POLITECNICO DI MILANO

Computer Science and Engineering

Requirements Analysis and Specifications Document

CodeKataBattle

Software Engineering 2 Project

Academic year 2023 - 2024

22 December 2023

Version 1.0

Authors:
Massara Gianluca
Nguyen Ba Chiara Thien Thao
Nicotri Flavia

Professor:
Matteo Giovanni Rossi

Contents

Chapter 1

Introduction

1.1 Purpose

In the last years, more and more students are becoming interested in programming and many educators have realized the need of new innovative methods to improve coding skills. CodeKataBattle aims to assist students in enhancing their programming skills by challenging them with creative tasks in a competitive and stimulating environment.

The following document wants to describe the system focusing on the requirements and specifications, providing scenarios and use cases to specify what the system must do and how it should interact with the stakeholders.

1.1.1 Goals

In the following table we describe the main goals that our system wants to achieve.

Goal	Description
G.1	Allow students to compete in a tournament.
G.2	Allow educators to create challenges for students.
G.3	Allow educators to grade students' projects.
G.4	Allow students to collect badges in a tournament proving their skills.

1.2 Scope

The main actors of the system are students and educators.
Educators can:

- **Create a tournament:** decide which colleagues can create battles within the tournament and define badges that represent the achievements of individual students;
- **Create a battle:** set configurations and rules for that battle;
- **Evaluate projects:** manually assign a personal score to the students' works.

Students can:

- **Join a tournament**;

- **Participate in a battle:** create a team and complete the project with his code;
- **Collect badges:** based on the rules of the tournament and his performance.

1.2.1 World Phenomena

W.P.	Description
WP.1	An educator wants to create a tournament.
WP.2	An educator wants to create a new battle.
WP.3	A student wants to join a tournament.
WP.4	A student wants to join a battle.
WP.5	An educator evaluates the works done by students.

1.2.2 Shared Phenomena

S.P.	Description	Controlled by
SP.1	The system notifies the student about upcoming battles.	Machine
SP.2	The student commits his code.	World
SP.3	The educator configures the tournament rules.	World
SP.4	The student forms a team.	World
SP.5	The educator grants other colleagues the permission to create battles within a tournament.	World
SP.6	The educator configures the battle.	World
SP.7	The student joins a team.	World
SP.8	The system creates a GitHub repository.	Machine
SP.9	The student forks and sets up the GitHub repository.	World
SP.10	GitHub Actions notifies the system about students' commits.	World
SP.11	The system shows the battle score of the team.	Machine
SP.12	The system notifies when the final battle rank becomes available.	Machine
SP.13	The system shows the tournament rank.	Machine
SP.14	The system shows the list of ongoing tournaments.	Machine
SP.15	The system shows the student's badges.	Machine
SP.16	The educator defines the badges of a tournament.	World
SP.17	All users can visualize the profile of a student.	World
SP.18	The educator evaluates the students' works.	World
SP.19	The system shows the tournament score of the students.	Machine

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Definitions	Meaning
Code Kata	A code kata battle is a programming exercise in a programming language of choice. The exercise includes a brief textual description and a software project with build automation scripts that contains a set of test cases that the program must pass, but without the program implementation.
Tournament	A tournament is a set of battles. It is created by an educator and it is composed by a set of rules and possible badges.
Battle	A battle is a competition between teams. It is created by an educator and it is composed by a set of rules.
Team	A team is a group of students competing in a battle.
Badge	A badge is a reward that a student can obtain by participating in a tournament.

1.3.2 Acronyms

Acronyms	Meaning
CKB	CodeKataBattle

1.3.3 Abbreviations

Abbreviations	Meaning
WP	World Phenomena
SP	Shared Phenomena
G	Goal
R	Requirement
UC	Use Case
D	Domain Assumption
i.e.	id est
etc.	etcetera

1.4 Reference Documents

- Course slides on WeBeep.
- RASD assignment document.

1.5 Document Structure

The structure of this RASD is the following:

1. **Introduction**: In this chapter the purpose of this document is presented highlighting in particular the main goals, the audience which is referred to, the identification of the product and application domain describing world and shared phenomena and, lastly, the terms definitions.
2. **Overall Description**: This chapter describes the possible scenarios of the platform, the shared phenomena presented at the beginning of the document and assumptions on the domain of the application.
3. **Specific Requirements**: This chapter includes all the requirements in a more specific way than the "Overall Description" section. Moreover, it is useful to show functional requirements in terms of use cases diagrams, sequence/activity diagrams.
4. **Formal Analysis Using Alloy**: This chapter includes Alloy models which are used for the description of the application domain and his properties, referring to the operations which the system must provide.
5. **Effort Spent**: This chapter shows the effort spent in terms of time for each team member and the whole team.
6. **References**: This chapter includes all documents that were helpful in drafting the RASD.

Chapter 2

Overall Description

2.1 Product perspective

In this section we analyze a list of real scenarios and diagrams illustrating further details about shared phenomena.

2.1.1 Scenarios

1. **Registration**
   Prof. White reads about CodeKataBattle in a online spot and decides to use the platform to challenge his students. So, he creates an account using his email address and personal information and convinces his colleagues to join the platform.
   Alice and her classmates, after prof. White's lesson, are interested in improving their coding skills and sign up to CKB. Alice provides her email address and personal information; after the email confirmation goes well, she can complete the registration process by linking her CKB account to her GitHub account.
   After the registration, both Prof. White and Alice can visualize the profile of all the other students and the information about the ongoing tournaments.

2. **Creation of tournaments**
   Prof. Brown and Prof. Smith want to decide whose class is better at programming. So Prof. Brown creates a tournament on CKB and invites his colleague to create battles within the tournament. Prof. Brown sets a deadline by which their students can subscribe. A notification is sent to all CKB students.

3. **Creation of battles**
   Prof. Bloom has been granted the permission to create battles in a tournament, so he uploads a code kata. First he writes a brief textual description of the exercise, then he includes the software project with build automation scripts and test cases. He also sets some rules: minimum and maximum number of students per group, registration deadline, final submission deadline, and automated evaluation parameters.
   Finally, he chooses to include an optional manual evaluation of his students' work.

4. **Students join a battle**
   Bob registered to the CKB platform and received a notification about an interesting tournament and he subscribed to it.
   He is notified about a new interesting upcoming battle in that tournament. Before the registration deadline expires, Bob can form a group, so invites his girlfriend Eva to join

him. After the registration deadline ends, they are sent the link of the GitHub repository of the code kata. To be able to start working on the project, Bob and Eva must fork the repository and set up an automated workflow through GitHub Actions to inform the CKB platform about their commits.

5. **Upload of a solution**
   Charlie has joined a battle with his friends and thinks that they have found a good solution to the assigned problem. So he pushes his code before the final deadline to the main branch of their repository.
   This battle does not expect any manual evaluation by the educator, so as soon as they push the solution, the platform evaluates it and they can visualize their updated battle score. After the final deadline has expired, Charlie can also see the current tournament rank.

6. **Evaluation of a project**
   When Prof. Cooper created the battle, he decided to include manual evaluation in the battle rules: he wants the system to assign up to 75 points, while he will assign the remaining 25 points. So, scores will always be between 0 and 100.
   After the final deadline, prof. Cooper can see all the groups' work and he evaluates them according to some personal parameters such that design quality, code cleanliness, compliance with specifications.

7. **Creation of gamification badges**
   Prof. Moore has created a tournament and to spice things up he decides to include badges. These are awarded according to the rules specified by him at tournament set up time: in particular, he wants to reward the student with most commits and the student who have attended the most battles.
   The badges are assigned at the end of the tournament and the collected badges can be seen by everyone visiting the student's profile.

2.1.2 Class diagrams

The UML class diagram below represents a conceptual, high-level model of the system to be. At this level, it does not include any references to methods and other low-level details, those will be detailed during the design phase.

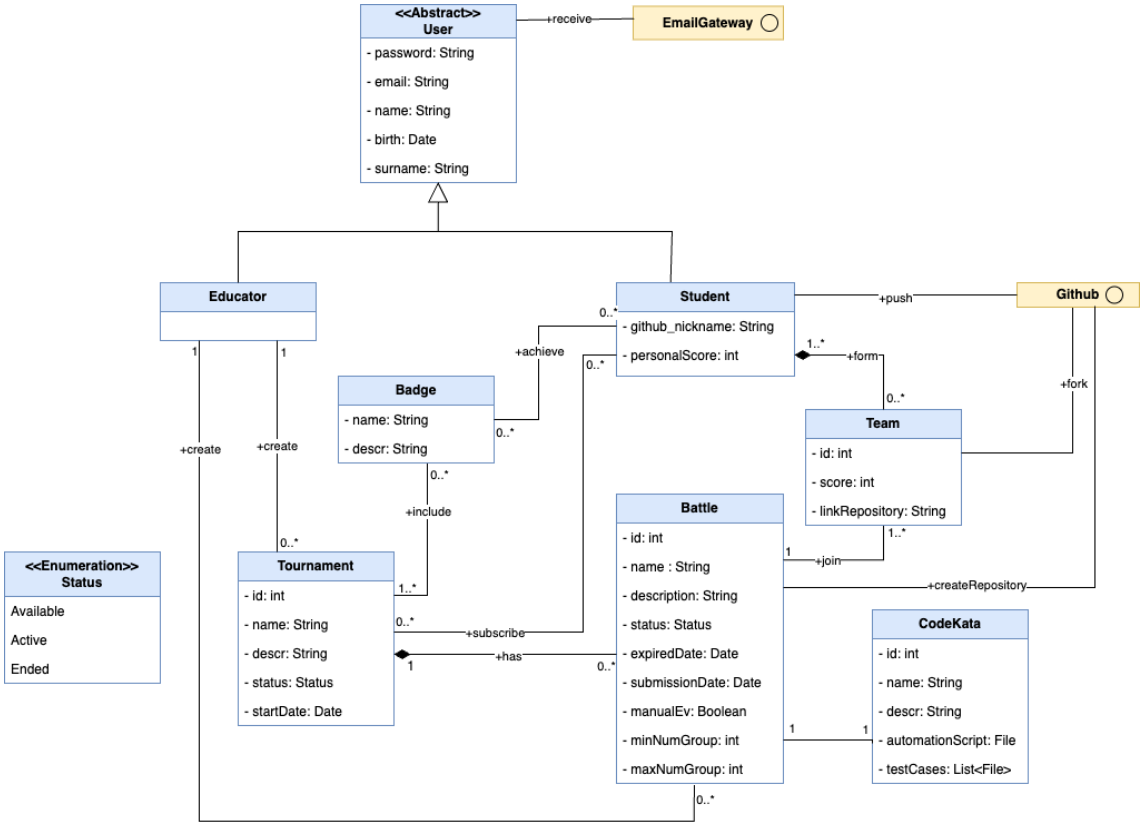

Figure 2.1: Class diagram

The main entities in the diagram are:

- **User:** represents an user of the system who can access to the platform with his credentials.
- **Student:** represents a student, a particular type of user who can join tournaments and battles.
- **Educator:** represents an educator, a particular type of user who can create tournaments, battles and badges.
- **Tournament:** represents a tournament, a programming competition composed of various battles created by an educator.
- **Battle:** represents a battle with a specific code kata created by an educator, in which teams of students can compete.
- **Team:** represents a group of students who can compete in a battle.
- **Badge:** represents an award that educator can create for a tournament and assign to eligible students.
- **CodeKata:** represents a programming exercise in a programming language chosen by its creator. It includes automation scripts and test cases.

2.1.3 State diagrams

State diagrams model the behavior of a single object and are used for objects with significant dynamic behavior; they also specify the sequence of states that an object goes through during its lifetime in response to stimuli from the environment. In particular, they show the life history of a given class, the events that cause a transition from one state to another and the actions that result from a state change.
In this section we will represent the main state diagrams of the whole system.

Tournament
In the following figure the possible status of a tournament is shown.

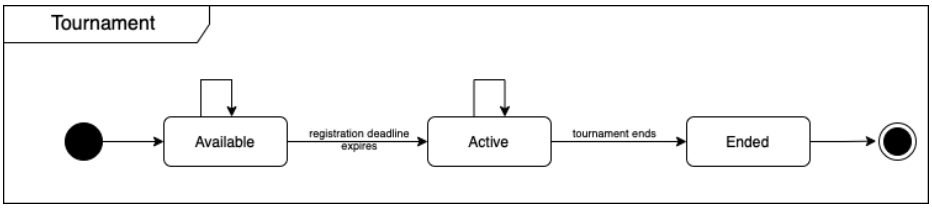

Figure 2.2: Tournament state diagram

Battle
In the following figure the possible status of a battle is shown.

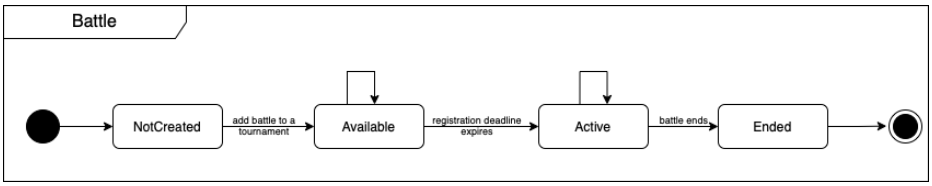

Figure 2.3: Battle state diagram

2.2 Product functions

2.2.1 Sign up and login

These functions are available to both students and educators.
The sign up functionality allows users to register on the platform: in particular, each user will provide an email and a password. Also, the user must provide their personal information (name, surname, date of birth) and if the user is a student they are required to provide a GitHub account.
Then a verification email is sent to the user. Once registered, both students and educators can visualize the profile of all the other students.

The login functionality allows users to access the platform using their email and password.

2.2.2 Creation of a tournament

Educators can create tournaments, programming competitions composed of various code kata battles.
The creator of the tournament can invite other educators and grant them the permission to create battles within the context of the tournament.
When they create a tournament, educators set a registration deadline and can also define badges, i.e., awards that represent particular achievements of individual students: the educator may use pre-defined variables and rules or create new ones. Badges are awarded automatically at the end of a tournament.
Information about ongoing tournaments is available to all subscribed users.

2.2.3 Creation of a battle

After being granted such permission from the creator of the tournament, educators can create code kata battles within the context of the tournament.
To create a battle, the educator must upload the code kata, set the rules regarding groups composition (minimum and maximum number of members), set a registration deadline, set a final submission deadline, and set evaluation parameters. They may also specify if they want to include a manual evaluation of the students' work, in addition to the mandatory automated one, after the submission deadline expires.
Users can see the current battle rank evolving during the battle.

2.2.4 Joining a tournament

Students can join any tournament before its registration deadline expires. By signing up to a tournament, they also subscribe to notifications regarding upcoming battles within the context of that tournament.
Students also receive a notification when the tournament ends.
By participating in a tournament, students can earn badges that can be visualized in their profile.

2.2.5 Joining a battle

Students subscribed to a tournament can join any upcoming battle before its registration deadline expires.
Students can participate alone or in groups to the battle, according to the rules set for that battle by its creator.
When the registration deadline expires, students are sent an email with the link of the GitHub repository containing the code kata. To start working on the project, they are required to fork the repository and set up an automated workflow, using GitHub Actions, to inform CKB about new commits into the main branch of their repository.

As soon as students push a solution it is automatically evaluated, so they can see their current battle score and their battle ranking.
When the submission deadline (and, if required, the manual evaluation stage) ends, students are notified about their final battle score and ranking and they can see their updated tournament score.

2.2.6 Evaluation of a project

According to the parameters set by the creator of the battle, the system automatically evaluates the students' work with regard to

1. functional aspects (i.e., the percentage of passed test cases);
2. timeliness (i.e., the time passed between the registration deadline and the last commit);
3. code quality (extracted using static analysis tools).

If the educator has decided to include manual evaluation in the battle rules, they can see all the groups' work and evaluate the students' work according to some personal parameters such as design quality, code cleanliness, compliance with specifications.
The final score is a natural number between 0 and 100 calculated as specified at battle creation time.

