CodeKataBattle

Software Engineering 2 Project - RASD v2.0

Mariotti Marco
Personal code: 10939419

7 January 2024

Contents

1 Introduction **2**
1.1 Purpose . . . 2
1.2 Scope . . . 2
1.3 Definitions, Acronyms, Abbreviations . . . 3
1.4 Revision history . . . 3
1.5 Reference Documents . . . 4
1.6 Document structure . . . 4

2 Overall Description **5**
2.1 Product perspective . . . 5
2.2 Product Functions . . . 9
2.3 User Characteristics . . . 12
2.4 Assumptions, Dependencies and Constraints . . . 12

3 Specific Requirements **13**
3.1 External Interface Requirements . . . 13
3.1.1 User Interfaces . . . 13
3.1.2 Hardware Interfaces . . . 13
3.1.3 Software Interfaces . . . 13
3.2 Functional Requirements . . . 13
3.3 Performance Requirements . . . 42
3.4 Design Constraints . . . 42
3.4.1 Standards Compliance . . . 42
3.4.2 Hardware Limitations . . . 42
3.5 Software System Attributes . . . 42
3.5.1 Reliability . . . 42
3.5.2 Availability . . . 42
3.5.3 Security . . . 42
3.5.4 Maintainability . . . 43
3.5.5 Portability . . . 43

4 Formal Analysis Using Alloy **43**

5 Effort Spent **48**

6 References **48**

1 Introduction

1.1 Purpose

Today, as the demand for the professional role of programmer is constantly increasing, so is the demand for applications that allow trained programmers as well as hobbyists to grow and hone their skills. Because of this need, *CodeKataBattle* accommodates a growing trend in software development education, where hands-on experience and teamwork play an important role in preparing students for real-world challenges.

The platform operates as a bridge between educators and students, allowing educators to test students from all over the world. The scope of these challenges is train future programmers using a test-first approach, and in particular training them with peers using code kata, which are small programming exercises for which test cases are already provided; in this way, the students can develop succesful routines to solve problems in a specific language without having to worry abbout testing.
CodeKataBattle encourages iterative development processes,teaching the importance of continuous improvement to meet all the requests the educator makes. Instead of more traditional methods, like lectures or online courses, *CodeKataBattle* aims to provide a modern learning experience; in this way students not only reinforce their coding skills but also improve critical aspects such as teamwork, time management, and communication, fundamental skills in many workplaces and vital for success in the dynamic field of software development.

The platform provides users with a user-friendly interface; in particular, the educators can create and manage code kata battles within specific tournaments, that students can participate in, alone or in teams. Educators can upload code katas, set submission deadlines, configure scoring parameters, and give permissions to colleagues to create other battles. Also, they can reward the students with badges, defining specific rules to achieve them, adding a competitive gamification element to the experience. The team score is update every time a team's member push a new commits to their *GitHub* repository (that is created automatically at the beginning of every battle). In this way, both students and educators can monitor the progress of all the teams during the battle. The tournament rank, that provide insights into individual performance of all the students involved in the tournament, is generated by *CodeKataBattle* summing the battle scores obtained from every battle the student participated in.

The goals of the application are the following:

G1: Allow students to partecipate in coding tournaments and battles.

G2: Allow students to deliver their solution to the platform.

G3: Allow educators to evaluate team's solution.

G4: Allow students and educators to monitor teams rank during a battle.

G5: Allow students and educatros to monitor tournament score of each student.

G6: Allow educators to create and manage tournaments and battles.

1.2 Scope

In this section, we analyse the world phenomena and the shared phenomena, according to the "World and the Machine" paradigm introduced by M.Jackson and P.Zave. In this context, the Machine is identified with the application to be developed and the world as the real-world environment in which it will be used.

World Phenomena

W1: A student wants to join a tournament

W2: A student wants to join a battle

W3: An educator wants to create a new tournament

W4: An educator wants to create a new battle

Shared Phenomena Shared phenomena are divided according to the controller of the phenomena.

Phenomena controlled by the world:

W1: A student wants to invite other students in a team

W2: An educator wants to grant to other educators the permission to create battles within a specific tournament

W3: A student set up automated workflow through GitHub Actions

W4: A student push a project

W5: An educator wants to manage settings and deadlines for a battle

W6: An educator wants to create a new badge

Phenomena controlled by the machine:

W7: A student wants to track their progress in a tournament rank

W8: Educators and students want to check teams rank during a battle

W9: Educators and students want to check active tournaments

W10: Educators and students want to check collected badges

W11: The app notifies all the students of the creation of a new tournament

W12: The app notifies all the students in a tournament of the creation of a new battle

W13: The app creates a GitHub repository and send the link to all the team's members

W14: The application assigns badges according to their rules

1.3 Definitions, Acronyms, Abbreviations

Definitions:

- **CodeKataBattle:** The software to be developed.
- **Student:** The participant in one or more tournaments and battles
- **Educator:** The creator of one or more tournaments and battles
- **Tournament:** A competition composed of multiple battles
- **Battle:** A contest of multiple teams to create a project that satisfies all the tests

Acronyms:

- **CKB:** CodeKataBattle

1.4 Revision history

- v1.0 : First version of the document released on December 22, 2023.
- v2.0 : Make minor adjustments to the document's formatting and address typing errors. Enhance the functional requirements section by incorporating additional details. Expand the product function section by subdividing existing subsections into more specific cases and providing more comprehensive descriptions for some of these cases.

1.5 Reference Documents

- The specification document ”Assignment RDD AY 2023-2024”.

1.6 Document structure

This document is composed of the chapters described below.

1. The first chapter contains a brief introduction to the problem with the goals of the project. It also contains analysis of world and shared phenomena, followed by a description of the document as a whole.
2. The second chapter contains a description of the system to be. Here are presented scenarios, a class diagram of the system, some statecharts and a list of domain assumptions and dependencies made for the project.
3. The third chapter contains a description of the requirements of the system. Here, use cases are presented with their diagrams. Then, functional and non-functional requirements are presented and mapped on goals.
4. The fourth chapter contains a formal analysis made with Alloy.
5. The fifth chapter contains the description of the effort spent by the group members.
6. The sixth chapter contains the eventual used references.

2 Overall Description

2.1 Product perspective

In the first part of this section, we will analyse some scenarios involving the usage of CKB.

- **User Registration on CKB:** Jamie, a computer science student, decides to check out the CodeKataBattle application under the recommendation of one of his professors, to improve his programming skills for one of his university courses. Through his laptop, he opens the CKB homepage and after reading the description on the CKB website, decides to sign up to the platform. He cicks on the "sign up" button on the homepage and he is redirected to the sign up form, through which he inserts all required personal data, like name, surname, email, a username and password. After linking his GitHub profile, he successfully created an account on the CKB platform.
- **Educator Creates a New Tournament:** John, a computer science professor, logs into the CKB application because he has decided to create a challenging code kata tournament for the students of his class as part of the final mark for his course. He clicks on "Create new tournament" banner, enters the required data and customizes all aspects of the tournaments according to his needs, like the tournament description and subscription deadlines. Lastly, John decide to give permission to other collegues to create new battles for the partecipants.
- **Educator Creates a New Battle:** Sarah, a computer science professor, decides to create a new battle for one of the tournaments she manages. To do so, she opens the CKB appliation, chooses one of the tournaments and clicks on the "create a battle" button. Because of the wide range of customization the application offers, she is able to organize and set the battle in the best possible way to accomodate her students' learning needs. After customizing the battle description and deadlines, she uploads the Code Kata including its test cases and build automation scripts. Then, she is able to customize the parameters used for evaluation and computation of the final score. Other than configuring the standard timeliness and functionality parameters, Sarah is also able to choose which aspects will be analyzed for the evaluation of the quality parameter, and which weight they will have on the final score automatically computed by the system.
- **Student Joins a Code Kata Battle:** Kelly logs into the CKB application after receiving a notification for the creation of a new battle in the tournamet in which she is involved. After reading the battle description and all related details, Kelly decides to form a team to participate in this battle. In the team creation section, Kelly selects the number of teammates she want for the team (ensuring that respect the minimum and maximum value for the battle) and sends an invitation to some of her friends, waiting for thir response. In the meantime, she also receives a request by another user to join her team , which she decides to decline. After her team is complete, her team officially joins the battle.
- **Student Submits Code Solution:** Jordan, a prticipant of a Code Kata battle, after finishing writing and modifying the code for his project locally on his IDE of choice, decides to push his work on the team's GitHub repository, eager to know the result. This action triggers an automated analysis of the code (considering factors such as functionality, timeliness, and code quality as set by the educator which created the battle). After the analisys is finished, the team's score is dynamically updated and dispayed on the rank of the battle on the platform.
- **User Visualizes Real-Time Battle Rank:** Taylor logs into the CKB application to visualize the position of her team in the battle they are involved. She navigates to the real-time battle rank section and the rank with all the teams's scores appears, providing a clear rappresentation of the situation of the team in the battle and allowing her to plan how to manage their work accordingly. Scrolling the leaderbord she finds out that they are in the middle of the list, so they need to put much more effort in the competition to obtain a good score.
- **User Receives Final Battle Rank Notification:** Mark is impatient to know the final result

of the battle he has participated in the past weeks. Some days after the battle concludes, Mark receives a notification from CKB application. Navigating in the website, on the tab of the tournament, Mark visualize the final rank, understanding the level of his coding skills and that he still has a lot to improve to reach the top.

- **Student Accumulates Tournament Scores:** Pat logs into CKB to visualize his personal score in the tournament he is involved. Pat navigates to the tournament scores section, visualizing his and other partecipants scores; in this way, Pat can see clearly his situation in the tournament and have a a clear view of all the other partecipants.
- **Educator creates a badge:** Morgan, an educator using CKB to challenge their students, decides to make the challenge more fun and competitive through the gamification feature offered by the platform. During the tournament creation phase, Morgan decides to create some badges associated to the tournament, which will be assigned to students who fullfill their requirements. In particular, he decides to create a new badge named "Unity is strength" to reward the collaboration between students. In the badge creation window within the tournament creation, he sets the name of the badge and the rule to achieve it: $team\_memebers > 3$, selecting $team\_members$ in the list of available attributes to create rules for badges. In this way all the students that partecipate in a battle with a team of four or more members receive this badge.

After the presentation of the above scenarios, we can formalize a **UML class diagram** for our system.

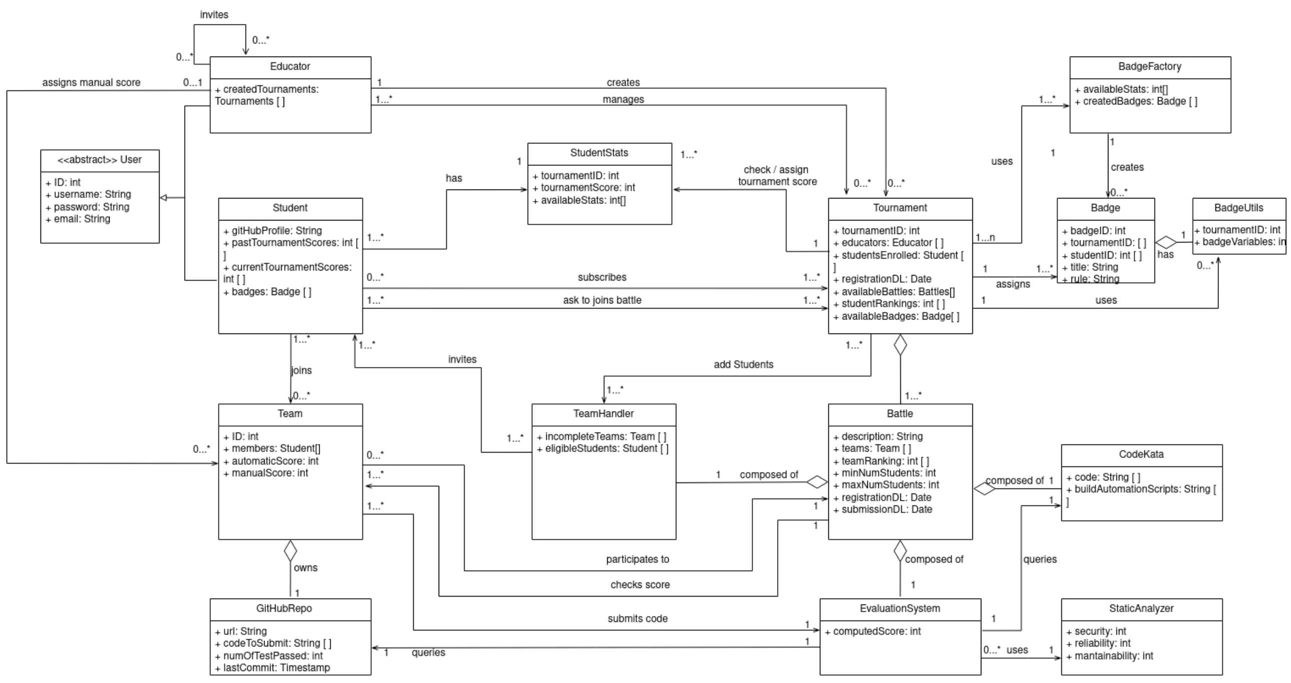


Figure 1: CKB Platform UML Class Diagram

Here are some notes regarding the most important entities of the diagram.

- The *User* abstract class is specialized into *Student* and *Educator*. Educators are the users who can create new Tournaments and Battles, while Students can partitipate to a Tournament, compete in Battles and receive Badges.
- The educator creates a Tournament by instantiating a *Tournament* class. This class is composed of different *Battle* classes, each one representing a battle created by an educator. The *Battle* class itself is composed of a *CodeKata* class containing the actual code of the battle, a *TeamHandler* class which is responsible for handling the creation of teams that will participate to the battle, and an *EvaluationSystem* class, used to compute the score of each team for the current battle.
- Each students subscribed to a Tournament has a *StudentStats* class associated to. This class

contains

  - the tournament score for the corresponding student
  - the *availableStats* list, containing all the statistics provided by the CKB platform, tailored for the current turnament. This list will be used to gather the necessary variables in order to assign Badges to each student.

- Whenever a student subscribed to a tournament wants to join a battle, he queries the *Tournament* class, which contains the list of available battles. In order to take part to the battle, the student has to either create a team or join an existing one. The *Tournament* class adds the student to the *TeamHandler* class associated with the battle picked by the student. *TeamHandler* contains a list of students that can be invited (which is a list of all students registered for the tournament, excluding students who are part of a complete team for the current battle) and a list of the incomplete teams. Note that a student part of an incomplete team can still be invited to or join another team. Whenever a team contained in *TeamHandler.incompleteTeams* is completed, it is moved to *Battle.teams* and its members are removed from *TeamHandler.eligibleStudents.* Students part of complete teams cannot switch to another team anymore.
- A tournament must have at least one battle available. If at the battle registration deadline no team has joined it, the battle is cancelled.
- Every time a team participating to a battle commits to the GitHub repository, the code is also submitted to the *EvaluationSystem* class, which compute the automatic score for that specific team by querying the *GitHubRepo* and *codeKata* classes in order to evaluate the functional aspect and also the timeliness (*GitHubRepo.lastCommit* attribute serves this purpose). The quality aspect of the score (security, reliability, mantainability) are evaluated using the *StaticAnalyzer* class, that exploits the API of Static analisys tools such as SonarQube to compute this portion of score. Ultimately, the manual portion of the code can be assigned directly by the *Educator* class. The *Battle* class retrieves the score from each team to compute the real time battle ranking.
- At the end of the battle, every team component has its own *StudentStats.tournamentScore* updated.
- The educator can create new Badges and add them to the current Tournament (the *Tournament* class has an *availableBadges* attribute). The process of creating a badge is handled through the *BadgeFactory* class: starting from a list of statistics provided by the CKB platform, the Educator can compute the required badge variables and the correspondig badge rules. Every *Badge* class has a *BadgeUtils* class associated to, which contains a list of the badge variables and the method to compute them.
- The assignment of badges is managed by the *Tournament* class, as it can access the statistics of all students (contained in *StudentStats.availableStats*) and all badge variables (contained in *BadgeUtils.badgeVariables*). For every Badge, the *Tournament* class:
  - retrieve how to compute the badge variables from the *BadgeUtils* class
  - retrieve the statistics of each student
  - compute the badge variable for each student
  - assigns the badge

Now we will analyze the behaviour of some object from the UML class diagram. In particular, we will use **state diagrams** to display the changes the object is subjected to over time.

