POLITECNICO
MILANO 1863

RASD: Requirement Analysis and Specification Document

Cotrone Mariarosaria, De Ciechi Samuele, Deidier Simone

Professor:

Elisabetta DI NITTO

Academic Year: 2023-2024

Contents

1 | INTRODUCTION

1.1. Purpose

The Italian school system often places excessive emphasis on theory, neglecting the importance of practice in scientific subject like informatics. This limits students' ability to apply knowledge in real-world contexts and develop key practical skills. If students don't see a direct connection between what they are learning and its practical usefulness in real life, they may lose interest in studying.

Our goal is to introduce practical experiences for stimulating curiosity with a new platform CodeKataBattle (CKB) that helps students improve their software development skills by training with peers on programming exercises known as "code kata". Each Code Kata Battle is a programming exercise that includes a description, test cases, and build automation scripts. In each battle, students are allowed to form a team by accessing a waiting room, where they await responses from friends and ensure that the number of participants falls within the acceptable range. Then students can start working on the project: they are asked to 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 students push a new commit into the main branch of their repository.

At the same time educators use the platform to challenge students by creating tournaments and challenges in which teams of students can compete against each other, thus proving (and improving) their skills. When a new tournament is created, all students subscribed to the CKB platform are notified and they can subscribe by a given deadline. If they subscribe, they are notified of all upcoming battles created within that tournament.

The final score of a battle is a natural number between 0 and 100 determined by considering some mandatory factors evaluated in a fully automated way, and optional factors evaluated manually by educators.

Furthermore, the platform supports the visualization of students' profiles that includes the display of badges earned during tournaments.

1.1.1. Goals

The following are the goals that we want to pursue with the implementation of the software:

- G1: allow students to participate individually or in teams in programming challenges to increase their skills and earn badges

- G2: allow educators to create new battles
- G3: allow educators to create a new tournament
- G4: allow educators to evaluate student's work
- G5: keep students updated on any new tournaments
- G6: keep students updated on the tournaments in which they are registered (for example if a new battle is created).
- G7: keep the scoreboard of tournaments and battles updated in real time
- G8: allow educators to create new badges and rules for assigning them
- G9: allow educators to close a tournament.

1.2. Scope

The CodeKataBattle software aims to create a stimulating and collaborative environment to enhance students' software development skills, providing them with a practical, competitive, and educational experience. To achieve this goal, the software is conceptualized as a web application with a user-friendly interface, allowing all users to effortlessly access information on active tournaments, scoreboards, and earned badges.

1.2.1. World phenomena

World phenomena	Description
WP01	A student wants to improve his software skills and so he decides to participate to a tournament or battle
WP02	An educator wants to create a tournament
WP03	An educator wants to create a battle
WP04	Educators decide to close a tournament
WP05	The student forks GitHub and setup automation with GitHub action
WP06	The educator wants to create new badges to reward students' efforts.

1.2.2. Shared phenomena

Shared phenomena	Description	Control
SP01	An educator enters their information to register	World controlled
SP02	A student enters their information to register	World controlled
SP03	An educator enters their credentials to login	World controlled
SP04	A student enters their credentials to login	World controlled
SP05	Educator logs out	World controlled
SP06	Student logs out	World controlled
SP07	Educator creates a tournament	World controlled
SP08	Educators create a battle: upload the code kata (description and software project, including test cases and build automation scripts), set minimum and maximum number of students per group, set a registration deadline, set a final submission deadline, set additional configurations for scoring	World controlled
SP09	Educator grants to other colleagues the permission to create battles within the context of a specific tournament	World controlled
SP10	Educator closes the battle and, if required, carries out the manual evaluation	World controlled
SP11	The app shows the evolving ranking during a battle	Machine controlled
SP12	The app shows the list of currently active tournaments and the corresponding ranking	Machine controlled
SP13	Educator closes a tournament	World controlled
SP14	Educator creates new rules, badge and variables	World controlled
SP15	The app allows to see the badges they have won on each student's profile	Machine controlled
SP16	Students decide to register for a tournament	World controlled
SP17	A student decides to invite their mate to form a team	World controlled

SP18	A student decides to join a battle	World controlled
SP19	A student does a commit	World controlled
SP20	The system sends notifications to all users if a tournament is created	Machine controlled
SP21	The system sends notifications when a user is invited to participate in a battle with another user	Machine controlled
SP22	The system sends notifications when a deadline has expired	Machine controlled
SP23	GitHub sends notifications every time a user does a commit	World controlled
SP24	The system sends notifications when the final rank of a battle is ready	Machine controlled
SP25	The system sends notifications when the final rank of the tournament is ready	Machine controlled

1.2.3. Definitions

Definition	Description
Student	A user registered in the system, who can participate in battles and tournaments
Educator	A user registered in the system, who can create new battles and tournaments
Code kata battle	Platform where you can sign up for tournaments and battles and do programming exercises. Depending on your skills you can also get various rewards.
Notification	An alert that a certain event occurred
Badge	Extraordinary reward awarded for certain merits in battles
Commit	Statements that make the changes effective and visible to other users.
Fork	Indicates the development of a new software project that starts from the source code of an existing one, by a programmer.

Tournament	A tournament is made up of one or more battles. It is created by an educator
Battle	A battle is a programming exercise with its own ranking. It is created by the educator who created the tournament or by a collaborator
Code kata	Programming exercise

1.2.4. Acronyms

Acronyms	Description
UML	Unified Modeling Language
WP	World Phenomena
SP	Shared Phenomena
G	Goal
D	Domain Assumption
R	Requirement
UI	User Interface
API	Application Programming Interface
DB	Data Base
CKB	Code Kata Battle
NFR	Non-Functional Requirements

1.2.5. Abbreviations

Abbreviations	Description
Alt	Alternative
Opt	Optional
Id	Identifier

1.3. Revision history

- December 21, 2023: version 1.0 (first release)

1.4. Reference documents

- Specification document: “R&DD Assignment A.Y. 2023-2024”
- Paper: "Jackson and Zave: the world and the machine”
- Alloy official documentation: https://alloytools.org/documentation.html

- UML official specification: https://www.omg.org/spec/UML/

1.5. Document structure

This RASD is structured in 5 chapter:

1. **INTRODUCTION**: in this chapter there is a general description of the goals of the project. The scope is described with the analysis of the world and shared phenomena. In this part there are also the descriptions of some abbreviations and acronyms.
2. **OVERALL DESCRIPTION**: the multiple interactions the system can do are described with scenarios and with class and state diagrams. In the last part there is a list of the domain assumptions and the characteristics of the user who will exploit the application.
3. **SPECIFIC REQUIREMENTS**: this is the main part of the document: here there are more details on all aspects of the previous section that could be useful for the development team. There is the description of hardware and software interfaces, functional requirements described with UML, performance requirements and at the end there are some design constraints.
4. **FORMAL ANALYSIS USING ALLOY**: this section contains a formal description of the software product using Alloy language. There are comments to clarify each part of the modeling.
5. **EFFORT SPENT:** in this section there is all the information about the time every student spent in each section.

2 | OVERALL DESCRIPTION

2.1. Product perspective

2.1.1. Scenarios

- **A new student wants to register at KCB**: Marco is a student who wants to improve his programming skills because theory classes alone bore him. He decides to go and search the web for the CodeKataBattle platform that was recommended to him at the beginning of the year by his professor. He decides to register, after hitting the button to register, the system asks him to enter all his personal information and in particular an e-mail, password and username that will be his login credentials. Marco fills all the text boxes and clicks 'Student sign up'. Then he receives a confirmation email from CodeKataBattle to conclude the registration. The system shows Marco that the registration has been completed.
- **New educator wants to register on KCB:** Nick is a professor who wants to try to get his students passionate about his subject, so he decides to sign up for the CodeKataBattle platform that was recommended to him by a colleague, so he can create challenges on the topics he explains in his computer science classes. He decides to register, after hitting the button to register, the system asks him to enter all his personal information and in particular an institutional e-mail, username and a password that will be his login credentials. Nick fills all the text boxes and clicks 'Educator sign up'. The system verifies that it is an institutional email. Then he receives a confirmation email from CodeKataBattle to conclude the registration. The system shows him that the registration has been completed.
- **Student registration for a tournament:** Anna is watching television when she gets an e-mail notification about the creation of a new tournament. Anna watches that it is a tournament in which the main objective is to improve her knowledge of client-server communication systems. Anna in very interested in this topic so she decides to sign up: she opens the web page, logs in and among the active tournaments looks for the one she is interested in. She clicks on the tournament, here all the information and objectives of the tournament appear to her. Since the deadline has not yet expired Anna sees a tournament registration button appear and decides to press it. Anna is registered for the tournament.
- **Student joins a Battle:** While Anna is studying, she gets an e-mail notification. In this e-mail Anna is informed that a new battle has been created in the tournament in which she has entered. She decides to open the computer to get more information about the battle, so she

logs into her profile on the CKB platform, searches the tournament in the "my tournaments" section, and after clicking the corresponding button, the new battle appears to her. Anna looks at the battle information and since the deadline for joining the competition has not yet expired, she decides to join.

- **Educator creates a new tournament:** Paul is a professor of software engineering at Politecnico of Milan. This year he has decided to make the course more interactive, so he has decided to open a tournament in which the battles will be about the topics he explains in class. To make sure that all students participate, he decides that the top 30 in the overall ranking will get an extra 3 points on the exam. Paul accesses his educator profile, now he sees all his tournaments, those he is a contributor to, and all active tournaments. He clicks on "Create tournament" and enters all the information.
- **Educator creates a new battle:** Letizia has a profile on the CKB platform as an educator. She has just finished explaining the basics of object-oriented programming to her students, so she decides to create a battle for them with the purpose of seeing if the topic is understood. Letizia prepares all the necessary materials: codeKata, tests, etc... She logs in to her profile, the screen of all tournaments appears, she presses on the tournament she created especially for her students and clicks on "Create battle". At this time the screen appears for her, where she must enter all the prepared materials, deadlines, and the minimum/maximum number of people the group can consist of. Afterwards she clicks on the confirmation button.
- **Educator creates a new badge:** Federico is a registered educator on the CKB platform and is creating a tournament for his students. He wants to create the badge to be awarded to the student with the highest winning percentage within the tournament. To do this, he clicks on the "Create new badge" button and uses the two existing variables "TOT_WON_BATTLE" and "TOT_BATTLES" describing the rules for winning the badge, and the MAX function.
- **Student creates a new team for a tournament:** Maria noticed that an educator had created a new battle in which the minimum number of students making up the team must be 2 and the maximum 5. Maria then, after logging in, clicks on the tournament and then on the battle that interests her, decides to sign up, and then clicks the button. At this moment a waiting room, in which she must assemble her team, appears. She wants to do this battle with her best friend Luca, but she's not sure if he will accept the invitation, so to be safe she also sends the invitation to Sara. Only when the team reaches the minimum number of members the button will appear to confirm the registration for the battle. If this does not happen by the deadline Maria will not be able to register for the battle.

- **Notification of new battle invitation:** Simone is studying when he receives a notification via e-mail from CodeKataBattle: Samuele has invited him to join his team to participate in a battle. Simone doesn't know whether to accept the invitation because he has many projects to think about, so he decides to watch the topic of the battle. He logs in, searches for the tournament, looks for the battle and clicks on the button to see the main information. Since he doesn't like the topic, Simone decides to refuse the invitation, so he goes back to the invitation and presses the 'reject' button.
- **Someone wants to see which badges a student has achieved**: Matteo and Silvia are two students registered on the CodeKataBattle platform and they decide to challenge each other: whoever wins the most badges by the end of the month will win a VIP ticket to their favorite singer's concert. The month is almost over, and Matteo hasn't won many badges, so he decides to visit Silvia's profile on the platform to see how many badges she has won. He logs in and enters Silvia's username in the search bar in the section on the right of the home page. Then his friend's profile appears, and he sees that she has won lots of badges.
- **Student wants to see all active tournaments and relative scoreboard**: All the teachers of the software engineering course decide to reward with 2 points all the students who are in first position in the current tournament ranking of each professor. Giulia is a teacher and is responsible for writing the names of the deserving students of each class on an Excel sheet. She enters the platform by entering her credentials, then searches for her colleague's tournament, clicks on the tournament and the updated ranking appears.
- **Educator closes a tournament:** It's December, Alessandro has finished his Phyton course. In January the students will have to take the exam, so he decides to close the tournament that he had created to help his students. He enters in his educator’s profile, then clicks on the tournament information and then on the “End tournament” option. Alessandro also looks at the final ranking and decides to reward the top 5 students.

2.1.2. Class diagram

In this section is reported the class diagram of the system, whose main concepts are presented from a high-level point of view.

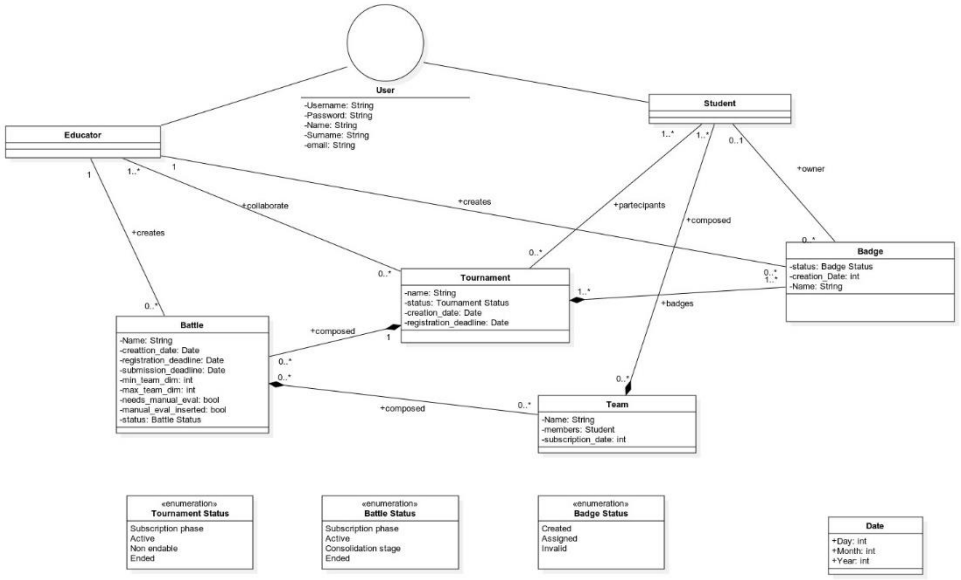

Figure 1: Class Diagram

2.1.3. State diagram

- **Tournament:** this state diagram shows a tournament's four possible states. When an educator creates a new tournament, it enters a "Subscription phase", until a given deadline. After that deadline is past the tournament is now in "Active" state. If an educator (the one who created the tournament or one who was invited to collaborate) creates a battle, the tournament enters a new state, called "Non endable state". It is also possible that, while this state is in progress, new battles are added for the future. The only way to exit this state is that all the battles in the tournament are in the state "ended" or "cancelled" (more on battles' possible states in the next point), leading the tournament state back to active. While the state is "Active" it is possible for an educator to terminate the tournament, leading it to its final state, "Ended".

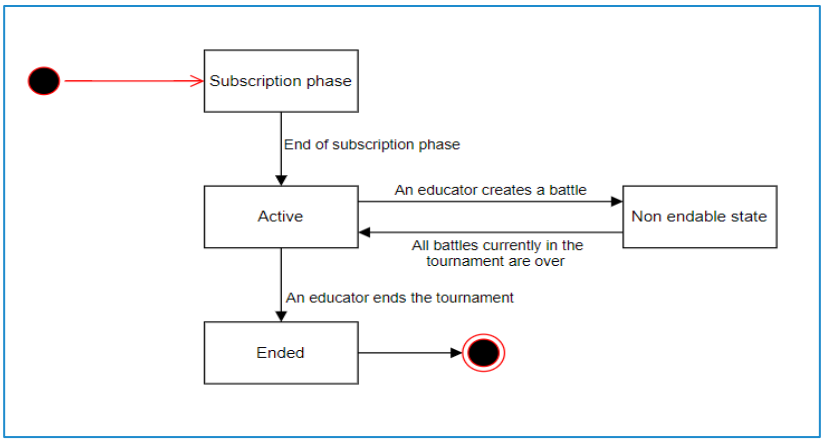

Figure 2: tournament's state diagram

- **Battle:** in the diagram it is possible to see that when a battle is created it enters a "Subscription phase". When the registration deadline expires, the battle's state goes to "Active". As soon as the submission deadline is over, there are two possibilities: if manual evaluation is not required, the final state "Ended" is reached immediately; otherwise, the battle's new state is "Consolidation stage" and for reaching the final state "Ended" an educator's review is necessary.

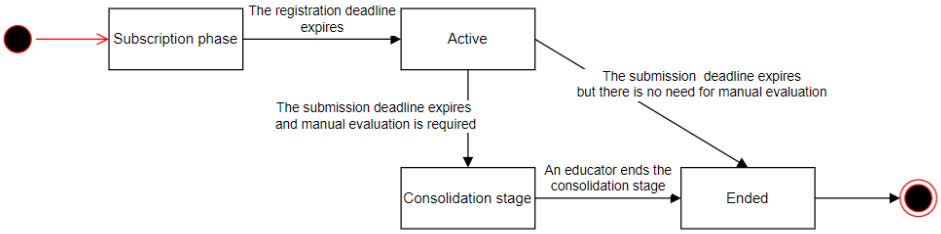

Figure 3: battle's state diagram

- **Badge:** as we can see below, when a tournament is created an educator can define the rules regarding a badge, thus leading it to the "Created" state. When a tournament ends, if there was at least one battle, the final state "Assigned" is reached, otherwise it ends in the final state "Invalid".

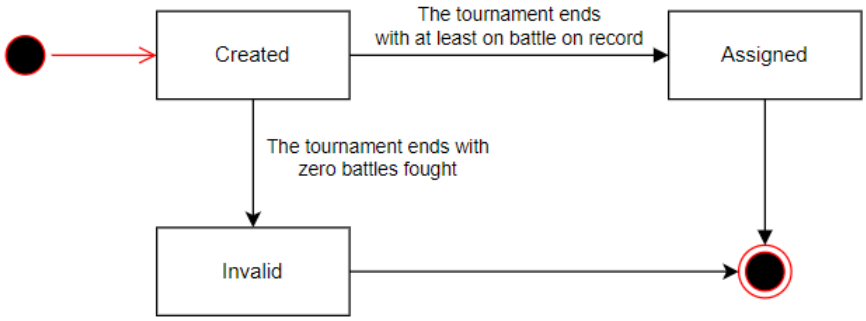

Figure 4: badge's state diagram

- **Invitation for battle:** after a battle is created, interested students can join it through a waiting room, where they can invite other students to create a team. After an invitation for the team is sent, it enters a "Valid" state. If the invitation is accepted, then the final state "Accepted" is reached. The final state "Invalid" instead, is reachable in different situations: if the invite is rejected, if the student joins another team competing in the same tournament, if the team that sent the invitation reaches the maximum number of participants allowed in that battle or if the registration deadline for the battle is over.

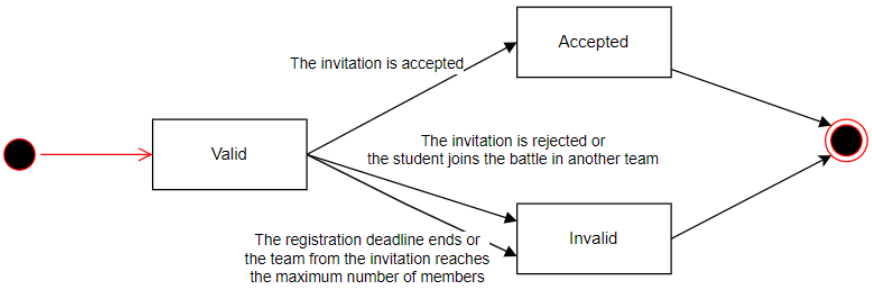

Figure 5: invitation for student state diagram

- **Invitation for collaboration on tournament:** An educator who handles a tournament can decide to invite other educators to provide new battles for students. When an invitation is created it reaches the initial state "Valid". If the invitation is accepted then it moves to the final state "Accepted", otherwise if it is rejected or the tournament ends before the educator gets a chance to reply it ends in the final state "Invalid".

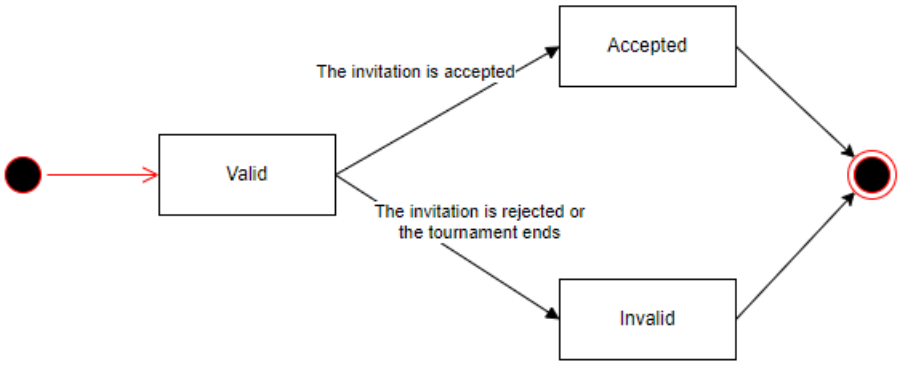

Figure 6: invitation for educator state diagram

2.2. Product functions

2.2.1 Student's product functions

- **Registration:**

To be able to use the web application all users are required to be registered. During the registration students will be asked for an email, a password, a username and their name and surname. Username shall be unique, even across the two different categories of users, students and educators. Before submitting the registration form, it's also required to accept terms and conditions. After the submission, if everything was correctly inserted, the user is redirected to a new page, reminding the user that, to finalize the registration, it is necessary to click on the link received on the email provided during the registration.

- **Register for a tournament:**

A registered student on the website can join every tournament he wants, if the tournament is still in the "Subscription phase". From his personal page, the student can browse through the tournaments and by clicking on "View tournament's informations" he can access the competition's page, where he has an opportunity to register for competing, by clicking on "Join tournament".

- **Register for a battle alone or in a team:**

A student registered in a tournament can participate in the battles of that competition. When a student decides to participate in a battle, to ease the creation of teams, all users must go through a waiting room, where they can form their teams inviting other students to join them in a team. They can invite as many students as they desire, a student can be invited only once for each battle and only students

participating in the tournament can be invited. When the team size reaches the minimum numbers of members for the battle, the creator of the team has the option to finalize the subscription to the battle, subscribing the team in the code competition. If the battle's minimum number of participants for a team is only one, a user who wants to compete alone can finalize his subscription for a team of one person right away after entering the waiting room.

- **Notifications:**

Students receive multiple notifications, regarding different aspects of the website's functionalities. All notifications are received both by email and in the section "My notifications", present in the user's personal page. When each new tournament is created by any educator, all students across the platform are informed through a notification. If a student signs up for a tournament he is notified about every upcoming battle for that tournament and about the final tournament rank availability, when the tournament ends. As described in the previous student's product function, a student can receive a notification regarding an invitation for a team. When a team finalizes their subscription in a battle, each member is notified when the final rank for the battle is available.

- **Visualize the leaderboard in a tournament or battle:**

A student can visualize any leaderboard from all tournaments and battles on the platform. A student can find a specific tournament in different ways: he can search for it on his personal page simply by scrolling or using the search bar available. He may also access it from a previously received notification regarding the tournament. After accessing it, he can see the leaderboard of the tournament and all battles in the competition. Selecting a battle, the student is then able to see the battle's page, thus being presented with the battle's leaderboard and all data available about that challenge.