2.3 User characteristics

The CKB system has two different types of actors who use the system:

- **Educator:** people who create tournaments and battles, and evaluate the students' work. To create a battle they set up some settings such as the code kata (description and software project, including test cases and build automation scripts), the minimum and maximum number of students per group, the registration deadline, the final submission deadline, and the configurations for scoring. They can also grant other educators the permission to create battles within a tournament. For each tournament, they may define specific badges establishing new rules and variables in order to reward capable students. Optionally, they can manually evaluate students' code based on their requirements and assign their personal score.
- **Student:** people who join tournaments and battles created by educators, and commit their code on GitHub. They can create a team inviting other students and compete in a battle with other teams. At the end of each tournament they can visualize the final rank and their current score. If educators has defined some specific badges for the tournament, students can achieve them if they have fulfilled the rules. They can visualize the achieved badges on their personal profile.

2.4 Assumptions, dependencies and constraints

2.4.1 Domain assumptions

D1	Users provide correct information during the registration process.
D2	Students have an email and a GitHub account.
D3	Users have a stable and reliable Internet connection.
D4	Students know the functioning of GitHub and GitHub Actions.
D5	Educators create at least one battle within a tournament.
D6	Students must give consent to access their GitHub account.
D7	Educators have a good knowledge of programming.
D8	Educators upload a correct code kata.

2.4.2 Dependencies

Dep1	The system requires access to GitHub API.
Dep2	The system will use a third party API to send emails to the users.
Dep3	The system will require internet connection to interact with all the users.
Dep4	The system requires access to a development environment to run students' code.

2.4.3 Constraints

- The system shall be compliant to local laws and regulations, in particular users data should be treated according to the GDPR. This means that users should be always able to request their data.
- The system should collect only necessary data, such as name, surname, date of birth, email address, etc.
- To better protect the users' sensitive information, such as their personal information and credentials, their data should be encrypted.
- The external APIs, especially those critical for the correct functioning of the system, must be chosen among those with the highest availability and reliability.

Chapter 3

Specific Requirements

3.1 External Interface Requirements

In this section details about user interfaces, hardware and application programming interfaces are described.

3.1.1 User Interfaces

In this subsection we present user interfaces for both types of users, educators and students.

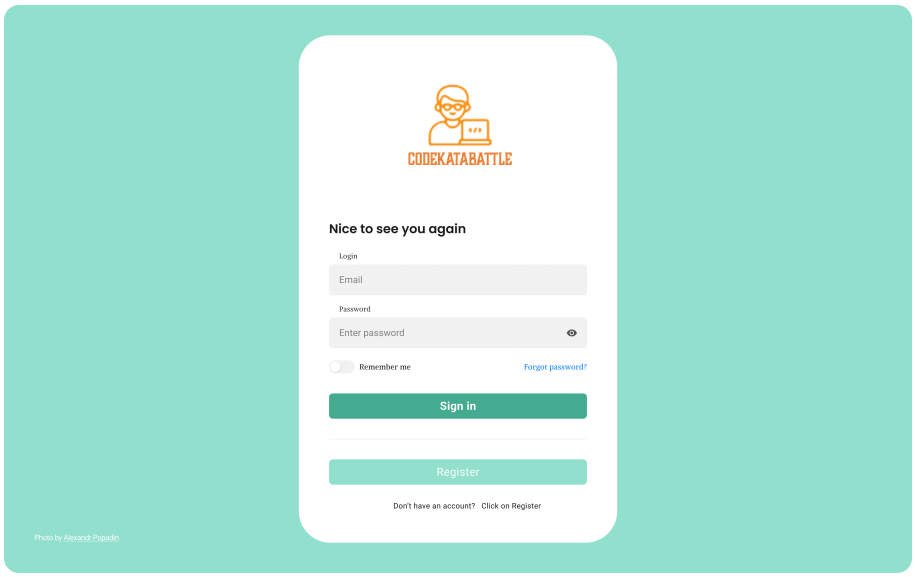

Figure 3.1: Login page - same for all types of users

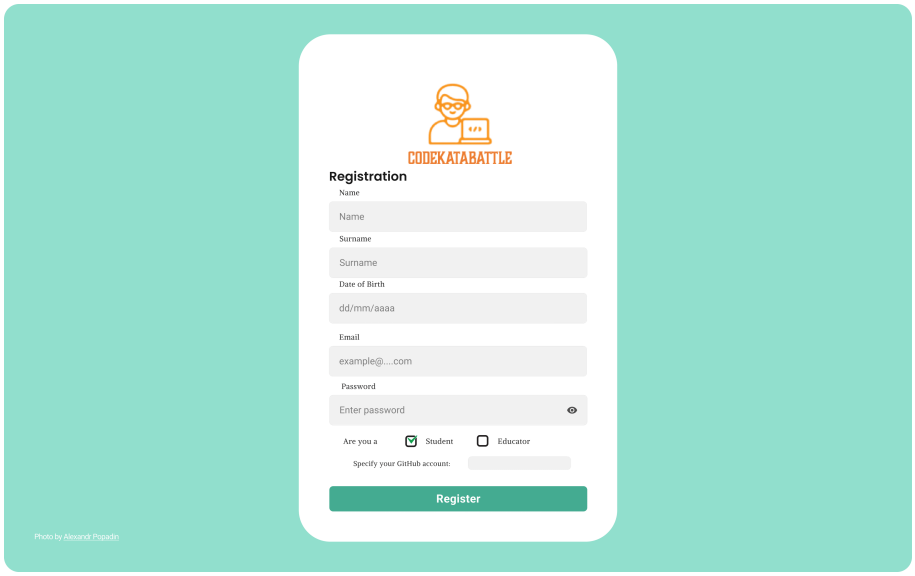

Figure 3.2: Registration page - same for all types of users

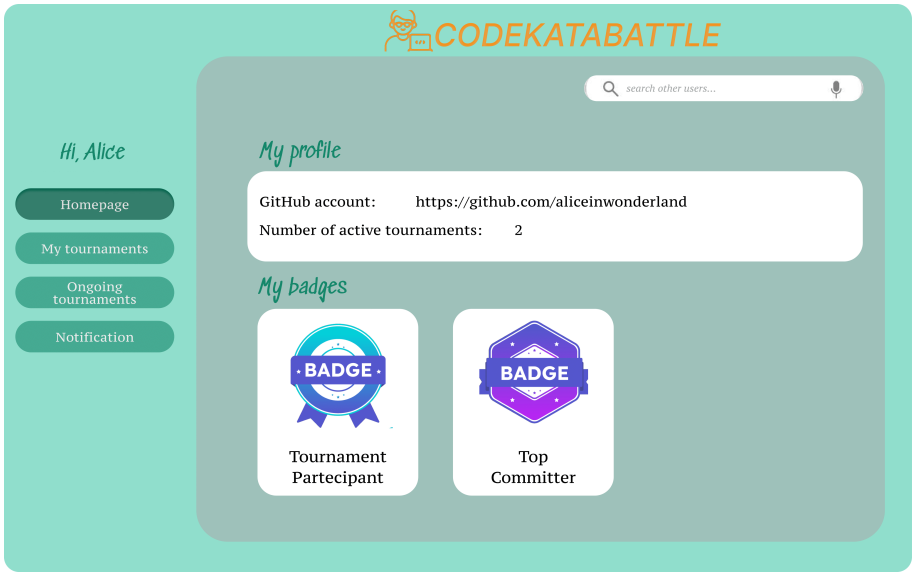

Figure 3.3: Student's homepage and profile

Figure 3.4: My tournaments - student's page

Figure 3.5: Tournament's details - student's page

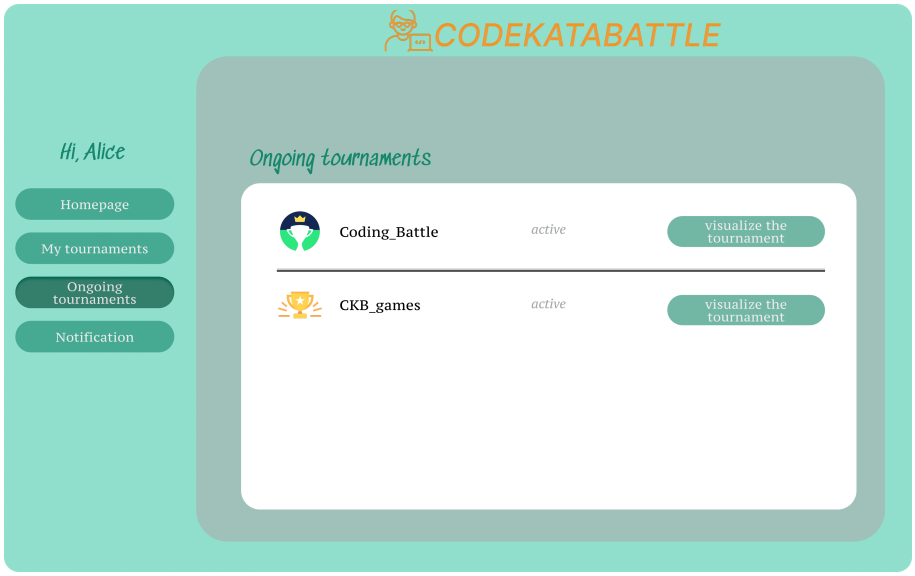

Figure 3.6: Ongoing tournaments - student's page

Figure 3.7: Join a new tournament - student's page

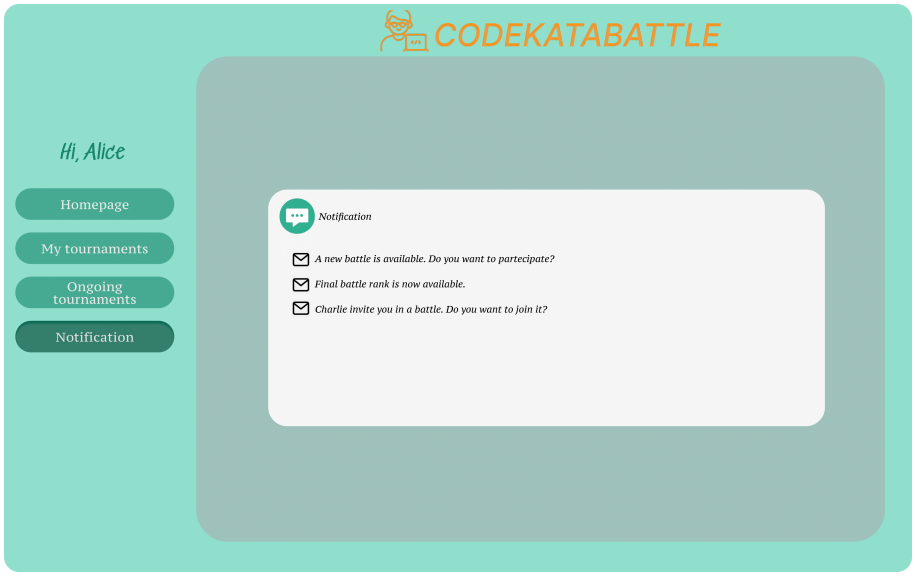

Figure 3.8: Notification page - student's page

Figure 3.9: Subscription to a new battle - student's page

Figure 3.10: Educator's homepage

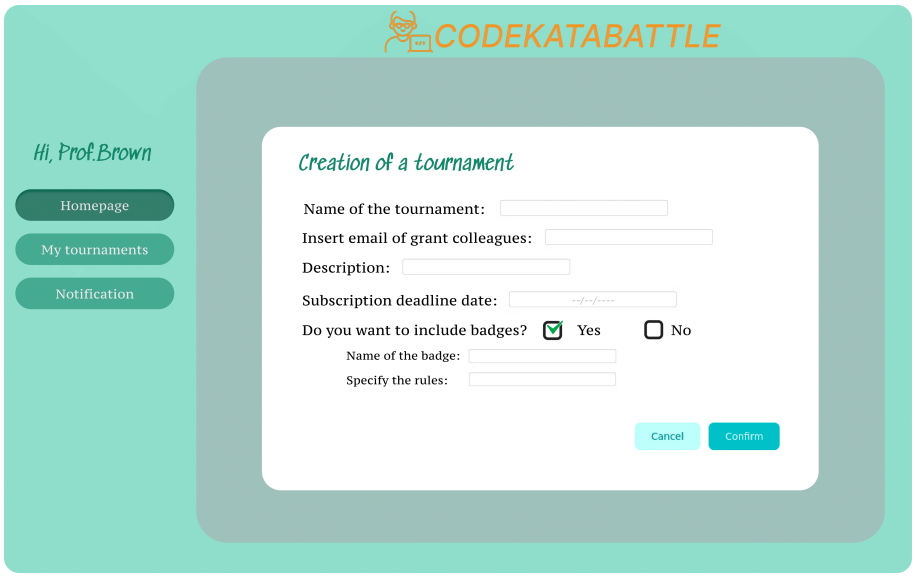

Figure 3.11: Creation of a tournament - educator's page

Figure 3.12: My tournaments page - educator's page

Figure 3.13: Tournament's details - educator's page

Figure 3.14: Creation of a battle - educator's page

Figure 3.15: Notification page - educator's page

3.1.2 Hardware Interfaces

This section describes the logical and physical characteristics of each interface between the hardware and software components of the system.
Both educators and students need to have a computer to use the CKB platform.

3.1.3 Software Interfaces

This section describes the connections between the system and other specific software components.
CodeKataBattle is a web application, so it needs a web browser to be used.

3.1.4 Communication Interfaces

This section describes the requirements associated with any communication function required by this system.
All the communications of the CKB infrastructure are made via the HTTPS application layer protocol: obviously, all the devices using the platform must be connected via WiFi or mobile network (LTE/3G/4G/5G).