The state diagram below shows the behaviour of the *User* object through the entire charging process. Please note that an user can decide to subscribe to CKB either as *Educator* or as *Student*.

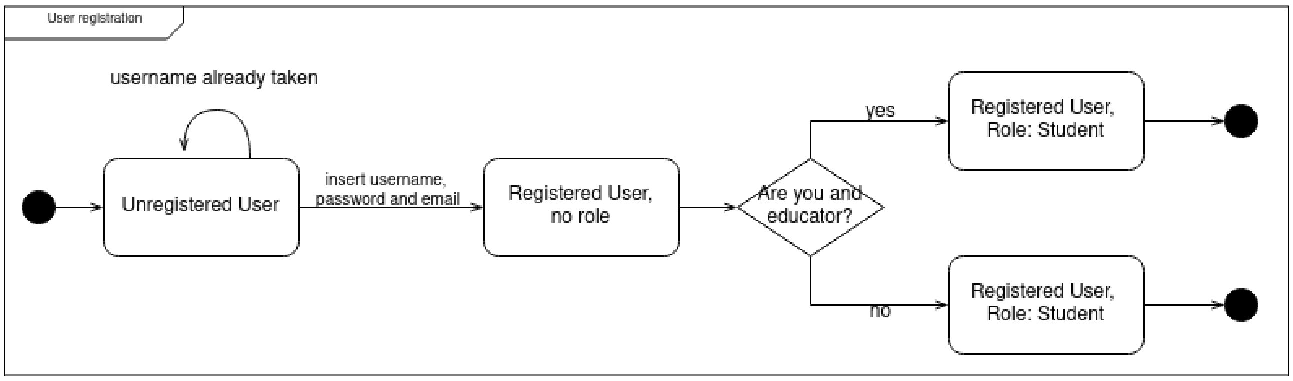
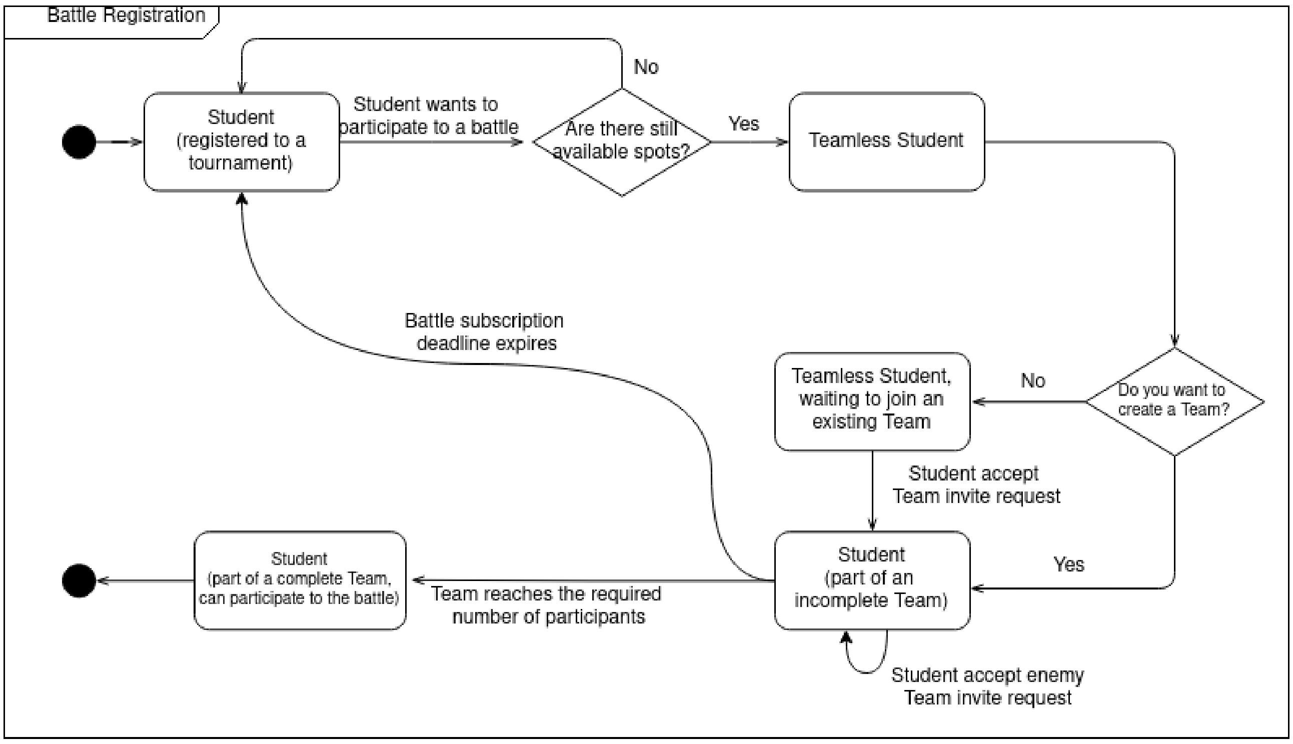


Figure 2: User registration state diagram

The state diagram below shows the process of joining a *Battle*. Please note that a student must be subscribed to the tournament hosting the battle in order to join it. Also, as long as the chosen *Team* is not completed, the student can leave it for another one. Lastly, if the battle subscription deadline expires and the Student is in an incomplete team it cannot participate to the battle.



Battle Registration
Student (registered to a tournament)
Student wants to participate to a battle
No
Are there still available spots?
Yes
Teamless Student
Battle subscription deadline expires
Teamless Student, waiting to join an existing Team
No
Do you want to create a Team?
Student accept Team invite request
Yes
Student (part of a complete Team, can participate to the battle)
Team reaches the required number of participants
Student (part of an incomplete Team)
Student accept enemy Team invite request

Figure 3: Battle registration state diagram

The state diagram below shows the process of updating the *Real Time Battle Score*. The process is triggered every time there is a commit to the Team's repo. When the battle deadline expires, the *Automatic Score* is finally computed and we enter the consolidation phase in which the *Manual Score* is assigned by an educator (if required).

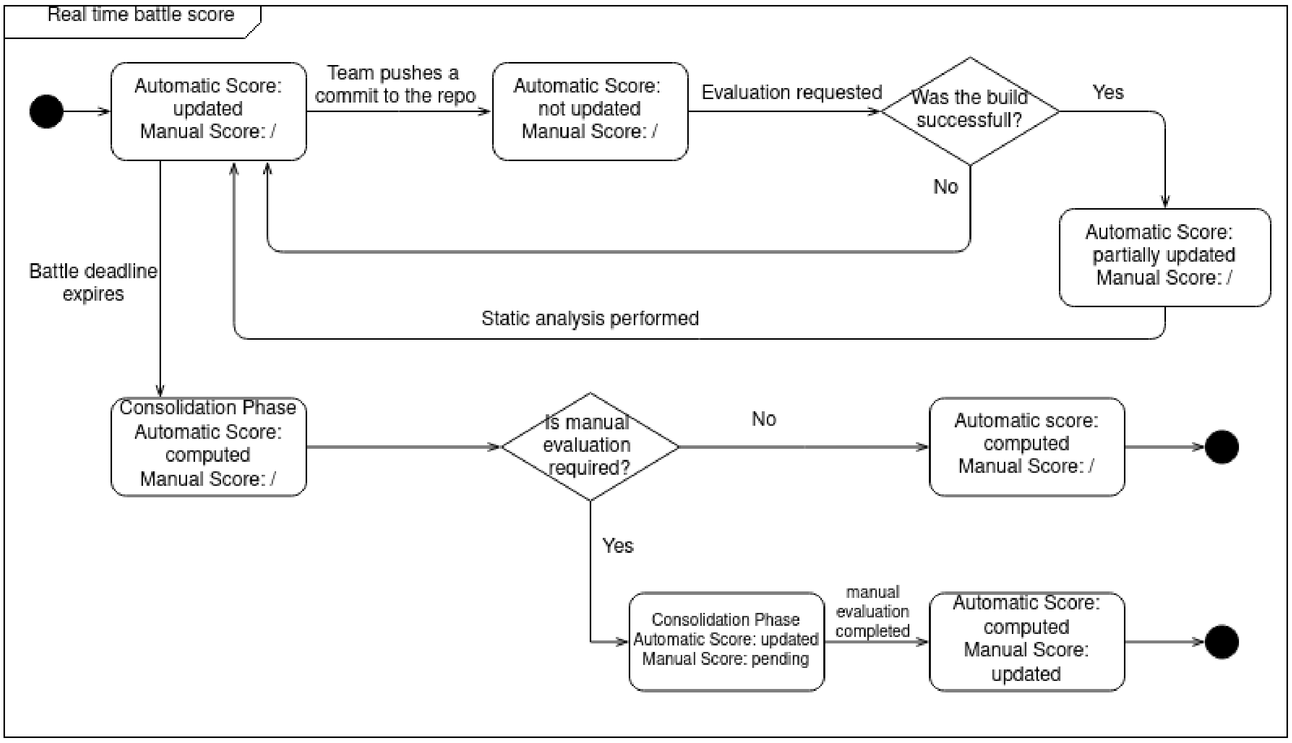

Real time battle score
Automatic Score: updated Manual Score: /
Team pushes a commit to the repo
Automatic Score: not updated Manual Score: /
Evaluation requested
Was the build successfull?
Yes
No
Automatic Score: partially updated Manual Score: /
Static analysis performed
Battle deadline expires
Consolidation Phase Automatic Score: computed Manual Score: /
Is manual evaluation required?
No
Automatic score: computed Manual Score: /
Yes
Consolidation Phase Automatic Score: updated Manual Score: pending
manual evaluation completed
Automatic Score: computed Manual Score: updated

Figure 4: Battle score computation state diagram

2.2 Product Functions

In this section, the main functionalities of CKB are presented and described in-depth.

- **Creation of a Tournament by an Educator:** CKB must provide already registered educators the possibility to create a personalized tournament in a simple way. On the main menu, they need to press the "create new tournament" tab and, in the newly opened window, they should specify all the fundamental information, such as the tournament name, description, and subscription deadline. Optionally, the system also allows educators to set an icon for the tournament. After customizing all mandatory settings, the tournament creation function should allow educators to choose whether to add collectible badges to the tournament. This function must be optional, as a tournament can be fully functional without badges. In case educators want to add badges to the tournment, an appropriate customization window should be provided to the user, in order to allow them to have a comprehensive and clear view of all parameters available to create rules.The system should allow the user to to create multiple badges within the same tournament. For each one, the application should provide functionalities to set a name, a brief description and an optional icon. Then, an arbitrary number of rules can be added, in the following form:

$$[system\_parameter] \leq, <, =, >, \geq [value, true, false]$$

in which true and false are only available for boolean parameters, while value can be a static value set by the educator or a dynamic value generated by CKB, for example *max-tot-commits*. Moreover, the user should be allowed to choose the relationship between rules with *and* and *or* logical operators, by writing simple logical expressions using previously defined rules and parenthesis, if needed:

(rule1 or rule2) and rule3

Rules will be evaluated and checked by a validator included in the badge creation functionality, in order to establish if the logical expression can be actually fulfilled and students can actually obtain the badge.

- **Manage Tournament:** CKB must allow educators to manage tournaments they have created. Each educator must be provided with the list of tournament they have created, and for which they are allowed to exploit all fuctionalities as administrators, and the list of tournaments to which they have been invited to collaborate. For created tournaments, educators must be allowed to invite other educators as collaborators, by going in the tournament management section and choosing the "invite educator" functionality. The invited educator should be allowed to accept or deny the request.
  In the tournament management section, educators should also be able to view the current status of the tournament rank and the status of all battles within the tournament. Educators that are also creators of a tournament must be allowed to close the tournament whenever they seem fit, as long as there are no active battles in the tournament. This function must not be provided to educators which have been invited to manage a tournament.
- **Creation of a Battle by an Educator:** CKB must provide already registered educators the possibility to create a new battle within tournaments they manage. By clicking the "create battle" tab, a window opens, and the educator is provided an interface through which they can enter all the necessary descriptive information, such as the name of the battle and a description of the challenge. After setting preliminar informations, the educator is asked to upload the software project (including the test cases and the automatic scripts). The educator should also be allowed to set requirements for the creation of teams, like the minimum and maximum number of members for each group. They also need to set two different deadlines, one for the subscription to the battle by students and the other for the end of the battle. Lastly, the aplication should provide all necessary functions to customize the evaluation system for the battle. The automatic evaluation system takes into account three different scoring parameters. Each parameter will have a weight on the final score set between 0 and 1, also customizable by the educator, and the final sum of all weights will be 1, as to obtain a total score of maximum 100 points. Available parameters are the following
  - *Functional.* This parameter measures the number of test cases passed out of all test cases. The maximum number of points that can be obtained through this parameter will be equally distributed among all test cases. As an example, if the functional parameter allows students to score a maximum of 30 points and the project provides 10 test cases, each test case passed will allow students to score 3 points. The only customization option provided by this paramter is its weight.
  - *Timeliness.* This parameter evaluates how much time was spent to complete the project with relation to the registration deadline. After half of the available days are passed, points will be equally distributed among the remaining days available to complete the project and will be progressively subtracted to the total number of obtainable points each time a push is executed in a new day. Educators can choose the starting time from which points will be detracted, other than the weight of this parameter on the total score.
  - *Quality.* Quality level of the sources can be evaluated according to multiple parameters, such as security, reliability and mantainability. The educator must be allowed to choose how many parameters to consider as they deem appropriate, to do so the
  - application must provide a toggle feature. The application must be connected through appropriate APIs to any third party application, such as *SonarQube*, that is able to provide static analisys of code based on the selected parameters.

  Other than the previously described parameters, the educator must indicate whether they want to provide a manual evaluation, whose points will contribute to the final score of the team. This optional evaluation will also have a weight that must conform to the previously stated rules.

- **Allow a Student to Join a Tournament:** CKB must allow students to browse and choose a tournament to join. To do so, a list of all ongonig tournaments must be available to all students subscribed to the platform, equipped with appropriate search and sort features to allow students to find the best tournament according to their skill level and their interests. On the tournament overview page, a join option must be provided if the tournament is still open for subscriptions.
- **Allow a Student to Join a Battle:** CKB must provide the students involved in a tournament with the opportunity to join a battle by different means. Any student that wants to join a battle must create or join an existing team first, according to the rules stated in the battle desciption. For this reason, different features are provided to the student to make the process of joining a battle as flexible as possible.
  - *Create a new team.* Each student is allowed to create their own team for any battle. This feature allows the student to choose a name for the team and the number of members. Within the creation context, if the team has more than one member (including the creator), the student can already send invites to other students subscribed to the same tournament by searching their username in a search bar. This feature must also be available after the team creation process is completed: incomplete teams are displayed in the battle overview page until they can join the battle. Any team member can send join invites to other students within the tournament. If the team has only one member and is compliant with battle rules, upon completing the team creation process the team immediately joins the battle.
  - *Join an existing team.* Upon visualizing the battle overview page, students are provided with the list of teams that have not reached ther maximum number of members yet. Students must be allowed to choose any team among the list, to visualize the team detailes, including the maximum number of members and current members, and must be allowed to send a join request to the team they prefer. At the same time, any member of a team that receives a join request is provided with the option to accept or decline a request. Once a team member performs one of those actions, other members cannot override that action.
  - *Quit team.* Students who are members of incomplete teams should also be allowed to quit their team. This feature must only be available for teams that have not joined the battle yet. If the only team member of a team decides to quit, the team is deleted.