2.2.2 Educator's product functions

- **Registration:**

To be able to use the web application all users are required to be registered. During the registration educators will be asked for an email, a password, a username and their name and surname. It is important to remember that an educator must provide an institutional email, or his subscription will not be accepted by the system. Username shall be unique, even across the two different categories of users, students and educators. Before submitting the registration form, it's also required to accept terms and conditions. After the submission, if everything was correctly inserted, the user is redirected

to a new page, reminding the user that, to finalize the registration, it is necessary to click on the link received on the email provided during the registration.

- **Notifications:**

An educator receives different notifications, all both by email and in his section “My notifications”. When another educator invites him to handle a tournament together, he is notified, and he can reply to the request by accessing his notification page. When an educator becomes an administrator of a tournament, he will also begin to receive notifications when a new battle for the tournament is created by another educator or when a battle that needs manual evaluation ends.

- **Create a tournament:**

An educator can decide to create a new tournament. To do so he needs to click from his personal page on “Create tournament". He is redirected to a new page, where he must fill in all the fields required, that are the name of the tournament, a brief description and the deadline for the students to subscribe. He also has the chance to create badges for this new tournament, an operation better described in the next product function. The user needs to remember that there should not be two tournaments with the same name created by the same educator. Once he is finished, he can click on “Confirm tournament’s creation” to finalize the operation.

- **Handle badge creation:**

During the previous product function, the badge creation was already mentioned since it is possible to create badges for a tournament only while the tournament is being created. There may also be competitions with no badges involved, it is merely a decision of the educator to add them to their challenge or not. A badge needs to have a name and one or more rules that determine whether a player gets the trophy or not. It is possible to define the rules over pre-defined variables, such as the total number of battles the student has been involved in or the number of commits carried out by the student, but it also possible for the educators to create new variables, combining existing variables for obtaining specific combinations of values. The process for creating new variables is explained in detail in the next paragraph.

- **Create a new variable:**

During the badge creation it is also possible to declare new variables. The users are provided with an extensive list of pre-defined variables, regarding nearly all aspects relevant to scoring. It is also possible to create new variables, combining the variables already defined using simple sign operators

(addition, subtraction, multiplication, division) or operations such as MAX, MIN, AVG to obtain the required new element. A table containing all pre-defined variables is available underneath.

Total number of battles the user participated in during the tournament.	TOT_BATTLE_PARTICIPATED
Total number of commits made by a student during the tournament.	TOT_COMMIT
Highest value among students' commits totals.	MAX_TOT_COMMIT
Total number of battles won by a student in the tournament.	TOT_WON_BATTLE
Total number of times a student finished in the top 5 in a tournament battle.	TOT_TOP5
Total number of times a student finished in the top 10 in a tournament battle.	TOT_TOP10
Total points obtained in all tournament battles in which a student participated.	TOT_POINTS
Highest final score a student achieved in the tournament battles they participated in.	MAX_FINAL_SCORE
Total number of battles created in the tournament.	TOT_BATTLES
Highest score achievable by always finishing first in all battles held in the tournament.	MAX_ACHIEVABLE_SCORE
Total number of different teammates with whom a student participated in battles in the tournament.	TOT_DIFFERENT_TEAMMATES
Total number of battles in the tournament that a student performed with the same team.	TOT_SAME_TEAM
Total number of lines of code a student wrote throughout the tournament.	TOT_LINES_OF_CODE
Average percentage of correctness of the code that is submitted at the end of each battle in the tournament.	AVG_CORRECTNESS_CODE
Average percentage of memory utilization (memory used/memory available) used by the final versions of each code submitted in the battles held in the tournament.	AVG_MEMORY_UTILIZATION
Average test case pass rate at the end of each battle in which a student participated during the tournament.	AVG_PASS_RATE
Total points obtained from manual evaluations at the end of the battles in which the student participated during the tournament.	TOT_MANUAL_EVAL
Time difference between the time and date of the start of the battle and the time and date of the last evaluated commit within the battle in the tournament.	TIME_DIFFERENCE
Average of the times when the student made commits in the battles in which they participated.	AVG_COMMIT_TIME

Average setting speed of GitHub Actions for each battle a student participated in during the tournament.	AVG_ACTIONS_SPEED
Total scores in the "security" category obtained by a student in the battles in which they participated in the tournament.	TOT_SECURITY
Total scores in the "reliability" category obtained by a student in the battles in which they participated in the tournament.	TOT_RELIABILITY
Total scores in the "maintainability" category obtained by a student in the battles in which they participated in the tournament.	TOT_MAINTAINABILITY
Total number of teammates with whom a student participated in battles during the tournament.	TOT_TEAMMATES

In case the user needs a new specific variable, in a situation where it is impossible to obtain the same result with a combination of pre-defined variables, it is possible to issue a ticket to the website for a new variable to be added to the already available one.

- **Close a tournament:**

An educator who is an admin of a tournament is allowed to end it. To do so, the tournament must not have any ongoing or upcoming battles in the future. If the tournament respects these conditions an educator can end a competition by accessing the tournament's page and clicking on "End tournament".

- **Create a battle:**

An educator who is an administrator of a tournament may create new battles for students to compete in. After accessing a tournament's page, an option is available (if the competition is still in progress and is past the subscription phase) for the educator to create a new battle. When "Create battle" is clicked, a new page is loaded containing a form with all fields required for the creation. An educator needs to fill these fields with the 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 whether manual evaluation is required. After everything is filled in and each file has been uploaded the educator can confirm the creation by clicking "Confirm battle's creation".

- **Enter manual evaluation:**

In the previous paragraph we discussed the fields needed to create a battle and we stated that the educator needed to insert whether manual evaluation is required. In case an educator selects this

option, after the submission deadline expires the leaderboard is not finalized until the manual evaluation takes place. To assign the scores due to manual evaluation only the educator that created the battle can access a battle's page and click "Assign manual evaluation". There is a list with the names of the groups, alongside a link to the GitHub repository of the group and the automatic evaluation they achieved. After reviewing the content, a group produced, he can add an evaluation in a text field in the line corresponding to the group. During this process the user may assign some evaluation, then click on "Save my modifications" and leave the rest of the correction for later. When all submissions are linked to a grade, after saving the modifications, the battle's final leaderboard is updated, and the battle officially ends.

- **Manage a tournament:**

After creating a tournament and the subscription of students, the educator needs to provide battles for the students to engage in. He can decide to take on this matter on his own or invite other educator to be administrators of the tournament with him: to do so, the tournament's page visible by an administrator of the tournament there is option "Add collaborator" that leads to a new page, where a search bar is present. The educator can use the search bar to look for other educators (by username or email) and the system retrieves possible matches who the educator may send an invite to by clicking on "Send invitation" next to the name of the educator. The invited educators receive a notification, which they can reply to by accessing their "My notifications" menu on their personal page.

2.3. User characteristics

The actors of the applications are the following:

- **Unregistered user:** a person who has not yet registered and is allowed to sign up to become either a Student or Educator.
- **User:** a person who is already registered into the system and is allowed to perform the actions available for their role (Educator or Student).
- **Educator:** a User of CKB, those who can create tournaments and battles within the platform. They can also create new badges within a tournament, along with new rules to define them, and, if it is required, they can manually evaluate student work in different challenges.
- **Student:** a User of CKB, are those who can participate in tournaments and battles. In each tournament they can win badges and see the current rankings.

2.4. Assumptions and dependencies

2.4.1. Domain assumptions

Domain Assumptions	Descriptions
D1	Students and Educators enter the correct credentials during the login phase
D2	Each educator has an institutional email
D3	All users have an internet connection
D4	All users have an account on GitHub
D5	CKB is able, via the GitHub API, to create a Repository
D6	All students have a computer that allows them to write code
D7	All students fork on GitHub when notification arrives
D8	Each tournament has its own unique ID
D9	Each battle has its own unique ID
D10	Each team has its own unique ID
D11	If a manual evaluation is planned, it is carried out
D12	Each tournament will end
D13	All notifications arrive correctly and on time to all users
D14	When an educator uploads data for a new battle, they are uploaded correctly
D15	Each badge has its own unique ID
D16	Each educator knows how to use the language used to create rules to assign badges

2.4.2. Dependencies

The web application relies on all users having an active Internet connection and an updated browser for surfing the web.

3 | Specific requirements

3.1. External interface requirements

3.1.1. User interfaces

Both students and educators access the CKB platform via the website. The two types of users, however, can perform different actions, so they have two different UIs depending on their role. The following mockups will give a more detailed description in terms of interfaces.

Common user interfaces

- **New user Sign Up:** this page is common to both users. A user can register as a student or educator by entering information such as email, first name, last name, username and the account password in the required fields. After accepting the terms of service and privacy policy the user can register as a student or as an educator. In the case of educator registration, a check is made to see that the email entered is an institutional or business one.

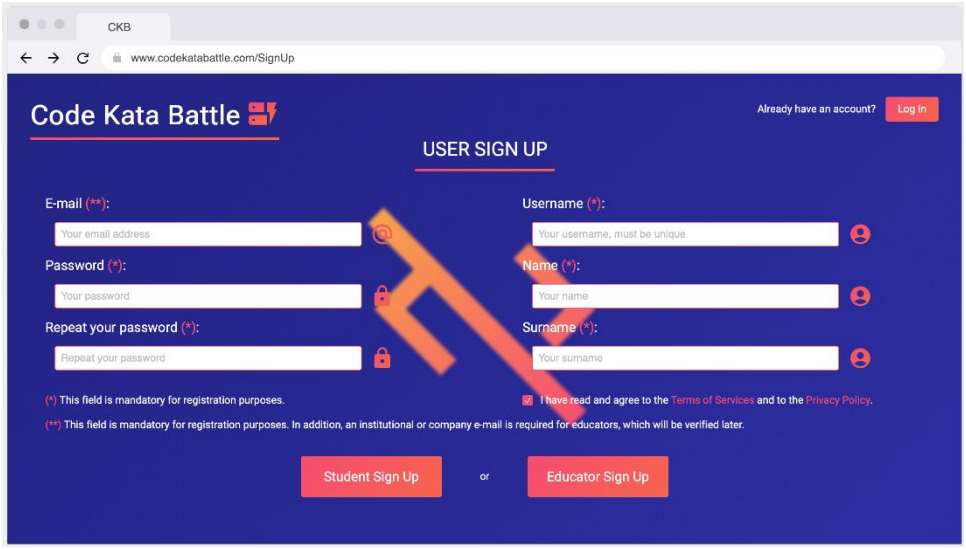

Figure 7: new user Sign Up web page

- **User Log In:** this is another page common to all users. The user fills in the form on the Log In page by entering the required information such as email and password with which he has previously registered. After clicking the "Log In!" button the system checks the data entered and redirects the user to the main page (educator's homepage if the data correspond to an educator's, student's homepage otherwise).

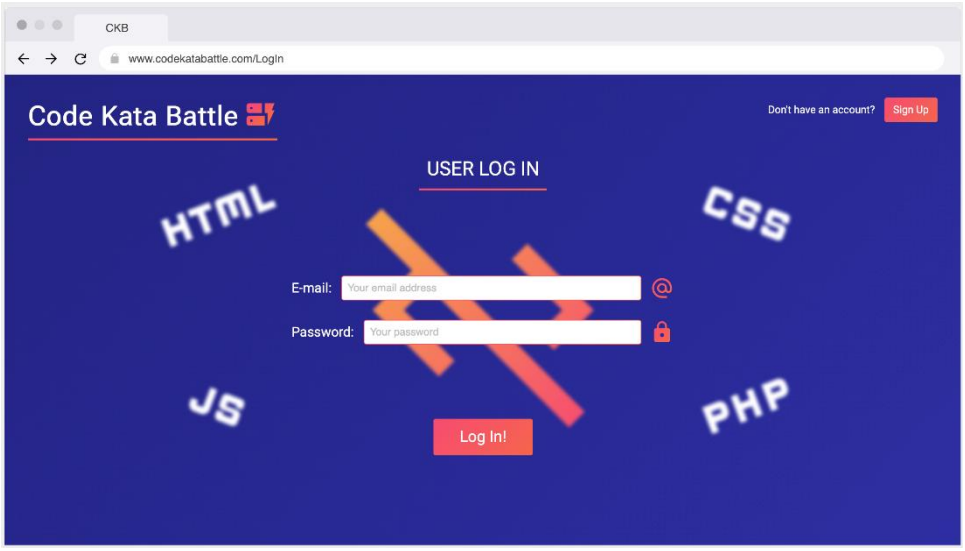

Figure 8: Log In webpage

Students' user interfaces

- **Student Home Page:** this is the main page for each student, as well as the page to which each student is redirected after logging in. On this page, the user can easily access his or her profile to edit information, the notification area, search for tournaments or user profiles, and possibly log out. In addition, the page presents a list of tournaments in which the student is registered and one containing all tournaments; the user can access more information on a particular tournament with the provided button.

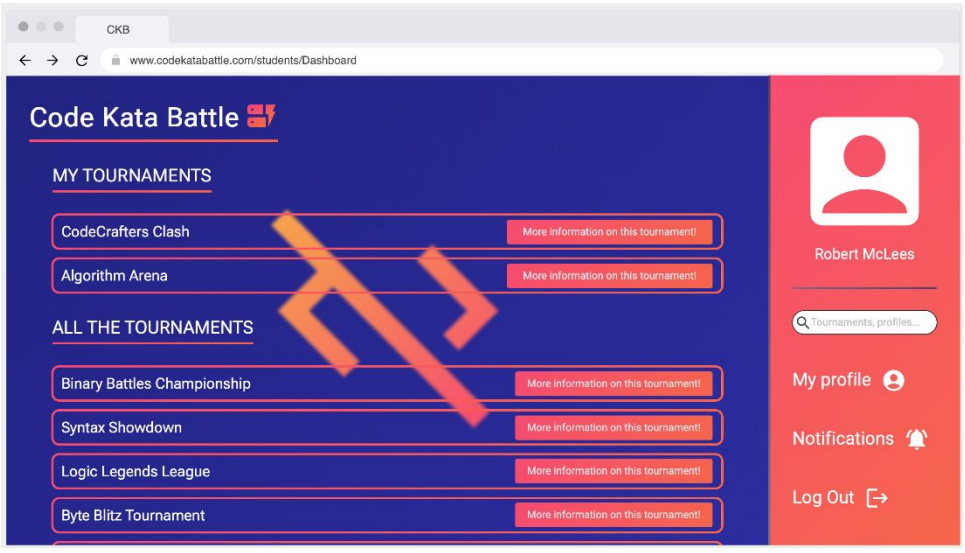

Figure 10: students' Home Page

- **Tournament information page:** this page presents the student with the main information about a tournament, such as the creator, the leaderboard, the tournament description and the

list of active and non-active battles. In addition, it is possible to see the list of educators who are moderators of the tournament and, in the case of a student who is not registered for the tournament, the registration deadline and the button to register (*in this case the student is already registered*). If the student wants more information about a battle, he or she can access it using the dedicated button.

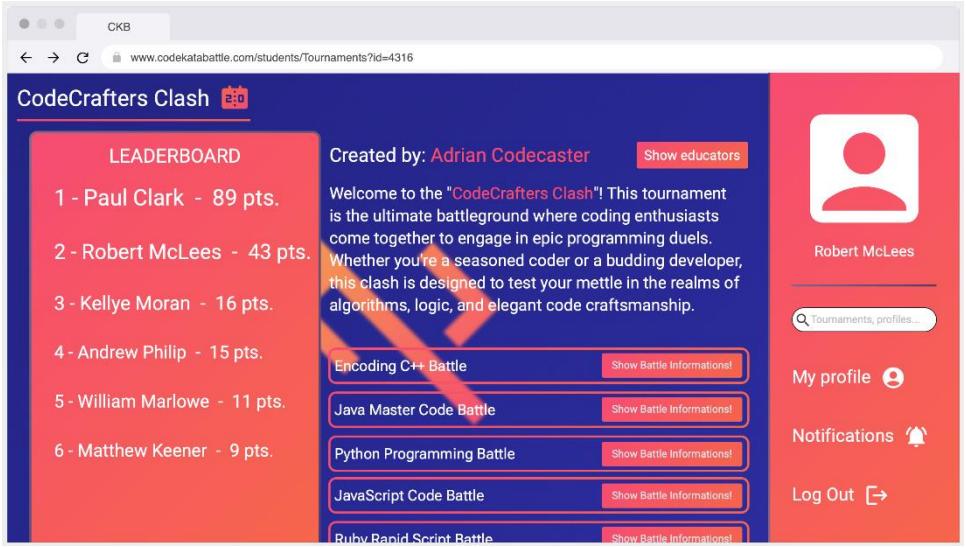

Figure 11: tournament informations webpage

- **Battle information page:** on this page, the student is presented with the main information of a battle, such as the team leaderboard, creation date, registration deadline, battle status and the task or problem on which the battle was created. In addition, if a student is not registered and it is still possible to do so, there is a button to register a new team in the battle (*in this case the user is already in a registered team*).

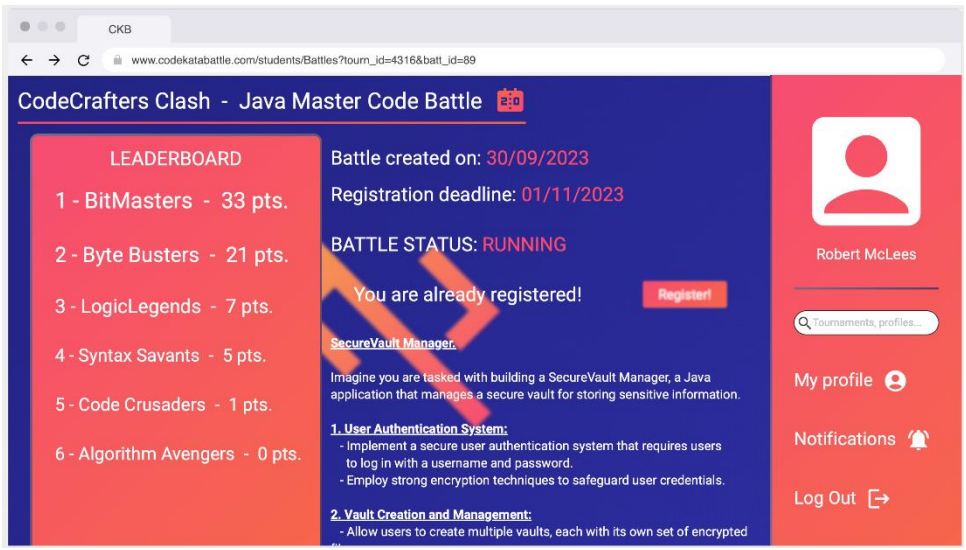

Figure 12: battle informations webpage

- **Team registration page:** this is the page from which a student can invite other students to form a registered team for a battle, accessed from the battle information page. The page presents some information such as the minimum and maximum number of members for a team for that battle, there is a search bar to search for other students and a button to send an invitation to the searched student. There is also a summary section showing current team members and invitations (accepted, declined or pending). Once you have reached the minimum number of members, you can register your team for the battle by entering the team's name and pressing the "Join the battle!" button.

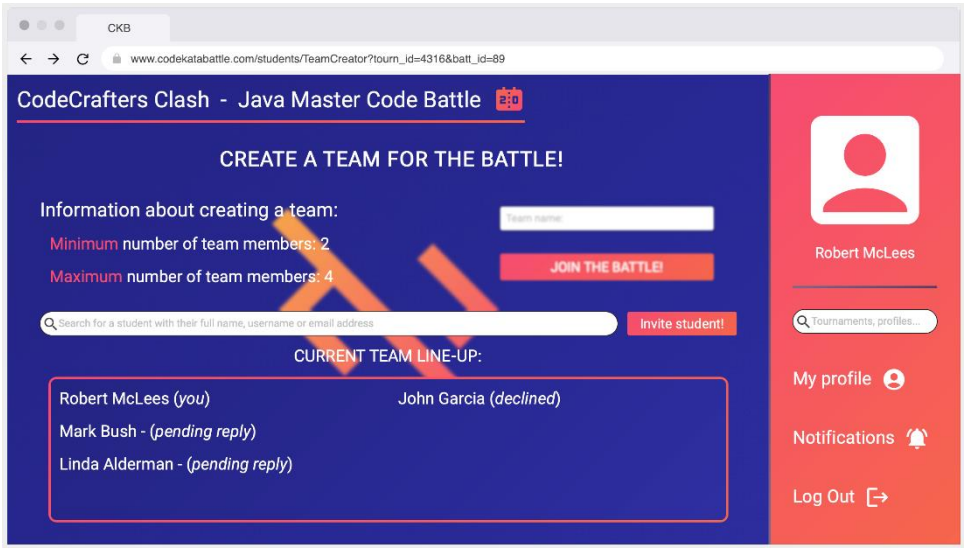

Figure 13: team registration webpage

- **Student profile page:** on this page, a student can easily access and, if necessary, modify his or her personal information, such as first name, surname, username, password and email. In addition, there is a list of the scores obtained by the student in past tournaments and a list of the badges obtained in the tournaments in which he or she has participated.

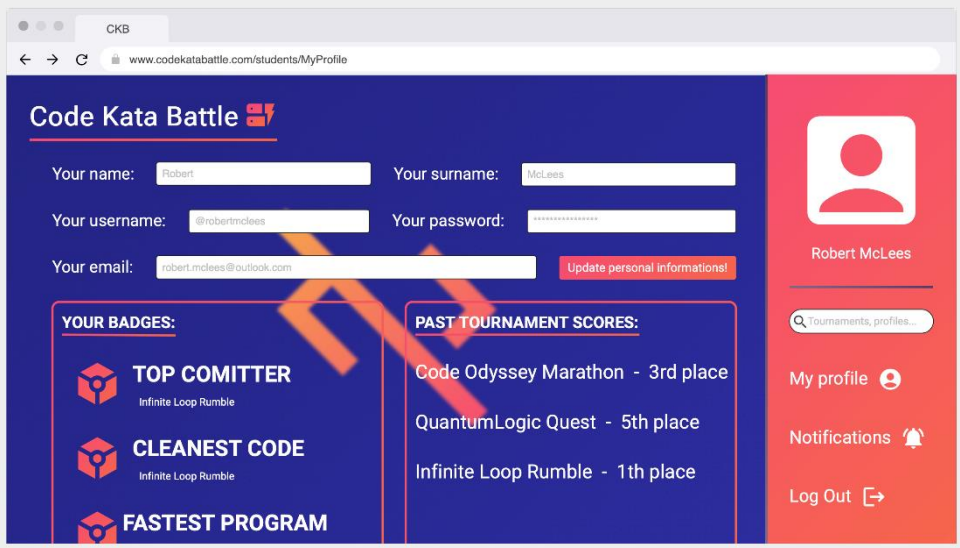

Figure 14: student profile webpage

Educators' user interfaces

- **Educator Home Page:** this is the main page for each educator, as well as the page to which each educator is redirected after logging in. On this page, the user can easily access his or her profile to edit information, the notification area, search for tournaments or user profiles, and possibly log out. In addition, the page presents the list of tournaments created by the user or where he's a collaborator and one containing all tournaments; the user can access more information on a particular tournament with the provided button or create a new tournament by clicking the "Create tournament" button.