3.2 Functional Requirements

3.2.1 Use Cases Diagrams

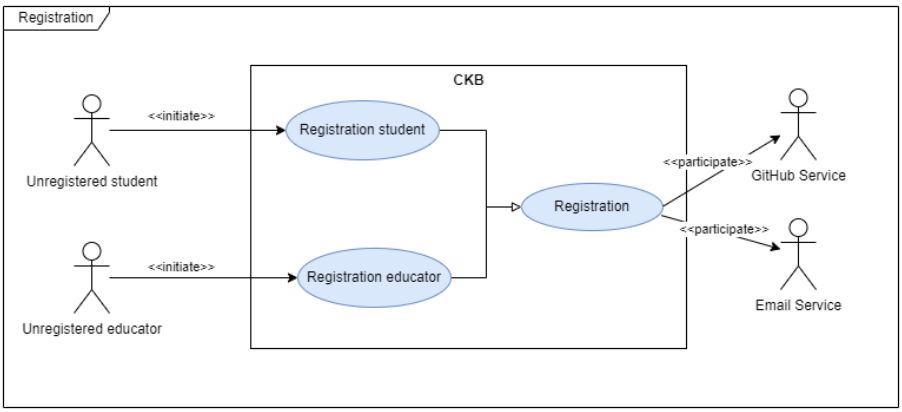

Figure 3.16: Unregistered user (student or educator) use case diagram

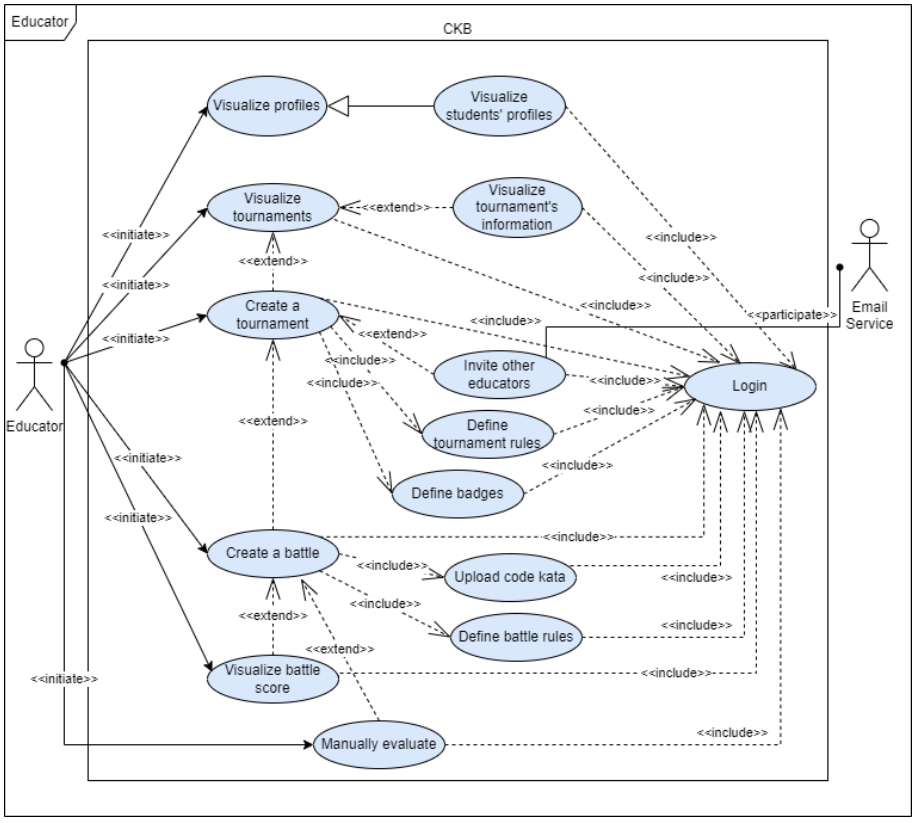

Figure 3.17: Educator use case diagram

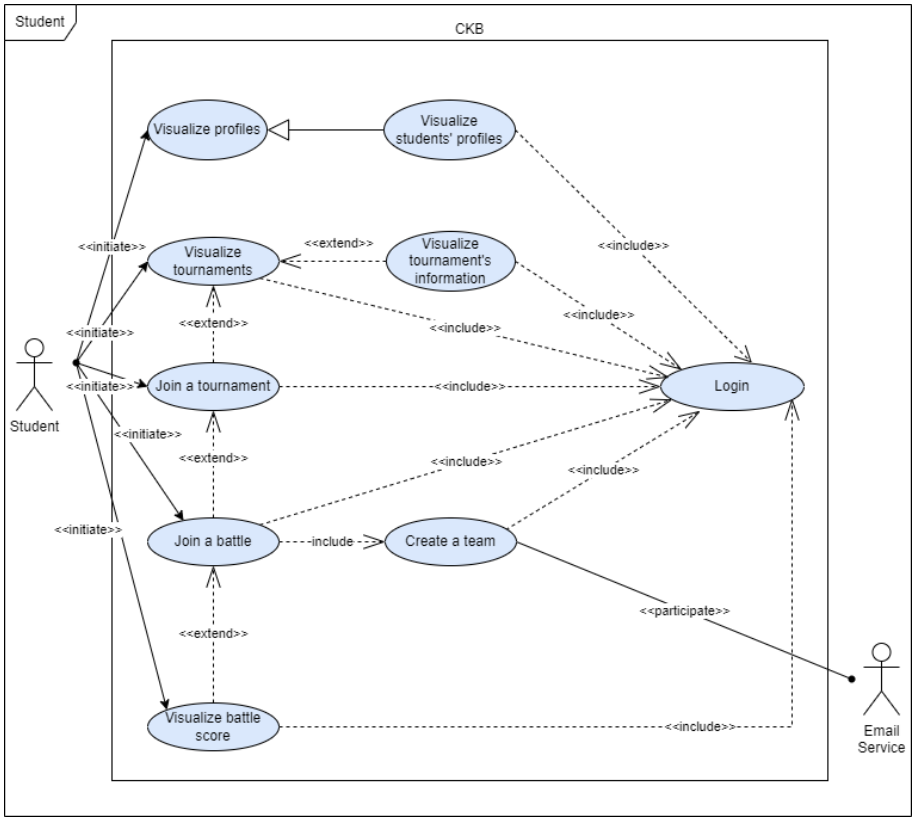

Figure 3.18: Student use case diagram

3.2.2 Use Cases Description

UC.1: Login

Actor(s)	Student, Educator
Entry Condition	The actor browses and opens the webapp.
Event Flow	1. The actor inserts the credentials (email and password) in the form. 2. The actor submits the form and sends it to the system. 3. The system processes the information and shows a success message redirecting the user to the homepage.
Exit Condition	The actor is correctly logged in and the homepage is displayed.
Exceptions	The provided email or password submitted is wrong.
Notes	In case of exception the system will notify user with a human-readable message.

UC.2: Registration

Actor(s)	Student, Educator, Email Service, GitHub Service
Entry Condition	The actor browses and opens the webapp.
Event Flow	1. The actor clicks the 'Register' button. 2. The actor fills the sign-up form with his personal information, email and password. 3. The student adds his GitHub username and submits the information. 4. The system calls GitHub Service API to check if the username is valid. 5. The GitHub Service sends the response to the system. 6. The system displays a success message about the verification of GitHub username. 7. The system calls Email Service API to send to the actor an email containing a secret code. 8. The Email Service sends the email to the actor. 9. The actor submits the received verification code. 10. The system displays a success message. 11. The system processes the provided information and creates a new account.
Exit Condition	The actor signed up correctly.
Exceptions	1. A required registration field is missing when the form is submitted. 2. A provided email is already registered in the system. 3. A wrong verification code is submitted. 4. A wrong GitHub username is submitted.
Notes	In case of exception the system will notify user with a human-readable message.

UC.3: Tournament creation

Actor(s)	Educator, Educator2, Student, Email Service
Entry Condition	The educator is already logged in the system.
Event Flow	1. The educator requires the "Create a new tournament" page. 2. The system shows the "Create a new tournament" page to the educator. 3. The educator fills the form with tournament name, description and deadline. 4. The educator can choose to define a set of badges that can be earned by the students. 5. The educator can invite other educators to join the tournament, giving them the possibility to create battles. 6. The educator clicks the "Confirm" button and submits the form. 7. The system sends an email to the invited educators. 8. The system sends an email to the students subscribed to the tournament.
Exit Condition	The tournament is created and the tournament page is shown.
Exceptions	• The educator provides an invalid name or description. • The educator provides a deadline that is not in the future. • The educator provides invalid emails of the invited colleagues.
Notes	In case of exception the system will notify user with a human-readable message.

UC.4: Battle creation

Actor(s)	Educator, Student, Email Service, GitHub Service
Entry Condition	The authorized educator is in the tournament page in which he wants to create the new battle.
Event Flow	1. The educator clicks the "Create a new battle" button. 2. The system shows the "Creation of a battle" page to the educator. 3. The educator gives a name to the battle. 4. The educator uploads the code kata. 5. The educator sets the minimum and maximum number of group members. 6. The educator sets a deadline to subscribe to the battle. 7. The educator sets a deadline to submit the solutions. 8. The educator can choose to include a manual evaluation of the students' work and specifies the battle rules. 9. The educator clicks the "Confirm" button and submits the form. 10. The system creates the battle repository on GitHub. 11. The system sends an email to all the students that are subscribed to the tournament.
Exit Condition	The battle is created.
Exceptions	• The educator provides an invalid name. • The educator does not upload a code kata. • The educator provides an invalid number of group members. • The educator provides a deadline that is not in the future.
Notes	In case of exception the system will notify user with a human-readable message.

UC.5: Tournament visualization

Actor(s)	Educator, Student
Entry Condition	The actor is already logged in the system.
Event Flow	1. The actor requires the "Ongoing tournaments" page. 2. The system shows the "Ongoing tournaments" page to the actor. 3. The actor selects a tournament.
Exit Condition	The tournament page is shown.
Exceptions	None.
Notes	

UC.6: Joining a tournament

Actor(s)	Student, Email Service
Entry Condition	The authorized student is in the tournament page.
Event Flow	1. The student clicks "Join a tournament" button. 2. The system signs up the student to the tournament. 3. The system sends an email to the student.
Exit Condition	The tournament page is shown.
Exceptions	None.
Notes	

UC.7.1: Joining a battle - Singleton

Actor(s)	Student, Email Service
Entry Condition	The authorized student is in the tournament page and the battle allows singleton groups.
Event Flow	1. The student clicks the "Join" button corresponding to the battle. 2. The system shows a form to join the battle. 3. The student fills the form as a singleton. 4. The system signs up the student to the battle. 5. The system sends an email to the student.
Exit Condition	The tournament page is shown.
Exceptions	None.
Notes	

UC.7.2: Joining a battle - Creating a group

Actor(s)	Students, Email Service
Entry Condition	The authorized student is in the tournament page.
Event Flow	1. The student clicks the "Join" button corresponding to the battle. 2. The system shows a form to join the battle. 3. The student fills the form with other students' email and clicks the "Confirm" button. 4. The system creates the team but does not sign it up to the battle yet. 5. The system sends an email to the invited students.
Exit Condition	The tournament page is shown.
Exceptions	1. The student does not provide valid emails. 2. The student does not provide a valid number of emails.
Notes	In case of exception the system will notify user with a human-readable message.

UC.7.3: Joining a battle - Joining a group

Actor(s)	Students, EmailProvider, Email Service
Entry Condition	The authorized student is logged in the system.
Event Flow	1. The student requires the "Notification" page. 2. The system shows the "Notification" page to the student. 3. The student selects an invite. 4. The student clicks the "Join" button. 5. The system updates the group. 6. If constraints are met, the system signs up the group to the battle. 7. The system sends an email to the students of the subscribed team.
Exit Condition	The tournament page is shown.
Exceptions	The battle registration deadline has expired.
Notes	In case of exception the system will notify user with a human-readable message.

UC.8: Battle execution

Actor(s)	Student, GitHub Service
Entry Condition	The student has received the email with the link to the code kata repository.
Event Flow	1. The student forks the code kata repository. 2. The student sets up the automated workflow on GitHub Actions. 3. The student pushes his code to the repository. 4. The system performs the automated evaluation.
Exit Condition	The system updates the battle score.
Exceptions	The battle submission deadline has expired.
Notes	In case of exception the system will notify user with a human-readable message.

UC.9: Conclusion of a battle

Actor(s)	Educator, Student, Email Service
Entry Condition	The submission deadline (and, if required, the manual evaluation stage) has expired.
Event Flow	If the manual evaluation is required: 1. The educator requests the code from the system. 2. The system provides the code to the educator. 3. The educator manually evaluates the code at the end of the battle and sets a score. 4. The system updates the battle score. Finally: 5. The system updates the tournament score and rank. 6. The system sends an email to the students subscribed to the battle.
Exit Condition	The battle is closed.
Exceptions	None.
Notes	

UC.10: Conclusion of a tournament

Actor(s)	Educator, Student, Email Service
Entry Condition	The authorized educator is in the tournament page.
Event Flow	1. The educator closes the tournament. 2. The system awards badges to eligible students. 3. The system sends an email to the subscribed students about the conclusion of the tournament.
Exit Condition	The tournament is closed.
Exceptions	None.
Notes	

UC.11: Profile visualization

Actor(s)	Student, Educator
Entry Condition	The actor is logged in the system.
Event Flow	1. The actor searches a student profile. 2. The system returns the selected student profile.
Exit Condition	The actor visualizes the personal information of the student.
Exceptions	The selected profile does not exist.
Notes	In case of exception the system will notify user with a human-readable message.