  Join requests by students and join invites by teams can be acceped or denied. As soon as a team reaches its maximum number of members, it is removed from the list of incomplete teams and joins the battle. At this point all information about the team cannot be edited, and team members cannot quit the team.
- **Allow Users to Check Rankings:** CKB applications provides a ranking system for both tournaments and battles.
  - The tournament rank is based on the personal score assigned to each student. Each time a student completes a battle and obtains a score, this score is summed to their personal score. For this reason, the battle rank is updated each time a battle is officially closed. The tournament overview page must allow both educators and students to check the current tournament rank.
  - The battle rank is based on the progressive score obtained by each team during a battle. Each time a push is executed ont he GitHub repository of a team, a new evaluation is performed and the team score is updated, thus updating the rank. The battle rank must be available in the battle overview page, and both students and educators must be allowed to check it whenever they seem fit.
- **Allow Students to be Notified of Relevant Actions:** CKB must provide a notification system to inform students of relevant actions and events. Multiple scenarios can trigger notifications, which will be displayed in an inbox on the student's account and sent by e-mail, to

the address specified during the registration process. The e-mail notification can be optional, users should be provided with the function to toggle them on or off. Actions that should trigger notifications are the following:

- *Creation of a tournament.* Each time a tournament is created, all students should receive a notification that a new tournament is available.
- *Expiration of battle subscription deadline.* When the battle subscription deadline expires, all subscribed students will receive a notification with the link of the GitHub repository for the project.
- *A battle is officially closed.* After the final battle rank is computed, all subscribed students will receive a notification informing them that the final rank is available.
- *Closing of a tournament.* When an educator decides to close their tournament, all subscribed students will be notified, as the final rank and assigned badges (if provided) are available.
- *Join requests or invites.* Whenever educators are invited to join a tournament, students are invited to join a battle or receive a join request for their team, they will be notified.

Student should also be able to choose for which one of these actions they want to receive real-time notifications, or if they prefer a daily or weekly recap.

2.3 User Characteristics

In our CKB system, the following type of users are considered:

- **Unregistred student:** students that are not yet register on CKB. They need to create an account to access the service offered by CKB.
- **Registered student:** students that have an account on CKB. They can use all the functionality offered by CKB for students, for example partecipates at a new tournament.
- **Unregisters educator:** educators that are not yet register on CKB. They need to create an account to access the service offered by CKB.
- **Registerd educator:** educators that have an account on CKB. They can use all the functionality offered by CKB for reducators, for example create a new tournament .

2.4 Assumptions, Dependencies and Constraints

Domain Assumptions:

D1: Users enters correct information regarding themself.

D2: Every user has an eletronic device with internet access.

D3: Educators provide a description of the tournament they are creating.

D4: Students joining battles have valid accounts.

D5: Students can partecipate on multiple tournaments at the same time.

D6: Students can partecipate on multiple battles at the same time, also in the same tournament.

D7: A tournament consists in different battles, and each battle can be on different programming language.

D8: The students that partecipate in a battle fork the GitHub repository of the team and set up an automated workflow through GitHub Actions.

D9: The battle rank updates after a commit during the battle.

D10: Personal tournament scores sums all the scores obtained by a students in the code kata battles within a tournament.

D11: Educators define accurate scoring parameters for code katas battles.

D12: Notifications are delivered promptly to participants after a new tournament or battle is created, or when a battle finish.

D13: Automated evaluation criteria, including functional aspects, timeliness, and source code quality, produce consistent results.

D14: Gamification badges have precise rules to satisfy to be assign to a student.

D15: Gamification badges rules refers on existing variables.

Dependencies:

De1: CKB integration with GitHub relies on its stability to works properly.

De2: Users depend on stable and secure internet connections to participate in code kata battles.

De3: Users depend on modern web browsers that support the website.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interface of CKB must satisfy the needs of both educators and students. The main user interface is a website, that must be available from any type of "common" device (like personal computers, tablets and smartphones). In a reasonable way, it is possible to assume that most educators will use an office dekstop while students might want to use also a mobile phone, so also a mobile-friendly version of the user interface must be be implemented.

3.1.2 Hardware Interfaces

CKB is a web-based system. Hence, in order to access it, a generic device, like personal computers, laptops, tablets, and smartphones, with an internet connection is needed.

3.1.3 Software Interfaces

To work properly, the CKB system relies on some external softwares:

- GitHub provides an API that enables the system to generate new repositories for each team involved in a battle.
- The system integrates with a static analysis tool, that provides an API for incorporating the static analysis of projects, contributing to the automatic score of the team.

3.2 Functional Requirements

In this section are presented the most relavant use cases for the CKB application. Each use case is illustrated through the associated sequence diagram, while all uses cases are grouped and presented together in the use case diagram. Entry conditions and exit conditions are the conditions that must hold for the actor to enter and exit the use case.

UC1: Student registration

Actors	Student
Entry condition	A user wants to register on the CKB platform as a student.
Event flow	1. The student opens the CKB home page and clicks on the sign up/sign in button. 2. The student chooses the ”sign up as a student” option and the registration form is loaded. 3. The student fills in the form with their personal data and account credentials, such as name, surname, e-mail, password and username. 4. The CKB platforms reteives the request and checks if the e-mail is valid and the password is conform to securety standards. 5. The student is asked to link their CKB account to their GitHub account. 6. The student is redirected to the GitHub login page. 7. The student inserts their GitHub credentials and logs into their account. 8. The student is redirected to the CKB registration page.
Exit condition	The student is successfully registered to the CKB platform and correctly linked their GitHub account.
Exception	1. The student can’t complete the registration process due to multiple reasons: • The credentials submitted are not adequate. • The e-mail is not well formatted. • The e-mail submitted is already used by another account. • The username is already used by another account. • The password used is not strong enough according to the platform’s security standards. • The user fails to log in their Github account because the credentials are wrong. 2. The platform fails to retreive the GitHub login page. The user is invited to refresh the page and try again. 3. Internet connection is lost during any phase before the completition of the registration process.

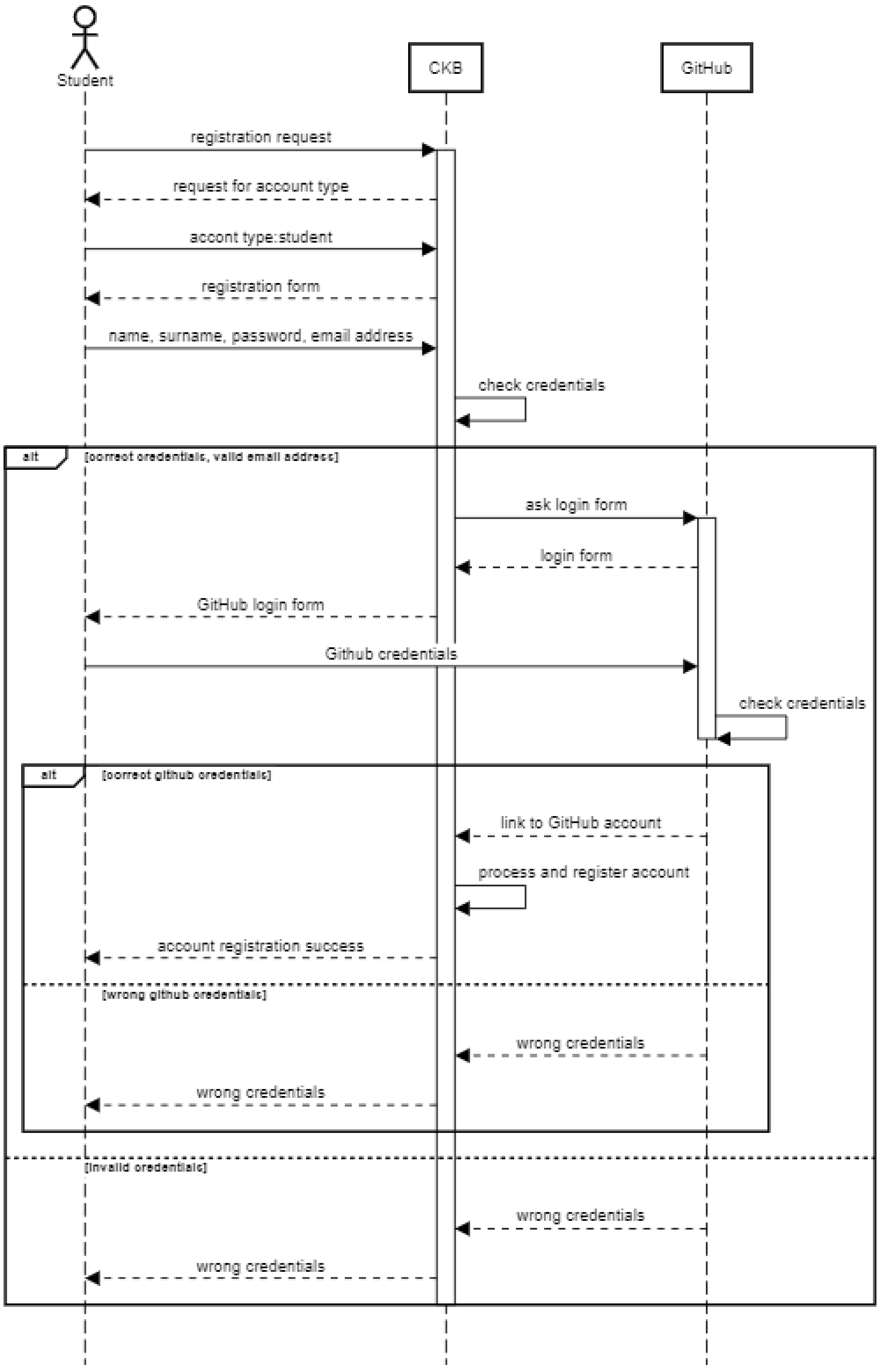


Figure 5: Student registration

UC2: User login

Actors	User
Entry condition	a user *studentoreducator* wants to log in the CKB platform.
Event flow	1. The user opens the CKB application and asks to log in. 2. The CKB application displays the login page. 3. The user inserts their account credentials. 4. The user logs in the platform.
Exit condition	The user logs in the CKB application.
Exception	• The user inserts wrong credentials. In this case an error message is displayed and the user is incvited to try again. • An error occurs during the authentication process. • Internet connection is lost before all oeprations are completed.

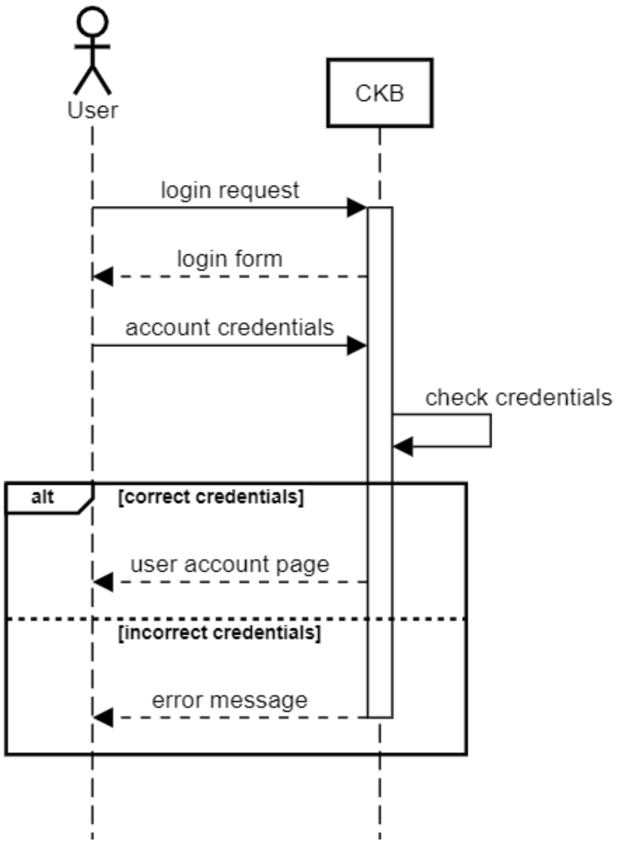


Figure 6: A user logs in the CKB platform

UC3: Join a tournament

Actors	Student
Entry condition	A student wants to join one of the open tournaments.
Event Flow	1. The student opens the CKB home page and logs in. 2. The student clicks on the "View active tournaments" button. 3. The list of active tornaments is displayed. 4. The student filters the list to see which tournaments are open to join. 5. The student sends a request to join the tournament by clicking on the join button. 6. The student is subscribed to the tournament.
Exit Condition	The student is successfully subscribed to the tournament.
Exception	• The student clicks on the join button seconds before the subscription deadline expires, in this way the platform does not have enough time to complete the subscription process before the deadline. • There are no open tournaments available to join. • Internet connection is lost before completing the operation.

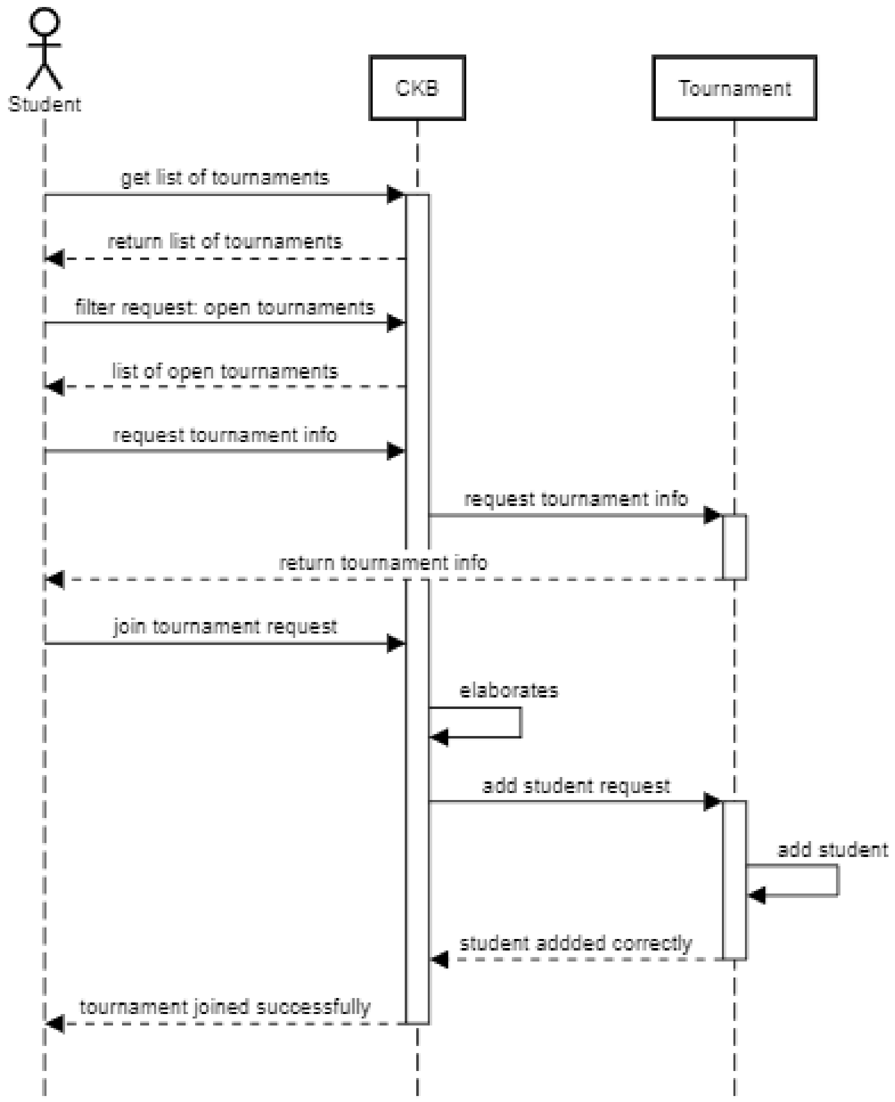


Figure 7: Student joins a tournament

UC4: Create a team

Actors	Student
Entry condition	The student wants to join a battle by creating a new team.
Event Flow	1. The student asks the platform to display the list of available battles for a certain tournament. 2. The student selects a battle. 3. The CKB application opens the battle overview page, displaying its details. 4. The student clicks on the button to join the battle. 5. The CKB application displays the "join battle" page, where the student is asked to coose between creating a new team or joinin an existing one. 6. The student decides to create a new team. 7. The student fills the form with the team settings, including the number of participants. 8. If the team has only one participant, the team is registered to the platform and joins the battle immediately. 9. If the team has more than one participant, the team is stored by the TeamHandler until the team is complete.
Exit Condition	The student creates a team and joins the battle. The student creates a team and is put in the waiting list.
Exception	• The application fails to retreive the list of available battles. • The application fails to load the batttle overview page. • The application fails to upload the team settings. • The application fails to register the team. • The internet connection is lost before the last operation si completed.