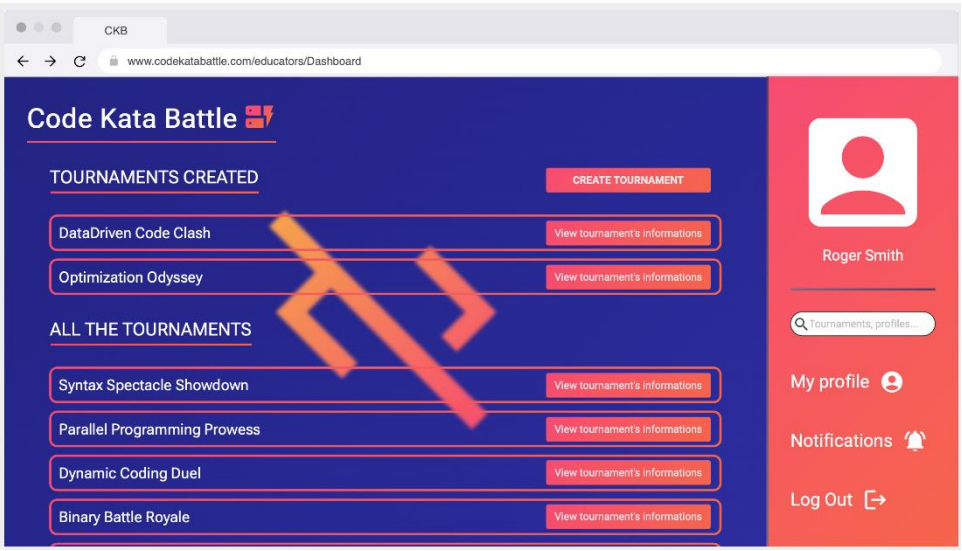

Figure 15: Educator home page

- **Tournament information page:** this is the page that an educator accesses when clicking on the "View tournament information" button of a tournament for which he or she is the creator or moderator. On this page the educator can view the live leaderboard, create new badges for the tournament, add other educators as collaborators, edit the tournament information and, if possible, end the tournament itself. In addition, the page contains the list of battles in the tournament and a button to create new ones.

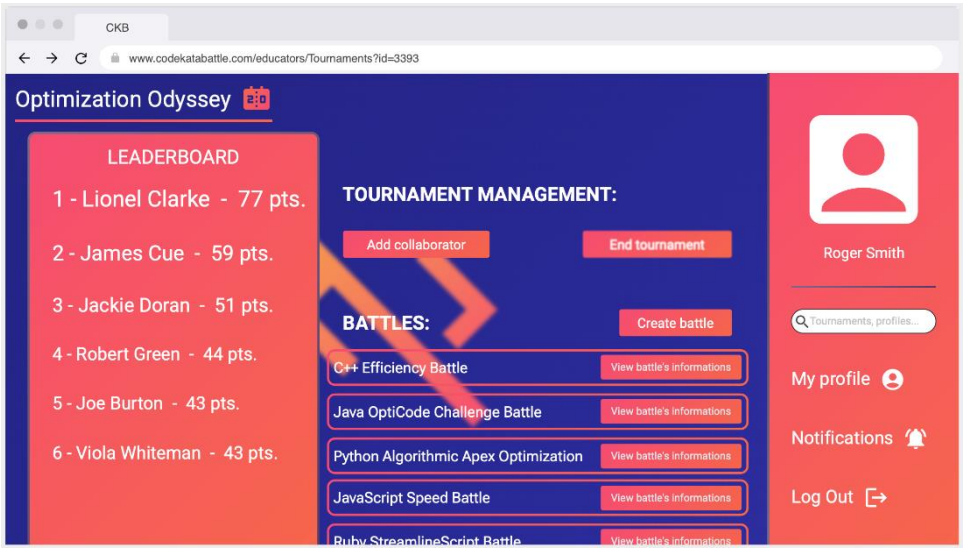

Figure 16: tournament information page

- **Battle information page:** on this page an educator who is a creator or collaborator for the tournament in which the battle was registered can view information about it such as the creation date, the registration deadline and the status of the battle. In addition, the educator can access the page to enter a manual evaluation once the battle has ended or edit information about the battle.

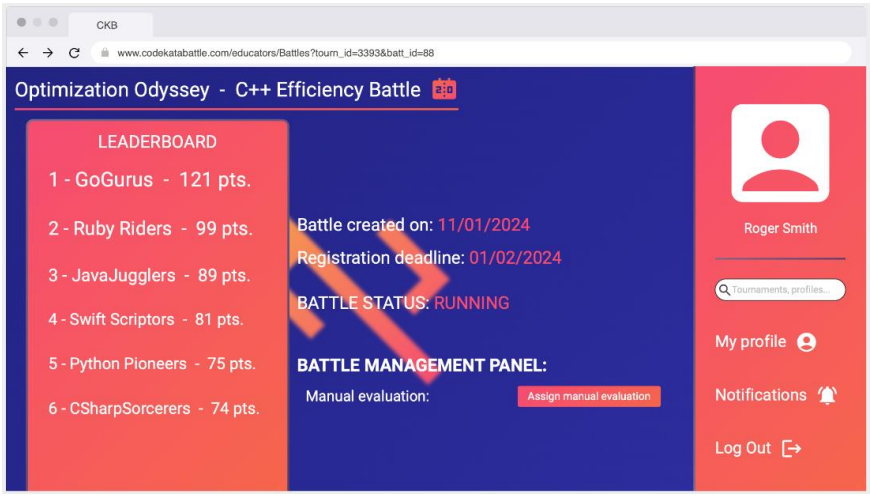

Figure 17: battle information page

3.1.2. Hardware interfaces

All users of the system are required to have a digital device able to navigate the internet. Both educators and students may access the website through either a computer or a mobile device. The system does not require the use of dedicated hardware.

3.1.3. Software interfaces

Users can access the web application without the need to install any program.

- GitHub APIs: to create repositories for all the teams participating in a battle when the registration phase ends and to get updates from each commit a team pushes in the main branch of their GitHub while the battle is in progress.

3.1.4. Communication interfaces

To maintain the correct behaviour of the application and up to date information the system requires a stable internet connection.

The two interfaces, one for the student and the other for the educator, are two web pages, therefore, they communicate using the HTTP protocol.

All communication interfaces and protocols will be specified in the DD document.

3.2. Functional requirements

3.2.1 List of Requirements

Students' requirements

Requirements	Description
R1.1	The system shall allow an unregistered student to register: • The system must allow the Customer to fulfill the registration form • The system must allow the Customer to read and accept the Terms and Conditions
R1.2	Allow Students to login using username/email and password
R1.3	Allow Students to log out
R1.4	The system allows each student to see the tournaments in which they are enrolled

R1.5	The system allows each student to see the rankings of the tournaments in which the student is entered
R1.6	The system allows each student to see if a battle is taking place within the tournament in which the student is enrolled
R1.7	The system allows each student to see the ranking of active battle
R1.8	The system allows each student to see all active tournaments
R1.9	The system allows each student to see the rankings of active tournaments
R1.10	The system allows each student to see if there are running battles in active tournament
R1.11	The system allows each student to see their badges
R1.12	The system allows each student to see the badges of the other students.
R1.13	The system allows each student to see all available badges in each tournament and the rules for winning them
R1.14	The system allows each student to see information and deadlines of each battle
R1.15	The system sends the repository link to all teams registered for the battle
R1.16	The system notifies when a new tournament is created
R1.17	The system notifies when a new battle is created
R1.18	The system notifies each student when they are invited to participate in a battle
R1.19	The system notifies each student when a deadline has passed
R1.20	The system notifies each student when the ranking of the battle, in which the player has participated, is ready
R1.21	The system notifies each student when the ranking of the tournament, in which the player has participated, is ready
R1.22	The system allows students to register for the tournament
R1.23	The system allows students to create their own team in a battle
R1.24	The system allows students to invite other students onto their team
R1.25	The system allows students to register for a battle

Educators' requirements

Requirements	Description
R2.1	The system shall allow an unregistered educator to register: • The system must allow the Customer to fulfill the registration form • The system must allow the Customer to read and accept the Terms and Conditions
R2.2	Allow Educators to login using username/email and password
R2.3	Allow Educators to log out
R2.4	The system allows educators to see the tournaments they have created
R2.5	The system allows educators to see the tournaments in which they were invited to participate as collaborators
R2.6	The system allows educators to see the active tournaments
R2.7	The system allows educators to see the updated ranking of the tournaments that he has created
R2.8	The system allows educators to see the updated ranking of tournaments in which they were invited to participate as collaborators
R2.9	The system allows educators to see the updated ranking of the active tournaments that he has created
R2.10	The system allows educators to see the running or finished battles of their tournament
R2.11	The system allows educators to see the running or finished battles of the tournaments in which they were invited to participate as collaborators
R2.12	The system allows educators to see the running or finished battles of all active tournaments
R2.13	The system allows each educator to see all available badges in each tournament and the rules for winning them
R2.14	The system allows each educator to see the badges of the students.
R2.15	The system notifies each educator when they are invited to participate in a tournament
R2.16	The system notifies each educator when a battle is created in a tournament that he has created or in a tournament in which he is invited to participate as collaborator

R2.17	The system notifies each educator when a manual evaluation is required in a battle that he has created
R2.18	The system allows each educator to create a new tournament
R2.19	The system allows each educator to create a new battle (each educator can insert codekata, set deadlines, set maximum and minimum number of students per group, set additional configurations for scoring).
R2.20	The system allows each educator to send an invitation to another educator for becoming a collaborator
R2.21	The system allows each educator to make a manual evaluation, if it is required
R2.22	The system allows each educator to close a tournament
R2.23	The system allows each educator to create badges
R2.24	The system allows each educator to create new variables

3.2.2. Mapping Requirements and Domain assumptions on Goals

- **G1: allow students to participate individually or in teams in programming challenges to increase their skills and earn badges**

Requirements/ Domain Assumptions	**Description**
R1.1	The system shall allow an unregistered student to register:
R1.2	Allow Students to login using username/email and password
R1.4	The system allows each student to see the tournaments in which they are enrolled
R1.6	The system allows each student to see if a battle is taking place within the tournament in which the student is enrolled
R1.8	The system allows each student to see all active tournaments
R1.11	The system allows each student to see their badges
R1.13	The system allows each student to see all available badges in each tournament and the rules for winning them
R1.14	The system allows each student to see information and deadlines of each battle
R1.15	The system sends the repository link to all teams registered for the battle

R1.16	The system notifies when a new tournament is created
R1.17	The system notifies when a new battle is created
R1.18	The system notifies each student when they are invited to participate in a battle
R1.22	The system allows students to register for the tournament
R1.23	The system allows students to create their own team in a battle
R1.24	The system allows students to invite other students onto their team
R1.25	The system allows students to register for a battle
R2.2	Allow Educators to login using username/email and password
R2.18	The system allows each educator to create a new tournament
R2.19	The system allows each educator to create a new battle (each educator can insert codekata, set deadlines, set maximum and minimum number of students per group, set additional configurations for scoring)
R2.23	The system allows each educator to create badges
D1	Students and Educators enter the correct credentials during the login phase
D3	All users have an internet connection
D4	All users have an account on GitHub
D5	CKB is able, via the GitHub API, to create a Repository
D6	All students have a computer that allows them to write code
D7	All students fork on GitHub when notification arrives
D8	Each tournament has its own unique ID
D9	Each battle has its own unique ID
D10	Each team has its own unique ID
D13	All notifications arrive correct and on time to all users
D15	Each badge has its own unique ID

- **G2: allow educators to create new battles**

Requirements/ Domain Assumptions	Description
R2.1	The system shall allow an unregistered educator to register:
R2.2	Allow Educators to login using username/email and password
R2.5	The system allows educators to see the tournaments in which they were invited to participate as collaborators
R2.10	The system allows educators to see the running or finished battles of their tournament
R2.11	The system allows educators to see the running or finished battles of the tournaments in which they were invited to participate as collaborators
R2.15	The system notifies each educator when they are invited to participate in a tournament
R2.18	The system allows each educator to create a new tournament
R2.19	The system allows each educator to create a new battle (each educator can insert codekata, set deadlines, set maximum and minimum number of students per group, set additional configurations for scoring).
D1	Students and Educators enter the correct credentials during the login phase
D2	Each educator has an institutional email
D3	All users have an internet connection
D5	CKB is able, via the GitHub API, to create a Repository
D8	Each tournament has its own unique ID
D9	Each battle has its own unique ID
D14	When an educator uploads data for a new battle, they are uploaded correctly

- **G3: allow educators to create a new tournament**

Requirements/ Domain Assumptions	Description
R2.1	The system shall allow an unregistered educator to register:
R2.2	Allow Educators to login using username/email and password
R2.18	The system allows each educator to create a new tournament
R2.23	The system allows each educator to create badges
D1	Students and Educators enter the correct credentials during the login phase
D2	Each educator has an institutional email
D3	All users have an internet connection
D8	Each tournament has its own unique ID

- **G4: allow educators to evaluate student's work**

Requirements/ Domain Assumptions	Description
R2.1	The system shall allow an unregistered educator to register:
R2.2	Allow Educators to login using username/email and password
R2.5	The system allows educators to see the tournaments in which they were invited to participate as collaborators
R2.11	The system allows educators to see the running or finished battles of the tournaments in which they were invited to participate as collaborators
R2.13	The system allows each educator to see all available badges in each tournament and the rules for winning them
R2.17	The system notifies each educator when a manual evaluation is required in a battle that he has created
R2.18	The system allows each educator to create a new tournament
R2.19	The system allows each educator to create a new battle (each educator can insert codekata, set deadlines, set maximum and minimum number of students per group, set additional configurations for scoring).
R2.21	The system allows each educator to make a manual evaluation, if it is required

R1.2	Allow Students to login using username/email and password
R1.4	The system allows each student to see the tournaments in which they are enrolled
R1.8	The system allows each student to see all active tournaments
R1.15	The system sends the repository link to all teams registered for the battle
R1.16	The system notifies when a new tournament is created
R1.17	The system notifies when a new battle is created
R1.15	The system sends the repository link to all teams registered for the battle
R1.16	The system notifies when a new tournament is created
R1.25	The system allows students to register for a battle
D1	Students and Educators enter the correct credentials during the login phase
D2	Each educator has an institutional email
D3	All users have an internet connection
D9	Each battle has its own unique ID
D10	Each team has its own unique ID
D11	If a manual evaluation is planned, it is carried out

- **G5: keep students updated on any new tournaments**

Requirements/ Domain Assumptions	**Description**
R1.1	The system shall allow an unregistered student to register:
R1.2	Allow Students to login using username/e-mail and password
R1.7	The system allows each student to see the ranking of active battle
R1.8	The system allows each student to see all active tournaments
R1.9	The system allows each student to see the rankings of active tournaments
R1.10	The system allows each student to see if there are running battles in active tournament
R1.16	The system notifies when a new tournament is created
R1.22	The system allows students to register for the tournament

R2.2	Allow Educators to login using username/e-mail and password
R2.18	The system allows each educator to create a new tournament
D1	Students and Educators enter the correct credentials during the login phase
D3	All users have an Internet connection
D8	Each tournament has its own unique ID
D13	All notifications arrive correct and on time to all users

- **G6: keep students updated on the tournaments in which they are registered.**

Requirements/ Domain Assumptions	**Description**
R1.1	The system shall allow an unregistered student to register:
R1.2	Allow Students to login using username/email and password
R1.5	The system allows each student to see the rankings of the tournaments in which the student is entered
R1.6	The system allows each student to see if a battle is taking place within the tournament in which the student is enrolled
R1.14	The system allows each student to see information and deadlines of each battle
R1.17	The system notifies when a new battle is created
R1.20	The system notifies each student when the ranking of the battle, in which the player has participated, is ready
R1.21	The system notifies each student when the ranking of the tournament, in which the player has participated, is ready
R1.22	The system allows students to register for the tournament
R1.23	The system allows students to create their own team in a battle
R1.24	The system allows students to invite other students onto their team
R2.2	Allow Educators to login using username/email and password
R2.17	The system notifies each educator when a manual evaluation is required in a battle that he has created
R2.18	The system allows each educator to create a new tournament

R2.19	The system allows each educator to create a new battle (each educator can insert codekata, set deadlines, set maximum and minimum number of students per group, set additional configurations for scoring).
R2.21	The system allows each educator to make a manual evaluation, if it is required
R2.22	The system allows each educator to close a tournament
D1	Students and Educators enter the correct credentials during the login phase
D2	Each educator has an institutional email
D3	All users have an internet connection
D8	Each tournament has its own unique ID
D9	Each battle has its own unique ID
D13	All notifications arrive correct and on time to all users

- **G7: keep the scoreboard of tournaments and battles updated in real time**

Requirements/ Domain Assumptions	**Description**
R1.5	The system allows each student to see the rankings of the tournaments in which the student is entered
R1.7	The system allows each student to see the ranking of active battle
R1.9	The system allows each student to see the rankings of active tournaments
R2.17	The system notifies each educator when a manual evaluation is required in a battle that he has created
R2.21	The system allows each educator to make a manual evaluation, if it is required
D3	All users have an internet connection
D5	CKB is able, via the GitHub API, to create a Repository
D8	Each tournament has its own unique ID
D9	Each battle has its own unique ID
D9	Each battle has its own unique ID
D10	Each team has its own unique ID

- **G8: allow educators to create new badges and rules for assigning them**

Requirements/ Domain Assumptions	Description
R2.1	The system shall allow an unregistered educator to register:
R2.2	Allow Educators to login using username/email and password
R2.13	The system allows each educator to see all available badges in each tournament and the rules for winning them
R2.18	The system allows each educator to create a new tournament
R2.23	The system allows each educator to create badges
R2.24	The system allows each educator to create new variables
D1	Students and Educators enter the correct credentials during the login phase
D2	Each educator has an institutional email
D3	All users have an internet connection
D8	Each tournament has its own unique ID
D15	Each badge has its own unique ID
D16	Each educator knows how to use the language used to create rules to assign badges

- **G9: allow educators to close a tournament.**

Requirements/ Domain Assumptions	Description
R2.1	The system shall allow an unregistered educator to register:
R2.2	Allow Educators to login using username/email and password
R2.5	The system allows educators to see the tournaments in which they were invited to participate as collaborators
R2.10	The system allows educators to see the running or finished battles of their tournament
R2.11	The system allows educators to see the running or finished battles of the tournaments in which they were invited to participate as collaborators
R2.18	The system allows each educator to create a new tournament
R2.22	The system allows each educator to close a tournament
D1	Students and Educators enter the correct credentials during the login phase
D2	Each educator has an institutional email
D3	All users have an internet connection
D8	Each tournament has its own unique ID
D11	If a manual evaluation is planned, it is carried out
D13	All notifications arrive correct and on time to all users

3.2.4. Use case diagrams

3.2.4.1 Student’s use case diagram

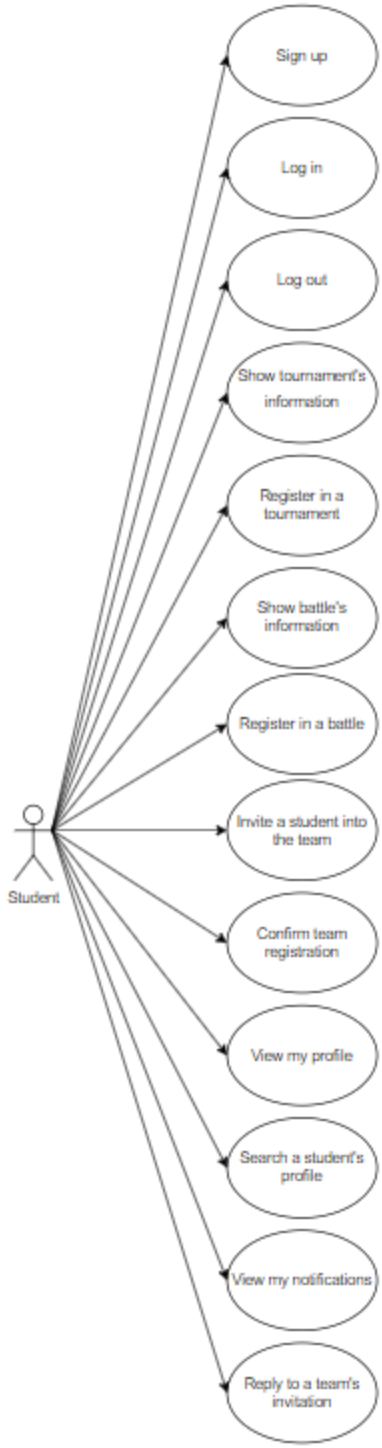

Figure 18: student’s use case diagram

3.2.4.2 Educator’s use case diagram

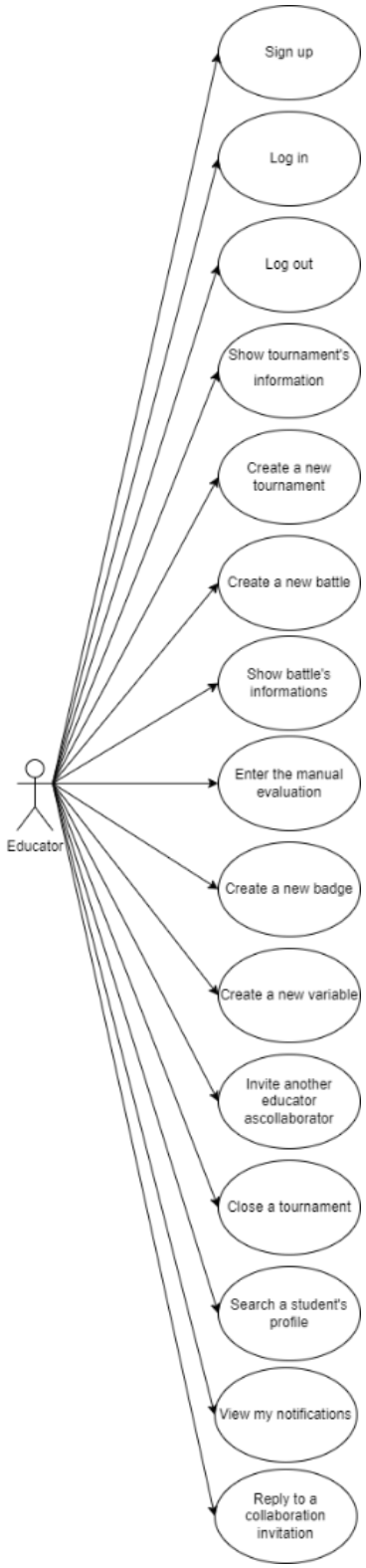

Figure 19: educator’s use case diagram

3.2.5. Students use case analysis

- **Sign up:**

UC	1
Name	Unregistered student signs up
Actor	Unregistered students
Entry conditions	• An unregistered student opens the web page
Input	Personal data (e-mail, password, name, surname, username)
Events flow	• User loads the website's main page (the login page) • User selects the option to signing up • The system shows the user the sign up page • User fills all data required by the registration form • User agrees to the Terms of Services and Privacy Policy • User clicks "Student sign up" • The system sends a confirmation email • The system shows a new page to the user, prompting him to check his email to finalize its subscription • The user, to confirm that he is in possess of the email, clicks the confirmation link received in the mail • The user is redirected to the final confirmation page, confirming he is now registered in the system as a student
Exit conditions	Registration successful, account created and stored in the website's database
Exceptions	• The user provides invalid data • The user doesn't fill all fields • The user clicks "Student sign up" before accepting the Terms of Services • The email is already present in the system • The user doesn't click the link the confirmation email within thirty (30) minutes The exception is notified to the user, and he's returned to the step where the error occurred

- **Log in:**

UC	2
Name	Student logs in
Actor	A registered student

Entry conditions	• The student has previously completed the registration process
Input	E-mail and password
Events flow	• User loads the website's main page (the login page) • User fills the fields with its email and password • User clicks "Log In!" • The system redirects the user to his personal page
Exit conditions	The system brings the students to his personal page
Exceptions	• The user fails to insert a correct combination of e-mail and password • The user fails to fill in any of the fields

- **Log out:**

UC	3
Name	Student logs out
Actor	Logged in student
Entry conditions	• Student has completed the log in procedure
Input	None
Events flow	• Student from his personal page selects the "Log out" option • The system handles the log out procedure, confirms to the user that he completed the log out and redirects him to the Login Page
Exit conditions	The user is now logged out and in the Login Page
Exceptions	• None