3.2.3 Sequence Diagrams

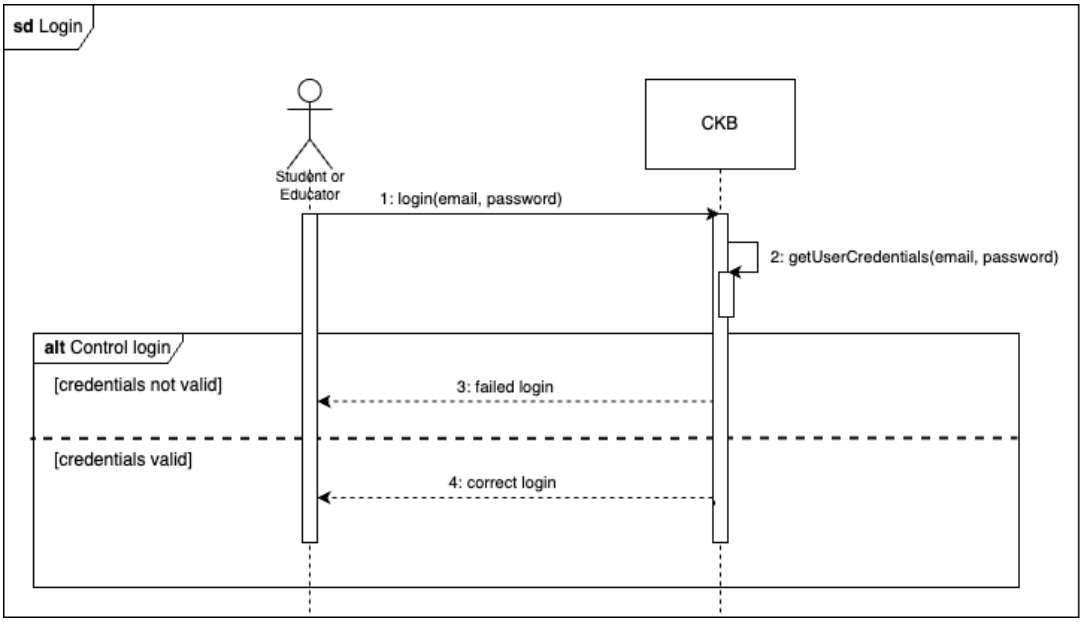

Figure 3.19: Login sequence diagram

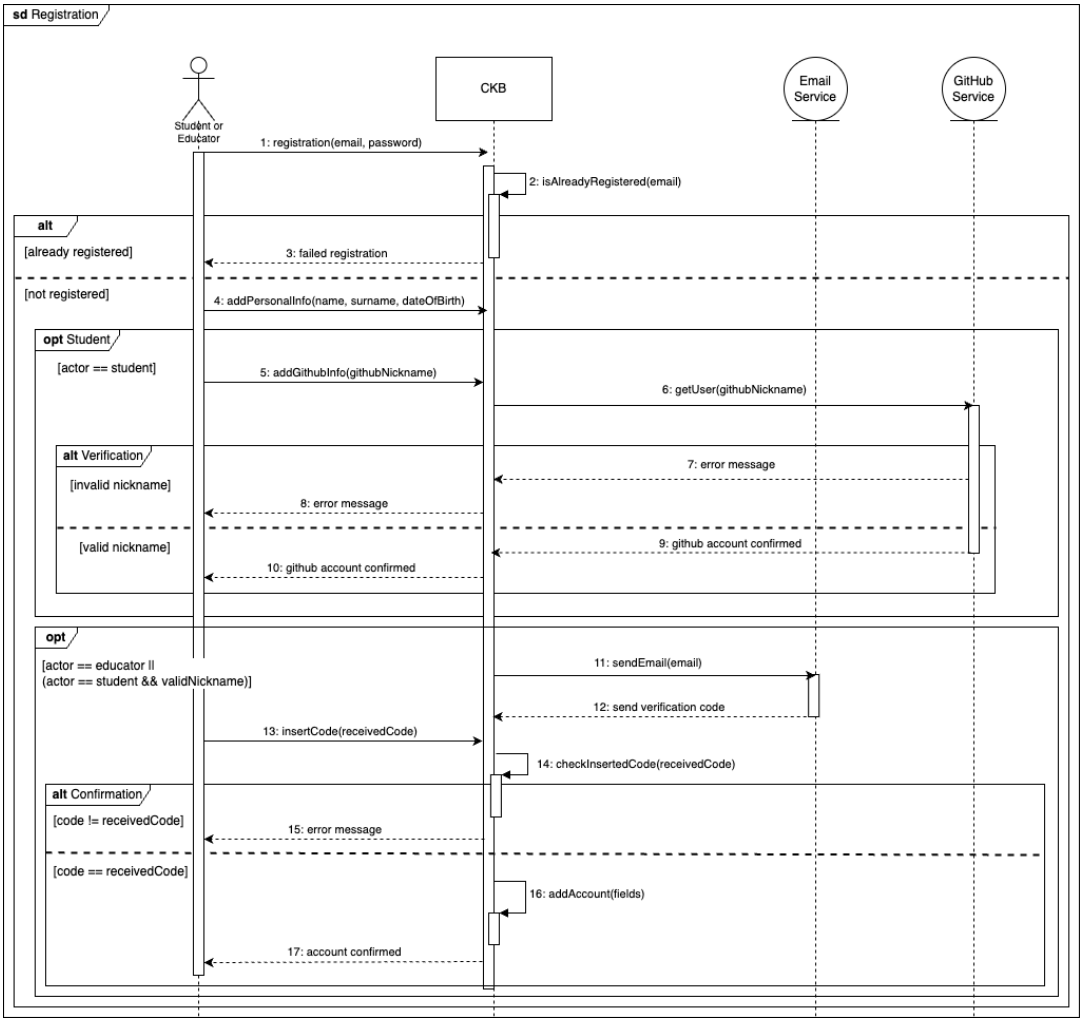

Figure 3.20: Registration sequence diagram

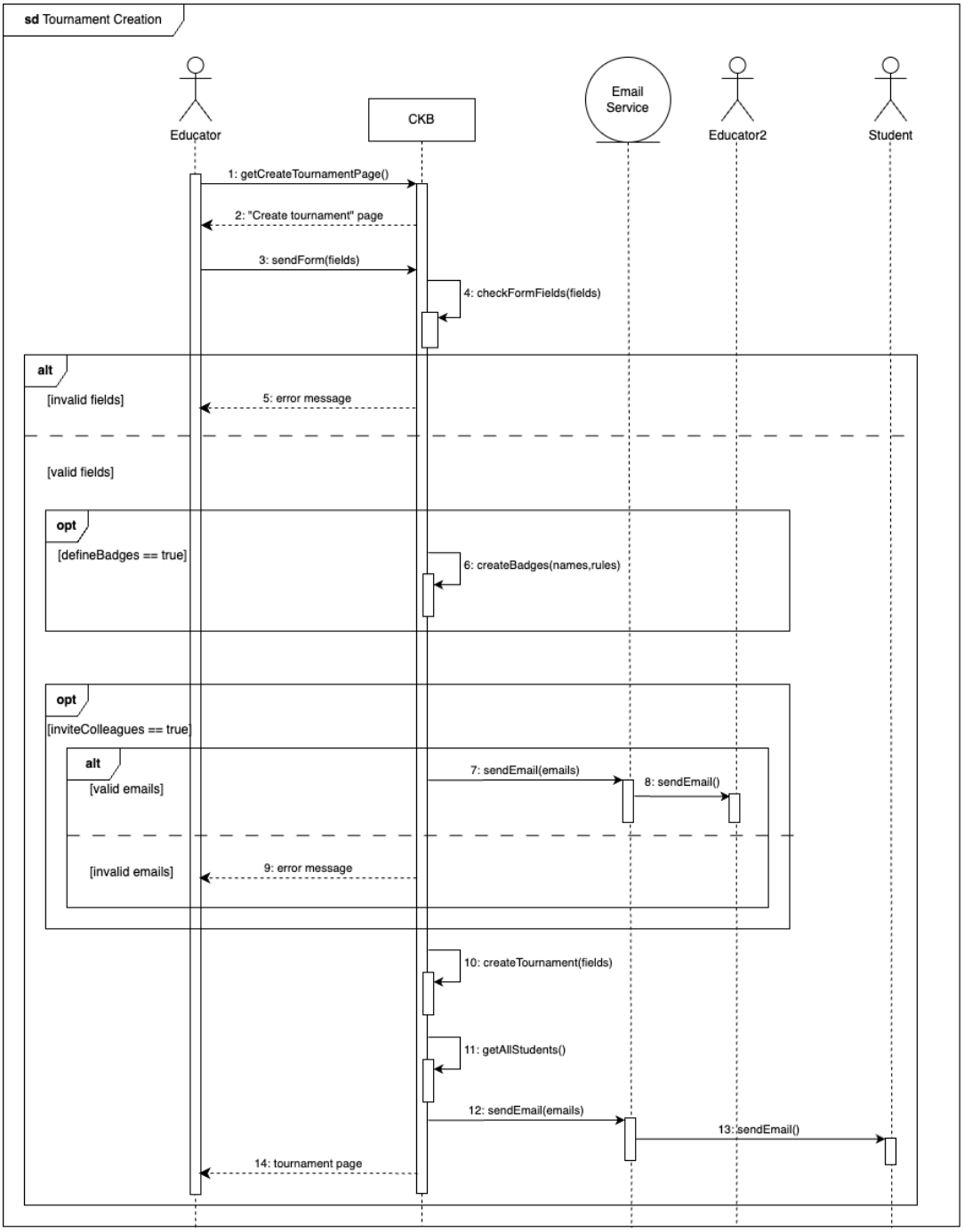

Figure 3.21: Tournament creation sequence diagram

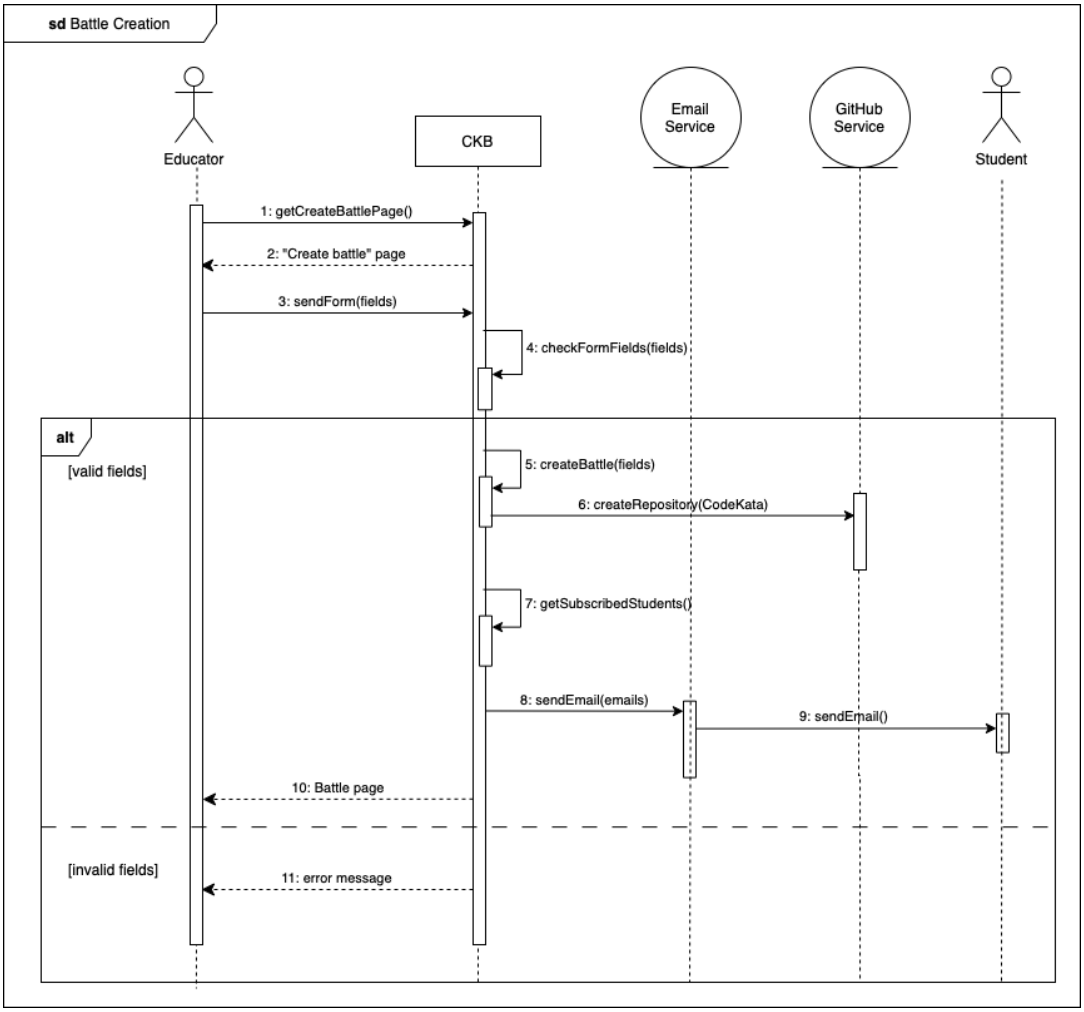

Figure 3.22: Battle creation sequence diagram

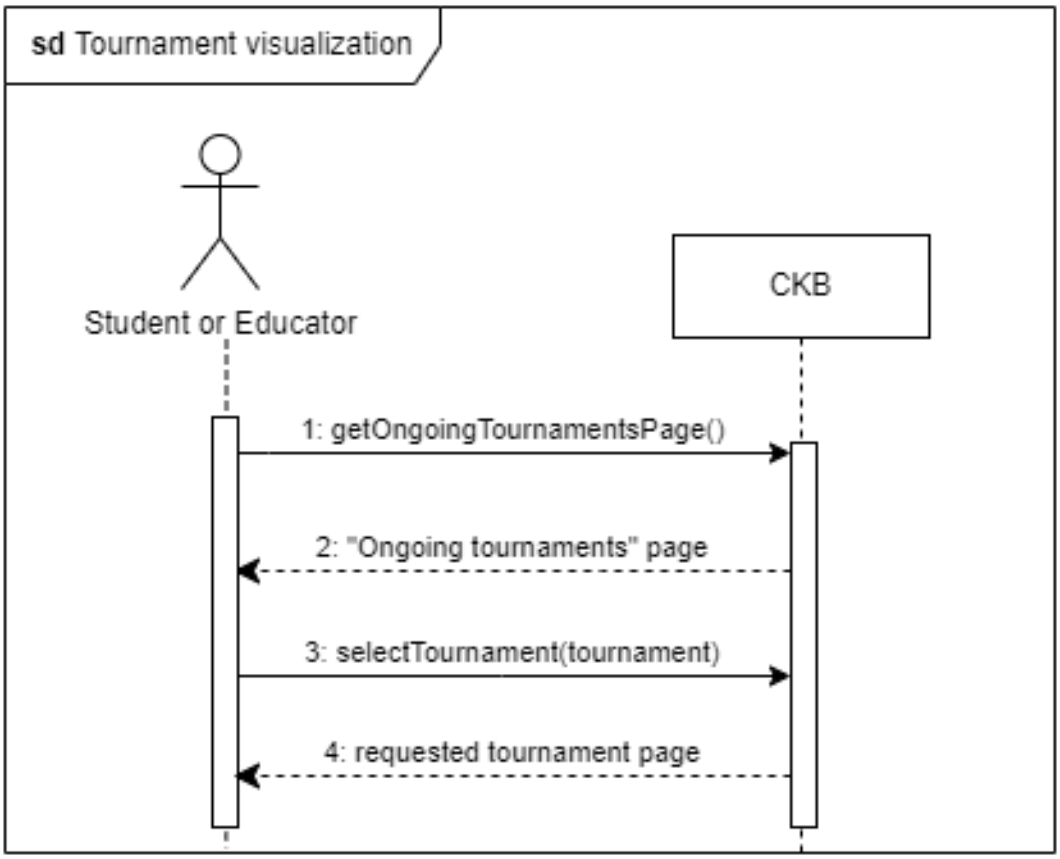

Figure 3.23: Tournament visualization sequence diagram

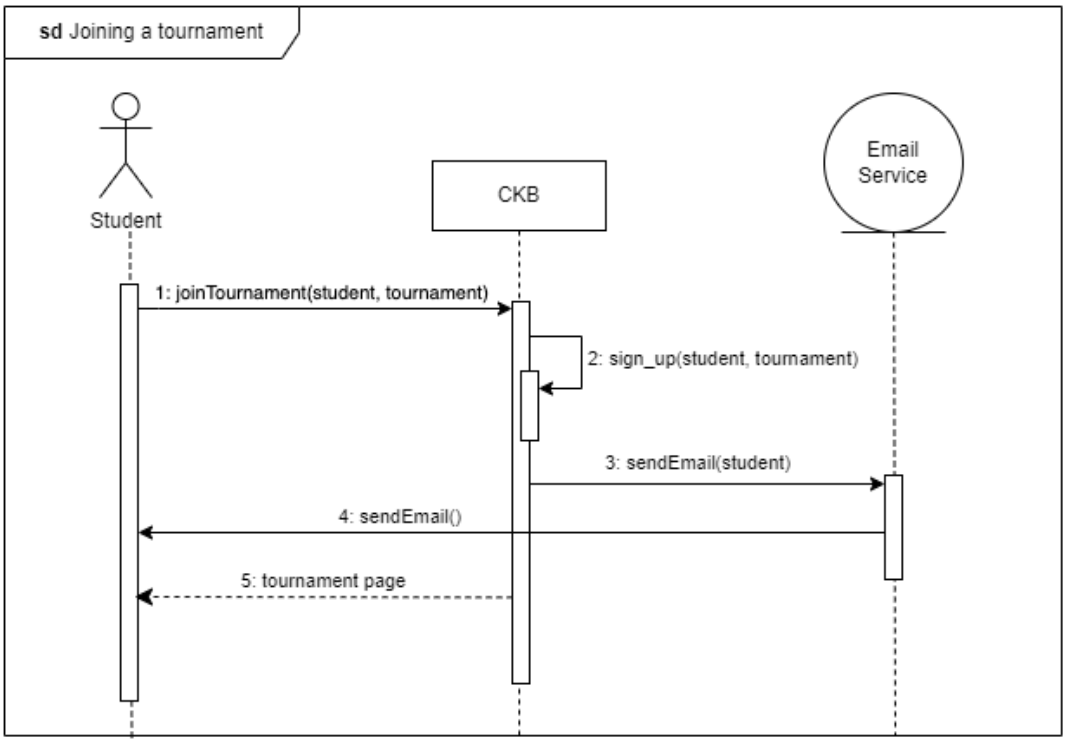

Figure 3.24: Joining a tournament sequence diagram

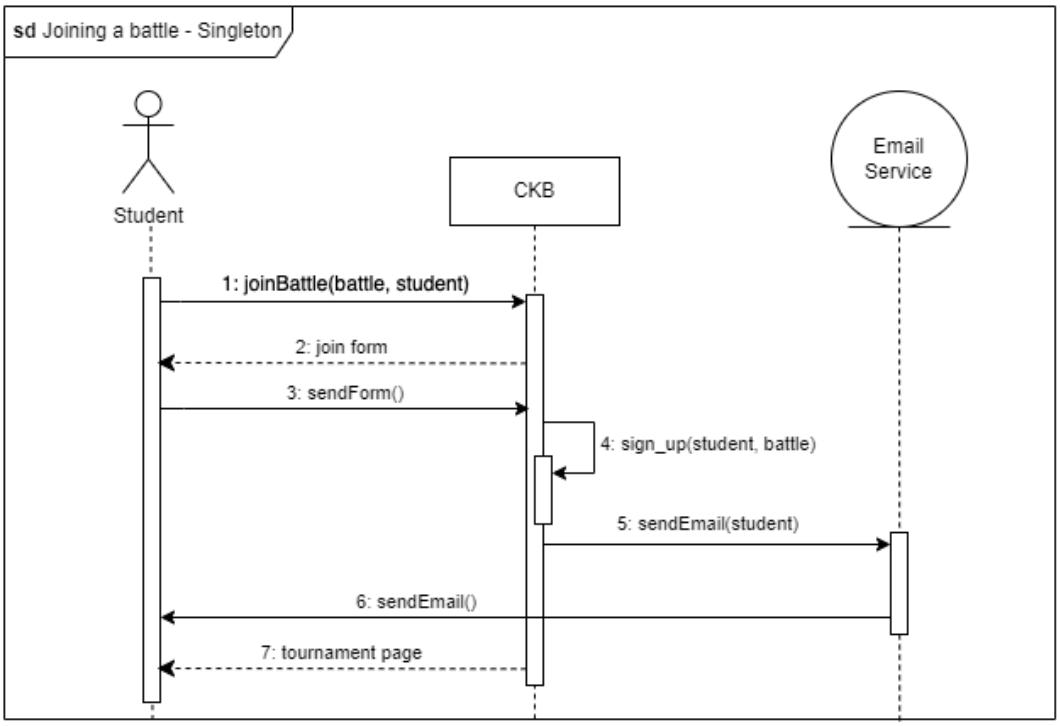

Figure 3.25: Joining a battle as a singleton sequence diagram

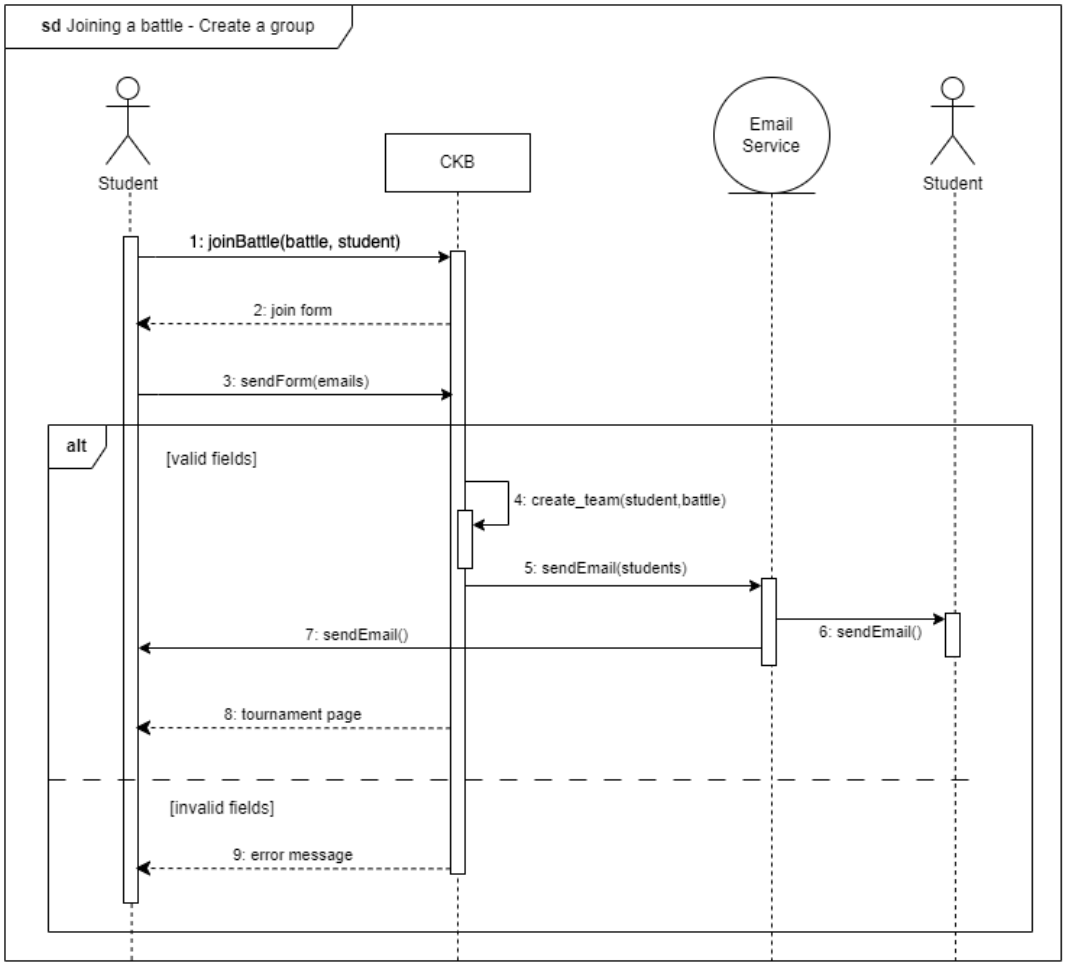

Figure 3.26: Joining a battle by creating a group sequence diagram

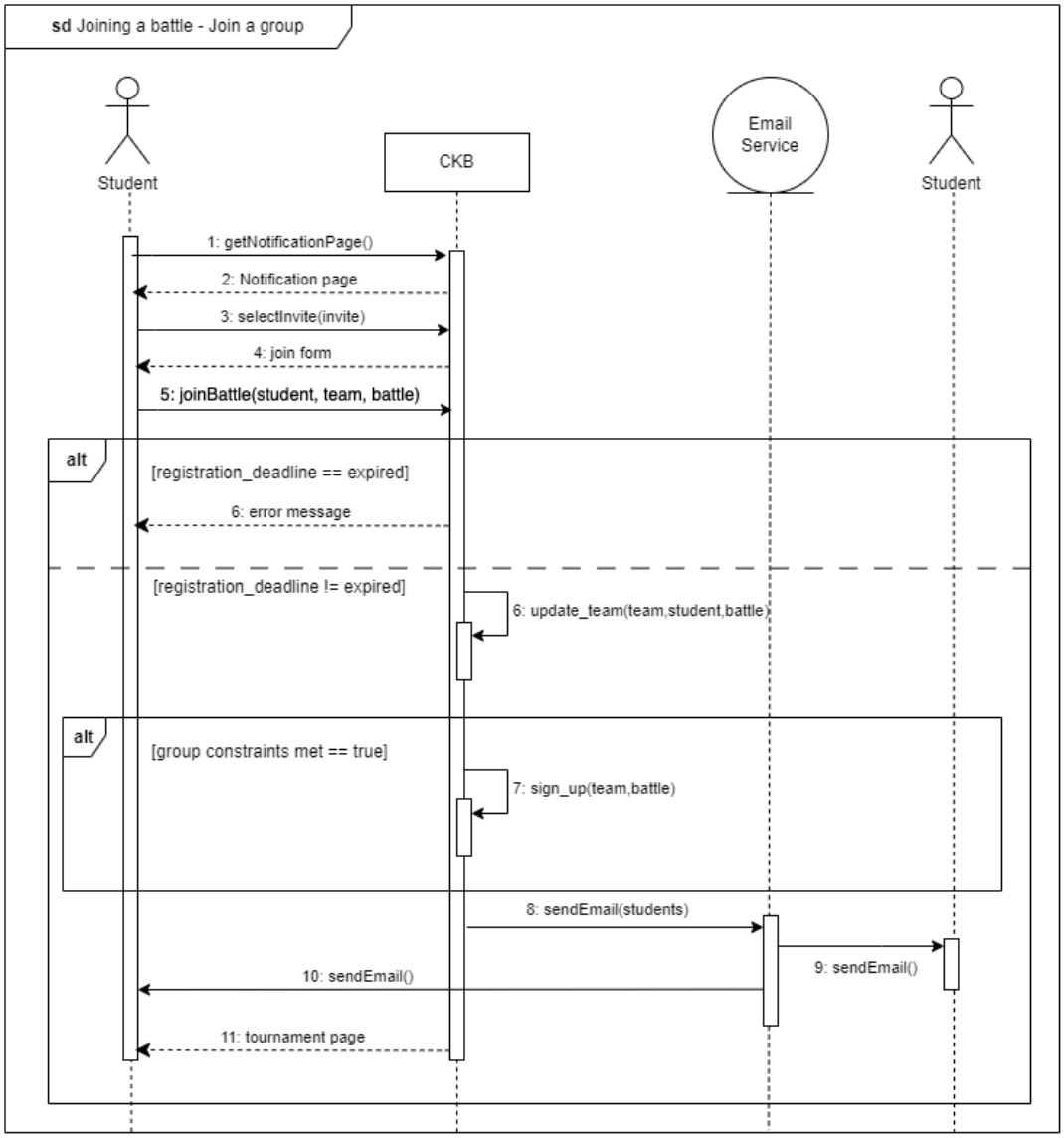

Figure 3.27: Joining a battle by joining a group sequence diagram

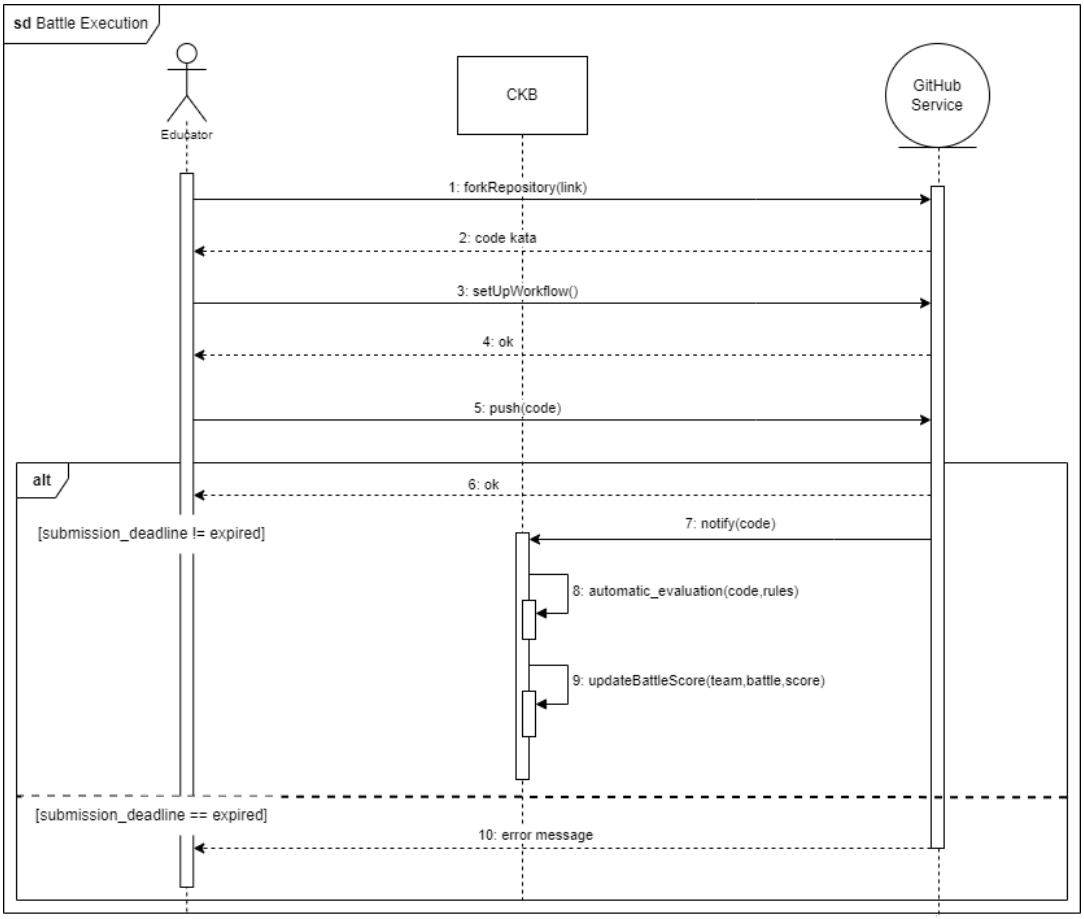

Figure 3.28: Execution of a battle sequence diagram

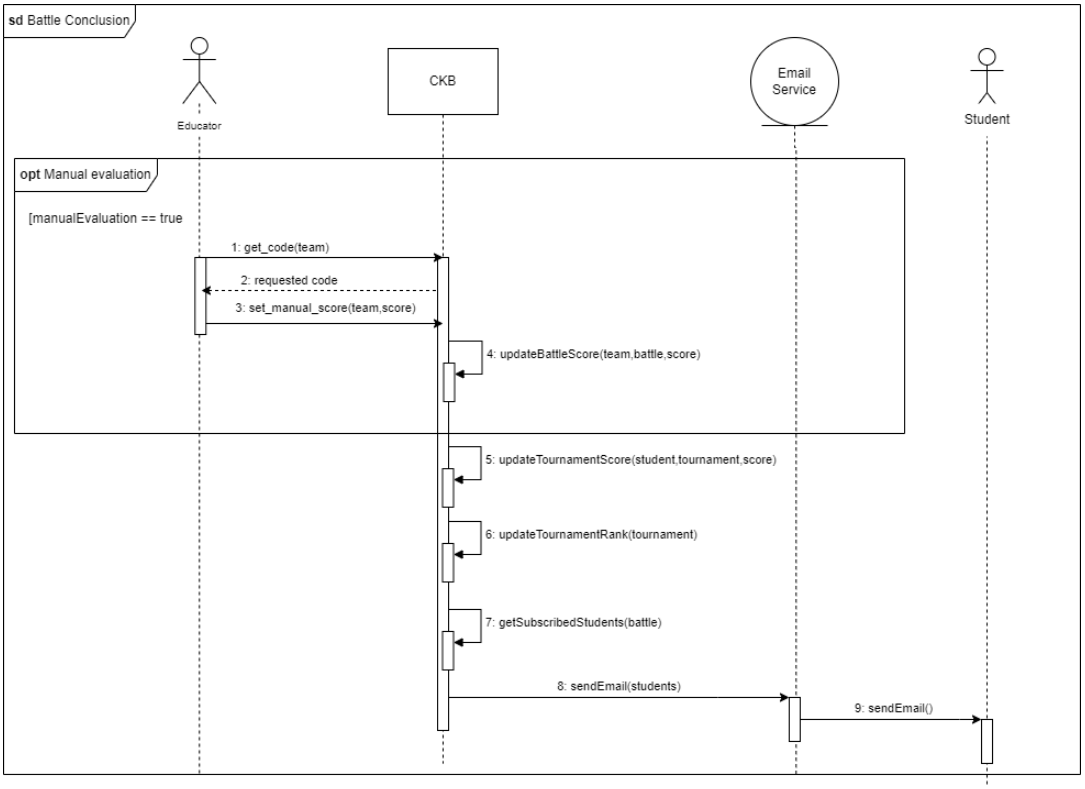

Figure 3.29: Conclusion of a battle sequence diagram

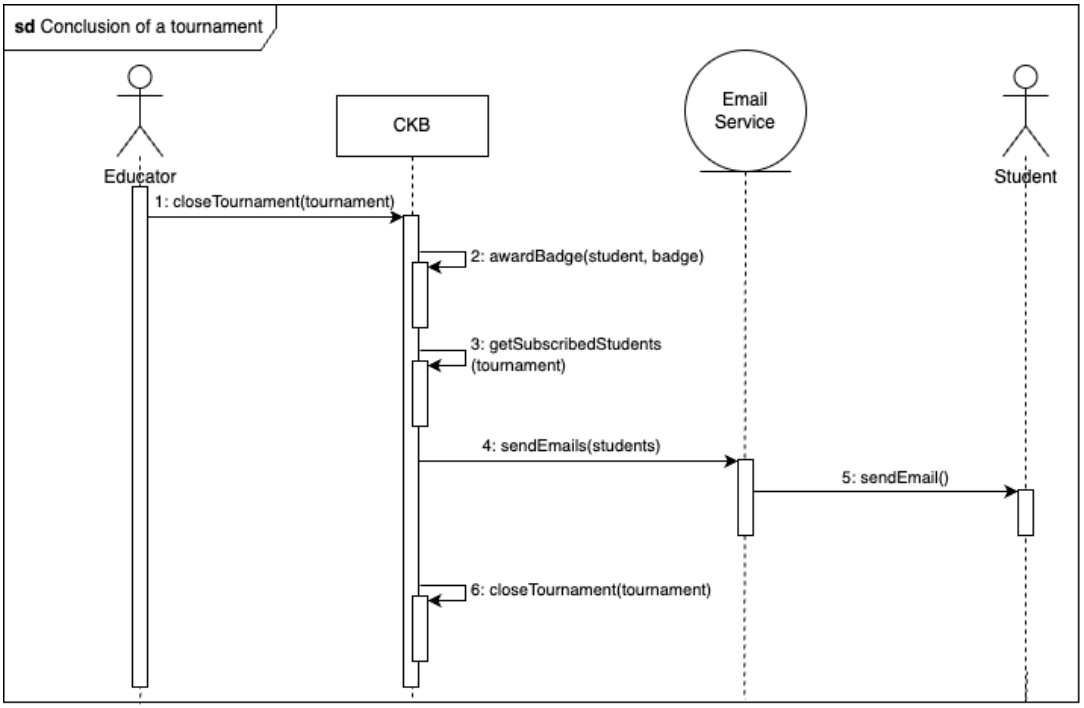

Figure 3.30: Tournament conclusion sequence diagram

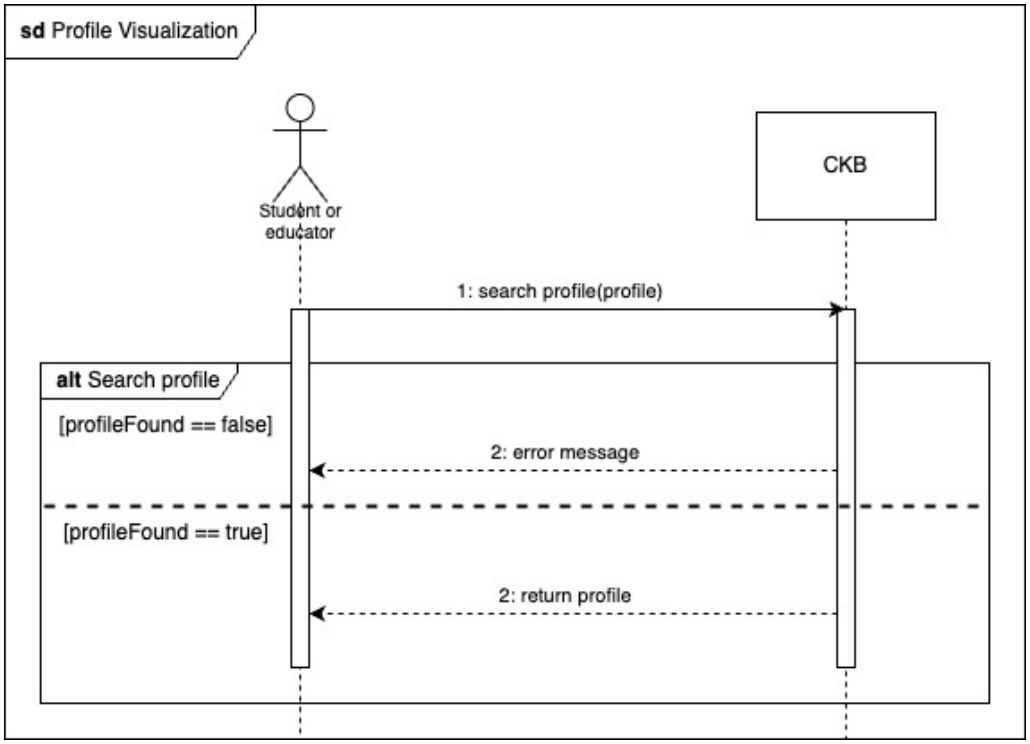

Figure 3.31: Profile visualization sequence diagram

3.2.4 Activity diagrams

Activity diagrams are used to model the workflow of a process, the operation that are needed to perform a task. They showcase the sequence and dependencies of various tasks, actions and processes within a system.

Registration
In the following figure the process of registration for both students and educators is shown. The user has to insert his personal information, his email, he has to choose password and specify if he is a student or an educator.
If he is a student, he has to link his GitHub account. If the email is incorrect, the system repeats the operation. Otherwise, the system sends a verification code and checks it.
If everything goes well, the system creates the new account.

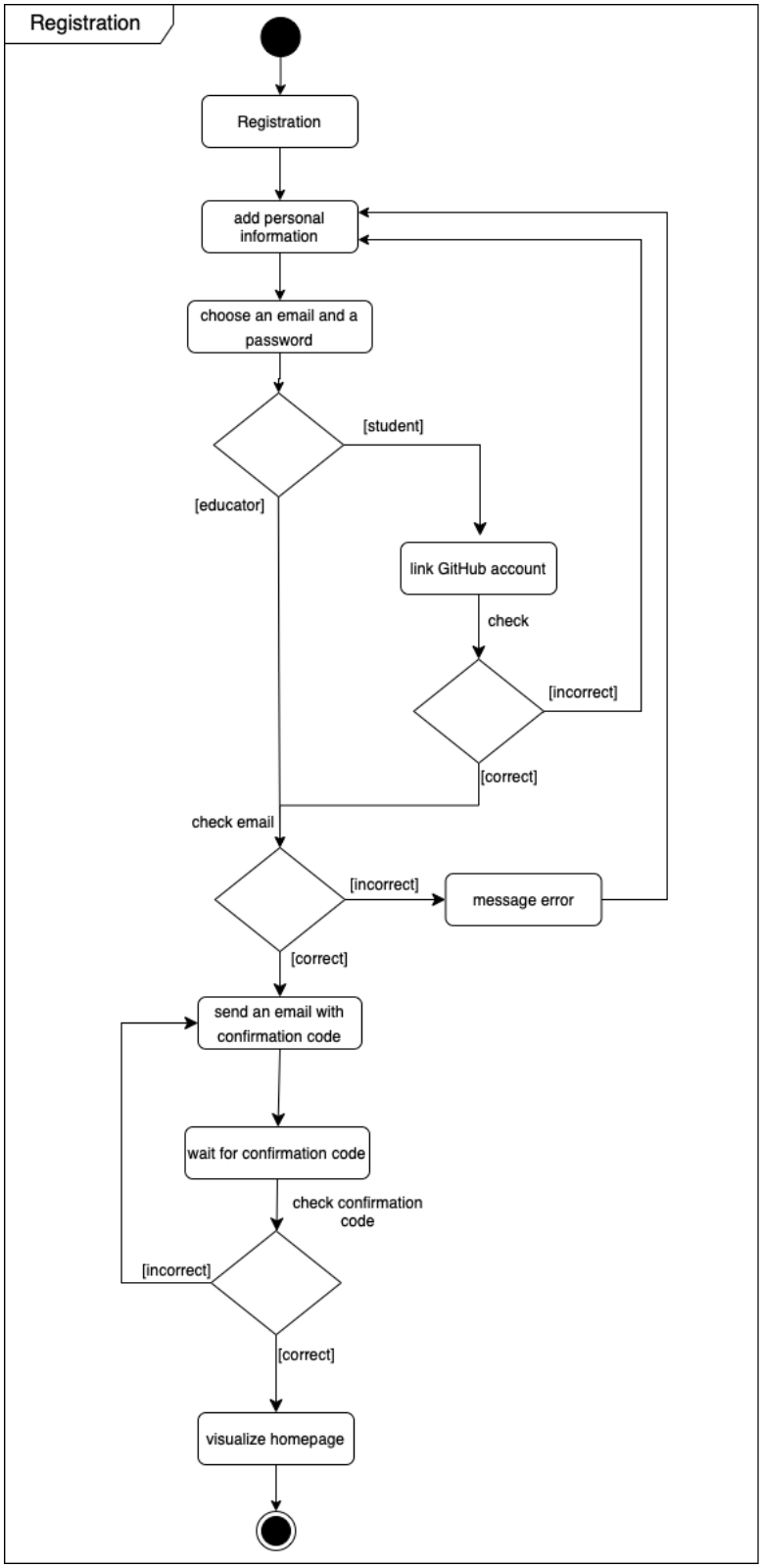

Figure 3.32: Registration state chart

Login