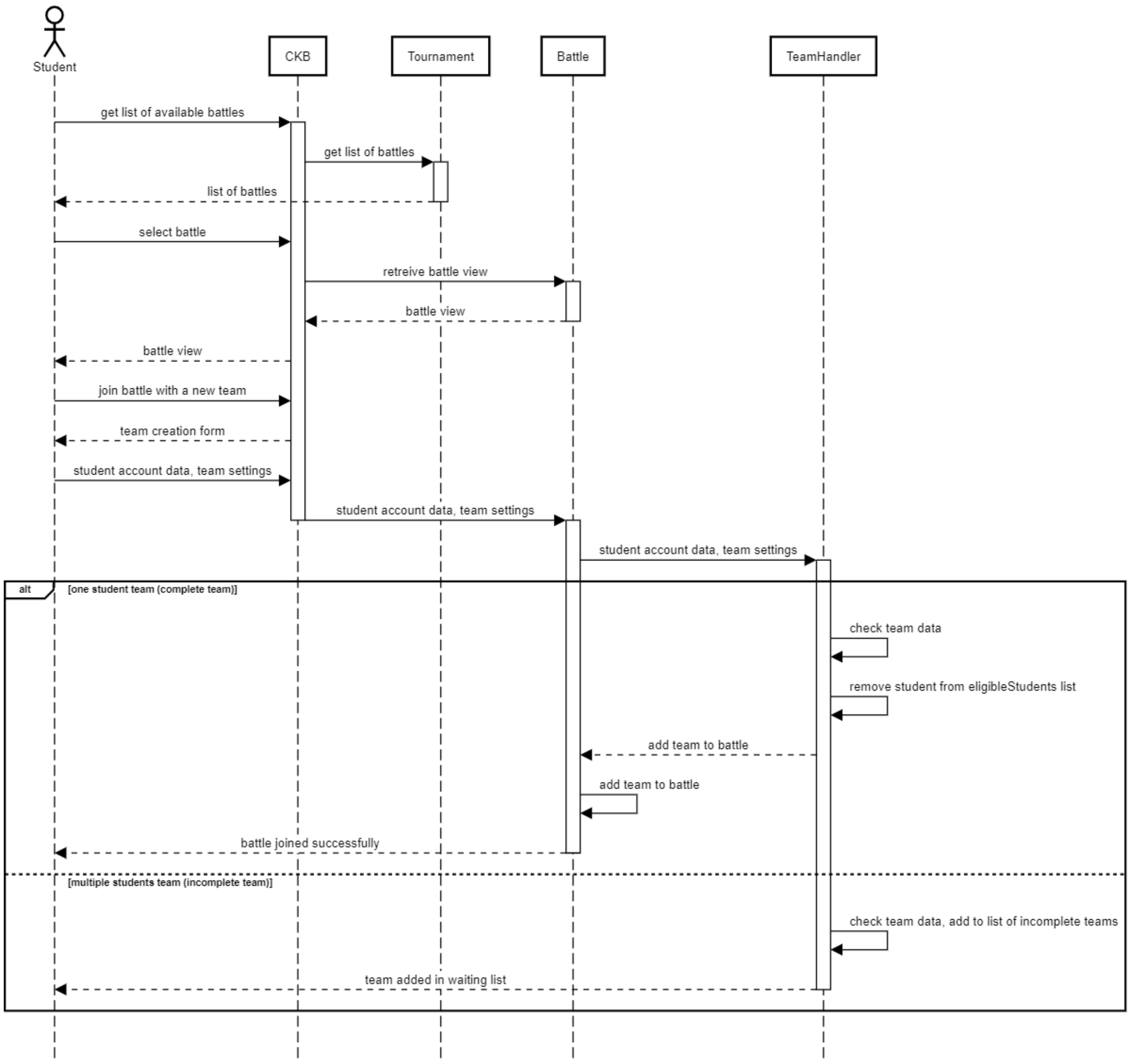

Student
CKB
Tournament
Battle
TeamHandler
get list of available battles
get list of battles
list of battles
select battle
retreive battle view
battle view
battle view
join battle with a new team
team creation form
student account data, team settings
student account data, team settings
student account data, team settings
alt
[one student team (complete team)]
check team data
remove student from eligibleStudents list
add team to battle
add team to battle
battle joined successfully
[multiple students team (incomplete team)]
check team data, add to list of incomplete teams
team added in waiting list

Figure 8: Student creates a team and joins a battle

UC5: Join a team

Actors	Student
Entry condition	The student wants to join a battle by joining an already existing team
Event Flow	1. The student asks the platform to display the list of available battles for a certain tournament. 2. The student selects a battle. 3. The CKB application opens the battle overview page, displaying its details. 4. The student clicks on the button to join the battle. 5. The CKB application displays the ”join battle” page, where the student is asked to choose between creating a new team or joinin an existing one 6. The student decides to join an existing team, so the platform shows them the list of incomplete teams. 7. The student selects a team and sens a join request. 8. If the team accepts the request, the student is added to the team. 9. If the team reaches the total number of participants specified in its settings, it automatically joins the battle. 10. If the team is still incomplete, it remains in the waiting list.
Exit Condition	• The student is added to the team and the team is still incomplete. • The team is complete and it joins the battle. • The request is rejected.
Exception	• The application fails to load the requested views and pages • The request is not accepted nor rejected before the battle subscription deadline expires. In this case, if the team is still incomplete, it cant join the battle. • If the team accepts other requests or successfully invites other students before the deadline expires, it can still join the battle. If the team joins the battle before answering the request, the request is considered as rejected. • Internet connection is lost before all operations are completed.

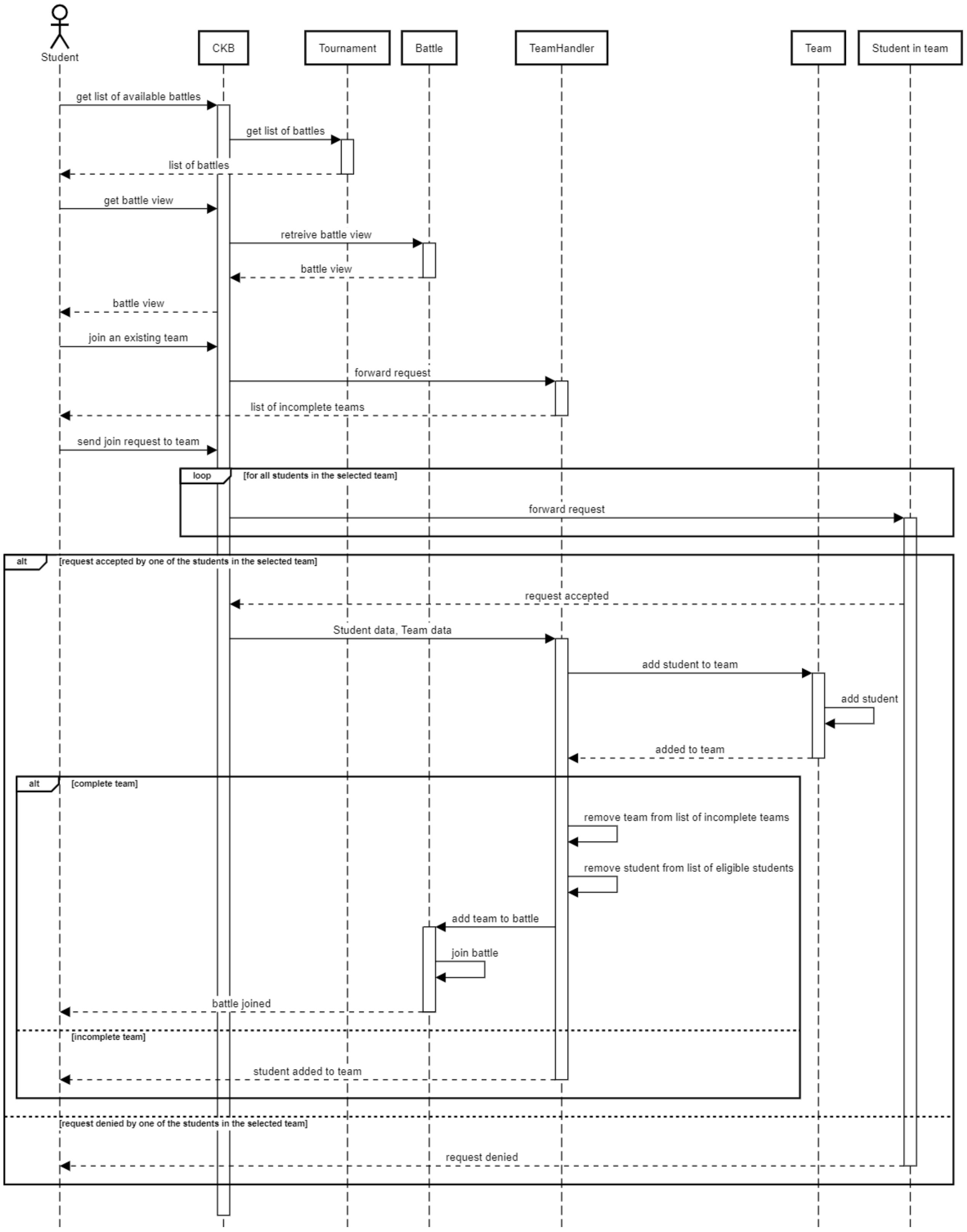


Figure 9: Student joins an existing team for a battle

UC6: Invite a student to join the team

Actors	Student
Entry condition	A team member invites a student to join their team
Event Flow	1. A student (team member) asks the platform to show the list of students subscribed to the tournament but that are not part of any team yet. 2. The student selects another student from the list and send them an invitation request to join their team. 3. If the receiving student accepts the request and the team is not complete yet, they're added to the team. 4. If the team is then complete, it automatically joins the battle. Otherwise, it remains in the waiting list. 5. If the request is rejected, team members are notified and the student is not added to the team.
Exit Condition	• The request is accepted, the team is complete and joins the battle. • The request is accepted, the team is incomplete and is kept in the waiting list. • The request is rejected.
Exception	• If the invited student accepts another request before the one considered in this use case, it is handled as rejected. • The student fails to accept/reject the request before the battle subscription expiration date. In this case the request is handled as rejected. • The application fails to deliver the request to the student. • The application fails to deliver the answer to the team. • Internet connection is lost before all operations are completed.

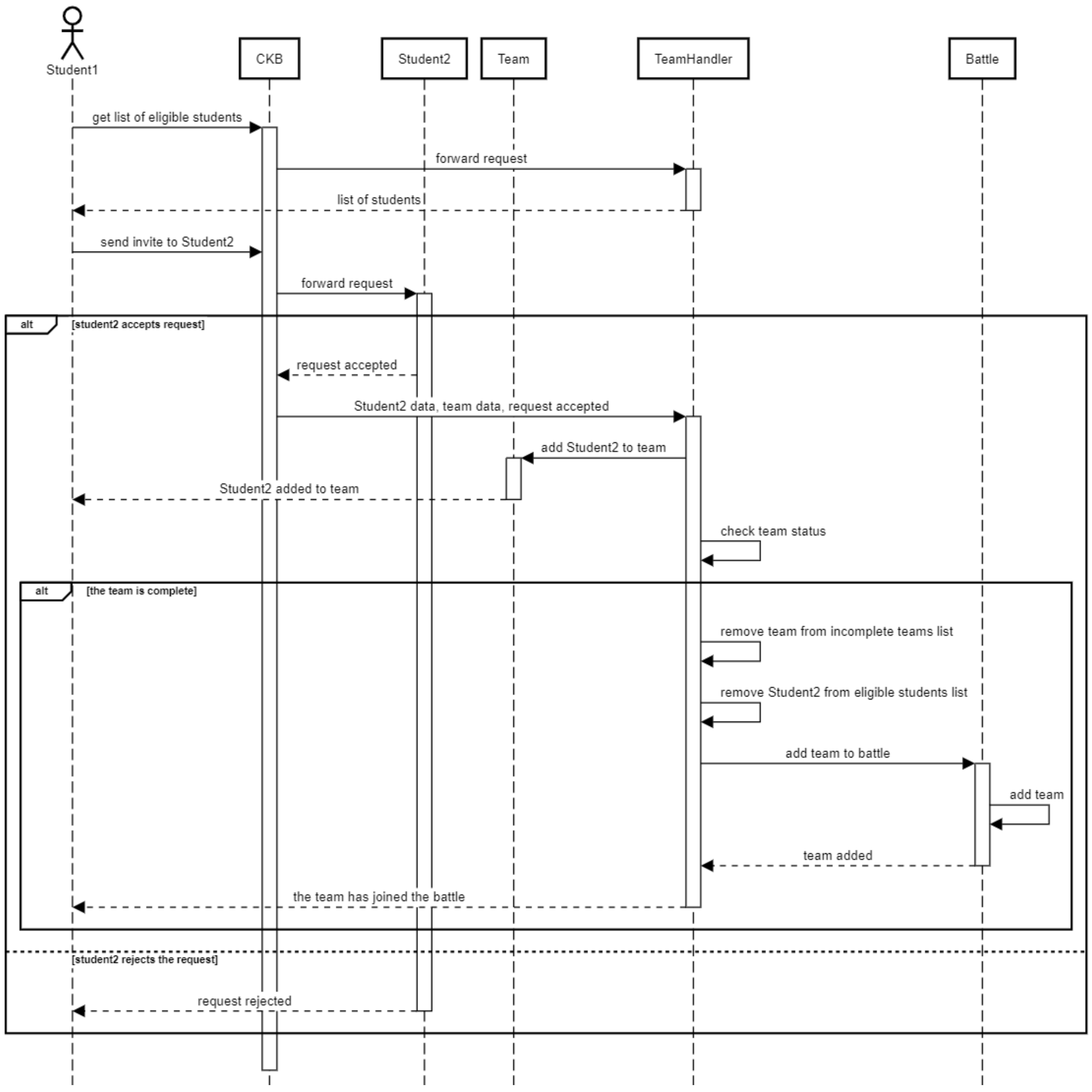


Figure 10: A team member invites another student to join their team

UC7: Quit team

Actors	Student
Entry condition	A student decides to quit their current team.
Event Flow	1. the student opens the overview of their team. 2. The student selects the ”quit team option” 3. The request is forwarded to the TeamHandler. 4. If the team has as its ony participant the student who wants to quit, the TeamHandler removes the team from the list of incomplete teams. 5. The TeamHandler deletes the team. 6. If the team has more than one participant, the TeamHandler deletes the student from the team and notifieas all other participants of the team.
Exit Condition	The student quits their team.
Exception	• An error occurs before the process is completed. • Internet connection is lost before all operations are completed.

