.testCases
}
```

```
--Battle
fact BattleExistsOnlyWithTournament{
  all b : Battle | one t : Tournament | b in t.battles
}
fact allBattlesAreUnique{
  no disj b1, b2 : Battle | b1.id = b2.id
}

--Team
fact TeamlExistsOnlyWithBattle{
  all t : Team | all b: Battle | t in b.subscribedTeams <=> b
      in t.battle
}
fact allTeamsAreUnique{
  no disj t1, t2 : Team | t1.battle = t2.battle and t1.
     students = t2.students
}

--Tournament
fact allTournamentsAreUnique{
  no disj t1, t2 : Tournament | t1.id = t2.id
}
--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| no s: Student | s in t.battles.
             subscribedTeams.students and s not in t.
             studentsSubscribed
      }
fact teamIsSubscribed{
  all t: Team| all b : Battle | t in b.subscribedTeams <=> b
     = 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 partecipate 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 partecipating to a battle if and only if it
           is in a subscribed team of that battle*/
        fact inBattleIfInTeam{
            all s : Student | all b : Battle | all t: Team| b
               in s.battles <=> ( s in t.students and t in b.
               subscribedTeams)
        }

pred show{
  #Battle = 2
  one b : Battle |  b.minStudents = 2 and b.maxStudents = 4
  one t1 : Team | #t1.students >= 2
}
        run show
```

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

Table 5.1: Effort Spent during team meetings

Tommaso Pasini

Topic	**Time**
Organizaionion document	1.5 h
Completion and correction chapter 1	4h

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	4h30m
Traeciability Matrix	30m
Class Diagram	1h
Alloy	9h

Table 5.4: Effort Spent by Michelangelo Stasi