- **Show tournament's information:**

UC	4
Name	Show tournament's informations
Actor	Logged in student
Entry conditions	• Student has completed the log in procedure
Input	None
Events flow	• The student looks for the tournament he is interested into by looking through his personal page • Along the tournament's name there is the option "More information on this tournament!" that now the user clicks • The system loads the tournament's info

Exit conditions	The user can visualize all tournaments' data
Exceptions	• None

- **Register in a tournament:**

UC	5
Name	Registration of a student in a tournament
Actor	Logged in student
Entry conditions	• Student has completed the log in procedure
Input	None
Events flow	• The student looks for the tournament he is interested into by looking through his personal page • Along the tournament's name there is the option "More information on this tournament!" that now the user clicks • The system loads the tournament's info like the deadline for the registration • The student clicks on the "Join tournament!" button • The system handles the registration and confirm to the user the successful registration
Exit conditions	The registration is completed
Exceptions	• The deadline to register for the tournament has passed, so the system sends an error message • The Customer inserts on the search bar a nonexistent tournament

- **Show battle's information:**

UC	6
Name	Show battle's information
Actor	Logged in student
Entry conditions	• Student has completed the login procedure
Input	None
Events flow	• Student, from his personal page, selects a tournament he is interested in, looking through the tournaments • Student clicks on "More information on this tournament!", next to the tournament's name • The website loads the tournament's page, with all its information

	• The student searches for the battle he is interested in and clicks on “Show Battle Informations!”, next to the battle’s name • The system loads the battle’s page, with all its information
Exit conditions	The user can see all information about the battle he selected
Exceptions	• None

- **Register in a battle:**

UC	7
Name	Registration in a battle
Actor	Logged in student
Entry conditions	• Student has already Logged in • Student is in the “More information on this tournament!” page of the desired tournament
Input	None
Events flow	• The student looks for the battle he is interested • Along the battle’s name there is the option “Show Battle Informations!” that now the user clicks • The system loads the battle’s info like the deadline for the registration • The student clicks on the 'Register!' button • The system loads the waiting room page for the student • The student assembles their team and sends an invitation • The student clicks on the 'Join the battle!' button • The system confirms to the user that he completed the registration for the battle and redirects him to the Home Page
Exit conditions	The registration is completed
Exceptions	• The deadline to register for the battle has passed and the student clicks on the 'Register!' button, so the system sends an error message • The student clicks on the 'Join the battle!' button and team’s size is not correct • If there is an issue during team invitation, such as a connection or system error, the system provides an appropriate error message

	• In the unique case where the minimum and the maximum number of students in the team are both one, the waiting room will not be used. After the user clicks “Register!”, he can automatically join the battle

- **Invite a student into the team:**

UC	8
Name	Invitation of a student into the team
Actor	• Logged in student
Entry conditions	• Student has already Logged in • Student is in the waiting room for the creation of a team for a battle
Input	Username/e-mail of the students he wants to invite into the team
Events flow	• Student uses the search bar and provides the student’s data. • Student clicks “Invite student!”
Exit conditions	The student can visualize the waiting room where there is information about the status of the invitations
Exceptions	None

- **Confirm team registration:**

UC	9
Name	Confirm team registration
Actor	Logged in student
Entry conditions	• Student has already Logged in • Student is in the waiting room for the creation of a team for a battle
Input	None
Events flow	• The student notices that his team has reached the minimum size for a team in that battle, so he clicks on “username, battle!” button • The system handles the procedure, confirms to the user that he completed the registration and redirects him to the battle page
Exit conditions	The student registered for the battle
Exceptions	• The deadline for registering in the battle has passed

	• The size constraints are not respected

- **View my profile:**

UC	10
Name	Student's looks for his personal profile
Actor	Logged in student
Entry conditions	• The student has completed the login procedure
Input	None
Events flow	• The student, from the Home Page, clicks on the button "My Profile" • The system shows a new page and provides all his personal information, including personal data, badge, his previous rankings
Exit conditions	The student can now visualize his profile and is able to view his data
Exceptions	• None

- **Search a student's profile:**

UC	11
Name	Student looks for a student's profile
Actor	Logged in student
Entry conditions	• The student has completed the login procedure
Input	Student's e-mail and/or username
Events flow	• The student, from his personal page, uses the search bar and provides the student's data. He then presses enter • The system provides the results in a new page, with all possible matches • The user can then open the student's profile by clicking on his username • The system provides all information, related to the website, of the student
Exit conditions	The student can now visualize the data of the other student, like his previous rankings and badges
Exceptions	• None

- **View my notifications:**

UC	12
Name	Student checks his notifications
Actor	Logged in student
Entry conditions	• Student has correctly completed the login procedure
Input	None
Events flow	• The student, from his personal page, selects “Notifications” • The system loads, in a new page, all the user’s notifications
Exit conditions	The student can check his notifications
Exceptions	• None

- **Reply to a team’s invitation:**

UC	13
Name	Student replies to a team’s invitation
Actor	Logged in student
Entry conditions	• Student has correctly completed the login procedure
Input	None
Events flow	• Student, from his personal page, selects “Notifications” • The system loads the student’s notifications • The student reads through the notifications and finds an invitation to participate in a battle of a tournament in which he is registered. Along with the pending invitations there are two options “Accept” or “Decline”. The student clicks the more appropriate option • The system refreshes the page containing the notifications of the student
Exit conditions	The student is viewing his notifications and if he checks an old invitation, he can see how he previously replied to it (there are three options for the text next to the invitation: accepted, rejected or invalid)
Exceptions	• There is a problem with the acceptance of the invitation. The user is notified of what happened and he may be able, based on the error, to retry to accept the invitation following the previously stated procedure

3.2.6 Educator’s use case analysis

- **Sign up:**

UC	1
Name	Unregistered educator signs up
Actor	Unregistered educators
Entry conditions	• An unregistered educator opens the web page
Input	Personal data (e-mail, password, name, surname, username)
Events flow	• User loads the website's main page (the login page) • User selects the option to signing up • The system shows the user the sign up page • User fills all data required by the registration form (an educator also must remember to insert an institutional email) • User agrees to the Terms of Services and Privacy Policy • User clicks "Educator sign up" • The system verifies that the email inserted is institutional, verifies its validity and sends a confirmation email to it • The system shows a new page to the user, prompting him to check his email to finalize its subscription • The user, to confirm that he is in possess of the email, clicks the confirmation link received in the mail • The user is redirected to the final confirmation page by the website, where it is confirmed that he is now registered in the system as an educator
Exit conditions	Registration successful, account created and stored in the website's database
Exceptions	• The user provides invalid data • The user doesn't fill all fields • The user clicks "Educator sign up" before accepting the Terms of Services • The email is already present in the system • The institutional check for the email fails • The user doesn't click the link the confirmation email within thirty (30) minutes • The username provided is not available

- **Log in:**

UC	2
Name	Educator logs in
Actor	A registered educator

Entry conditions	• The educator has previously completed the registration process
Input	E-mail/username and password
Events flow	• User loads the website's main page (the login page) • User fills the fields with its email and password or with its username and password • User clicks "Log In!" • The system redirects the user to his personal page
Exit conditions	The system brings the educator to his personal page
Exceptions	• The user fails to insert a correct combination of e-mail/username and password • The user fails to fill in any of the fields

- **Log out:**

UC	3
Name	Educator logs out
Actor	Logged in educator
Entry conditions	• Educator has completed the log in procedure
Input	None
Events flow	• Educator from his personal page selects the "Log out" option • The system handles the log out procedure, confirms to the user that he completed the log out and redirects him to the Login Page
Exit conditions	The user is now logged out and in the Login Page
Exceptions	None

- **Create a new tournament:**

UC	4
Name	Educator creates a new tournament
Actor	Logged in educator
Entry conditions	• Educator has completed the log in procedure
Input	The new tournament's name, brief description and deadline for subscription
Events flow	• Educator, from his personal page, selects "Create tournament"

	• The system shows the user the page, containing a form to be filled with all data required to create a tournament • The user fills all fields • The user clicks on "Confirm tournament's creation" • The system confirms to the user that the page was created and redirects him to his personal page
Exit conditions	The new tournament is created, and all its data are saved in the database
Exceptions	• The user fails to fill all fields • The user inserts wrong data (e.g. wrong type of input, like numbers instead of text) • There is already a tournament with the same name created by the same educator

- **Show tournament's information:**

UC	5
Name	Show tournament's informations
Actor	Logged in educator
Entry conditions	• Educator has completed the log in procedure
Input	None
Events flow	• The educator looks for the tournament he is interested into either by looking through his personal page • Along the tournament's name there is the option "View tournament's informations" that now the user clicks • The system loads the tournament's info, checking if the logged in educator is part of the admin's team, showing additional options in the case that he is
Exit conditions	The user can visualize all tournaments' data
Exceptions	None

- **Create a new battle:**

UC	6
Name	Educator creates a new battle
Actor	Logged in educator
Entry conditions	• Educator has completed the log in procedure • Educator is part of the admin's team of the tournament he is trying to create a battle for • The tournament he is trying to create a battle for is past the subscription phase and it has not ended
Input	The code kata, the minimum and maximum number of students per group, the registration deadline, the final submission deadline and whether manual evaluation is required
Events flow	• The educator looks for the tournament he is interested into by looking through his personal page in the "My tournaments" tab • Along the tournament's name there is the option "View tournament's informations" that now the user clicks • The system loads the selected tournament, along with all the informations and options available to the admin's team • The educator clicks on "Create battle" • The system shows a new page containing a form with fields related to the new battle • The user fills all fields and clicks "Confirm battle's creation" • The system handles the creation and, after confirming to the user the successful creation, redirects the user to the tournament page
Exit conditions	The battle has been created and saved in the database. The educator is now looking at the tournament's data updated with also the new battle
Exceptions	• The educator isn't part of any admin's team • The educator doesn't fill all fields • The educator inserts wrong data in some field

- **Show battle's information:**

UC	7
Name	Show battle's information
Actor	Logged in educator
Entry conditions	• Educator has completed the login procedure
Input	None
Events flow	• Educator, from his personal page, selects a tournament he is interested in, looking through the tournaments • Educator clicks on "View tournament's informations", next to the tournament's name • The website loads the tournament's page, with all its information • The educator searches for the battle he is interested in and clicks on "View battle's informations", next to the battle's name • The system loads the battle's page, with all its information
Exit conditions	The user can see all information about the battle he selected
Exceptions	None

- **Enter the manual evaluation:**

UC	8
Name	Educator proceeds with the manual evaluation
Actor	Logged in educator
Entry conditions	• Educator has completed the login procedure • Educator is part of the admin's team of the tournament • One battle, created by the logged in educator, has just ended, and it needs manual evaluation
Input	Points awarded, regarding each submission he corrected
Events flow	• The educator, from his personal page, selects the tournament he is interested in looking through the section "My tournaments". He then clicks on "View tournament's informations" • The system loads the tournament's information • The educator now looks through the battles' list for the battle that has ended, then clicking "View battle's informations" next to it • Since the battle has ended and it's a battle requiring manual evaluation, he can see and click on "Assign manual evaluation"

	• There is a list with the names of the groups, alongside a link to the GitHub repository of the group and the automatic evaluation they achieved. After reviewing the content, a group produced, he can add an evaluation in a text field in the line corresponding to the group • When he is done with all the corrections or he wants to take a break, at the end of the page he can select "Save my modifications". If all the evaluations for all groups were provided the battle now enters the "Ended" state and notifies the users about the change of state, otherwise it remains in the "Waiting for evaluation" state.
Exit conditions	The educator can see the current situation about the grading process in the battle's page
Exceptions	• The grades need to be provided in form of a number, letters are unallowed

- **Create a new badge:**

UC	9
Name	Educator creates a new badge
Actor	Logged in educator
Entry conditions	• Educator has completed the login procedure • Educator is creating a new tournament, thus he is in the creation tournament page
Input	Name for the new badge, new rules (*optional*) regarding the badge
Events flow	• The educator clicks on "add badges to the tournament" • The system loads the linking badges page, composed of a form for creating new badges, containing two fields: name and first rule, and a clickable button to go back to the tournament creation page. There is also a list of the system's pre-defined variables and the educator's personal defined variables. To add a new variable, he can click on "Add new variable", discussed in the following use case analysis. • For creating a new badge, the user needs to fill the fields of the form, providing a name and the first rule of the new badge. Upon doing so the system loads a new field for a new rule, in case the user wants to add multiple rules, and a clickable button the user can click

	to confirm the modifications and create a badge. When clicked, the system confirms the creation and reloads the linking badges page
Exit conditions	The user can see the updated linking badges page and, when satisfied with the added badges, can click on the button for going back to the tournament creation page
Exceptions	• The educator, while creating a new badge, fails to fill all fields • The educator, while creating a new badge, fails to insert in the "rule" field a valid rule • The educator, while creating a new badge, fails to insert in the "rule" field a valid variable name

- **Create a new variable:**

UC	10
Name	Educator creates a new variable
Actor	Logged in educator
Entry conditions	• Educator has completed the login procedure • Educator is creating a new tournament • Educator is in the linking badges page
Input	New variable name, code for using the new variable
Events flow	• The educator clicks on "Add new variable" • The system loads the variable creation page, containing a form, with "name" and "lookup rule" as fields • The user fills the fields of the new variable and clicks "Add new variable" • The system confirms the creation, adds the new variable to the user's personal variables and redirects the user to the linking badges page
Exit conditions	The user is in the updated linking badges page with the new variable visible in his personal variables section
Exceptions	• The educator writes an invalid lookup rule • The educator fails to fill in all fields

- **Invite another educator as collaborator:**

UC	11
Name	Educator invites a collaborator
Actor	Logged in educator

Entry conditions	• Educator has completed the login procedure • Educator is one of the admins for the selected tournament
Input	Username and/or e-mail of the educator to be invited
Events flow	• The educator, from his personal page, selects one of the tournaments he is an admin of and clicks on “View tournament’s informations” • The system loads the tournament’s page • The educator clicks on “Add collaborator” • The system loads a new page, where a search bar is present • The educator uses the search bar, using the other educator’s data (username and/or e-mail) to be able to find the other educator • The system shows possible matches for the data • The educator, once found the profile, clicks on “Send invitation” next to the name of the educator
Exit conditions	The system shows a confirmation of the invitation and redirects the educator on the tournament’s page
Exceptions	• The invited educator is already an admin in the tournament

- **Close a tournament:**

UC	12
Name	Educator closes a tournament
Actor	Logged in educator
Entry conditions	• Educator has completed the login procedure • Educator is one of the admins of the tournament
Input	None
Events flow	• The educator, from his personal page, selects the tournament he is interested in looking through the section “My tournaments”. He then clicks on “View tournament’s informations” • The system loads the tournament's page • The educator clicks on “End tournament” • The system confirms the tournament has ended and redirects the educator to his personal page
Exit conditions	The tournament now results “Ended” and users are notified of the final rankings

Exceptions	• The “End tournament” option is present only if the tournament is in the “Active” state

- **Search a student’s profile:**

UC	13
Name	Educator looks for a student’s profile
Actor	Logged in educator
Entry conditions	• The educator has completed the login procedure
Input	Student’s username and/or e-mail
Events flow	• The educator, from his personal page, uses the search bar and provides the student’s data. He then clicks “Search” • The system provides the results in a new page, with all possible matches • The user can then open the student’s profile by clicking on his name • The system provides all information, related to the website, of the student
Exit conditions	The educator can now visualize the data of the student, like his previous rankings and badges
Exceptions	None

- **View my notifications:**

UC	14
Name	Educator checks his notifications
Actor	Logged in educator
Entry conditions	• Educator has correctly completed the login procedure
Input	None
Events flow	• The educator, from his personal page, selects “My notifications” • The system loads, in a new page, all the user’s notifications
Exit conditions	The educator can check his notifications
Exceptions	None

- **Reply to a collaboration invitation:**

UC	15

Name	Educator replies to a collaboration invitation
Actor	Logged in educator
Entry conditions	• Educator has correctly completed the login procedure
Input	None
Events flow	• Educator, from his personal page, selects “My notifications” • The system loads the educator’s notifications • The educator reads through the notifications and finds an invitation to collaborate in the handling of a tournament. Along with the pending invitations there are two options “Accept” or “Reject”. The educator clicks the more appropriate option • The system refreshes the page containing the notifications of the educator
Exit conditions	The educator is viewing his notifications and if he checks an old invitation, he can see how he previously replied to it (there are three options for the text next to the invitation: accepted, rejected or invalid)
Exceptions	• There is a problem with the acceptance of the invitation. The user is notified of what happened and he may be able, based on the error, to retry to accept the invitation following the previously stated procedure