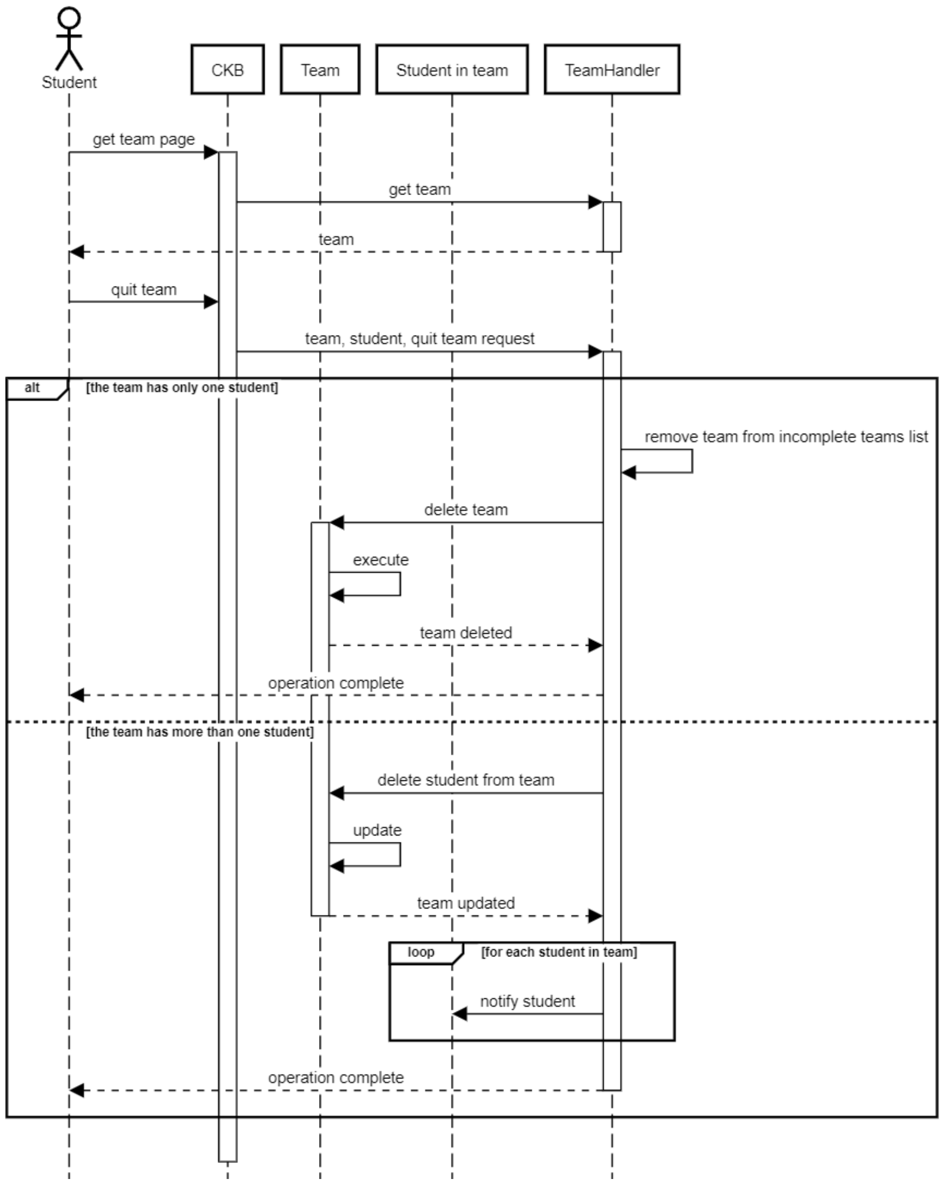


Figure 11: A student decides to quit their team

UC8: Join battle

Actors	Student
Entry condition	A student is part of a team that fulfills all team requirements to join the battle for which it was created.
Event flow	1. The student joins a team or creates a new one. The team is complete with respect to its settings. 2. The team is added to the list of team that particpate to the battle. 3. The platform notifies the student that they joined the battle.
Exit condition	The team joins the battle
Exception	• An error occurs during the process. • Internet connection is lost before all operations are completed.

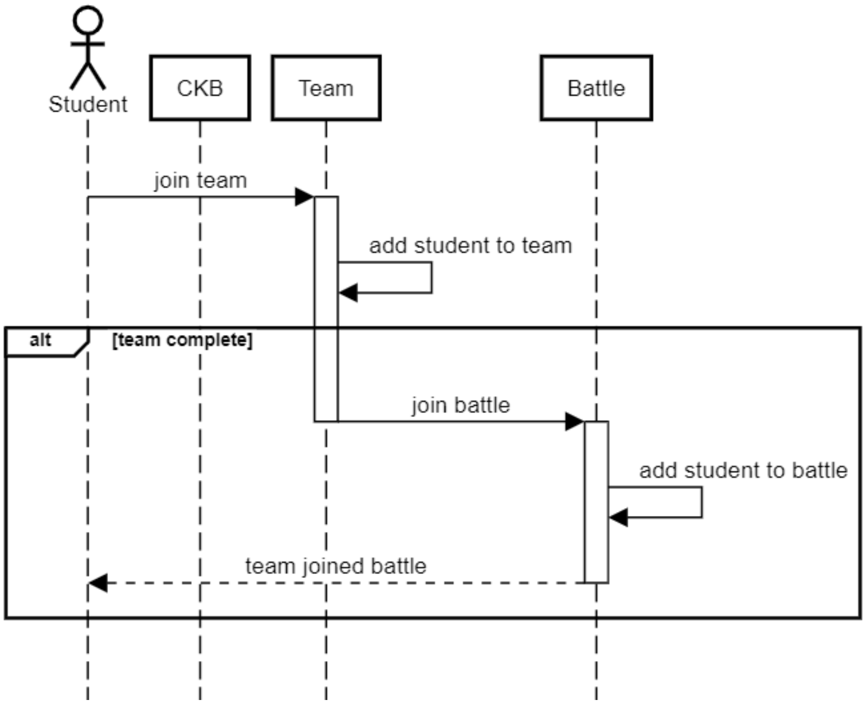


Figure 12: A student joins a battle

UC9: Commit and push on GitHub repository

Actors	Student
Entry condition	A student pushes on the team's GitHub repository.
Event Flow	1. The student executes a push on their team's repository. 2. The code stored in the repository is updated. 3. GitHub notifies the CKB platform that the code has been updated. 4. CKB pulls the code from the repository. 5. CKB sends the code and an evaluation request to the evaluation system associated to the battle. 6. The evaluation system analyzes the performance based on functional and timeliness parameters. 7. The evaluation system analyzes the performance with relation to quality parameters through third party static analisys tools. 8. The evaluation system computes the total automatic score based on the performance for the three main parameters. 9. The automatic score of the team is updated. 10. CKB sends a request to update the battle rank.
Exit Condition	The battle rank is updated.
Exception	• CKB fails to pull the code from the repository. • Some error occurs during the evaluation of the parameters. • The team score is not updated. • The battle rank is not updated.

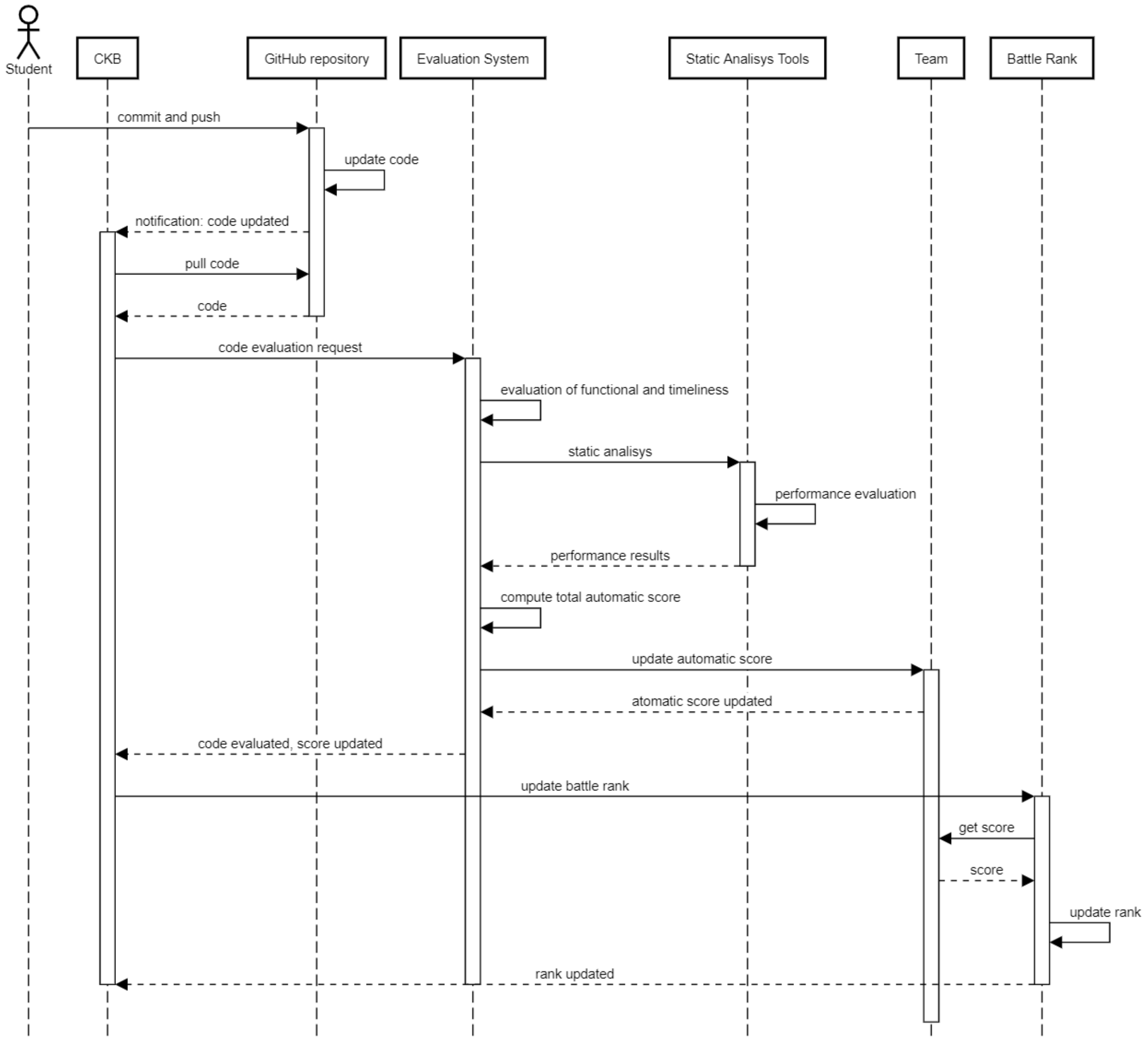


Figure 13: a student executes a commit and push on the GitHub repository

UC10: Badge assignment

Actors	Student
Entry condition	The tournament is closed and there are badges associated to the tournament
Event Flow	1. The tournament instance reteives statistics for each student from the StudentStats class. 2. The tournament instance reteives for each badge the variables and values associated. 3. According to each variable and student statistic, the tournament checks if the student fulfills the requirements to be assigned the badge. 4. The badge is assigned to the student and added to the student profile.
Exit Condition	• The badge is assigned to the student. • The badge is not assigned to the student.
Exception	• An error occurs before the process is complete. • Internet connection is lost before the process is complete.

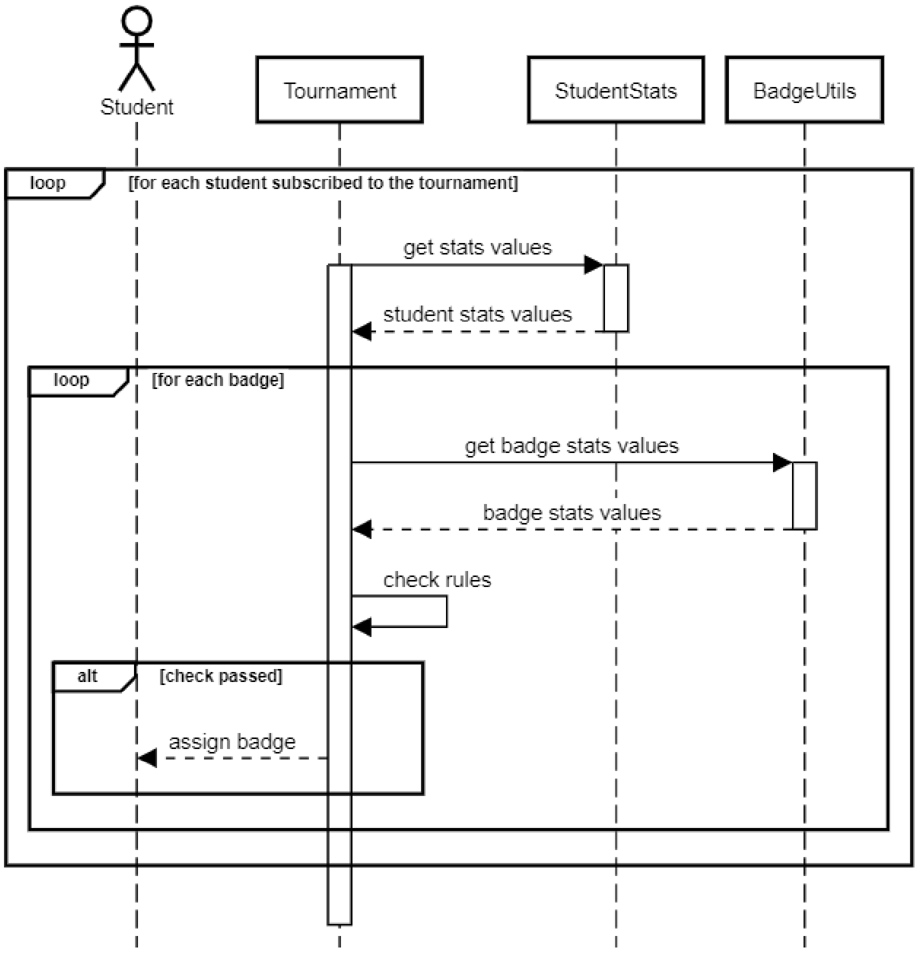


Figure 14: Badge assignment process

UC11: Educator registration

Actors	Educator
Entry condition	A user wants to register on the CKB application as an educator.
Event Flow	1. The user clicks on the "sign up as an educator" option on the CKB homepage. 2. The user is redirected to the registration page. 3. The user inserts all requested personal data, as name, surname, username, a valid e-mail address and a password. 4. The CKB platform checks the validity of credentials. 5. The user is regstered as an educator on the CKB platform.
Exit Condition	The user is registered as an educator.
Exception	• The platform fails to retreive the registration form. • The e-mail is not valid. • The password is not strong enough according to the platform's security standards. • The username is already in use. • The platform fails to register the user correctly. • Internet connection is lost before all operations are completed.

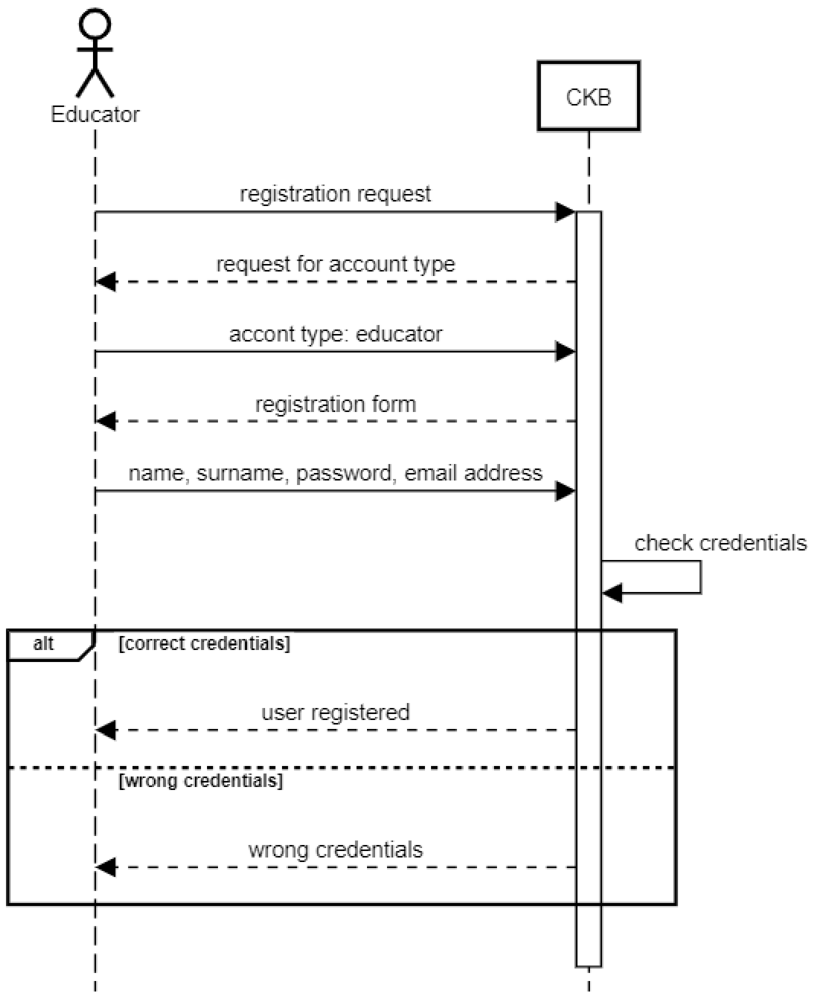


Figure 15: Educator registers on CKB

UC12: Create a tournament

In the following sequence diagram, the participant *badge creation system* is a high-level entity used to represent the system of classes used for badge creation functionalities. In this use case the badge creation pahse is purposefully kept as simple as possible to emphasize the main steps involved in the creation of a tournament. The badge creation system will be further analyzed in a separate sequence diagram.