In the following figure the process of login for both students and educators is shown. The user has to insert his email and the password associated to his account.
If the credentials are incorrect, the system repeats the operations.
Otherwise, the user visualizes the homepage.

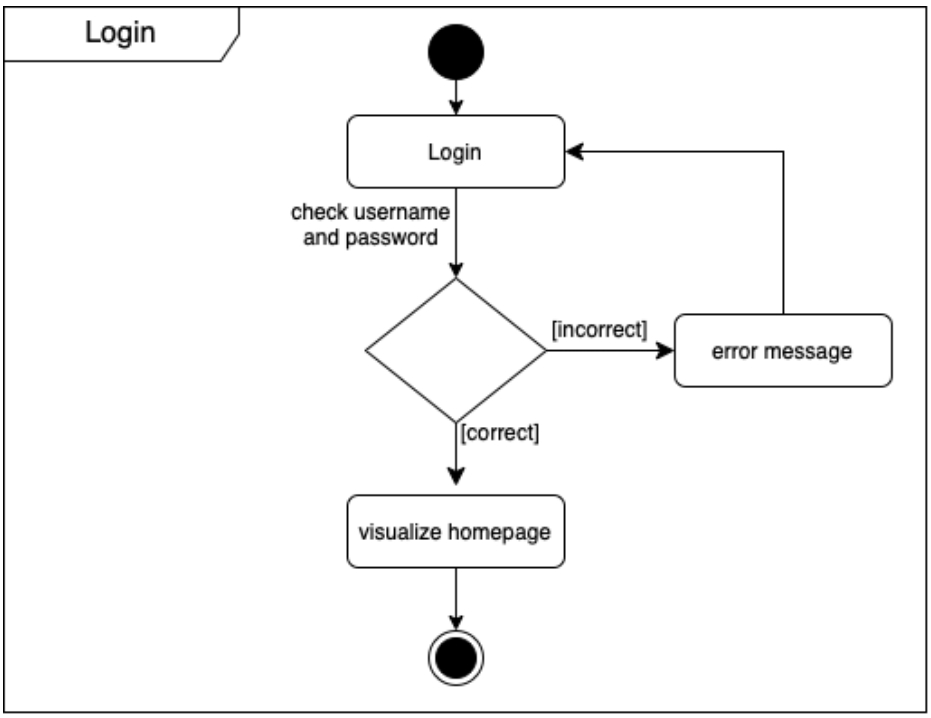

Figure 3.33: Login state chart

Tournament evolution

In the following figure the evolution of a tournament is shown. If the user is a student, he can only join the tournament, choose which battle he wants to attempt and invite friends.
If the user is an educator, he can create a tournament or only create the battles (if he has the permission). He can also evaluate the student if the settings of the tournament allow it.

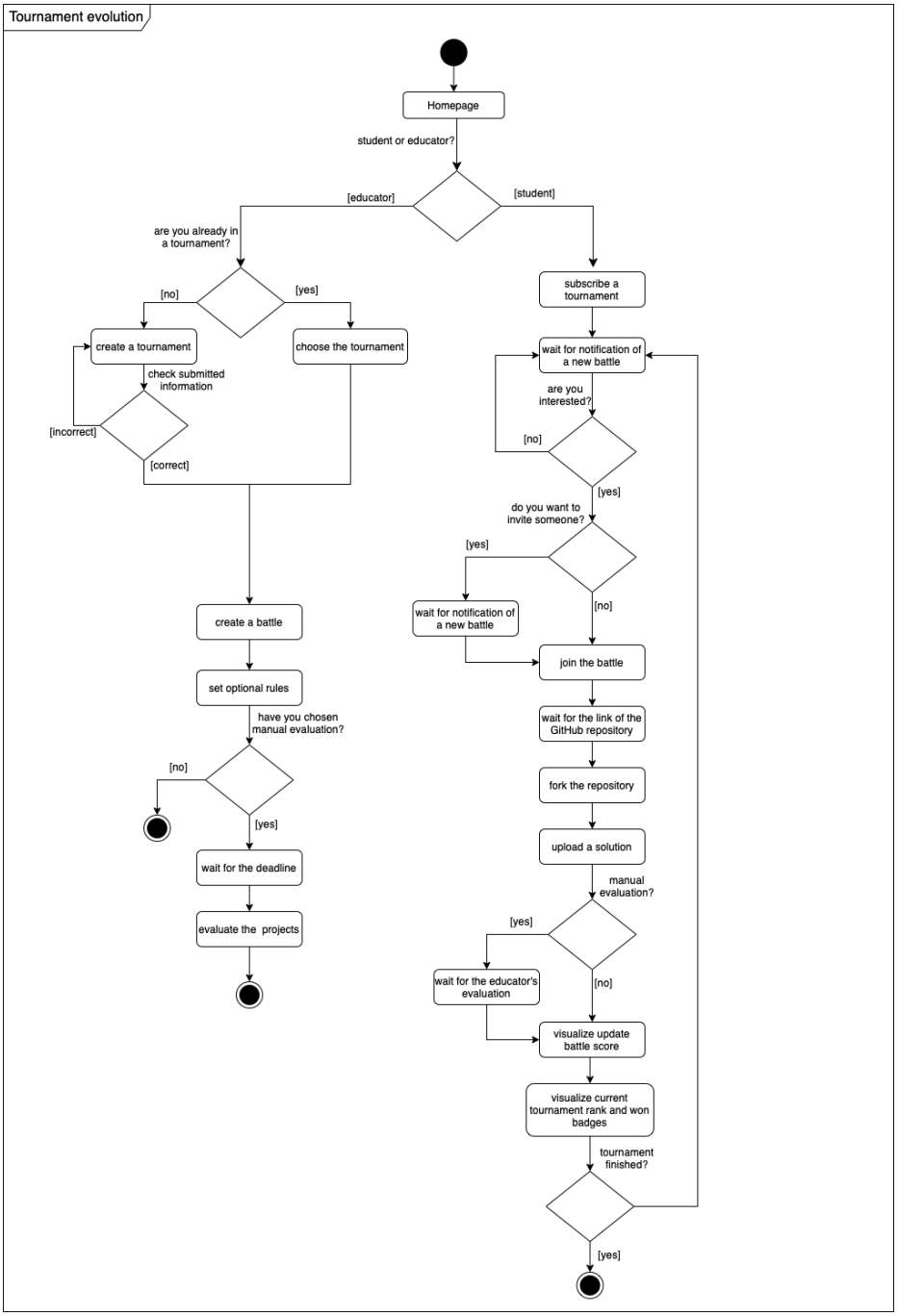

Figure 3.34: Tournament evolution state chart

3.2.5 List of functional requirements

Requirements	Description
R1	The system must allow the user to register himself to the system by providing all the mandatory information.
R2	The system must allow the students to link their GitHub account.
R3	The system must verify the uniqueness of the email of the user to be registered.
R4	The system must send a confirmation on the user's email after his registration.
R5	The system must allow the user to log into their account by entering their email and password.
R6	The system must allow educators to create tournaments.
R7	The system must allow the tournament creator to invite other educators.
R8	The systems must allow the tournament creator to end the tournament.
R9	The system must allow the tournament creator to set the tournament configuration.
R10	The system must allow the tournament creator to define badges.
R11	The system must allow students to collect badges.
R12	The system must assign badges to eligible students.
R13	The system must notify via email students when a tournament they subscribed to ends.
R14	The system must notify via email students when they are awarded a badge.
R15	The system must notify via email educators when it is time to evaluate students' project.