3.2.7. Student's Sequence Diagram

- **Sign up**

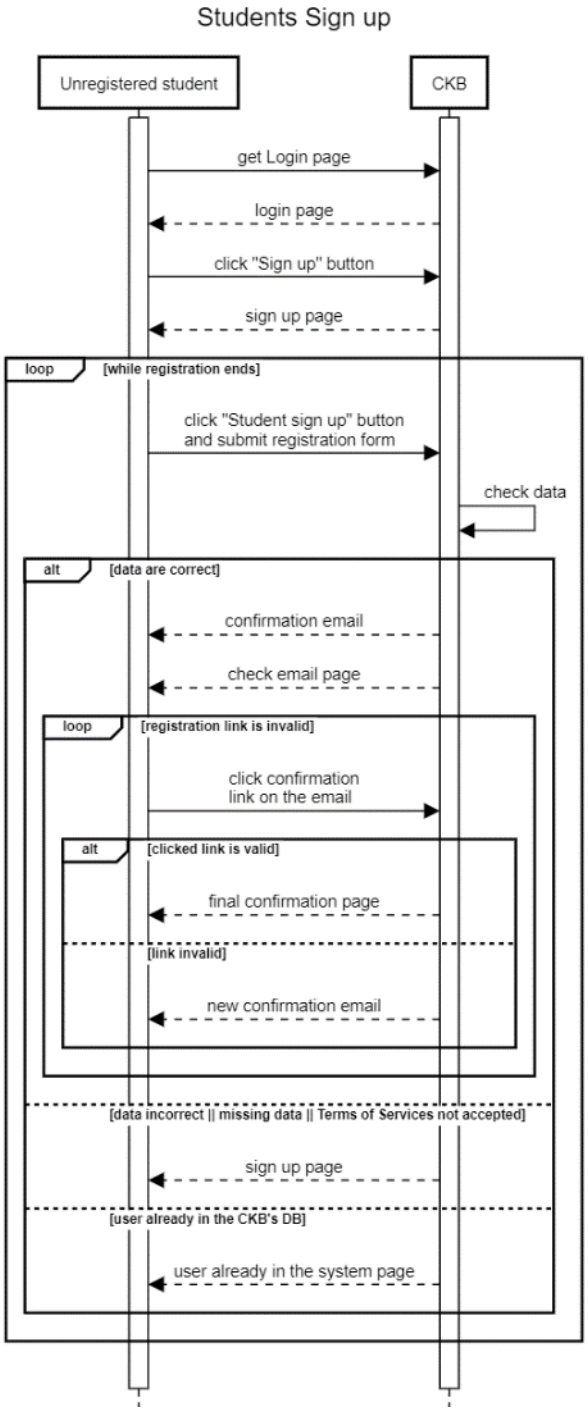

Figure 20: Student Sign Up

- **Log in**

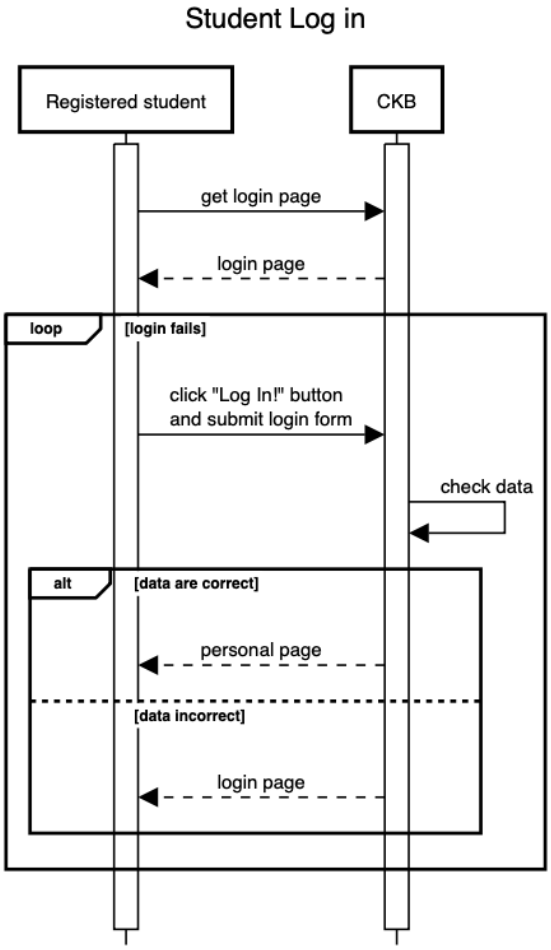

Figure 21: Student Log in

- **Log out**

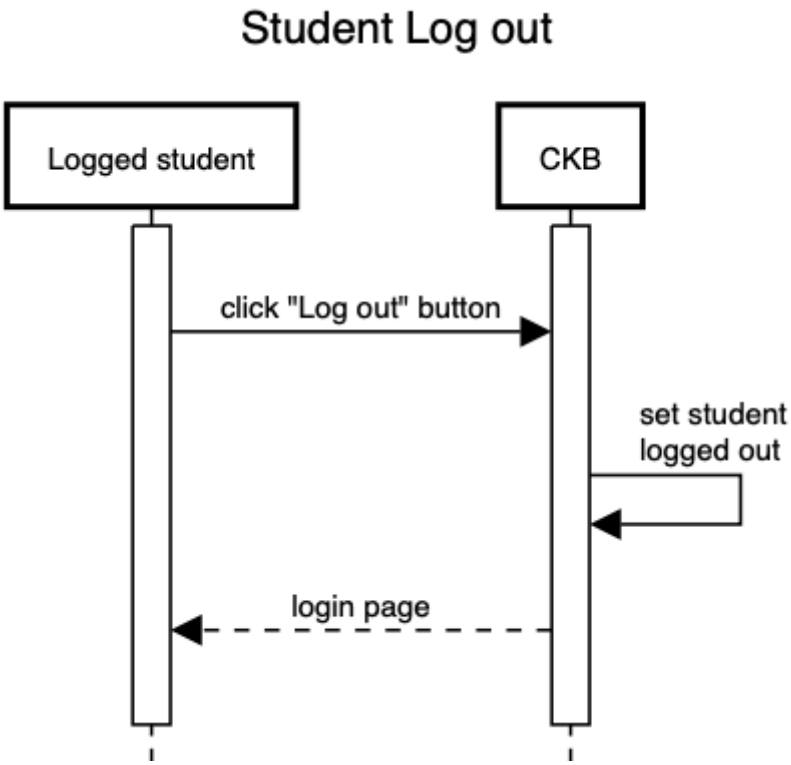

Figure 22: Student Log Out

- **Show tournament information**

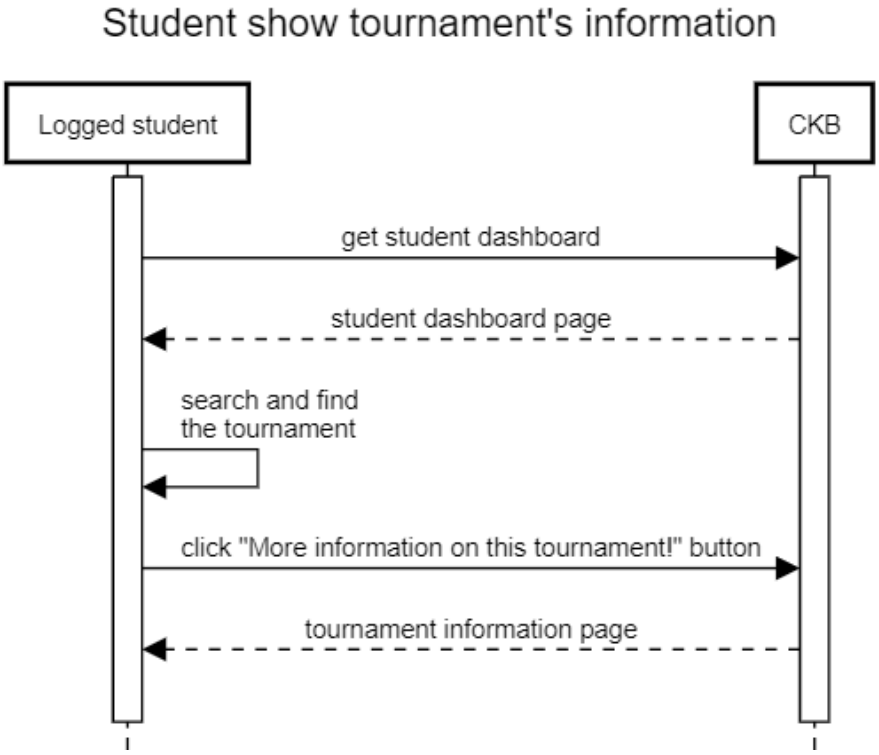

Figure 23: Student tournament information

- **Register in a tournament**

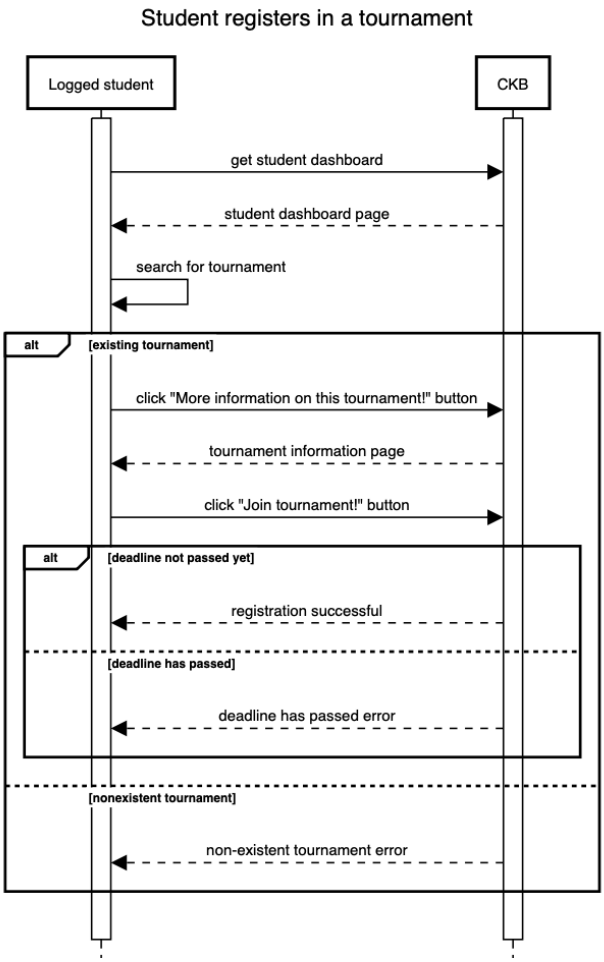

Figure 24: Student tournament registration

- **Show battle information**

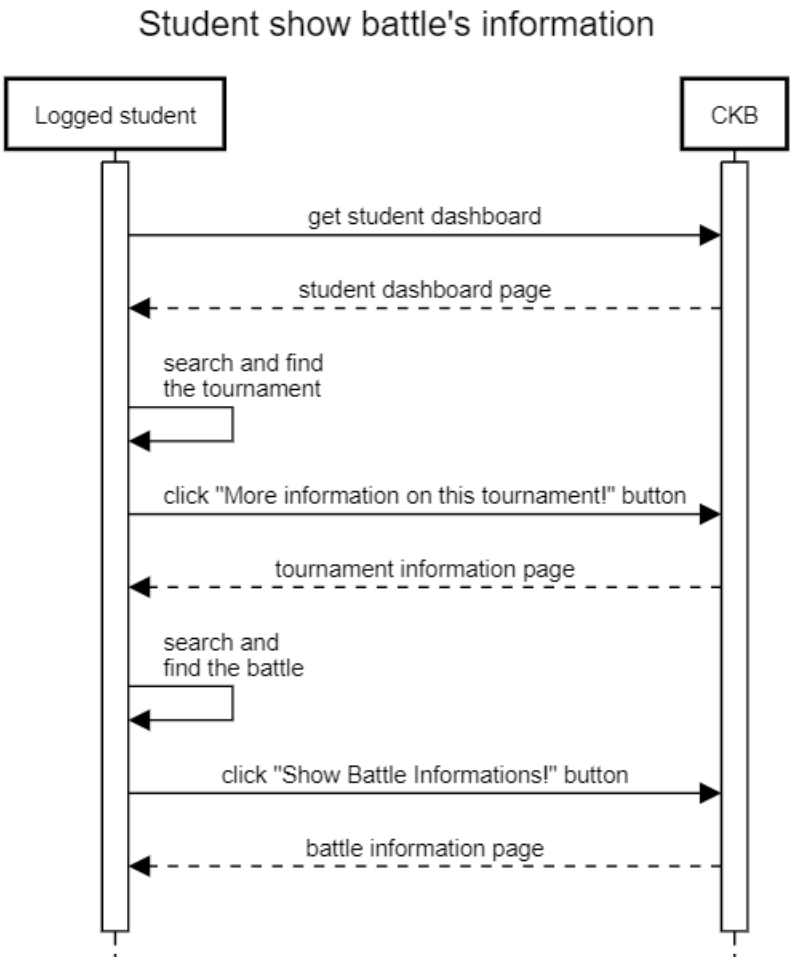

Figure 25: Student battle information

- **Register in a battle**

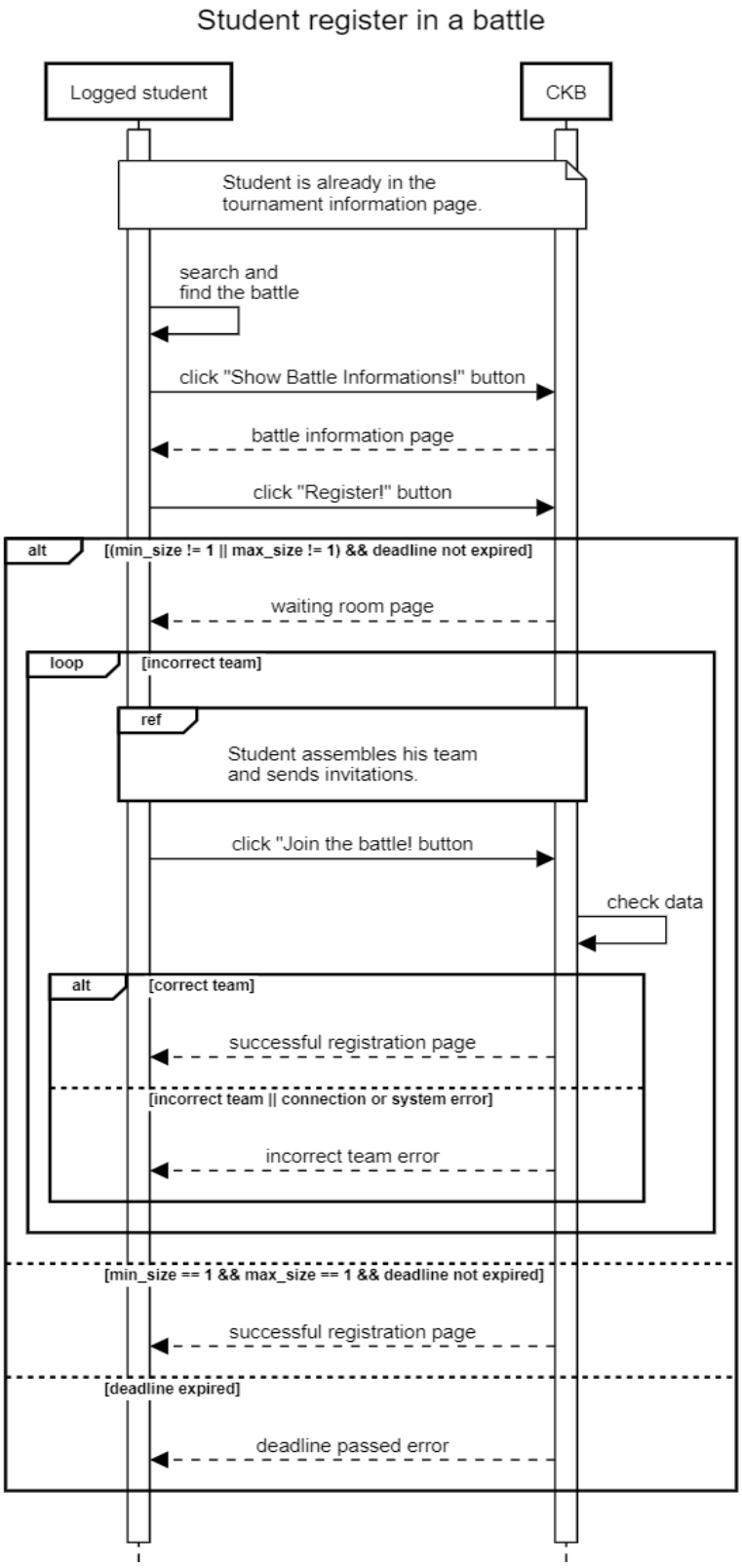

Figure 26: Student battle registration

- **Student invites a student into a team**

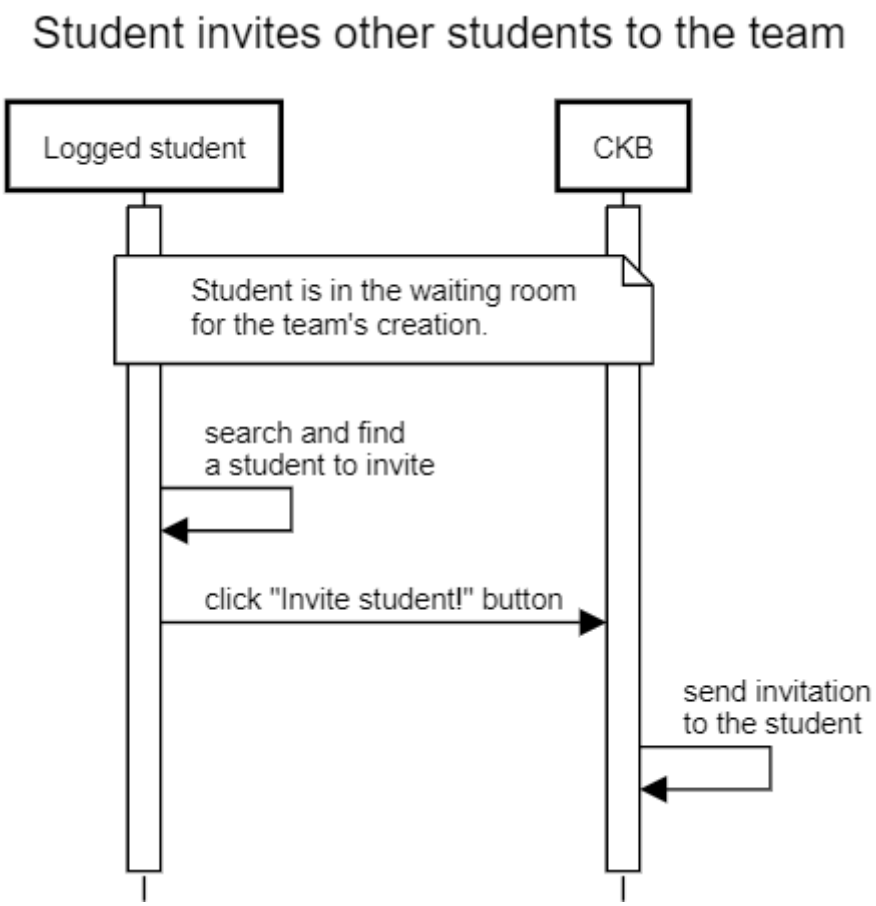

Figure 27: Student invites other students to the team

- **Student confirms a team registration**

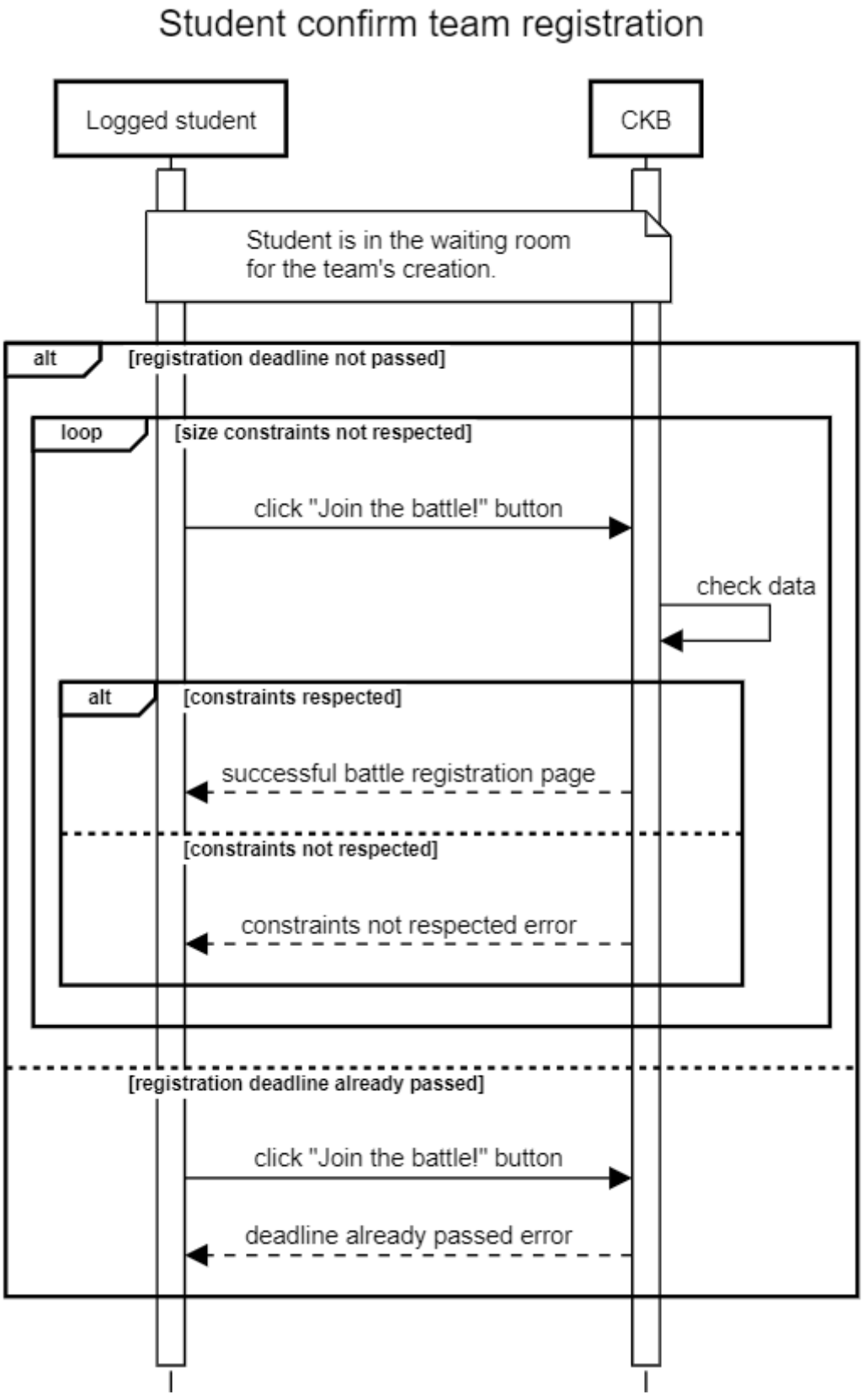

Figure 28: Student confirms team registration

- **Student visits his personal profile**

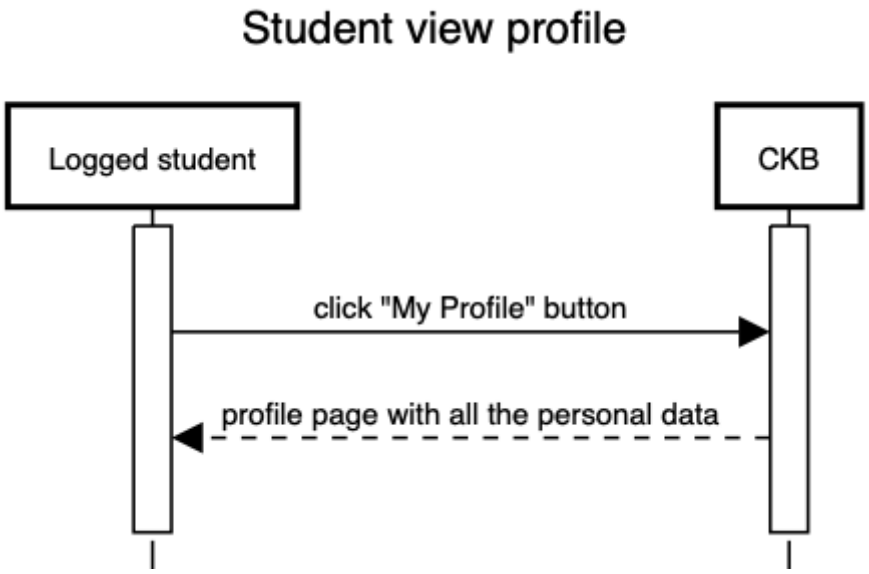

Figure 29: Student personal profile

- **Student searches for another student's profile**

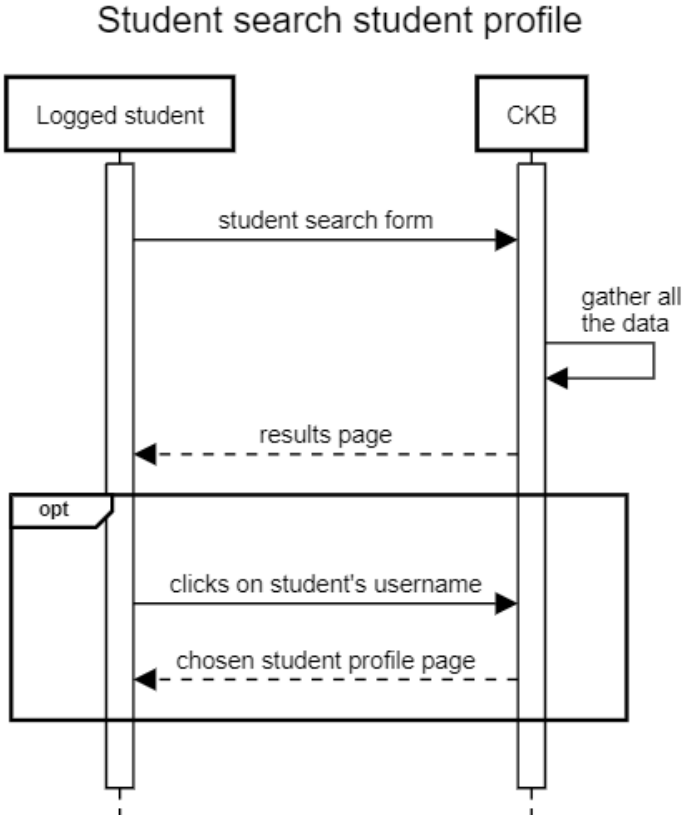

Figure 30: Student search student profile

- **Student visits his notifications page**

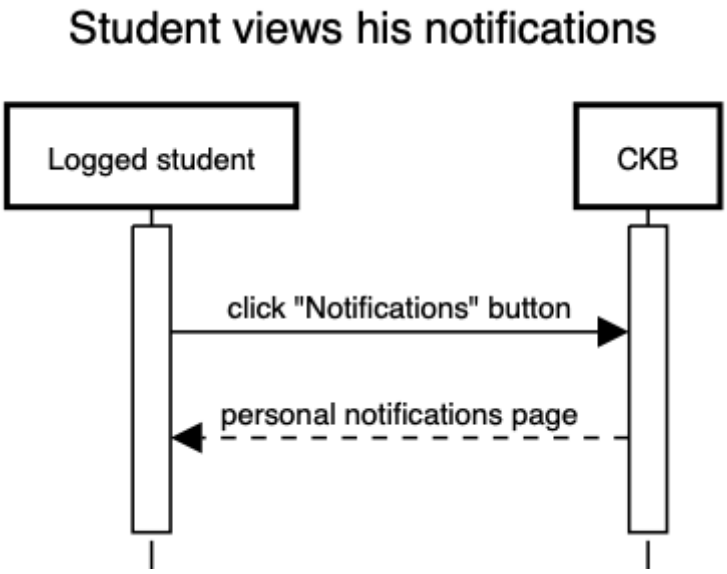

Figure 31: Student notifications

- **Student accepts an invitation to a team**

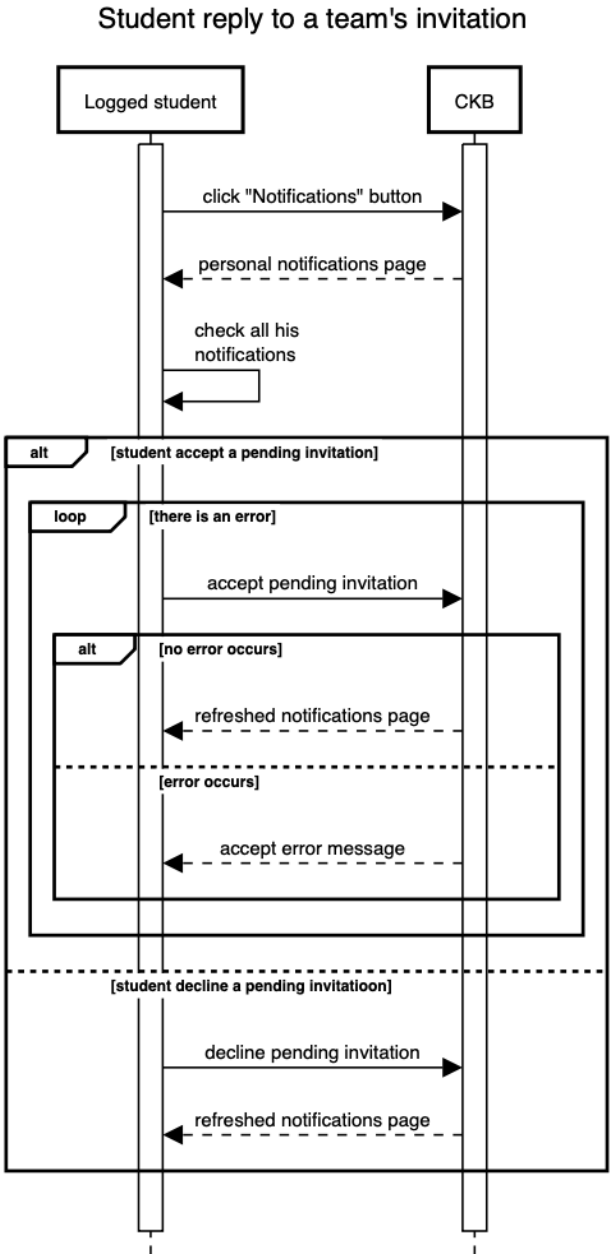

Figure 31: Student accepts invitations

3.2.8. Educator's Sequence Diagram

- **Educator Sign Up**

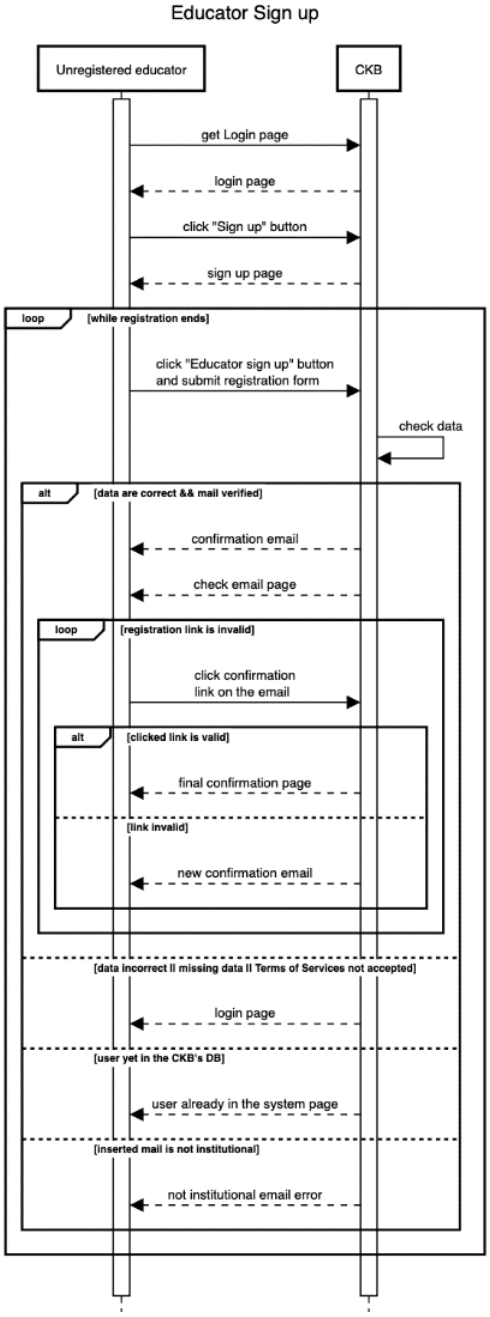

Figure 32: Educator Sign Up

- **Educator Log In**

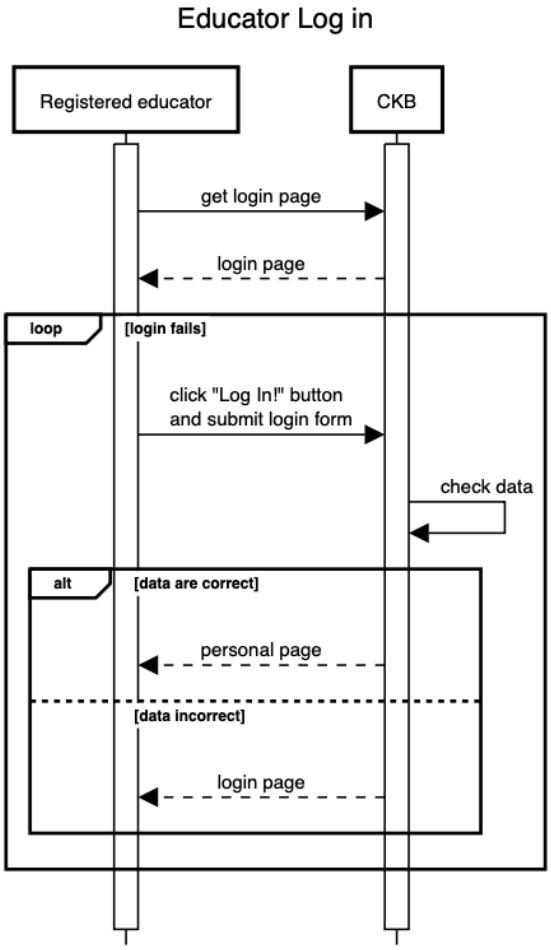

Figure 33: Educator Log In

- **Educator Log out**

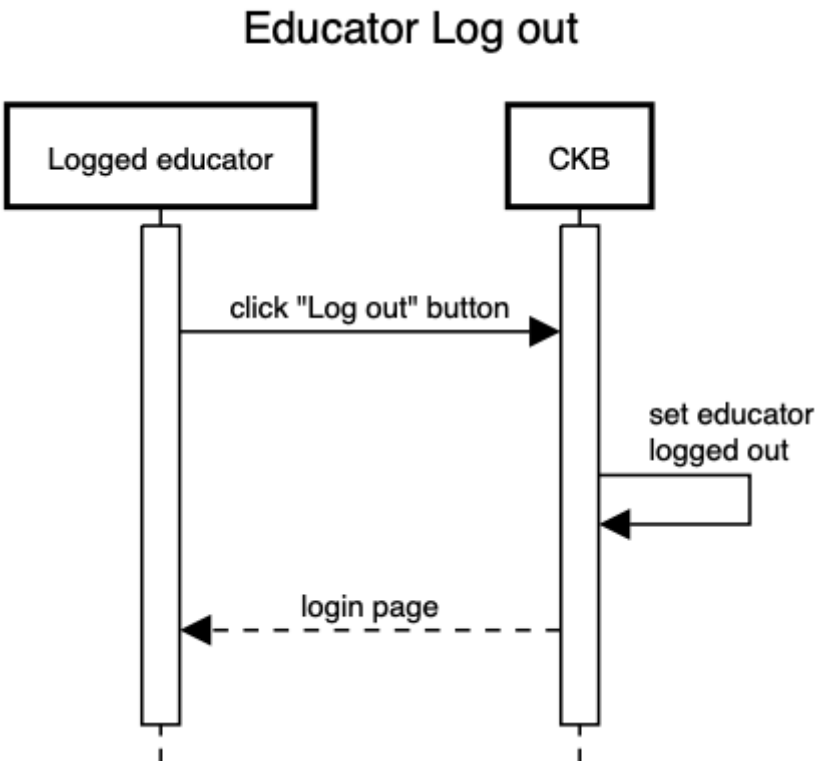

Figure 34: Educator Log Out

- **Educator creates a new tournament**

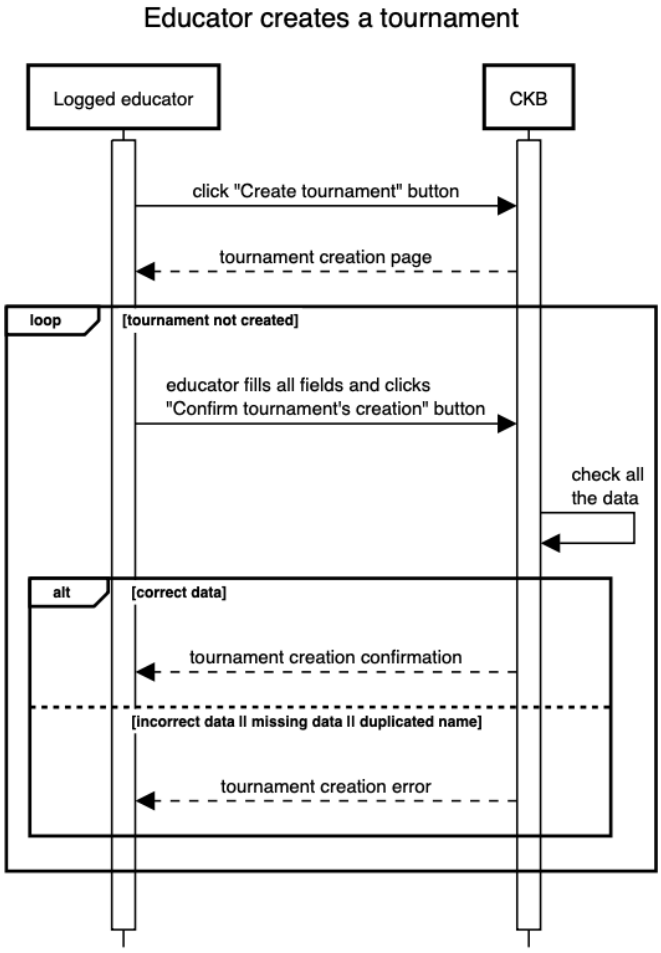

Figure 35: Educator creates tournament

- **Educator looks at tournament’s informations**

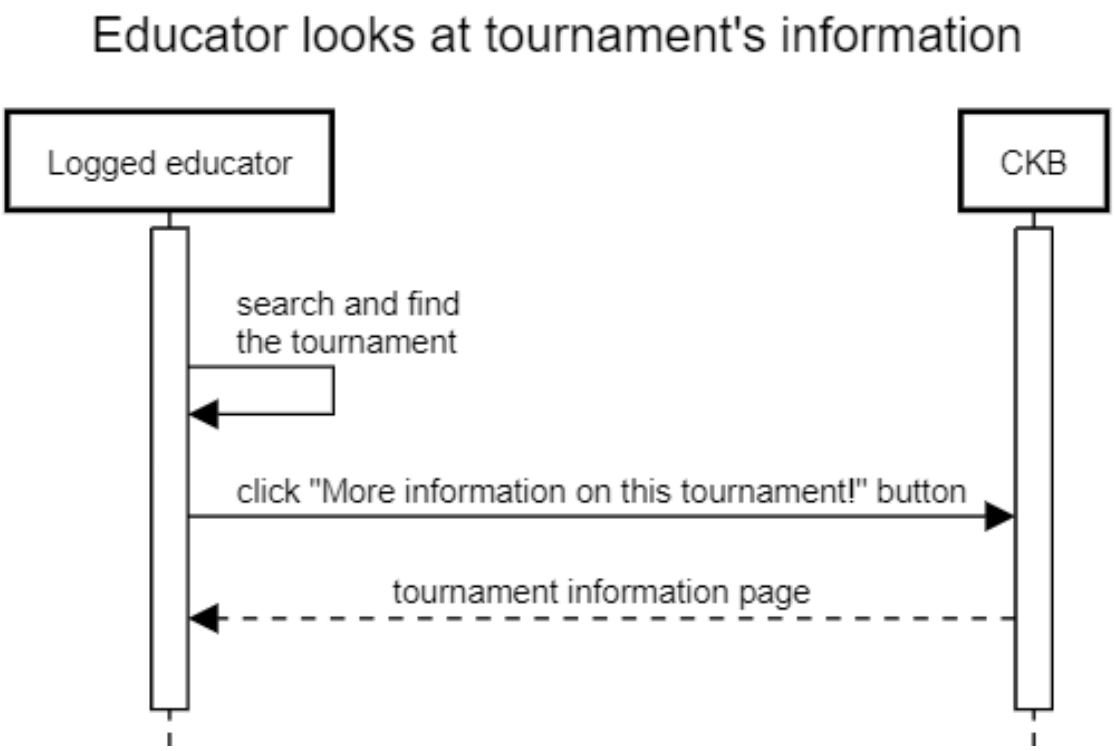

Figure 36: Educator tournament information

- **Educator creates a battle**

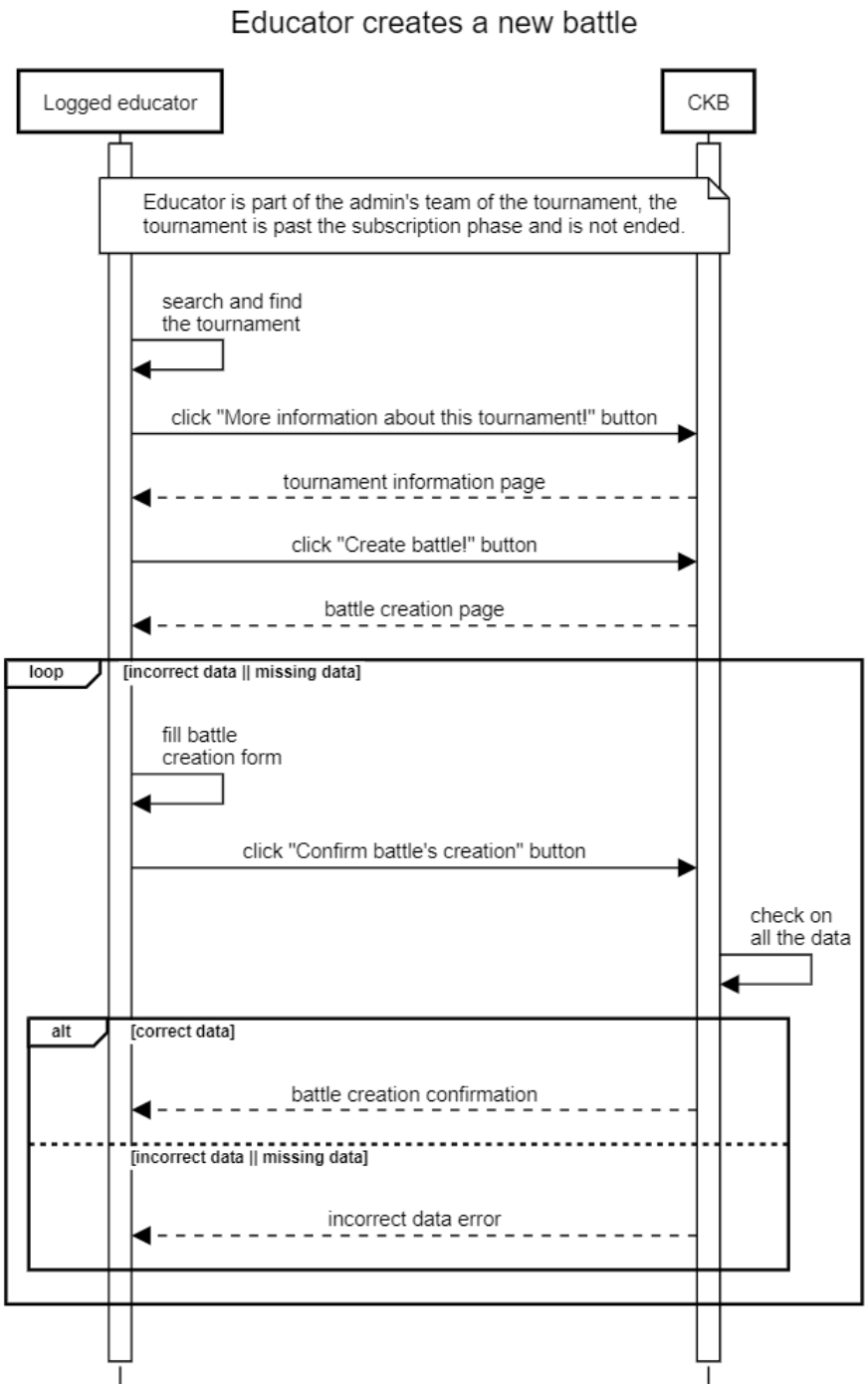

Figure 37: Educator creates a battle

- **Educator looks at battle's information**

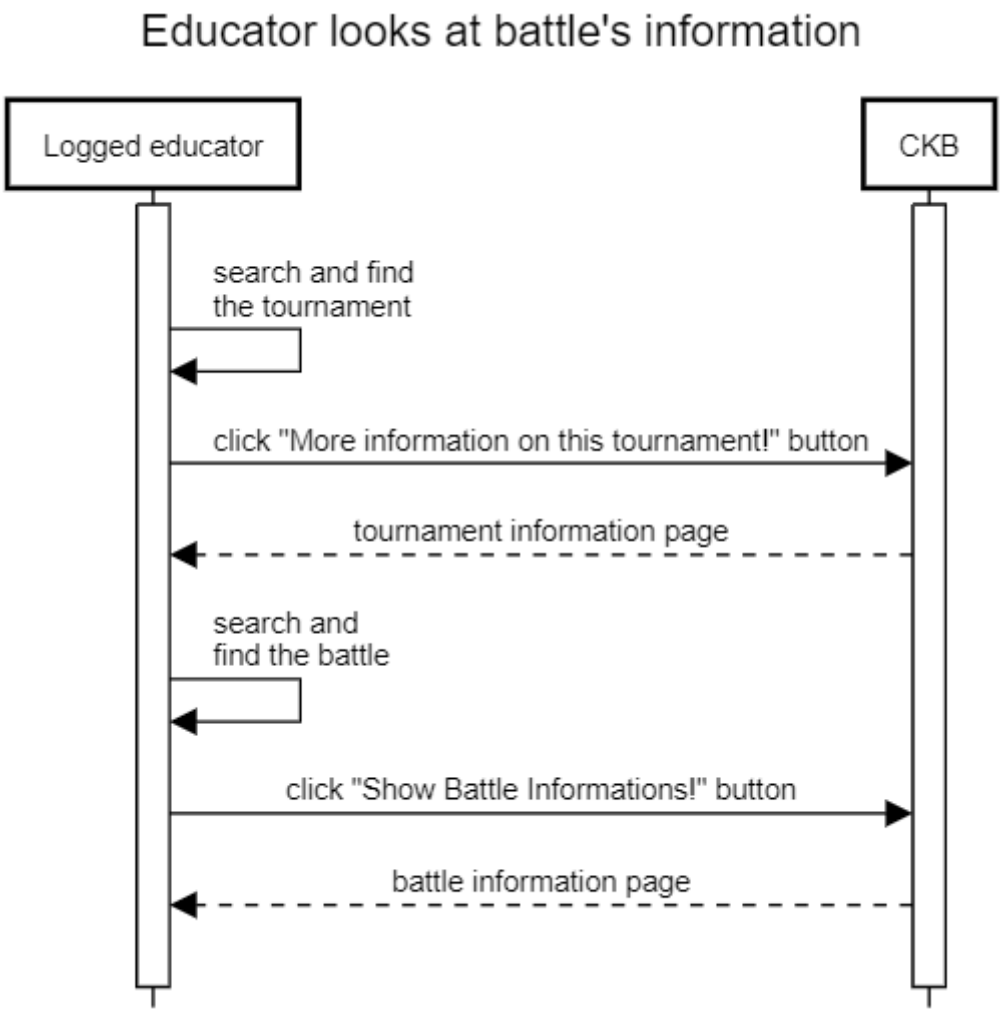

Figure 38: Educator looks battle's information

- **Educator enters a manual evaluation**

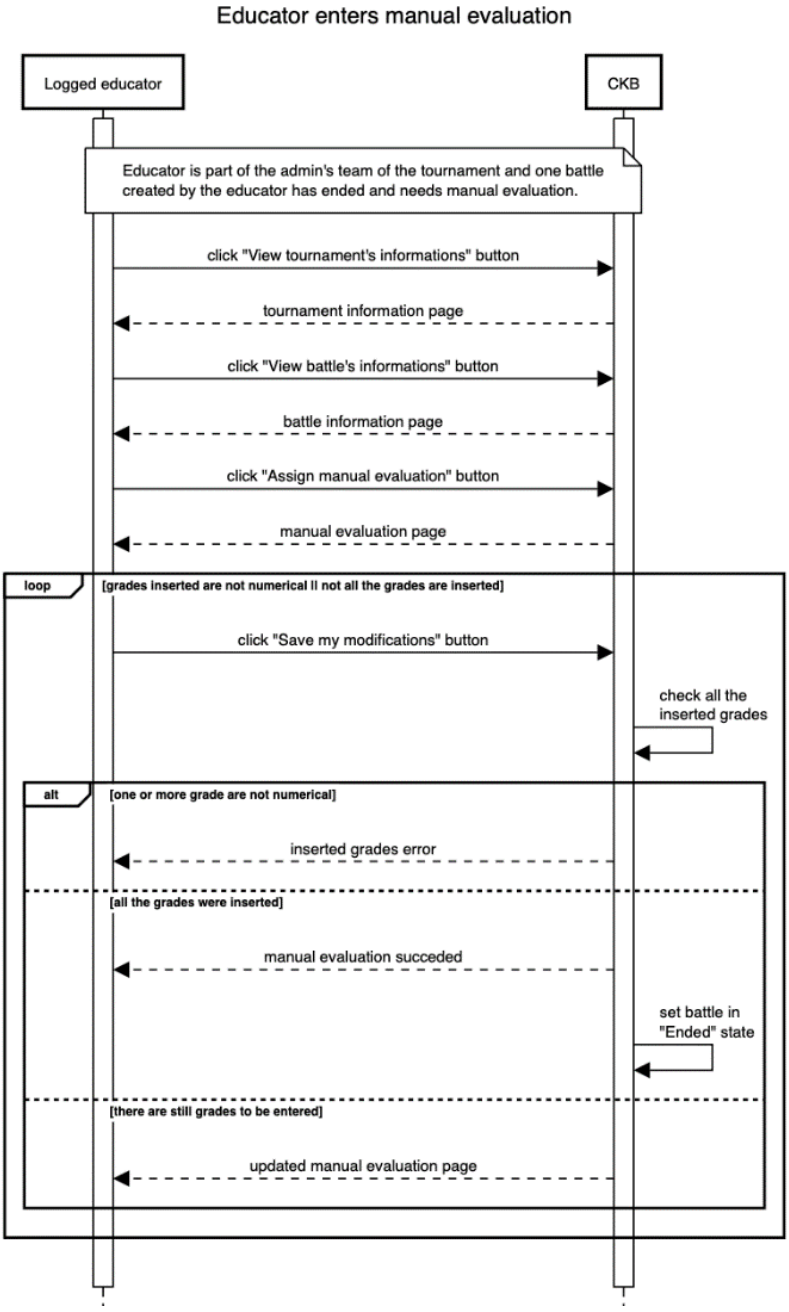

Figure 39: Educator enters a manual evaluation

- **Educator creates a new badge**

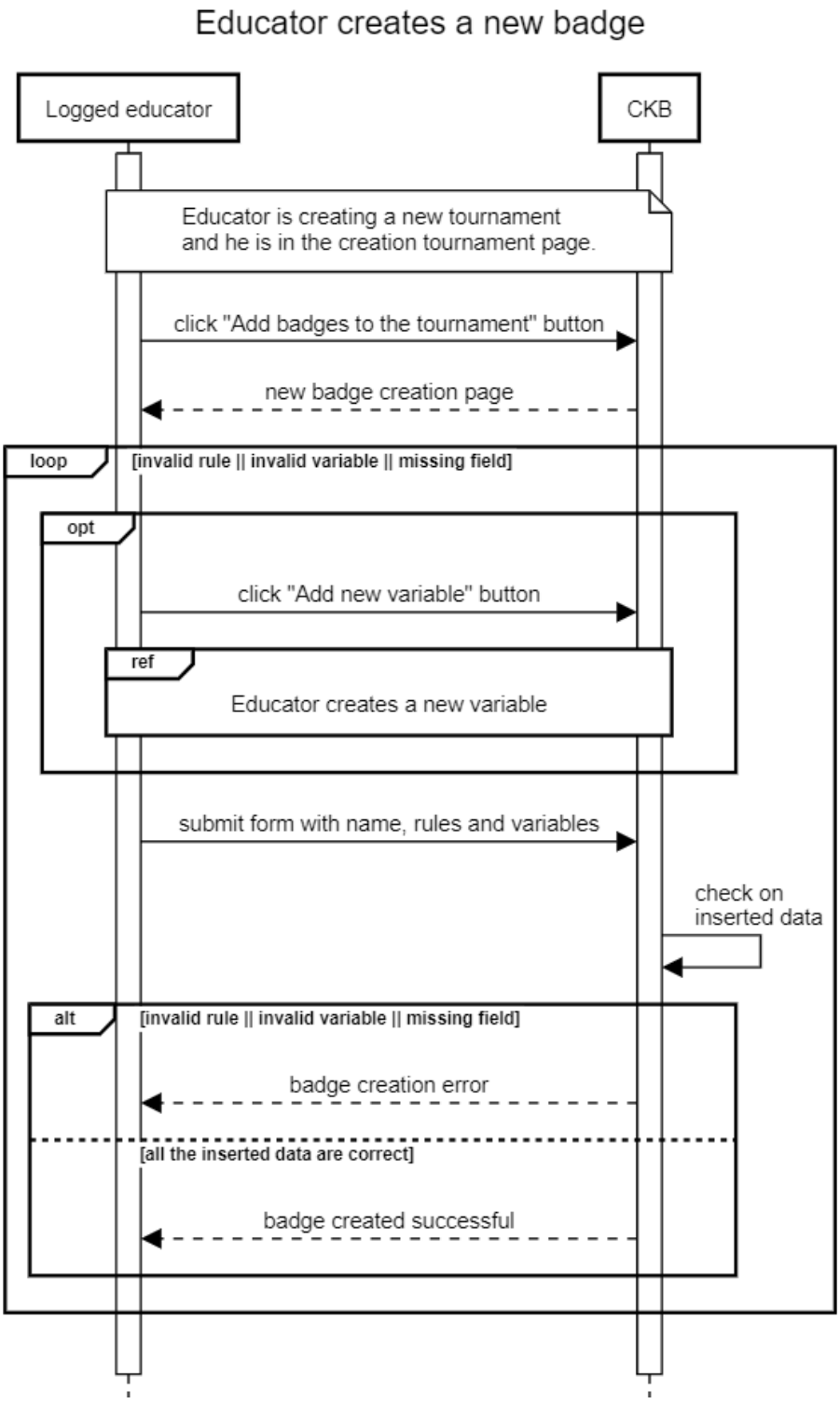

Figure 40: Educator creates a new badge

- **Educator creates a new variable**

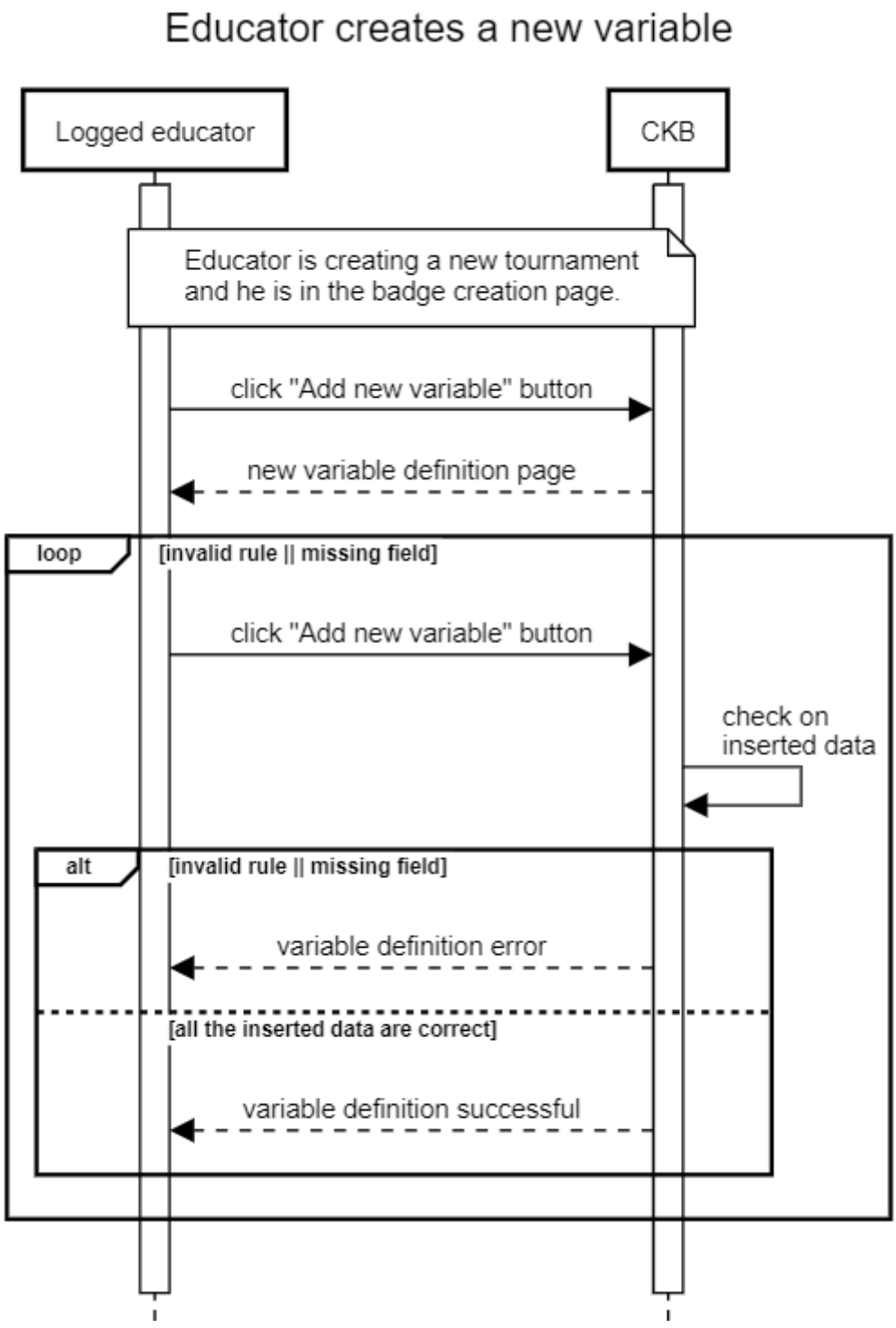

Figure 41: Educator creates a new variable

- **Educator invites other educators as collaborators in a tournament**

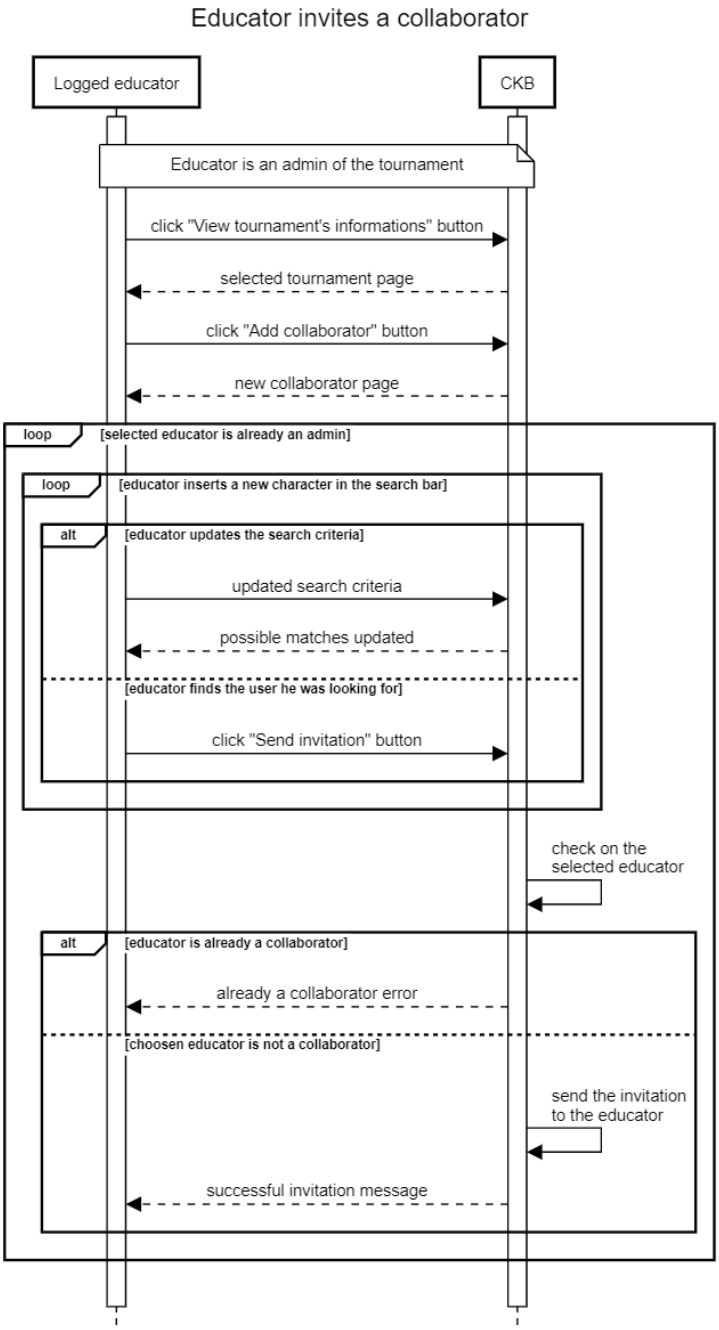