Actors	Educator
Entry condition	An educator wants to create a tournament.
Event Flow	1. The eucator clicks on the "create new tournament" button. 2. CKB returns the educator the tournament creation page. 3. The educator chooses all available settings to create a tournament, including description, purpose and deadlines. 4. The badge creation view is displayed to the educator. 5. If the educator decides to add badges to the tournament, they ask the system to display the avilable badge settings. 6. The educator sets the values necessary to the creation of a badge and clicks the confirmation button. 7. Once the educator has finished the badge creation process, they exit the badge creation view. 8. The settings are used by the platform to instantiate a new tournament. 9. Once the tournament is created, the platform notifies the educator and sets them as the creator.
Exit Condition	The tournament is created successfully.
Exception	• Some settings are missing. The user is asked to complete all fields before resubmitting the request. • The tournament is not instantiated correctly. The user is informed of the failure and sked to try again. • Internet connection is lost before all operations are completed.

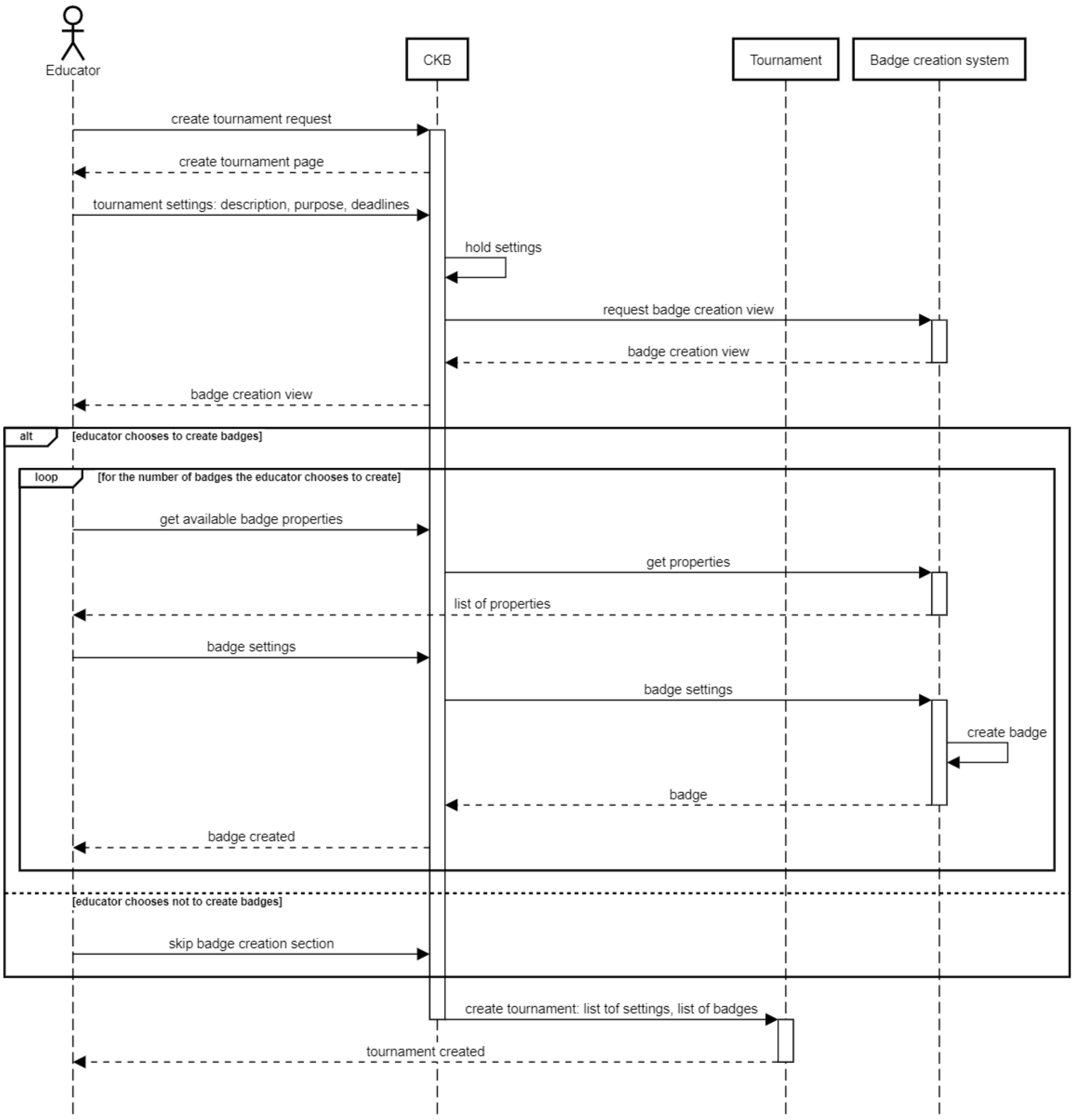

Educator
CKB
Tournament
Badge creation system
create tournament request
create tournament page
tournament settings: description, purpose, deadlines
hold settings
request badge creation view
badge creation view
badge creation view
alt
[educator chooses to create badges]
loop
[for the number of badges the educator chooses to create]
get available badge properties
get properties
list of properties
badge settings
badge settings
create badge
badge
badge created
[educator chooses not to create badges]
skip badge creation section
create tournament: list tof settings, list of badges
tournament created

Figure 16: Educator creates a tournament

UC13: Create a badge

Actors	Educator
Entry condition	The educator decides to associate one or more badges to the tournament they're creating.
Event Flow	1. The educator is displeyed the badge creation page during the tournament creation process. 2. The educator enters a title for the badge and a rule description. 3. The educator is asked to choose the statistics that define the badge between a list of available variables provided by the platform. 4. For each statistic chosen, if necessary the eduactor is asked to provide some custom bounding values. 5. The set of variable, value is sent to the BadgeFactory to create the badge.
Exit Condition	The badge is created successfully
Exception	• The educator does not choose at least one rule for the baadge. In this case the application returns an error maessage and the user is asked to try again. • The educator chooses conflicting variables or values for the badge. In this case the application returns an error message and the user is asked to try again. • The educator exits the badge creation page before completing the creation process. In this case the badge is not created. • Internet connection is lost before completing the process.

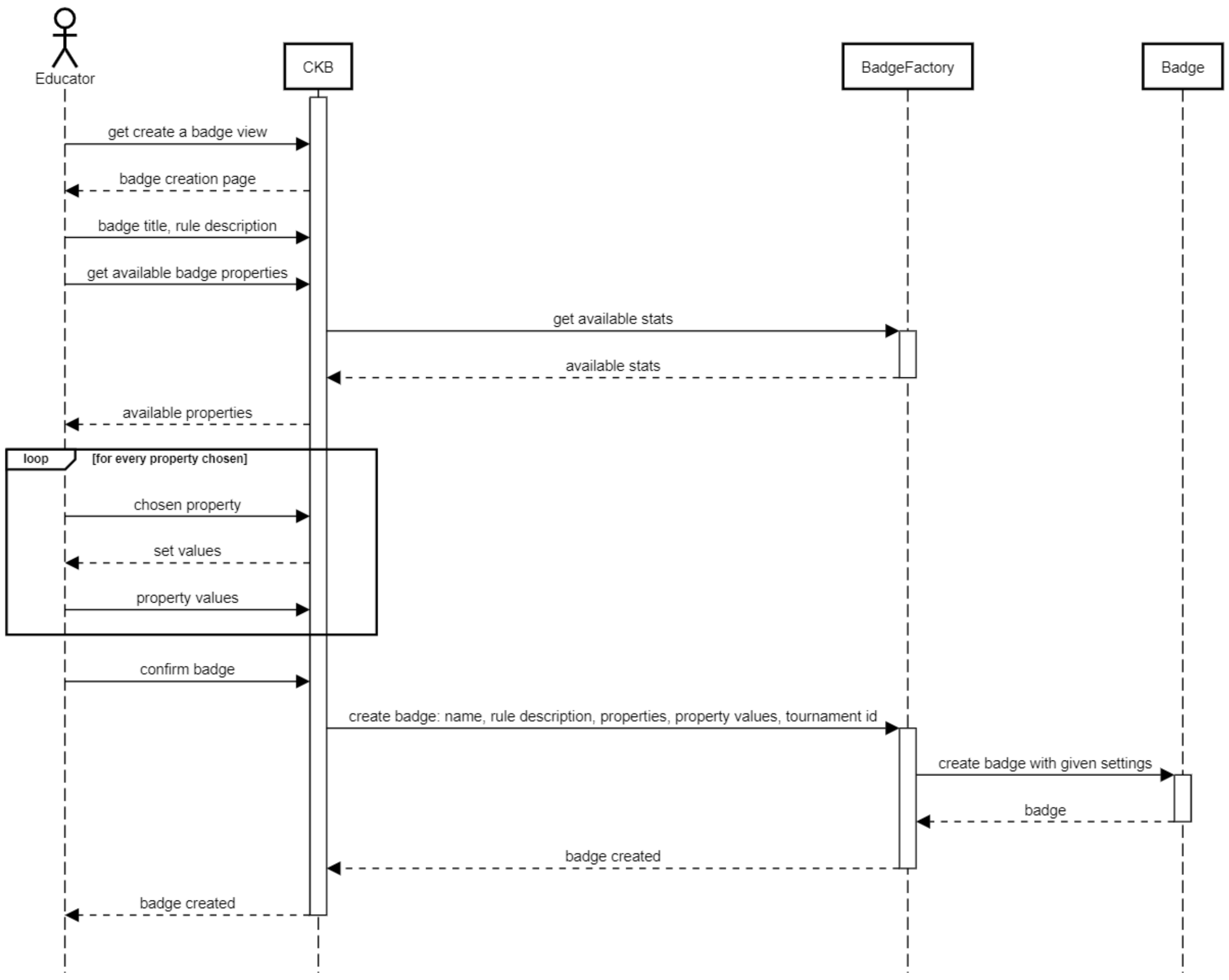


Figure 17: Educator creates a badge for a tournament

UC14: Create a battle

Actors	Educator
Entry condition	An educator wants to create a battle
Event Flow	1. The educator chooses a tournament between those he manages by clicking on it. 2. The CKB application redirecs the user on the tournament page. 3. The educator clicks on the "create battle" button. 4. CKB retreives the battle creation page and shows it to the user. 5. The educator sets the description and deadlines for the battle, uploads the code and build automation scripts and chooses the settings for the evaluation system. 6. The battle settings are sent to the Tournament class that creates a new battle in the context of the tournament. 7. The application notifies the educator thet the battle has been created.
Exit Condition	The battle is created.
Exception	• The educator doesnt fill all the fields in the battle creation phase. This fires an error massage and the user is asked to fulfill all the required settings. • Some error occurrs during the creation of the battle by Tournament. The user is informed that an error occuredd and is asked to try again. • The tournament is closed before the battle creation is completed. The user is informed that the tournament has been closed and no more battles can be created. • Internet connection si lost before all operations are completed.

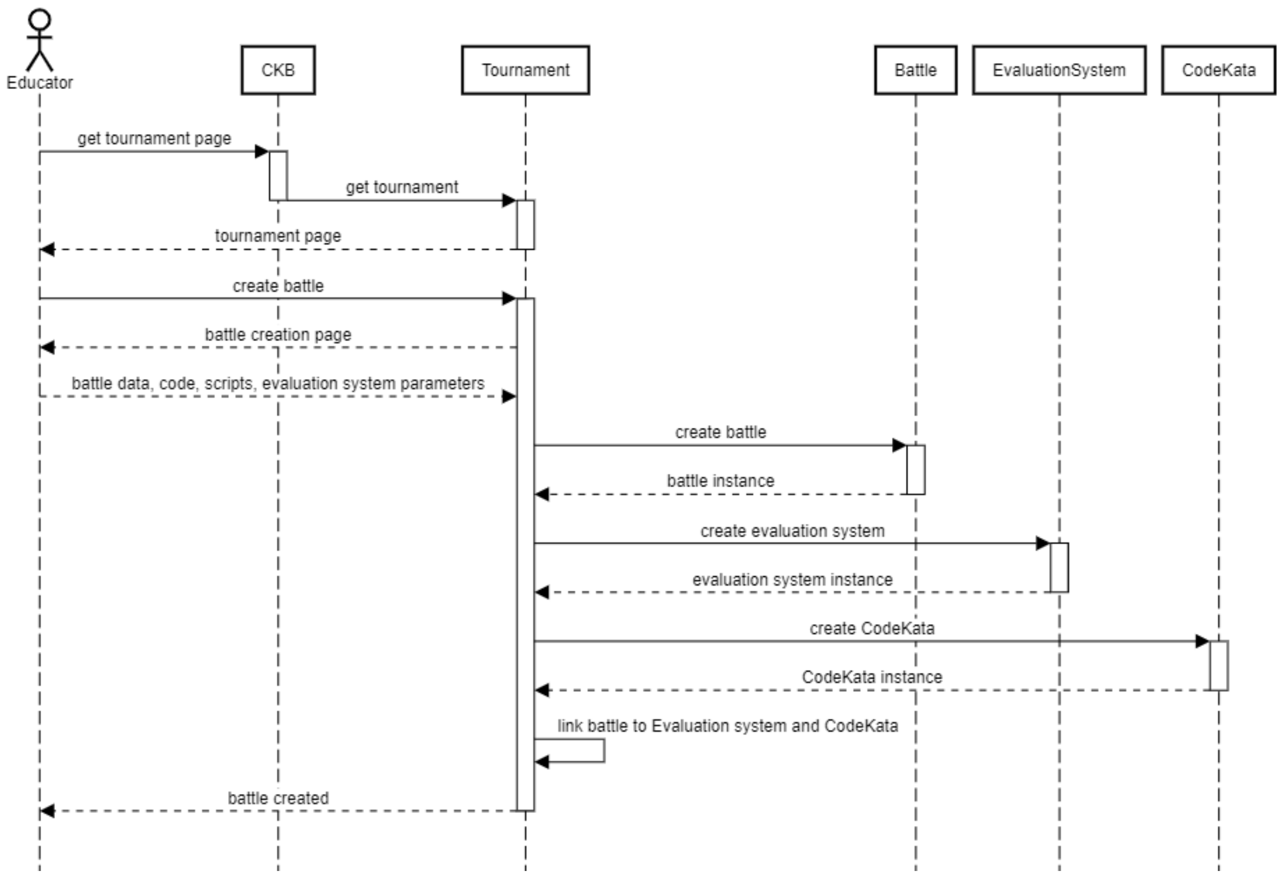


Figure 18: Educator creates a battle

UC15: Invite educators to manage the tournament

Actors	Educator
Entry condition	An educator invites another educator to manage one of their tournaments
Event Flow	1. The educator opens their profile and gets the list of their created tournaments. 2. The educator clicks on the "add manager" button. 3. The educator is asked to enter the username of the educator they want to add as a manager. 4. After entering the username, the request is forwarded to the selected educator. 5. If the educator accepts the request, the platform adds the educator as a tournament manager. 6. A notification is sent both to Educator1 and Eduactor2 to confirm the operation.
Exit Condition	• Educator2 accepts the invitation and is added as a tournament manager. • Educator2 rejects the request.
Exception	• Educator2 is already a tournament manager. In this case Educator1 is notified and the request is not sent. • The selected username does not exist. • Some error occurs during the registration of Educator2 as a tournament manager. • The tournament is closed before Educator2 can i