R16	The system must notify via email educators when they are invited to join a tournament.
R17	The system must allow authorized educators to create battles.
R18	The system must allow the battle creator to upload the code kata.
R19	The system must allow the battle creator to set the battle configuration.

R20	The system must allow the educator to manually evaluate the students' project.
R21	The system must send the link to the repository to all students subscribed to the battle.
R22	The system must create the GitHub repository containing the code kata.
R23	The system must notify via email students when a new tournament is created.
R24	The system must allow students to join a tournament.
R25	The system must notify via email students when a new battle is created within the context of a tournament they signed-up to.
R26	The system must allow students to subscribe to a battle within the context of a tournament they signed-up to.
R27	The system must allow students to invite other students to a battle.
R28	The system must notify via email students when they are invited to join a battle by another student.
R29	The system must notify vie email students when the final battle rank becomes available.
R30	The system must notify via email students when the final tournament rank becomes available.
R31	The system must pull the latest submitted sources before the submission deadline expired.
R32	The system must analyze the students' projects.
R33	The system must run the students' projects.
R34	The system must evaluate the students' projects.
R35	The system must update the battle scores of the teams.
R36	The system must compute the battle rank.
R37	The system must update the students' personal tournament scores.
R38	The system must compute the tournament rank.
R39	The system must allow users to visualize ongoing tournaments.
R40	The system must allow users to visualize ongoing tournament ranks.
R41	The system must allow users to visualize students' profiles.

R42	The system must notify via email students when they join a tournament.

3.2.6 Mapping on requirements

G1 Allow students to compete in a tournament.