Figure 42: Educator invites other educators

- **Educator closes a tournament**

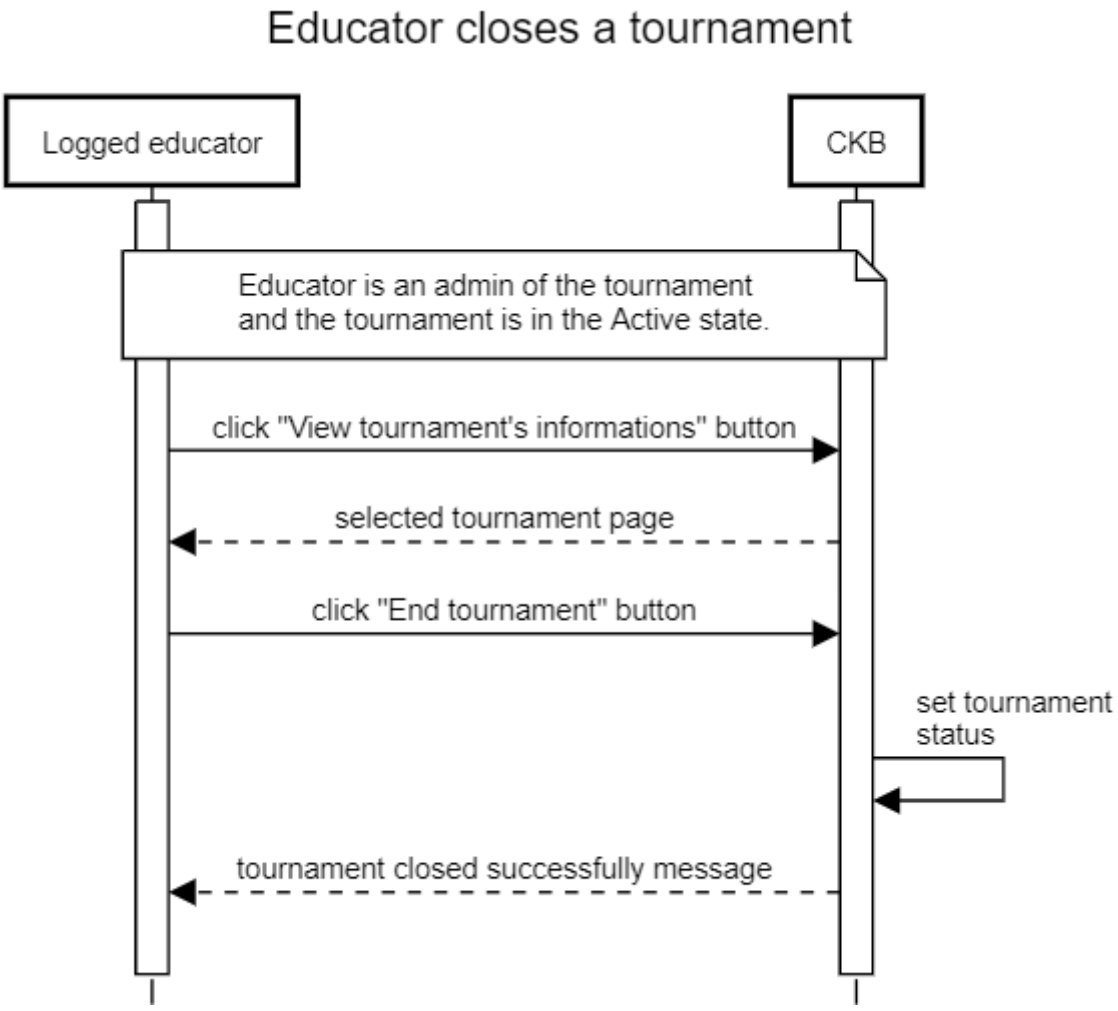

Figure 43: Educator closes a tournament

- **Educator searches a student's profile**

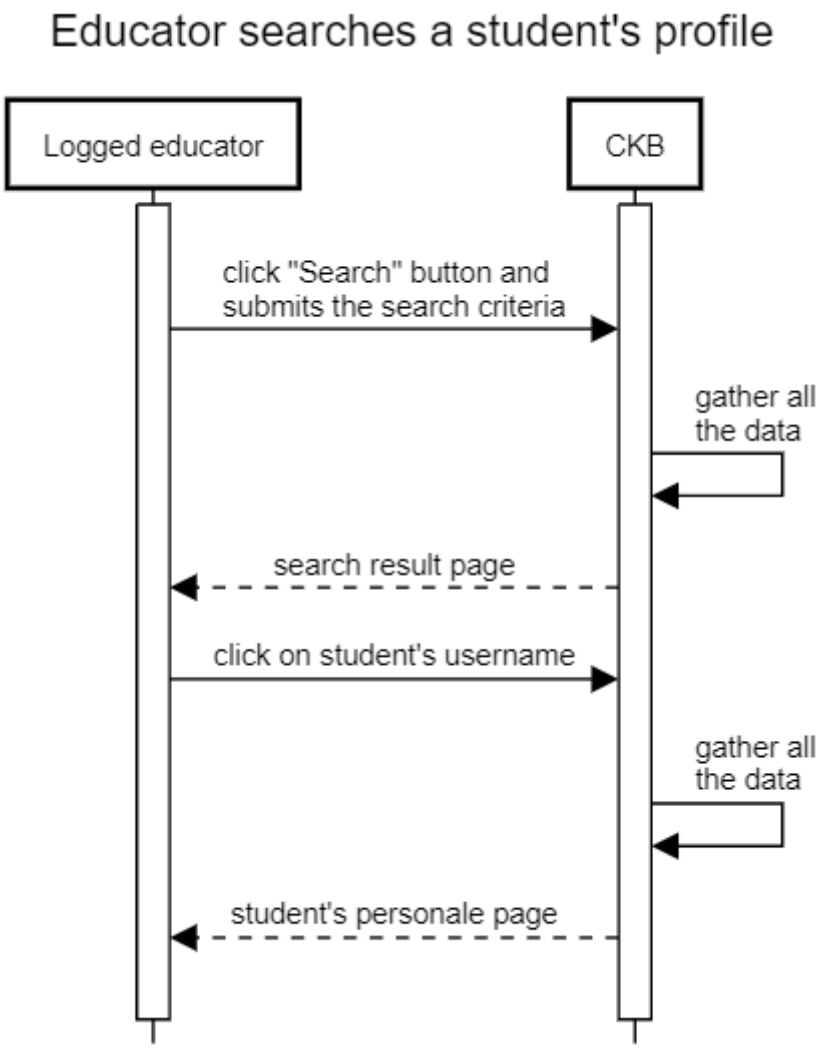

Figure 44: Educator searches a student's profile

- **Educator checks his notifications**

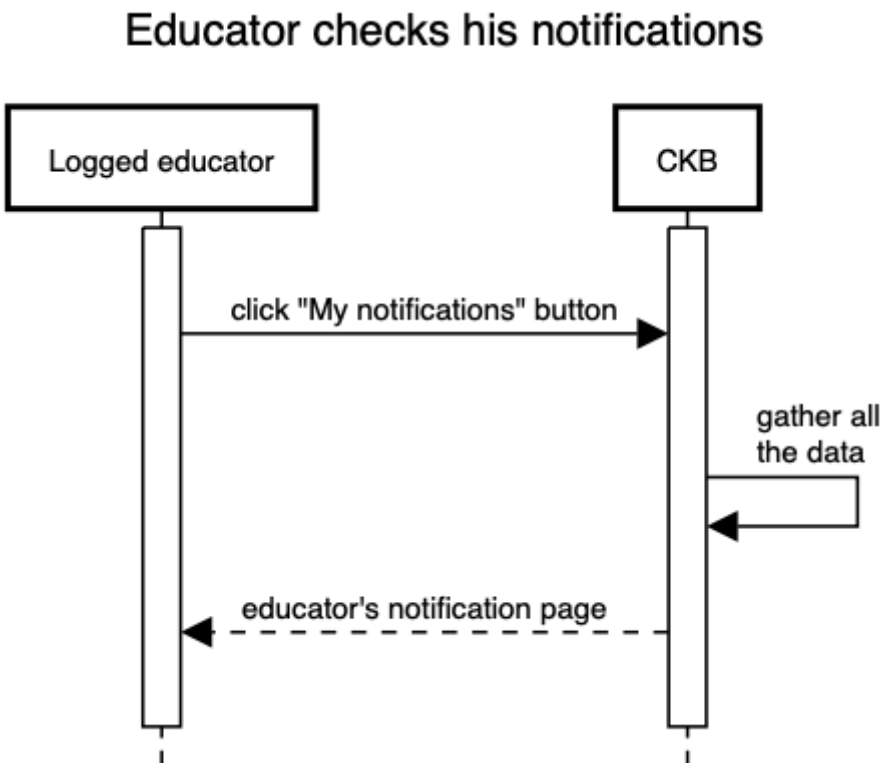

Figure 45: Educator checks his notifications

- **Educator replies to an invitation as collaborator**

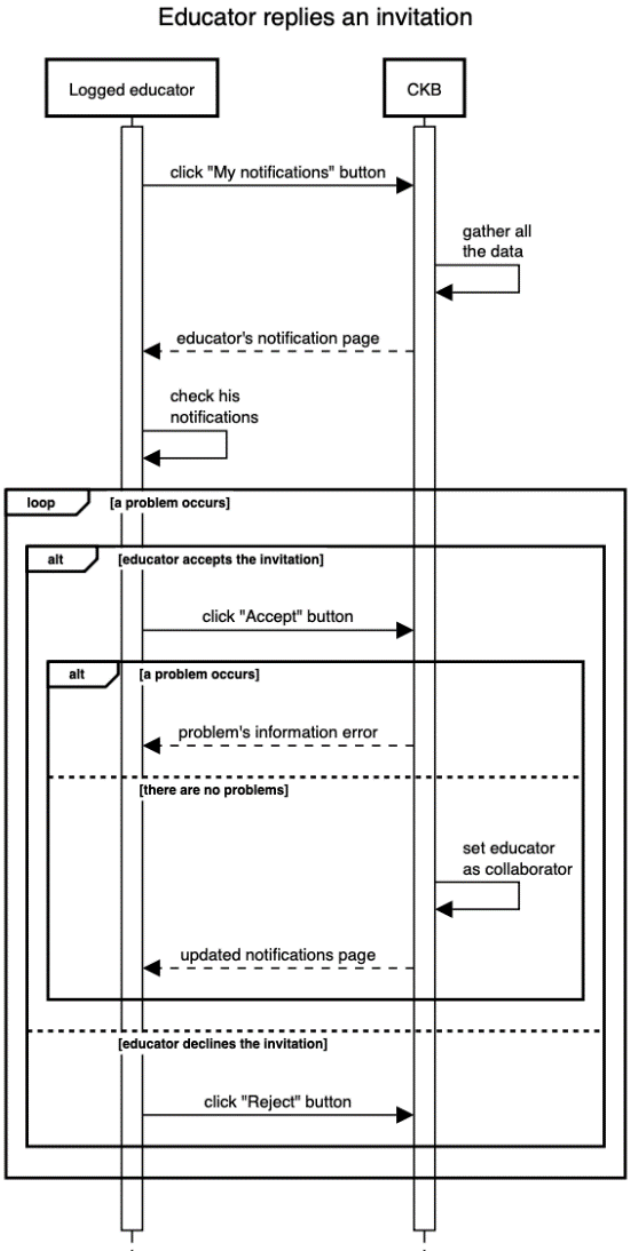

Figure 46: Educator replies to an invitation as collaborator

3.3. Performance requirements

Our expectations for the traffic during the day are around three hundred thousand simultaneous connections from the users. To provide a reliable service and enjoyable experience, the appropriate response time we evaluated and should be guaranteed is 1 second at most for each operation. This allows users to properly navigate through all the web application's available services without any delay, ensuring pleasant navigation.

The application will have to handle all the users' requests and at the same time react to the triggers the GitHub's APIs will send regarding the last students' commits. It is important that, even with this many interactions with the DBMS, the response time should not be affected and so it is vital that the retrieval and storage of data is as efficient as possible. To ensure a response time of 1 second the server must be able to receive up to seven thousand concurrent requests and elaborate them in between 0.45 and 0.7 second.

3.4. Design constraints

All Design Constraints that the system should comply with are listed in this section.

3.4.1. Standards compliance

The system must be GDPR compliant to safeguard all personal data that is stored in the system.

3.4.2. Hardware limitations

To access the web application, it is only needed to run a supported and updated browser. It is mandatory for the hardware used by the users to be able to download and update the preferred user's browser.

3.4.3. Any other constraint

No additional constraints are needed.

3.5. Software System Attributes

3.5.1. Reliability

The system needs to be reliable, delivering the expected results even in case of high traffic. A reliable system needs to be always available and to be fault tolerant, never impacting the user experience. The system must have offline backups of the data storage to exploit in disaster recovery after a data loss.

3.5.2. Availability

Our system isn't linked to emergencies so adequate availability for it is 99,9%. This means that the average time between the occurrence of a fault and service recovery (MTTR) must be contained at around 0.365 days per year. It is important to plan the maintenance when there would be less users connected to the platform, studying the incoming traffic in the previous days.

3.5.3. Security