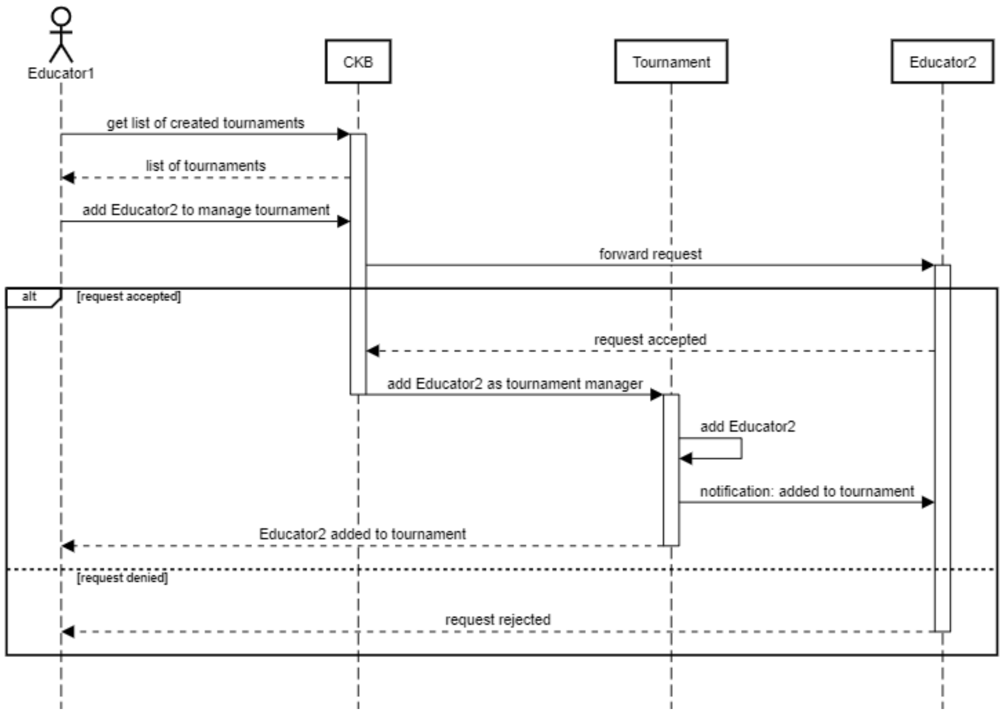

Educator1
CKB
Tournament
Educator2
get list of created tournaments
list of tournaments
add Educator2 to manage tournament
forward request
alt
[request accepted]
request accepted
add Educator2 as tournament manager
add Educator2
notification: added to tournament
Educator2 added to tournament
[request denied]
request rejected

Figure 19: Educator invites another educator to manage a tournament

UC16: Close battle

Actors	Educator
Entry condition	The battle closing deadline expires
Event Flow	1. The battle closing deadline expires, notifying the educator which created the tournament. 2. If the educator decides to assign a manual score to a team, they ask the CKB platform to display the view of the team's project 3. After reviewing the source code, the educator sets a value for the manual score. 4. The manual score for the team is updated and the educator is notified of the success of the operation. 5. After the manual evaluation phase is over, the platform computes the new total score of the team and updates the battle rank. 6. For each student in a team, the sccore of the team is added to the personal score of the team. 7. After updating the personal score of a stuent, the tournament rank is updated.
Exit Condition	The tournament rank is updated for every student subscribed to the battle
Exception	• The educator decides not to set a manual score: in this case the manual score value is defaulted to 0. • An error occurs during any step of the process, before the update of the tournament rank is completed for every student. • Internet connection is lost before all operations are completed.

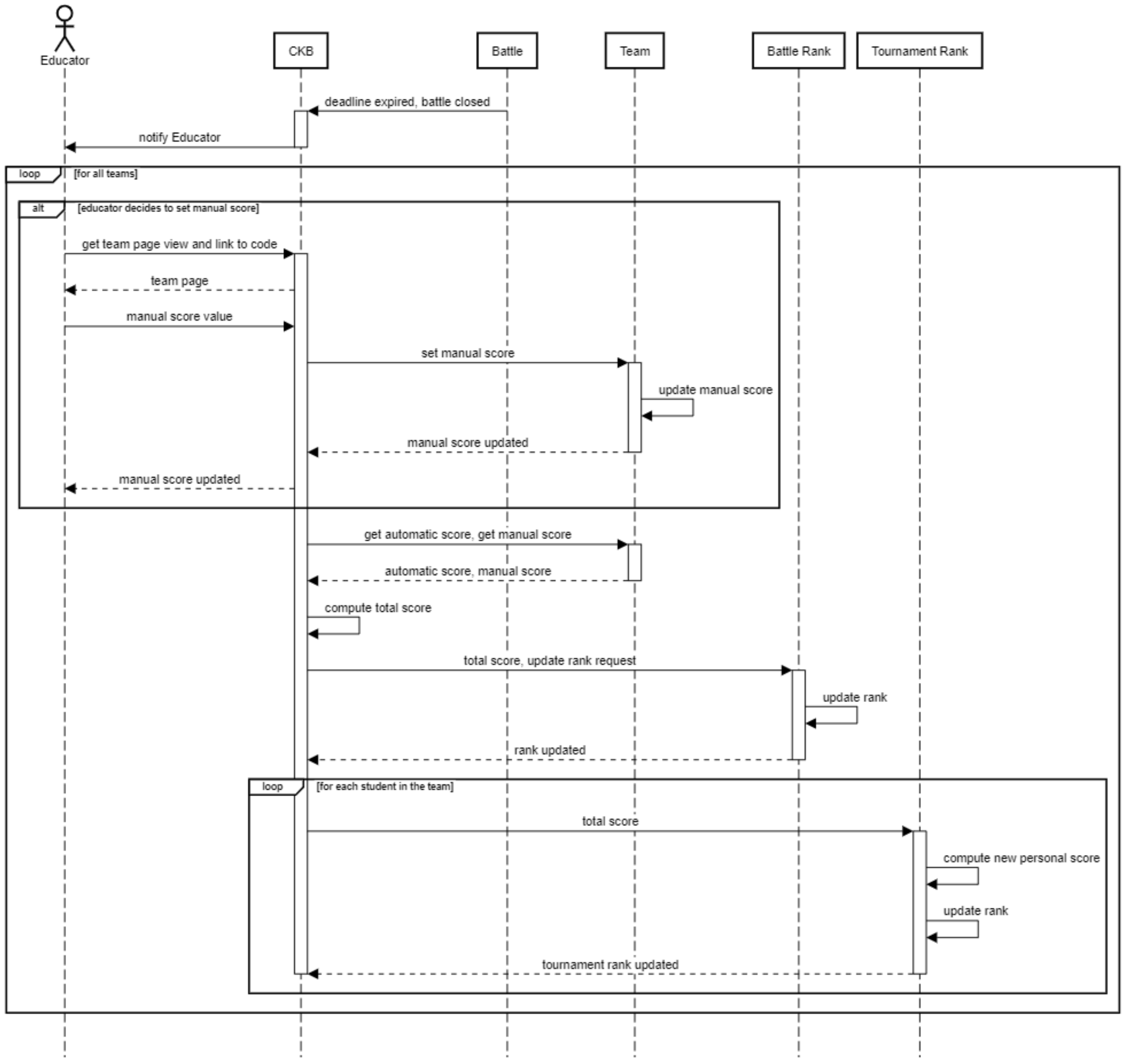


Figure 20: The deadline of a battle expires, closing the battle

UC17: Close a tournament

Actors	Educator
Entry condition	The educator that created the tournament decides to close it.
Event Flow	1. The eduactor clicks on the ”close tournament” button 2. The request is forwarded to the tournament, which sets its status as closed. 3. The tournament rank cannot be modified, so the application retreives it for publishing. 4. The final rank is publisked on the tournament page. 5. All students subscribed to the tournament are notified.
Exit Condition	All students subscribed to the tournament are notified of the final rank.
Exception	• There are still ongoing battless in the tornament. In this case the educator is notified and the tournament cannot be closed. • An educator different from the one that created the tournament tries to close the battle. This option shouldnt be provided to an educator which is not the creator of the tournament • An error occurs before the publishing of the final rank and the notification of students. • Internet connection is lost before all operations are completed.

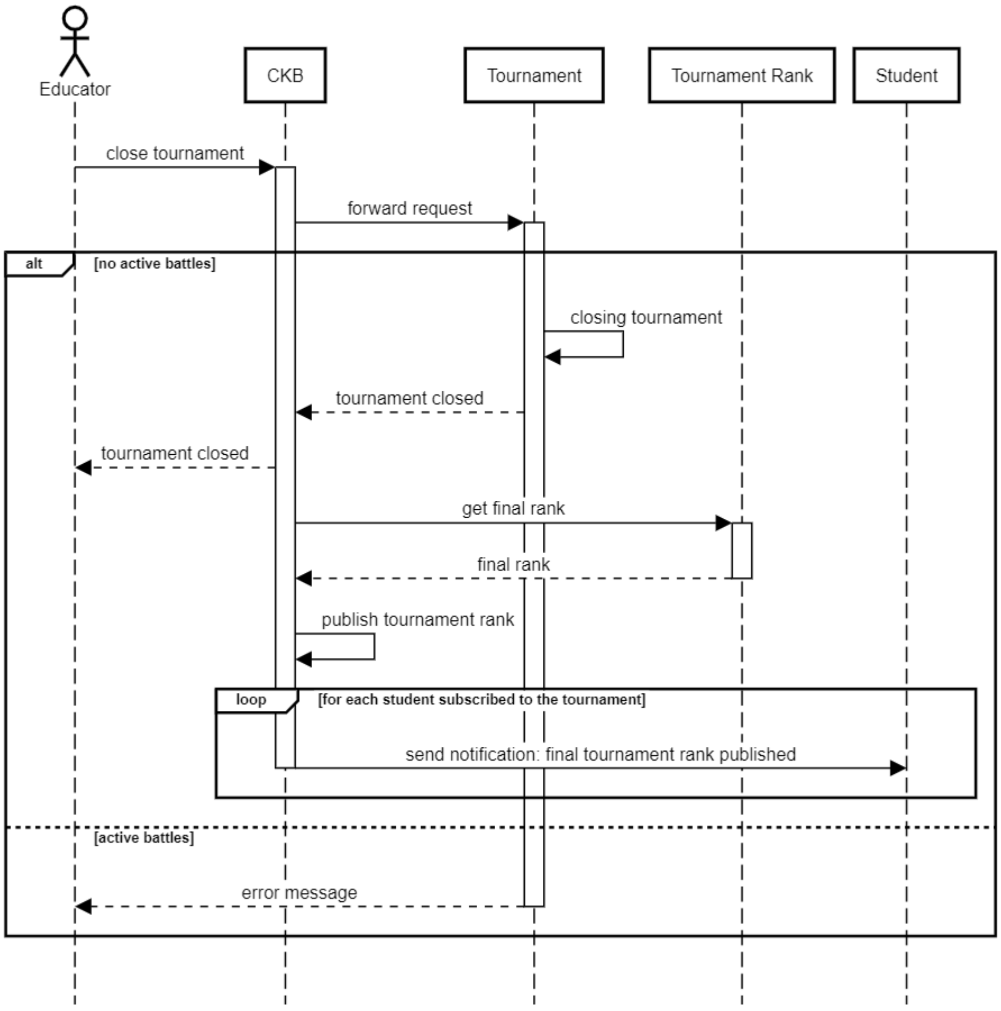


Figure 21: Educator closes a tournament

The **Use-Case Diagram** that represents all stated use cases is the following:

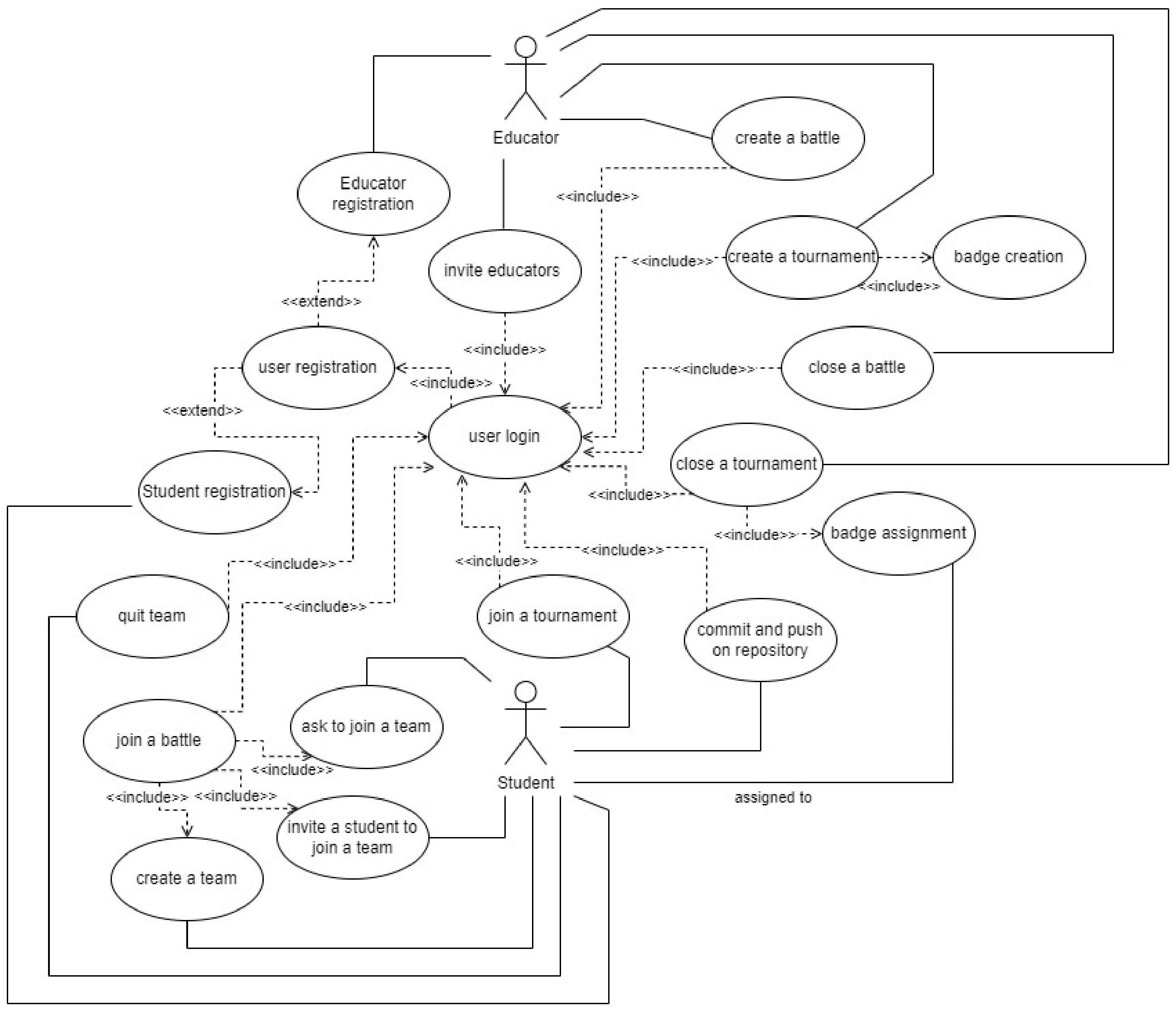


Figure 22: Use Case diagram

The following table represents the mapping between Use Cases and Functional Requirements:

Use Cases	**Requirements**
UC1: Student registration	R1
UC2: User login	R2
UC3: Join a tournament	R10
UC4: Create a team	R17
UC5: Request to join an existing team	R20, R21
UC6: Invite a student to join a team	R18, R19
UC7: Quit a team	R22
UC8: Join a battle	R23
UC9: Commit and push on GitHub repository	R24
UC10: Badge assignment	R29
UC11: Educator registration	R1
UC12: Create a tournament	R3, R4, R5, R9
UC13: Create a badge	R6
UC14: Create a battle	R11, R12, R13, R14, R15, R16
UC15: Invite educators to manage the tournament	R7, R8
UC16: Close battle	R25, R26
UC17: Close a tournament	R27, R28

The following table shows the mapping between Goals, Functional Requirements, Domain Assumptions and Dependencies.