Requirements

R1 The system must allow the user to register himself to the system.

R2 The system must allow the user to register himself to the system.

R3 The system must verify the uniqueness of the email of the user to be registered.

R4 The system must send a confirmation on the user's email after his registration.

R5 The system must allow the user to log into their account by entering their email and password.

R11 The system must allow students to collect badges.

R12 The system must assign badges to eligible students.

R13 The system must notify via email students when a tournament they subscribed to ends.

R17 The system must allow authorized educators to create battles.

R18 The system must allow the battle creator to upload the code kata.

R19 The system must allow the battle creator to set the battle configuration.

R20 The system must allow the educator to manually evaluate the students' project.

R21 The system must send the link to the repository to all students subscribed to the battle.

R22 The system must create the GitHub repository containing the code kata.

R23 The system must notify via email students when a new tournament is created.

R24 The system must allow students to join a tournament.

R25 The system must notify via email students when a new battle is created within the context of a tournament they signed-up to.

R26 The system must allow students to subscribe to a battle within the context of a tournament they signed-up to.

R27 The system must allow students to invite other students to a battle.

R28 The system must notify via email students when they are invited to join

a battle by another student.

R29 The system must notify via email students when the final battle rank becomes available.

R30 The system must notify via email students when the final tournament rank becomes available.

R31 The system must pull the latest submitted sources before the submission deadline expired.

R35 The system must update the battle scores of the teams.

R36 The system must compute the battle rank.

R37 The system must update the students' personal tournament scores.

R38 The system must compute the tournament rank.

R39 The system must allow users to visualize ongoing tournaments.

R40 The system must allow users to visualize ongoing tournament ranks.

R41 The system must allow users to visualize students' profiles.

R42 The system must notify via email students when they join a tournament.

Domain Assumptions

D1 Users provide correct information during the registration process.

D2 Students have an email and a GitHub account.

D3 Users have a stable and reliable Internet connection.

D4 Students know the functioning of GitHub and GitHub Actions.

D5 Educators create at least one battle within a tournament.

D6 Students must give consent to access their GitHub account.

D8 Educators upload a correct code kata.

G2 Allows educators to create challenges for students.

Requirements

R1 The system must allow the user to register himself to the system.

R3 The system must verify the uniqueness of the mail of the user to be registered.

R4 The system must send a confirmation on the user's email after his registration.

R5 The system must allow the user to log into their account by entering their email and password.

R6 The system must allow educators to create tournaments.

R7 The system must allow the tournament creator to invite other educators.

R8 The systems must allow the tournament creator to end the tournament.

R9 The system must allow the tournament creator to set the tournament configuration.

R10 The system must allow the tournament creator to define badges.

R11 The system must allow students to collect badges.

R12 The system must assign badges to eligible students.

R14 The system must notify via email students when they are awarded a badge.

R15 The system must notify via email educators when it is time to evaluate students' project.

R16 The system must notify via email educators when they are invited to join a tournament.

R17 The system must allow authorized educators to create battles.

R18 The system must allow the battle creator to upload the code kata.

R19 The system must allow the battle creator to set the battle configuration.

R23 The system must notify via email students when a new tournament is created.

R41 The system must allow users to visualize students' profiles.

Domain Assumptions

D1 Users provide correct information during the registration process.

D3 Users have a stable and reliable Internet connection.

D5 Educators create at least one battle within a tournament.

D7 Educators have a good knowledge of programming.

D8 Educators upload a correct code kata.

G3 Allows educators to grade students' projects.

Requirements

R1 The system must allow the user to register himself to the system.

R3 The system must verify the uniqueness of the email of the user to be registered.

R4 The system must send a confirmation on the user's email after his registration.

R5 The system must allow the user to log into their account by entering their email and password.

R6 The system must allow educators to create tournaments.

R7 The system must allow the tournament creator to invite other educators.

R20 The system must allow the educator to manually evaluate the students' project.

R31 The system must pull the latest submitted sources before the submission deadline expired.

R32 The system must analyze the students' projects.

R33 The system must run the students' projects.

R34 The system must evaluate the students' projects.

R38 The system must compute the tournament rank.

Domain Assumptions

D1 Users provide correct information during the registration process.

D2 Students have an email and a GitHub account.

D3 Users have a stable and reliable Internet connection.

D5 Educators create at least one battle within a tournament.

D7 Educators have a good knowledge of programming.

G4 Allows students to collect badges in a tournament proving their skills.

Requirements

R1 The system must allow the user to register himself to the system.

R2 The system must allow the students to link their GitHub account.

R3 The system must verify the uniqueness of the email of the user to be registered.

R4 The system must send a confirmation on the user's email after his registration.

R5 The system must allow the user to log into their account by entering their email and password.

R10 The system must allow the tournament creator to define badges.

R11 The system must allow students to collect badges.

R12 The system must assign badges to eligible students.

R14 The system must notify via email students when they are awarded a badge.

R31 The system must pull the latest submitted sources before the submission deadline expired.

R32 The system must analyze the students' projects.

R38 The system must compute the tournament rank.

Domain Assumptions

D1 Users provide correct information during the registration process.

D3 Users have a stable and reliable Internet connection.

3.2.7 Mapping on use cases

Use Case	Requirements
UC1 - Login	R5
UC2 - Registration	R1 R2 R3 R4
UC3 - Tournament creation	R6 R7 R9 R10 R16 R23
UC4 - Battle creation	R17 R18 R19 R21 R25
UC5 - Tournament visualization	R39 R40
UC6 - Joining a tournament	R24 R42
UC7 - Joining a battle	R26 R27 R28
UC8 - Battle execution	R22 R31 R32 R33 R34 R35
UC9 - Conclusion of a battle	R15 R20 R29 R31 R35 R36 R37 R38
UC10 - Conclusion of a tournament	R8 R11 R12 R13 R14 R30
UC11 - Profile visualization	R41

3.3 Performance Requirements

This section presents the performance requirements which need to be fulfilled to make the whole system as usable as possible.
To provide the best user experience, the system must be able to provide a scalable, reactive and capable of load-balancing backend, assure protection against DDoS attacks and provide a responsive and fluid frontend.
The frontend must handle correctly asynchronous interactions with the server, even when the connection quality is bad, in order to provide the best possible experience.

3.4 Design Constraints

3.4.1 Standards compliance

Specifications described in this document must be respected by the system. Source code of the application must be commented on and documented adequately. The system should respect the guidelines described by the GDPR.

3.4.2 Hardware limitations

The system requires a computer with a web browser and a stable internet connection.

3.5 Software System Attributes

3.5.1 Reliability

In order to guarantee better reliability performances, all the scheduled maintenance of the system should be done during the night. Some functionalities of the system rely on external APIs, though the system should not completely fail because of failures in one of those. It is also critical to avoid data loss through redundant storage methods.

3.5.2 Availability

The system should offer its functionalities with an availability equal to 99.7%. This means that, on average, the system must be inaccessible for at most one day every year. To achieve this goal, the system should provide a high redundancy for the most critical components.

3.5.3 Security

To protect users sensible data the system must collect the minimum data possible, and store them in an encrypted database. The connection between the application must follows modern standard secure protocols as HTTPS. Not only passwords should be considered as sensible data, but also users' email addresses and GitHub accounts. All passwords must be salted and hashed.

3.5.4 Maintainability

The system should be designed to be easily maintainable. This means that the code should be well documented and commented, and the architecture should be modular and well designed.

3.5.5 Portability

The application must run on any OS (Windows, Linux, MacOS) that supports a web browser.

Chapter 4

Formal Analysis

In this chapter the Alloy model of the CKB system is implemented, describing the main constraints and focusing in particular on the consistency of the world.

```
-- Signatures

sig Nickname {}
sig EmailAddr {}
sig Password {}
sig Name {}
sig Surname {}
sig GitHubAccount {}
sig Date {
        value: Int   -- value represents the distance in days from 1-1-1900
} {value ≥ 0}
sig CodeKata {}
sig BadgeName{}
sig BadgeDescription{}

sig Badge {
        name: BadgeName,
        description: BadgeDescription
}

abstract sig User {
        nickname: Nickname,
        emailAddr: EmailAddr,
        password: Password,
        name: Name,
        surname: Surname,
        dateOfBirth: Date
}

sig Educator extends User {}

sig Student extends User {
        githubAccount: GitHubAccount,
        badges: set Badge
}

sig SubscribedStudent {
        student: Student,
        var score: Int
} {score ≥ 0 and score ≤ 100}

abstract sig TournamentStatus{}
one sig TAvailable, TActive, TEnded extends TournamentStatus{}
abstract sig BattleStatus{}
one sig BNotCreated, BAvailable, BActive, BEnded extends BattleStatus{}

sig Tournament {
```

```
        name: Name,
        creator: Educator,
        educators: some Educator,
        badges: set Badge,
        subscribedStudents: set SubscribedStudent,
        registrationDeadline: Date,
        var battles: set Battle,
        var status: TournamentStatus
}

sig Rules {
        minimumNumberOfStudents: Int,
        maximumNumberOfStudents: Int
} {minimumNumberOfStudents > 0}

sig Team {
        students: some Student,
        var score: Int
} {score ≥ 0 and score ≤ 100}

sig Battle {
        creator: Educator,
        codeKata: CodeKata,
        registrationDeadline: Date,
        submissionDeadline: Date,
        subscribedStudents: set Student,
        rules: Rules,
        teams: set Team,
        var status: BattleStatus
}

-- Facts

-- Date
fact dateIsUnique {
        no disj d1 , d2 : Date | d1.value = d2.value
}

-- User
fact nicknamesAreUnique {
        no disj u1 , u2 : User | u1.nickname = u2.nickname
}

fact emailsAreUnique {
        no disj u1 , u2 : User | u1.emailAddr = u2.emailAddr
}

fact usernameExistsOnlyWithUser {
        all nn : Nickname | one u : User | nn in u.nickname
}

fact emailAddrExistsOnlyWithUser {
        all email : EmailAddr | one u : User | u.emailAddr in email
}

fact passwordExistsOnlyWithUser {
        all pw : Password | one u : User | u.password in pw
}

fact nameExistsOnlyWithUser {
        all nm : Name | one u : User | u.name in nm
}

fact surnameExistsOnlyWithUser {
        all sm : Surname | one u : User | u.surname in sm
}
```

```
fact dateOfBirthExistsOnlyWithUser {
        all dob : Date | one u : User | u.dateOfBirth in dob
}

-- Student

fact githubAccountsAreUnique {
        no disj u1 , u2 : Student | u1.githubAccount = u2.githubAccount
}

fact githubAccountExistsOnlyWithUser {
        all gha: GitHubAccount | one s : Student | s.githubAccount in gha
}

-- Code Kata

fact codeKataExistsOnlyWithBattle {
        all ck: CodeKata | one b : Battle | b.codeKata in ck
}

-- Tournament

fact tournamentNameIsUnique {
        no disj t1, t2: Tournament | t1.name = t2.name
}

fact tournamentBadgeIsUnique {
        all t: Tournament | no disj b1, b2: Badge | b1 in t.badges and b2 in t.
            ↪ badges and b1.name = b2.name and b1.description = b2.description
}

fact tournamentExistsOnlyWithCreator {
        all t: Tournament | one e : Educator | e in t.creator
}

fact badgeExistsOnlyWithTournament {
        all b: Badge | one t: Tournament | b in t.badges
}

fact studentBadgeIsAmongTournamentBadges {
        all s: Student, b: s.badges | one t: Tournament | b in t.badges and s in t.
            ↪ subscribedStudents.student
        and t.status = TEnded
}

fact creatorHasAccessToTournament {
        all t: Tournament | t.creator in t.educators
}

fact registrationDeadlineIsAfterCreatorDateOfBirth {
        all t: Tournament | t.registrationDeadline.value > t.creator.dateOfBirth.
            ↪ value
}

fact registrationDeadlineIsAfterEducatorsDatesOfBirth {
        all t: Tournament, e: t.educators | t.registrationDeadline.value > e.
            ↪ dateOfBirth.value
}

fact registrationDeadlineIsAfterStudentsDatesOfBirth {
        all t: Tournament, s: t.subscribedStudents.student | t.registrationDeadline
            ↪ .value > s.dateOfBirth.value
}

fact battlesCannotBeRemoved {
        all t: Tournament | always (t.battles in t.battles')
}
```

```
fact battlesCannotBeAddedIfTournamentEnded {
        all t: Tournament | always (t.status = TEnded implies t.battles' = t.
            ↪ battles)
}

fact endedTournamentCannotGoBackToAPreviousStatus {
        all t: Tournament | always (t.status = TEnded implies t.status' = TEnded)
}

fact activeTournamentCannotGoBackToAPreviousStatus {
        all t: Tournament | always (t.status = TActive implies (t.status' = TActive
            ↪  or t.status' = TEnded))
}


-- Rules
fact minimumNumberOfStudentsIsLessThanMaximumNumberOfStudents {
        all r: Rules | r.minimumNumberOfStudents ≤ r.maximumNumberOfStudents
}

fact rulesExistsOnlyWithBattle {
        all r: Rules | one b: Battle | b.rules in r
}

-- Battle
fact creatorHasPermissionToCreateBattle {
        all t: Tournament, b: t.battles | b.creator in t.educators
}

fact battleExistsOnlyWithTournament {
        all b: Battle | one t: Tournament | eventually (b in t.battles)
}

fact battleIsNotCreatedIfItDoesNotBelongToATournamentYet{
        all b: Battle | b.status = BNotCreated implies (all t: Tournament | b not
            ↪ in t.battles)
}

fact registrationDeadlineIsAfterTournamentRegistrationDeadline {
        all t: Tournament, b: t.battles | b.registrationDeadline.value > t.
            ↪ registrationDeadline.value
}

fact submissionDeadlineIsAfterRegistrationDeadline {
        all b: Battle | b.submissionDeadline.value > b.registrationDeadline.value
}

fact subscribedStudentsAreSubscribedToTournament {
        all t: Tournament, b: t.battles, s: b.subscribedStudents | s in t.
            ↪ subscribedStudents.student
}

fact teamExistsOnlyWithBattle {
        all t: Team | one b: Battle | t in b.teams
}

fact teamHasAtLeastMinimumNumberOfStudents {
        all b: Battle, t: b.teams| #t.students ≥ b.rules.minimumNumberOfStudents
}

fact teamHasAtMostMaximumNumberOfStudents {
        all b: Battle, t: b.teams| #t.students ≤ b.rules.maximumNumberOfStudents
}

fact teamStudentsAreSubscribedToBattle {
        all b: Battle, t: b.teams, s: t.students | s in b.subscribedStudents
```

```
}

fact allSubscribedStudentsAreInATeam {
        all b: Battle | all s: b.subscribedStudents | one t: b.teams | s in t.
            ↪ students
}

fact availableTournamentHasNoBattle {
        all t: Tournament | t.status = TAvailable implies t.battles = none
}

fact endedTournamentHasNoaActiveBattle {
        all t: Tournament | t.status = TEnded implies (all b: t.battles | b.status
            ↪ = BEnded)
}

fact activeBattleImpliesActiveTournament {
        all t: Tournament | (some b: t.battles | b.status = BActive) implies t.
            ↪ status = TActive
}

fact availableBattleImpliesActiveTournament {
        all t: Tournament | (some b: t.battles | b.status = BAvailable) implies t.
            ↪ status = TActive and t.status' = TActive
}

fact activeOrAvailableBattleImpliesNotEndedTournament {
        all t: Tournament | (some b: t.battles | (b.status = BActive or b.status =
            ↪ BAvailable)) implies t.status ≠ TEnded
}

fact endedBattleCannotGoBackToAPreviousStatus {
        all b: Battle | always (b.status = BEnded implies b.status' = BEnded)
}

fact activeBattleCannotGoBackToAPreviousStatus {
        all b: Battle | always (b.status = BActive implies (b.status' = BActive or
            ↪ b.status' = BEnded))
}

fact availableBattleCannotGoBackToAPreviousStatus {
        all b: Battle | always (b.status = BAvailable implies (b.status' =
            ↪ BAvailable or b.status' = BActive))
}


-- Predicates

pred tournamentDeadlineExpires[t: Tournament] {
        t.status = TAvailable
        t.status' = TActive
        all s: t.subscribedStudents | s.score' = 0
}

pred tournamentEnds[t: Tournament] {
        t.status = TActive
        t.status' = TEnded
}

pred battleRegistrationDeadlineExpires[b: Battle] {
        b.status = BAvailable
        b.status' = BActive
        all team: b.teams | team.score' = 0
}

pred battleEnds[b: Battle] {
        b.status = BActive
```

```
        b.status' = BEnded
}

pred addABattleToTournament[t: Tournament, b: Battle] {
        t.status = TActive
        b.status = BNotCreated
        t.battles' = t.battles + b
        b.status' = BAvailable
        t.status' = TActive
}

-- Assertions

assert activeTournamentWasOnceAvailable {
        all t: Tournament | tournamentDeadlineExpires[t] implies once t.status =
            ↪ TAvailable
}

assert endedTournamentWasOnceActive {
        all t: Tournament | tournamentEnds[t] implies once t.status = TActive
}

assert activeBattleWasOnceAvailable {
        all b: Battle | battleRegistrationDeadlineExpires[b] implies once b.status
            ↪ = BAvailable
}

assert endedBattleWasOnceActive {
        all b: Battle | battleEnds[b] implies once b.status = BActive
}

assert endedTournamentDoesNotGoBackToPreviousStatus{
        all t: Tournament | always (t.status = TEnded implies t.status' = TEnded)
}

assert activeTournamentDoesNotGoBackToPreviousStatus{
        all t: Tournament | always (t.status = TActive implies (t.status' = TEnded
            ↪ or t.status' = TActive))
}

assert endedBattleDoesNotGoBackToAPreviousStatus {
        all b: Battle | always (b.status = BEnded implies b.status' = BEnded)
}

assert activeBattleDoesNotGoBackToAPreviousStatus {
        all b: Battle | always (b.status = BActive implies (b.status' = BEnded or b
            ↪ .status' = BActive))
}

check activeTournamentWasOnceAvailable
check endedTournamentWasOnceActive
check activeBattleWasOnceAvailable
check endedBattleWasOnceActive
check endedTournamentDoesNotGoBackToPreviousStatus
check activeTournamentDoesNotGoBackToPreviousStatus
check endedBattleDoesNotGoBackToAPreviousStatus
check activeBattleDoesNotGoBackToAPreviousStatus

-- Run

pred example[b: Battle, t: Tournament] {
        t.status = TAvailable
        tournamentDeadlineExpires[t];
        addABattleToTournament[t, b];
        battleRegistrationDeadlineExpires[b];
        battleEnds[b];
        tournamentEnds[t]
```

```
}

pred example1[b, b1: Battle, t: Tournament] {
        t.status = TAvailable
        tournamentDeadlineExpires[t];
        addABattleToTournament[t, b];
        addABattleToTournament[t, b1];
        battleRegistrationDeadlineExpires[b];
        battleEnds[b];
        tournamentEnds[t]
}

run example for 5 but 1 Tournament, 1 Battle
run example1 for 5 but 1 Tournament, 2 Battle
```