Security is an essential feature that every application these days can't ignore. It is related to all the features that prevent malicious users from stealing personal data from the users or being able to appear in the system as someone else.

The system needs to grant access only to the users who already own an account and are in possession of a valid combination of email/username and password. After five (5) invalid combinations over the same email/username the account will be temporarily blocked, to prevent brute force attacks.

The data stored in the central database must be protected and it is fundamental for all passwords to be stored cryptographically, never allowing anyone, even in possession of the data, to be able to decrypt it.

3.5.4. Maintainability

Maintainability two crucial aspects regard the choices of implementation and the documentation. While it is fundamental that the documents regarding the application must be precise, analyze all the software's aspects and be kept up to date during each future update of the system, the initial choices for the development deeply influence this feature. Robust and widely supported technologies are preferable, and the system must be developed using modular and scalable principles. A testing routine must be provided, and it must cover at least 80% of the entire codebase.

3.5.5. Portability

The web application should be accessible through all browsers able to support the chosen security protocols.

4 | Formal analysis using Alloy

This section will include a formal modeling of the system using the Alloy tool. Alloy is an open-source language and analyzer for software modeling that allows to describe the system's properties and check for inconsistencies. The following image (Fig. 47) displays the Metamodel, giving a clear view of all components involved in the simulation. Some attributes have been omitted to avoid cluttering while preserving the model's ability to reflect the system's status. For example, all Human agents interacting with the system are represented as Accounts containing only a username and password, omitting their personal information like names and surnames, all the informations about tournaments and battles like descriptions, names and so on are also omitted since they were less useful for the modeling compared to other informations.

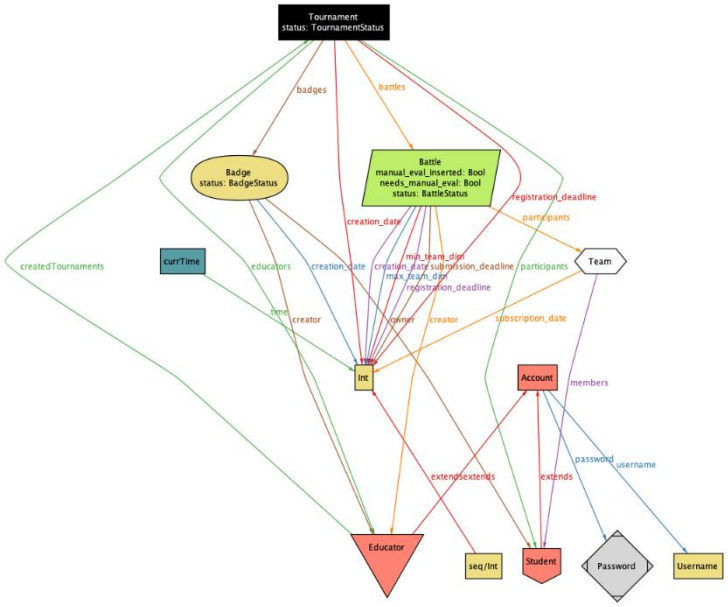

Figure 47: Metamodel

The model will focus primarily on Educators, Tournaments, Battles, Teams and Badges covering the following Goals:

- G1: allow students to participate individually or in teams in programming challenges to increase their skills and earn badges
- G2: allow educators to create new battles
- G3: allow educators to create a new tournament
- G8: allow educators to create new badges and rules for assigning them

- G9: allow educators to close a tournament.