Goals	**Functional Requirements**	**Domain Assumptions**	**Dependencies**
G1	R1, R2, R9, R10, R16, R17, R18, R19, R20, R21, R22, R23	D1, D2, D4, D5, D6, D8, D12	De1, De2, De3
G2	R2, R24	D2, D8, D9	De1, De2, De3
G3	R2,	D2, D8, D11, D13	De1, De2, De3
G4	R2, R15, R25, R26	D2, D9	De2, De3
G5	R2, R24, R28, R29	D2, D10	De2, De3
G6	R1, R2, R3, R4, R5, R6, R7, R8, R11, R12, R13, R14, R15, R27	D1, D2, D3, D7, D11, D12, D13, D14, D15	De1, De2, De3

The following list shows the **Functional Requirements** of the CKB system:

R1 : The system must allow an unregistered user to register an account with personal informations and define a username.

R2 : The system must allow a registered user to login into his account.

R3 : The system must allow registered educators to create a new tournament.

R4 : The system must allow registered educators to write a description for the tournament during a tournamnet creation process.

R5 : The system must allow registered educators to set a subscription deadline during a tournament creation process.

R6 : The system must allow registered educators to define badge's names and rules during a tournamnet creation process.

R7 : The system must allow registered educators to give permission to other educators to create battles in the tournament they have created.

R8 : The system must allow registered educators to accept or deny the permission for the creation of battles by another educator.

R9 : The system must allow registered students to be notified (also via e-mail) when a new tournament is created.

R10 : The system must allow registered students to subscribe to a new tournament until the registration deadline expires.

R11 : The system must allow registered educators with permission to create a new battle within an existing tournament.

R12 : The system must allow registered educators to upload the code kata during a battle creation process.

R13 : The system must allow registered educators to set minimum and maximum team's members number during a battle creation process.

R14 : The system must allow registered educators to set a submission deadline and a finish deadline during a battle creation process.

R15 : The system must allow registered educators to set scoring parameters during a battle creation process.

R16 : The system must allow registered students to be notified (also via e-mail) when a new battle is created in a tournament they are involved.

R17 : The system must allow registered students to form teams for a battle.

R18 : The system must allow registered students to invite other students to join their team.

R19 : The system must allow registered students to to accept or deny the invite by another student to be part of his team.

R20 : The system must allow registered students to send a request to join an existing team for a battle they want to join.

R21 : The system must allow registered students to accept or deny a request to join their team by another student.

R22 : The system must allow registered students to quit their team before joining the battle.

R23 : The system must allow registered students to join battles within tournaments they're subscribed to.

R24 : The system must allow registered students to track their score every time they perform a push on the GitHub repository of their project.

R25 : The system must allow registered educators to perform the manual evaluation of the codes and assign a personal score during the consolidation stage.

R26 : The system must allow registered students to be notified (also via e-mail) when the final battle rank is available.

R27 : The system must allow registered educators to close tournaments (if they are the one that have created the tournament).

R28 : The system must allow registered students to be notified (also via e-mail) when the final tournament rank is available.

R29 : The system must allow registered users to visualize badges obtained by the students (in their personal profile and also in the final tournament rank).

3.3 Performance Requirements

As far as performance is concerned, the system must be able to satisfy its users needs at any time of the day. In general, we can formulate the following performance-related non-functional requirements:

Nfr1 : The system must be available 99.9% of the time in a year.

Nfr2 : If the system goes down, it must recover in less than 15 minutes.

Nfr3 : The system must be able to link with external services in less than a minute.

Nfr4 : The system must be able to fullfill the requests of the educators in less than a minute.

Nfr5 : The system must be able to fullfill the requests of the students in less than a minute.

Nfr6 : The system must be able to support the simultaneous connection of at least 100.000 users.

3.4 Design Constraints

3.4.1 Standards Compliance

The application must comply with the standards of security, privacy or other particular directives imposed in the states where it will operate. The application must be usable on the most popular browsers, therefore it must comply with the standards of those browsers. The application must respect the rules of the external APIs it interfaces with.

3.4.2 Hardware Limitations

To be used, the application requires that the user's device has access to the internet.

3.5 Software System Attributes

3.5.1 Reliability

The system must be able to tolerate possible failures that would increase downtime. To do so, a backup system should be created, that allows the replication of the data contained in the main server but also of the processes that provide the services of the system. If the system has a scheduled maintenance, the system must notify the users at least 2 days before. In any case, even if the system goes down unexpectedly, it must notify the users that it is back up running.

3.5.2 Availability

We deem that an availability of 99.99% for the system is reasonable, which would corresponds to roughly 52 minutes/years of downtime. To allow such availability the system should have a server running parallel to the main one so, in case the central server goes down, it can do his stead.

3.5.3 Security

The personal informations of the users should be protected. To do so, encryption is applied to the login credentials of each user, in this way the system is able to prevent unauthorized accesses to confidential information. The database should be encrypted at rest and in transit. Encrypt all database connections using protocols such as the Transport Layer Security protocol, protecting data in transit.

3.5.4 Maintainability

The system should be implemented by using design patterns that allows the code to be reusable and easily modified in case of future extension or malfunctioning. It should also be implemented using an object oriented language and by documenting most of the classes and methods, so that it easy to read and understand for other engineers in case of need.

3.5.5 Portability

The system must be compatible with different operating system such as Android, iOS for mobile devices and Windows, Linux and Mac for the desktop.

4 Formal Analysis Using Alloy

This section is dedicated to the static modeling of the CKB platform, performed using the `Alloy` language (alloytools.org). The goal of this analysis is to show the relations between the most important components of the CKB platform. In particular, the focus on the analysis is to represent a "snapshot" of the system in which are present two `Tournaments`, some `Battles`, and the corresponding `Students` participating to the battles. The "snapshot" is taken after the battles are concluded (and hence their score is assigned to the students).
In the following, a list of the most important properties is provided.

1. A `Battle` is associated to exactly one `Tournament`.
2. Only `Students` subscribed to a `Tournament` can participate to `Battles`.
3. A `Team` is formed in the context of a specific `Battle` and must comply to the battle criteria w.r.t. the number of participants.
4. A `Student` cannot participate to the same battle with multiple `Teams`
5. The tournament score for each `Student` (contained in the `StudentStats` entity) must be equal to the sum of the scores obtained by the `Student` in the battles in which he participated

The Alloy code is included below, and in Figure 23 a graphical representation of the Alloy World is provided.

```

    // Signatures code

    sig Student {}
    sig Educator {}

    sig StudentStats {
      student: Student,
      score: Int
    }{
      score > 0
    }

    sig Tournament {
      students: set Student,
      educators: set Educator,
      scores: set StudentStats,
      battles: set Battle
    }{
      #educators > 0
      #students > 0 //the Tournament must have at least one Student enrolled
      #battles > 0 // the Tournament must have at least one Battle
      #scores = #students
    }

    sig CodeKata{}

    sig Battle {
      creator: Educator,
      teams: set Team,
      maxNumOfStudents: Int,
      minNumOfStudents: Int,
      codeKata: one CodeKata

    }{
      #teams > 0
      maxNumOfStudents >= minNumOfStudents
      maxNumOfStudents > 0
      minNumOfStudents > 0
    }

    sig GitHubRepo {}

    sig Team {
      members: set Student,
      repo: GitHubRepo,
      score: Int
    }{
      score > 0
    }


    // Facts code

    // Every Battle must exist in the context of a Tournament
    fact noTournamentlessBattle {
      no b: Battle | not b in Tournament.battles
    }

    // Every Battle must be part of one and only one Tournament
    fact battleBelongsToOneTournament {
      all disj t1, t2: Tournament, b: Tournament.battles |
      b in t1.battles and b in t2.battles iff t1=t2
    }

    // Each Battle has its own CodeKata
    fact codeKataBelongsToOneBattle {
      all disj b1, b2: Battle, ck: Battle.codeKata |
      ck = b1.codeKata and ck = b2.codeKata iff b1=b2
    }

    // An Educator can create a Battle if he has been granted access
    // (i.e. he is part of the educators for one specific Tournament)
    fact educatorCanCreateBattle {
      all t: Tournament, e: Educator, b: t.battles |
        e in b.creator iff e in t.educators
    }

    // Every Team must exist in the context of a Battle
    fact noBattlelessTeam{
      no t: Team | not t in Battle.teams
    }

    // Every Team must participate in one and only one Battle
    fact TeamBelongsToOneBattle {
      all disj b1, b2: Battle, t: Battle.teams |
      t in b1.teams and t in b2.teams iff b1=b2
    }

    // Every Team must comply to its Battle criteria w.r.t. the number of
       participants
    fact NumOfParticipants {
      all b: Battle, t: b.teams |
        #t.members <= b.maxNumOfStudents and #t.members >=
           b.minNumOfStudents
    }

    // Every GitHubRepo must exist in the context of a Team
    fact noRepoLessTeam{
      no g: GitHubRepo | not g in Team.repo
    }

    // Each Team has its own GitHubRepo
    fact gitHubRepoBelongsToOneTeam {
      all disj t1, t2: Team, g: Team.repo |
      g = t1.repo and g = t2.repo iff t1=t2
    }

    // A Student can participate to a Battle if he subscribed to a
       Tournament
    fact StudentCanParticipateToBattle {
      all t: Tournament, s: Student, team: Team |
        s in team.members implies s in t.students
    }

    // A Student cannot participate to the same battle with multiple Teams
    fact OneTeamForEachStudent {
      all b: Battle, disj t1, t2: b.teams, s: b.teams.members |
        s in t1.members iff not s in t2.members
    }

    // Only the students subscribed to a Tournament own a StudentStats
       signature
    fact noStudentStats {
      all s: Student | not s in Tournament.students implies not s in
         StudentStats.student
    }

    // There are no multiple StudentStats associated to the same Student
    fact oneStudentOneStudentStats {
      all t: Tournament, disj ss1, ss2: t.scores, s: t.students |
        s in ss1.student and s in ss2.student iff ss1=ss2
    }

    // Every student subscribed to a Tournament must have a StudentStats
       signature associated to that Tournament
    fact studentHasStudentStats {
      all t: Tournament, s: t.students |
        s in t.scores.student and #t.students = #t.scores
    }

    // Every StudentStats is associated to only one Tournament
    fact oneStudentStatsForEachTournament {
      all disj t1, t2: Tournament, ss: Tournament.scores |
      ss in t1.scores and ss in t2.scores iff t1=t2
```

```
}

// Every StudentStats must exist in the context of a Tournament
fact noTournamentlessStudentStats {
  no ss: StudentStats | not ss in Tournament.scores
}

// retrieves all the teams in a Tournament t where Student s is a member
fun getTeams[s: Student, t: Tournament]: set Team {
    let teams = {tm: Team | tm in t.battles.teams and s in tm.members}
       | teams
}

// returns the sum of the scores of a set of Teams
fun computeScore[teams: set Team]: Int {
    sum t: teams | t.score
}

// the score of a Student in a Tournament is equal to the sum of the
   Battle's score in which the Student has participated
fact score {
  all t: Tournament, s: t.students, ss: t.scores |
    //ss.student = s implies ss.score = sum[getTeams[s, t].score]
    ss.student = s implies ss.score = computeScore[ getTeams[s,t] ]
}


// World creation
run ckb {
  // costraints
  #Tournament = 2
  some t: Tournament | #t.battles > 3
  some s: Student | not s in Tournament.students
  #Student > 5
  some b: Battle | #b.teams > 1
  some t: Team | #t.members > 3

} for 3 but 5 Educator, 20 Student, 40 StudentStats, 10 Team, 10
   GitHubRepo, 5 Battle, 5 CodeKata
```

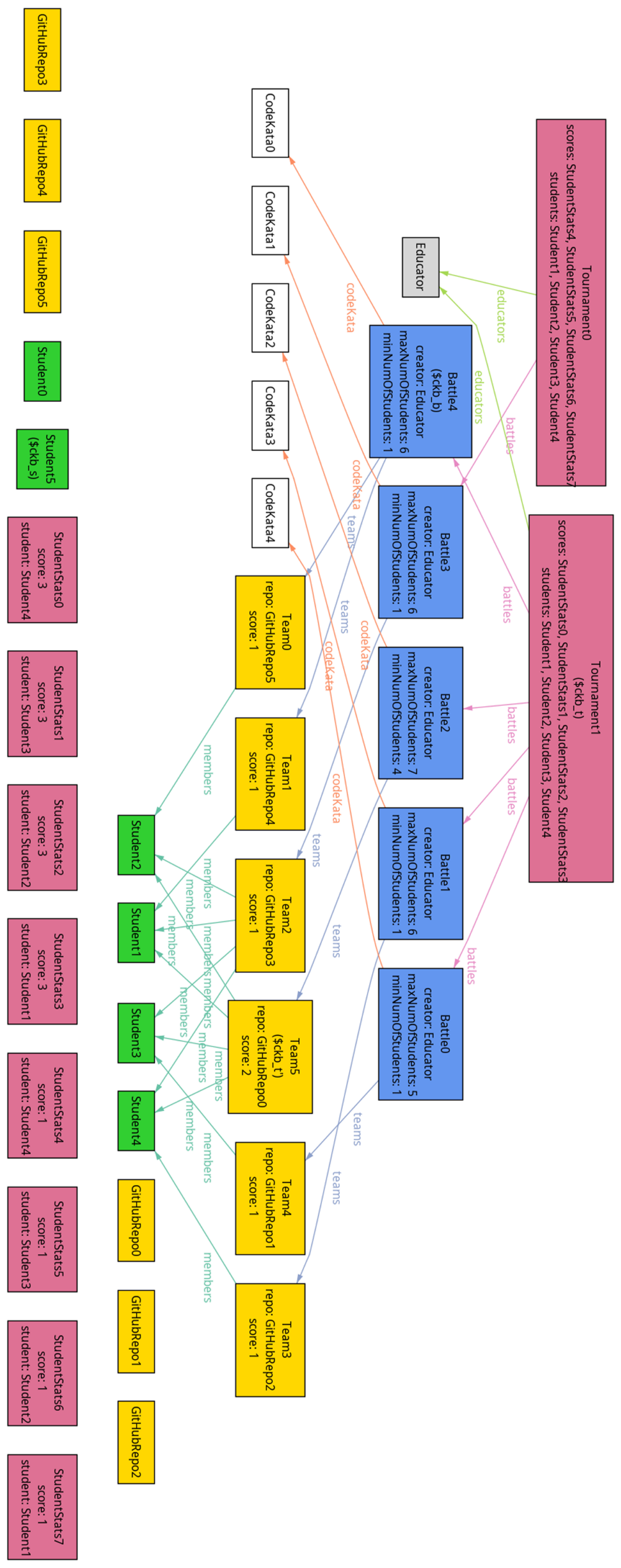


Figure 23: World generated by the Alloy Analyzer

5 Effort Spent

The following tables display the effort spent by each member of the group.

Daniele Gagni

Chapter	**Hours**
Introduction	9h (7h together)
Overall Description	21h (18h together)
Specific Requirements	15h (11h together)
Formal Analysis Using Alloy	12h (6h together)

Guglielmelli Isabella

Chapter	**Hours**
Introduction	11h (7h together)
Overall Description	24h (18h together)
Specific Requirements	18h (11h together)
Formal Analysis Using Alloy	9h (6h together)

Mariotti Marco

Chapter	**Hours**
Introduction	13h (7h together)
Overall Description	23h (18h together)
Specific Requirements	14h (11h together)
Formal Analysis Using Alloy	8h (6h together)

6 References

- SonarQube
- Alloy