The following figures show the evolution of a tournament: starting as available, then the registration deadline expires and it becomes active; then a battle is created and once its registration deadline expires it becomes active too. When the submission deadline ends, the battle closes. Finally, the tournament ends.

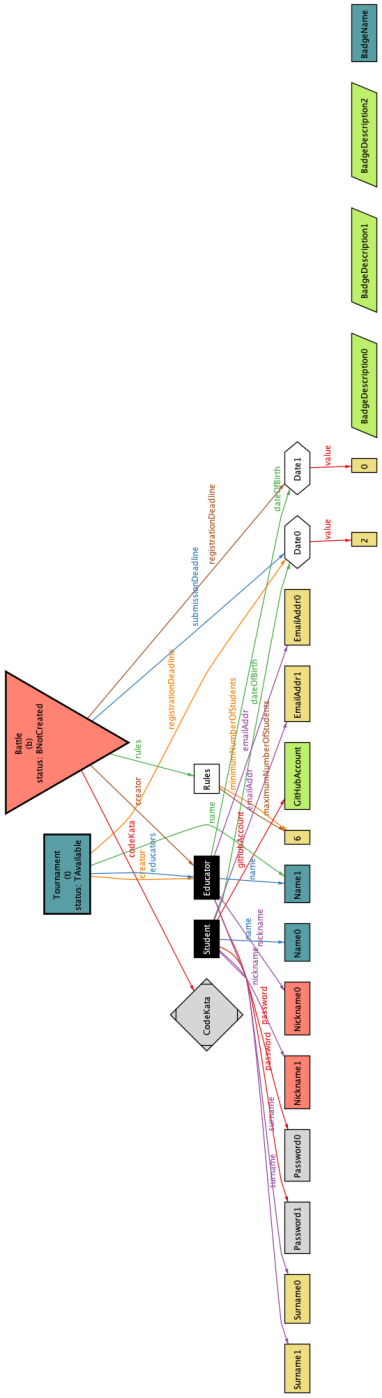

Figure 4.1: Example 1 - Instant 0

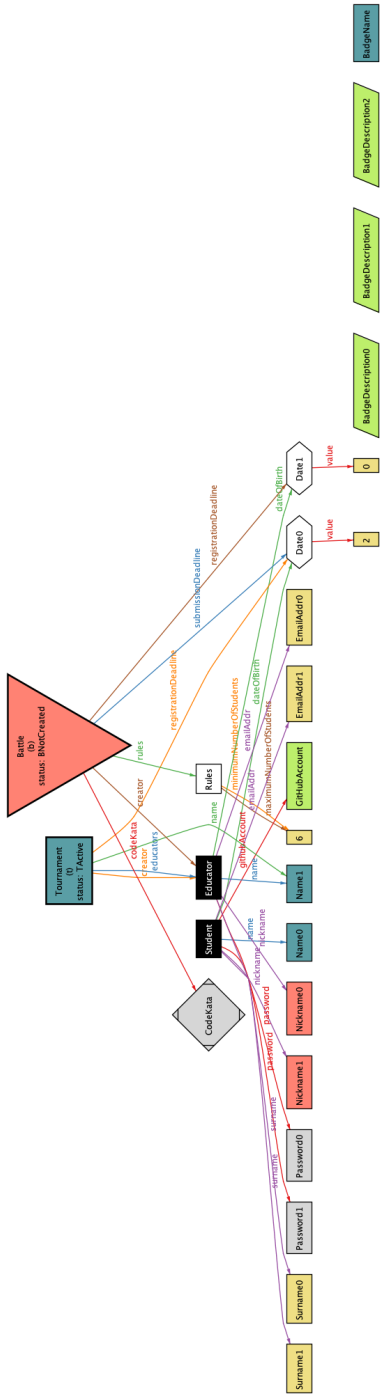

Figure 4.2: Example 1 - Instant 1

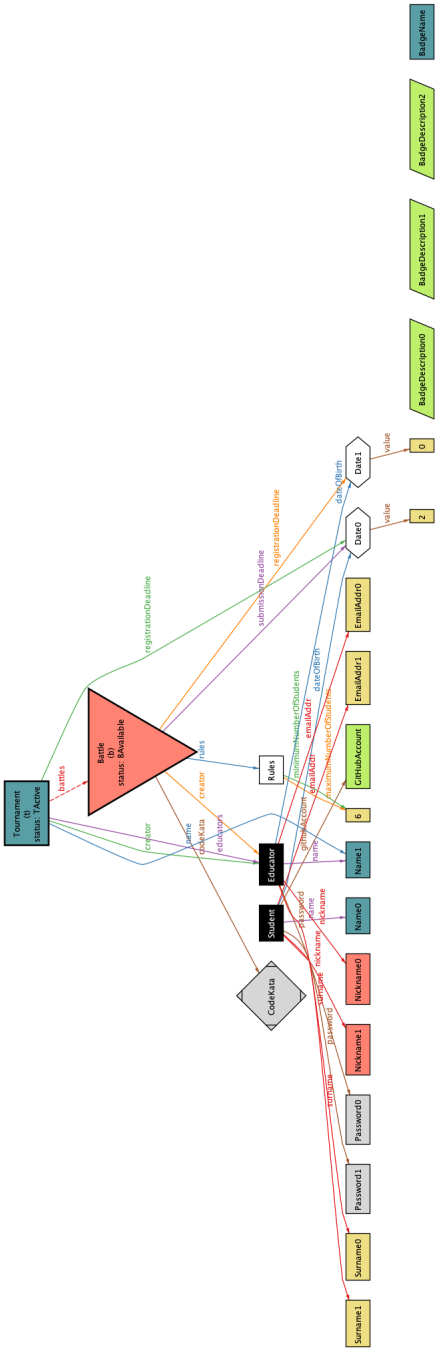

Figure 4.3: Example 1 - Instant 2

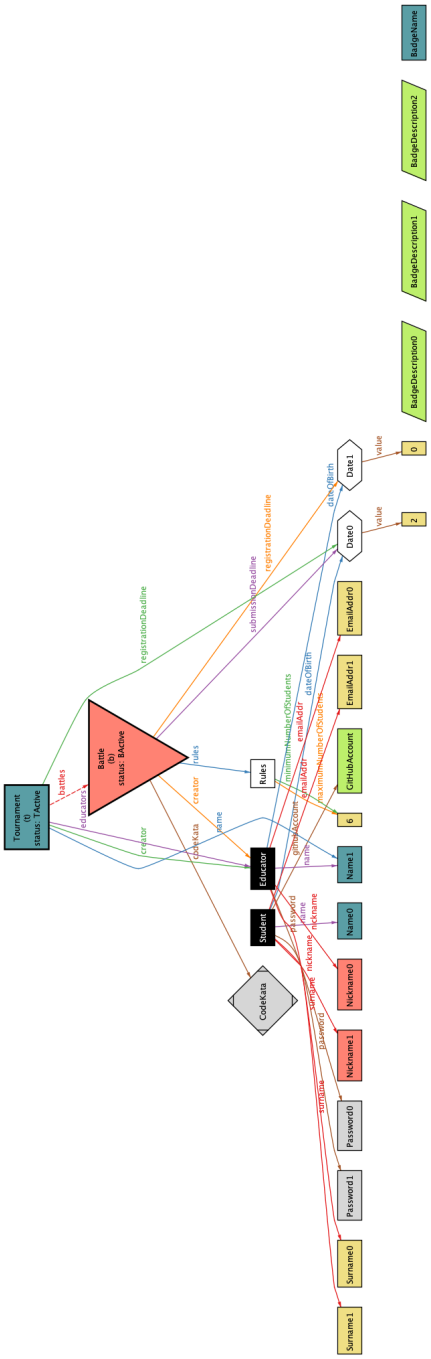

Figure 4.4: Example 1 - Instant 3

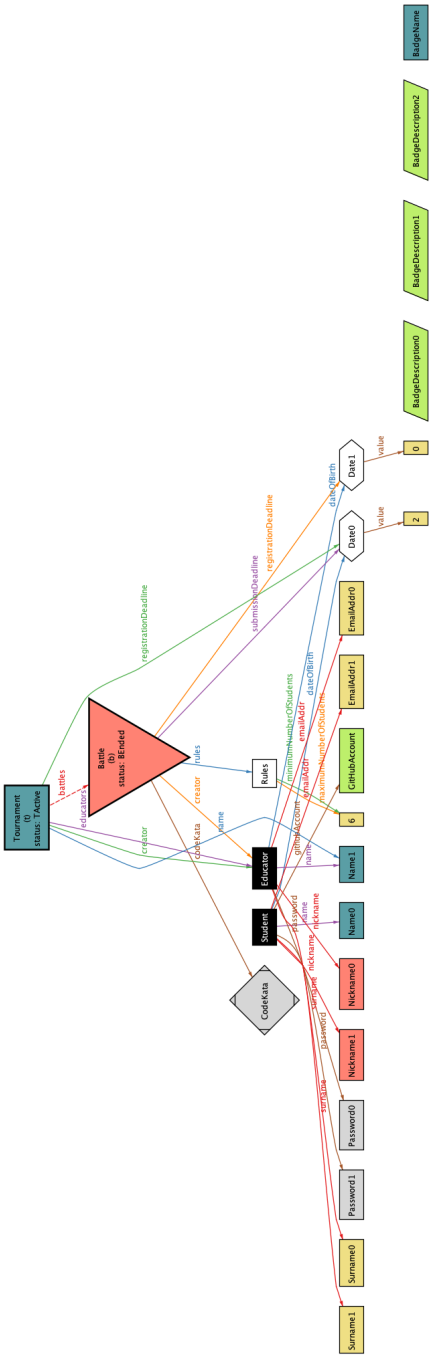

Figure 4.5: Example 1 - Instant 4

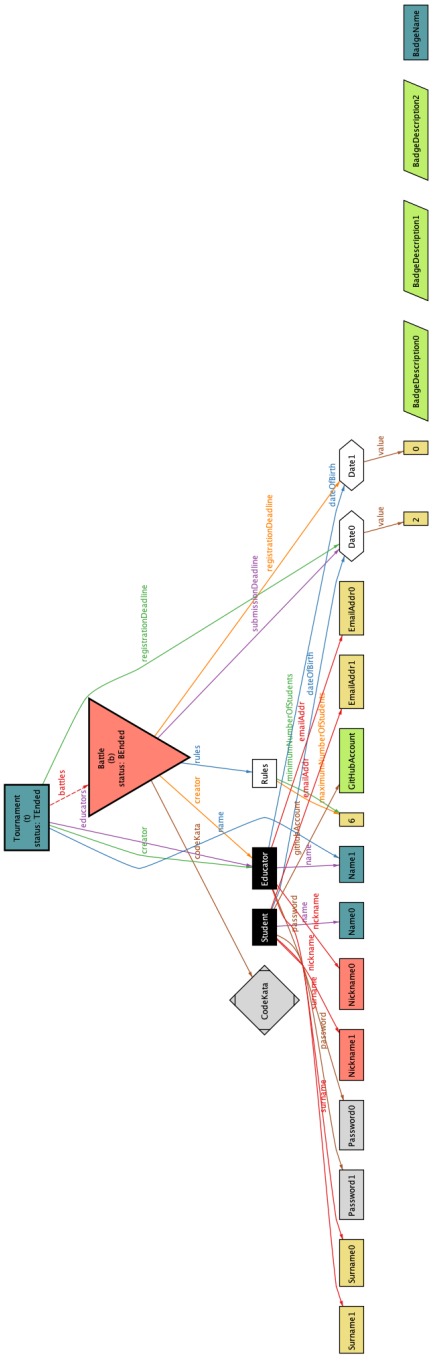

Figure 4.6: Example 1 - Instant 5

Chapter 5

Effort Spent

Group working

Project Specification Analysis, Brainstorming, RASD structure	**4h**
Study of the context and existing protocols	**3h**
Goals and Phenomena elicitation	**5h**
Use cases diagrams and description	**7h**
Definitions, Acronyms	**1h**
Assumptions, dependencies, constraints	**2h**
List of functional requirements	**2h**
Formal Analysis with Alloy	**4h**
Scenarios	**3h**
Final Review	**4h**

Chiara

Sequence diagrams	**4h**
Class Diagram	**2h**
UML	**3h**
User characteristics	**1h**
Mapping on use cases	**2h**

Flavia

User Interfaces	**6h**
State Diagrams	**2h**
Mapping on goals, on requirements	**3h**

Gianluca

Sequence diagrams	**4h**
Formal Analysis with Alloy	**4h**
Product Functions	**2h**
Hardware Interfaces, Software Interfaces, Communication Interfaces	**2h**
Software system attributes	**1h**

Chapter 6

References

- Software Engineering II course slides
- Use Case Diagrams and Sequence Diagrams were made using draw.io
  https://app.diagrams.net/
- Alloy Analyzer
  https://github.com/AlloyTools/org.alloytools.alloy
- UI Mockup
  https://www.figma.com/