Integers were used to model time and deadlines to avoid the complexity of modeling a full date and time format that would have only made reading the results more confusing.

The following is a list of the main points being analyzed:

- Deadline constraints on Tournaments and Battles with respective statuses
- Assignment of badges
- Managing teams and relative constraints
- Creation, lifecycle and closure of Tournaments and Battles

4.1 Signatures

```
-- All possible Tournament Status
abstract sig TournamentStatus {}
one sig SUBSCRIPTION_TOURNAMENT extends TournamentStatus {}
one sig ACTIVE_TOURNAMENT extends TournamentStatus {}
one sig NON_ENDABLE extends TournamentStatus {}
one sig ENDED_TOURNAMENT extends TournamentStatus {}

-- All possible Battle Status
abstract sig BattleStatus {}
one sig SUBSCRIPTION_BATTLE extends BattleStatus {}
one sig ACTIVE_BATTLE extends BattleStatus {}
one sig CONSOLIDATION_STAGE extends BattleStatus {}
one sig ENDED_BATTLE extends BattleStatus {}

-- All possible Badge Status
abstract sig BadgeStatus {}
one sig CREATED extends BadgeStatus {}
one sig ASSIGNED extends BadgeStatus {}
one sig BADGE_INVALID extends BadgeStatus {}

--Boolean definition
abstract sig Bool {}
```

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

--Simplified Account view: only Username and Password
sig Username {}
sig Password {}

abstract sig Account {
username : one Username,
password : one Password
}

sig Educator extends Account {
createdTournaments : set Tournament
}
sig Student extends Account {}

sig Tournament {
status : one TournamentStatus,
battles : set Battle,
educators : some Educator,
participants : set Student,
creation_date : one Int,
registration_deadline : one Int,
badges : set Badge
} { registration_deadline > creation_date and creation_date >= 0 }

sig Battle {
status : one BattleStatus,
creation_date : one Int,
registration_deadline : one Int,
submission_deadline : one Int,
creator : one Educator,
participants : set Team,
```

```
min_team_dim : one Int,
max_team_dim : one Int,
needs_manual_eval : one Bool,
manual_eval_inserted : one Bool
} { registration_deadline > creation_date and
submission_deadline > registration_deadline and
max_team_dim >= min_team_dim and
min_team_dim > 0 and
creation_date >= 0
}

sig Badge {
status : one BadgeStatus,
creator : one Educator,
owner : lone Student,
creation_date : one Int
} { creation_date >= 0 }

sig Team{
members : some Student,
subscription_date : one Int
} { subscription_date >= 0 }

-- Used to model current time
sig currTime {
time : one Int
} { time > 0 }
```

4.2 Facts

```
-- Account:

-- No duplicate Usernames
fact {no disj a1, a2 : Account | a1.username = a2.username }
```

```
-- All Passwords belong to an Account
fact {all p : Password | one a : Account | a.password = p }

-- Tournament:

-- No two tournaments are equal
fact { no disj t1, t2 : Tournament | t1 = t2 }

-- All battles of one tournament has the creator that is one educator in
that tournament
fact { all b : Battle | all t : Tournament | b in t.battles implies
( one e : Educator | e = b.creator and ( e in t.educators )) }

-- If there is at least one battle not ended the tournament can't be closed
fact { all t : Tournament | ( t.status = NON_ENDABLE ) iff ( one b : Battle
| b in t.battles and
b.status != ENDED_BATTLE ) }

-- One tournament has subscriptions open if and only if the deadline for
subscriptions has not been reached yet
fact { all t : Tournament | t.status = SUBSCRIPTION_TOURNAMENT iff
t.registration_deadline > currTime.time }

-- If a tournament has been closed, it will be always closed
fact { all t : Tournament | always( t.status = ENDED_TOURNAMENT implies
after always t.status = ENDED_TOURNAMENT ) }

-- During the subscription phase a tournament can't have battles
fact { all t : Tournament | t.status = SUBSCRIPTION_TOURNAMENT implies
# t.battles = 0  }
```

```
-- If and only if a tournament is active or ended it has no battle that are not ended
fact { all t : Tournament | ( t.status = ACTIVE_TOURNAMENT or t.status = ENDED_TOURNAMENT ) iff
no b : Battle | b in t.battles and b.status != ENDED_BATTLE }

-- For all tournaments exists only one creator
fact { all t : Tournament | one e : Educator | e in t.educators and t in e.createdTournaments }

-- All the battles in a tournament are created after the tournament creation
fact  { all t : Tournament | all b : Battle | b in t.battles implies b.creation_date >= t.creation_date }

-- There are no tournaments created by two or more educators
fact { no disj e1, e2 : Educator | all t : Tournament | e1 != e2 and t in e1.createdTournaments and t in e2.createdTournaments }

-- A battle can be part only of one tournament
fact { no disj t1, t2 : Tournament | all b : Battle | t1 != t2 and b in t1.battles and b in t2.battles }

-- A tournament will be closed if after the registration deadline there are no students registrated
fact { all t : Tournament | currTime.time >= t.registration_deadline and # t.participants = 0 implies t.status = ENDED_TOURNAMENT }

-- Battle:

-- No two battles are equal
fact { no disj b1, b2 : Battle | b1 = b2 }

-- All battles belong to a tournament
```

```
fact { all b : Battle | one t : Tournament | b in t.battles }

-- One battle has subscriptions open if and only if the deadline for
subscriptions has not been reached yet
fact { all b : Battle | b.status = SUBSCRIPTION_BATTLE iff
b.registration_deadline > currTime.time }

-- A team can't register in a battle after the subscription phase
fact { all b : Battle | all t : Team | t in b.participants implies
t.subscription_date < b.registration_deadline }

-- One battle is active if and only if the deadline for submissions has
not been reached yet
fact { all b : Battle | b.status = ACTIVE_BATTLE iff
b.submission_deadline > currTime.time and currTime.time >=
b.registration_deadline }

-- One battle is in consolidation phase if and only if the deadline for
submissions has been reached and it needs the manual evaluation
fact { all b : Battle | b.status = CONSOLIDATION_STAGE iff
b.submission_deadline <= currTime.time and b.needs_manual_eval = True }

-- One battle is ended if and only if the deadline for submissions has
been reached and it doesn't need the manual evaluation or
--   the manual evaluation is inserted
fact { all b : Battle | b.status = ENDED_BATTLE iff
b.submission_deadline <= currTime.time and ( b.needs_manual_eval = False
or
( b.needs_manual_eval = True and b.manual_eval_inserted = True )) }

-- If a battle has ended, it will be always ended
fact { all b : Battle | always( b.status = ENDED_BATTLE implies
after always b.status = ENDED_BATTLE ) }
```

```
-- All the teams in a battle satisfy the members count constraints
fact { all b : Battle | all t : Team | t in b.participants implies
( # t.members >= b.min_team_dim ) and ( # t.members =< b.max_team_dim ) }

-- If in a battle there are no registered teams, it will be closed
automatically
fact { all b : Battle | ( b.registration_deadline <= currTime.time and #
b.participants = 0 ) implies
b.status = ENDED_BATTLE }

-- If a battle don't need a manual evaluation, a manual evaluation can not
be inserted
fact { all b : Battle | b.needs_manual_eval = False implies
b.manual_eval_inserted = False}

-- There are no battles created before the tournament subscription phase
fact { all t : Tournament | no b : Battle | b in t.battles and
b.creation_date <= t.registration_deadline }

-- Badge:

-- No two badges are equal
fact { no disj b1, b2 : Badge | b1 = b2 }

-- All badges belong to a tournament
fact { all b : Badge | one t : Tournament | b in t.badges }

-- A badge can be created only by the educator that has created the
tournament
fact { all e : Educator | all t : Tournament | all b : Badge | ( t in
e.createdTournaments and
b in t.badges ) iff e = b.creator }
```

```
-- A badge can be created only when a tournament is created
fact { all b : Badge | all t : Tournament | b in t.badges implies
b.creation_date = t.creation_date }

-- If and only if a badge is not already linked to a student, it's CREATED
fact { all b : Badge | # b.owner = 0 iff
( b.status = CREATED or b.status = BADGE_INVALID ) }

-- A badge is assigned if and only if there is a student of the closed
tournament that has been linked to the badge
fact { all b : Badge | all t : Tournament | b.status = ASSIGNED iff
( b in t.badges and t.status = ENDED_TOURNAMENT and  # b.owner = 1 and
b.owner in t.participants ) }

-- If and only if the tournament is closed and there are no battles in it,
the badge is invalid
fact { all b : Badge | all t : Tournament | ( b in t.badges and
t.status = ENDED_TOURNAMENT and # t.battles = 0 ) iff
b.status = BADGE_INVALID }

-- If the badge is assigned it will always be that
fact { all b : Badge | always( b.status = ASSIGNED implies
after always b.status = ASSIGNED ) }

-- If the badge is invalid it will always be that
fact { all b : Badge | always( b.status = BADGE_INVALID implies
after always b.status = BADGE_INVALID ) }

-- If a badge is invalid it can't have an owner
fact { all b : Badge | no s : Student | b.status = BADGE_INVALID and s =
b.owner }
```

```
-- If a tournament is closed and has at least one battle all the badges are valid and must be assigned to a student in the tournament
fact { all t : Tournament | ( t.status = ENDED_TOURNAMENT and # t.battles > 0 ) implies
( all b : Badge | one s : Student | b in t.badges and b.status = ASSIGNED and s = b.owner and s in t.participants ) }

-- Team:

-- No two teams are equal
fact { no disj t1, t2 : Team | t1 = t2 }

-- All the teams are registered at least in one battle
fact { all t : Team | one b : Battle | t in b.participants }

-- All the teams are made up by only students that are registered in the tournament of that battle
fact { all t : Tournament | all b : Battle | all team : Team | all s : Student |
b in t.battles and team in b.participants and s in team.members implies
s in t.participants }

-- If a student is in a team for a battle, it can't be also a member of another team in the same battle
fact { all b : Battle | all s : Student | no disj t1, t2 : Team | t1 != t2
and (t1 in b.participants and t2 in b.participants ) and
( s in t1.members and s in t2.members ) }

-- A team can be registered in a battle only after the battle creation and before the deadline
fact { all b : Battle | all t : Team | t in b.participants implies
( t.subscription_date > b.creation_date and t.subscription_date < b.registration_deadline ) }
```

4.3 Assertions and Predicates

```
-- Check that every active battle are in a not closed tournament and they
are created by an educator that can manage that tournament and that
--    all the teams in that battle are formed by students registered in the
tournament
assert activeBattle {
all b : Battle | ( b.status = ACTIVE_BATTLE implies (( one t : Tournament
| b in t.battles and t.status = NON_ENDABLE and
( one e : Educator | e = b.creator and e in t.educators ) and ( some team
: Team | team in b.participants and
( all s : team.members | s in t.participants )))))
}
check activeBattle

-- Check that all the closed tournament are those with no registrations at
the end of the subscription time and those without active battle
assert closeTournament {
all t : Tournament | ( t.status = ENDED_TOURNAMENT implies (( currTime.time
>= t.registration_deadline and
# t.participants = 0 ) or ( no b : Battle | b in t.battles and b.status !=
ENDED_BATTLE )))
}
check closeTournament

-- Check that in every closed tournament all the badges are invalid or all
the badges are assigned at one student
assert assignedBadges {
all t : Tournament | ( t.status = ENDED_TOURNAMENT  implies
(( all b : t.badges | b.status = BADGE_INVALID ) or
( all b : t.badges | one s : t.participants | b.status = ASSIGNED and s =
b.owner )))
}
check assignedBadges
```

```
-- Check that if a tournament is not endable is because there are active
battles or battles that are waiting for a manual evaluation
--    or battle that are in subscription phase
assert nonClosableTournaments {
all t : Tournament | ( t.status = NON_ENDABLE implies
( some b : Battle | b.status = ACTIVE_BATTLE or ( b.needs_manual_eval =
True and b.manual_eval_inserted = False ) or
b.status = SUBSCRIPTION_BATTLE ))
}
check nonClosableTournaments

-- Check that if a badge has not an owner is because the badge is invalid
or because it's a badge of a tournament that is not already closed
assert notAssignedBadges {
all b : Badge | ( # b.owner = 0 implies
(( b.status = BADGE_INVALID ) or ( one t : Tournament | t.status !=
ENDED_TOURNAMENT and b in t.badges )))
}
check notAssignedBadges

-- PREDICATES

-- Generate a World to show the system's Tournament
pred tournamentWorld {
# Tournament = 1
# Battle = 2
# Badge = 3
# Team = 1
# Student = 1
# Educator = 1
all t : Tournament | t.status = NON_ENDABLE
}
```

```
run tournamentWorld

-- Generate a World to show the system's Battle
pred battleWorld {
# Tournament = 1
# Battle = 3
# Badge = 0
# Team = 3
}
run battleWorld

-- Generate a World to show the system's Educator
pred educatorWorld {
# Battle = 0
# Badge = 0
# Educator = 3
one t : Tournament | all e : Educator | e in t.educators
}
run educatorWorld

-- Generate a World to show the system's Student
pred studentWorld {
# Student = 3
# Battle = 1
# Team = 2
one t : Tournament | all s : Student | s in t.participants
all s : Student | one t : Team | s in t.members
}
run studentWorld for 4

-- Generate a complete Worlds
pred completeWorld {
# Tournament = 1
```

```
# Battle = 3
# Badge = 3
# Student = 3
# Educator = 3
# Team = 2
all s : Student | one t : Team | s in t.members
}
run completeWorld for 6
```

```
10 commands were executed. The results are:
  #1: No counterexample found. activeBattle may be valid.
  #2: No counterexample found. closeTournament may be valid.
  #3: No counterexample found. assignedBadges may be valid.
  #4: No counterexample found. nonClosableTournaments may be valid.
  #5: No counterexample found. notAssignedBadges may be valid.
  #6: Instance found. tournamentWorld is consistent.
  #7: Instance found. battleWorld is consistent.
  #8: Instance found. educatorWorld is consistent.
  #9: Instance found. studentWorld is consistent.
  #10: Instance found. completeWorld is consistent.
```

Figure 48: Results of assertions and predicates

4.4 Worlds

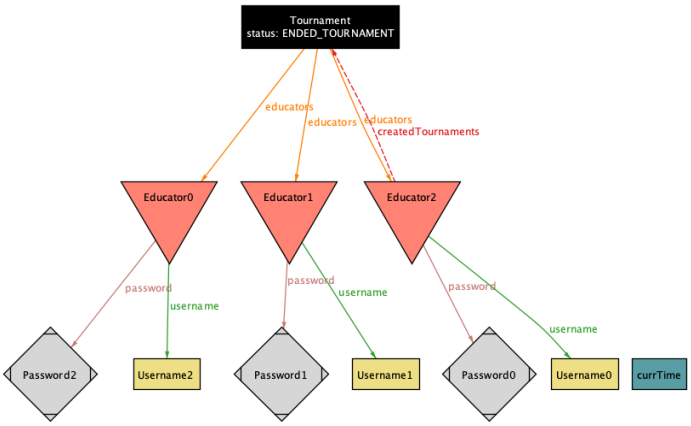

Figure 49: A simple world with Educators related to a Tournament

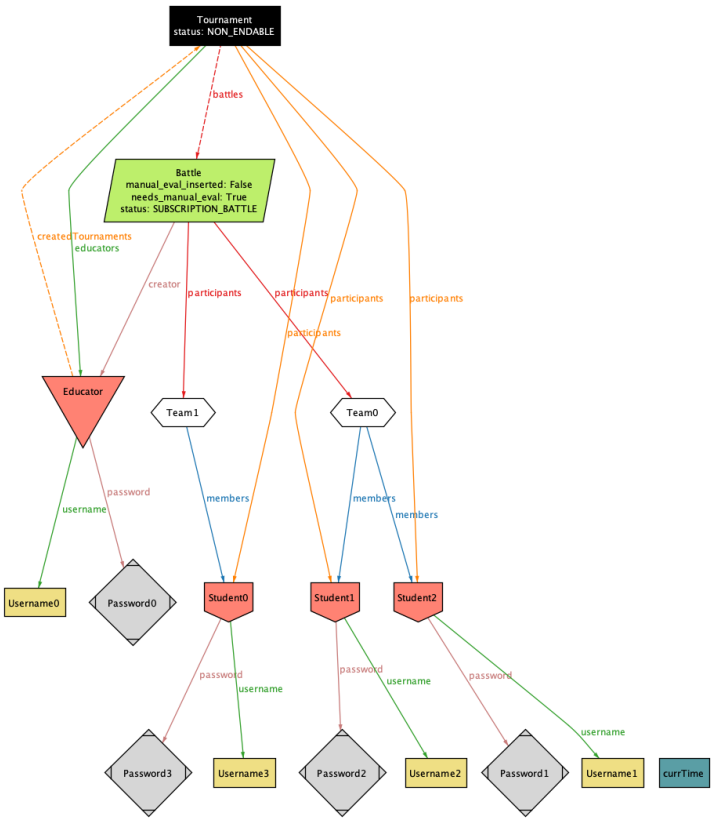

Figure 50: A simple world with Students that participate in teams in a Battle of a Tournament

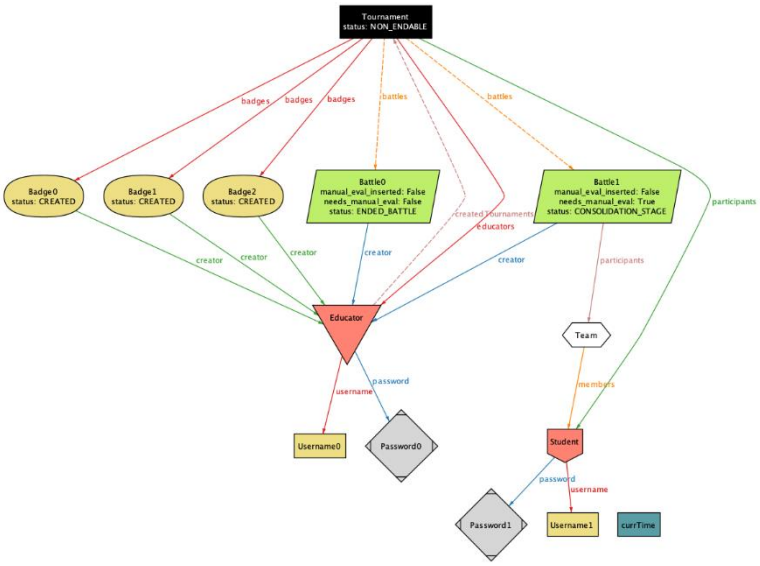

Figure 51: A simple World with a Tournament that contains Battles, Badges, a Team and an Educator

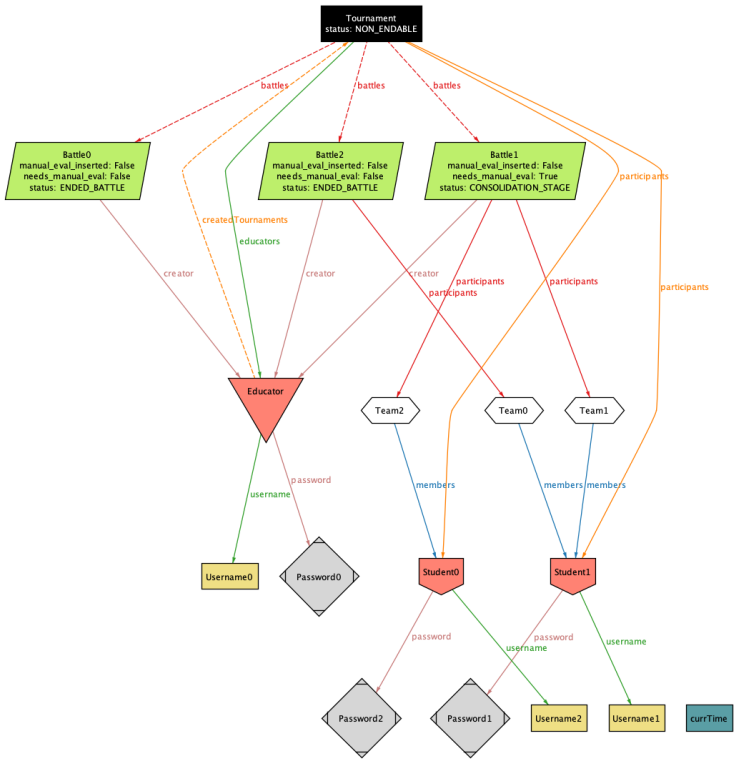

Figure 52: A simple world with a Tournament that contains several Battles in different states

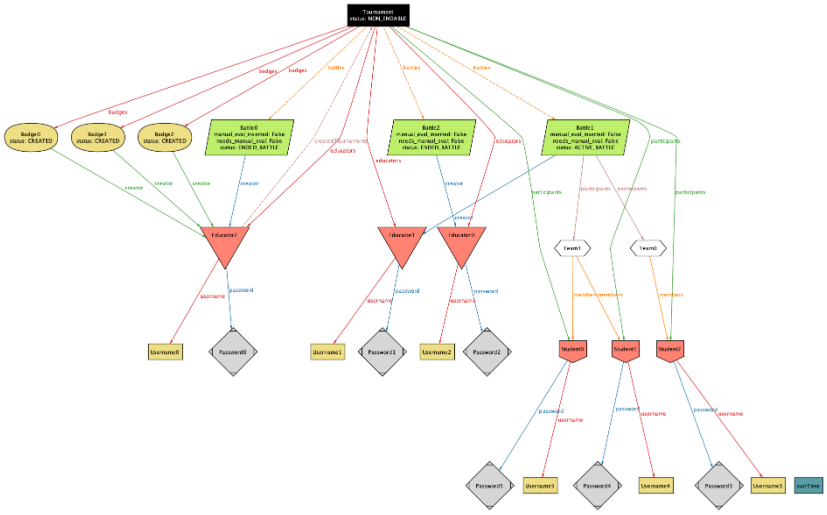

Figure 53: A simple example of a complete World in Alloy modeling

5 | Effort spent

5.1 Cotrone Mariarosaria

Task	Hours
Introduction	3
Goals and phenomena	4
Product Prospective	5.30
User characteristics	0.10
Requirements and Domain Assumptions	4.30
Mapping	4
Alloy Model	11.30
Use Cases Tables	5.30
Use Cases Diagrams	3
Document revision	6

5.2 De Ciechi Samuele

Task	Hours
Goals and phenomena	2
Requirements and Domain Assumptions	3
Alloy Model	11.30
Use Cases Tables	2.30
Document revision	5.30
Use Case Analysis	6.30
State Diagrams	3
Specific Requirements section	5.30
Class Diagram	0.30
Product functions	4

5.3 Deidier Simone

Task	Hours
Goals and phenomena	2
Requirements and Domain Assumptions	2.30
Alloy Model	16.30
Use Cases Tables	2.30
Use Cases Diagrams	8
Product Functions	0.45
Mock-ups	13
Alloy Revision	1.45
Document revision	4