POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software engineering II
Computer Science and Engineering

Requirement Analysis and Specification Document
Code Kata Battle

Authors:

Andaloro Emanuele 10692544
Galantino Claudia 11011392
Lew Deveali Simon 10986407

Academic Year:

2023-24

Deliverable: RASD
Title: Requirements Analysis Specification Document
Authors:
Andaloro Emanuele 10692544
Galantino Claudia 11011392
Lew Deveali Simon 10986407
Version: 2.0
Date: 21-December-2023
Download page: https://github.com/simonlew/AndaloroLewDevaliGalantino

Contents

1 | Introduction

1.1. Purpose

As the software industry grows, training better developers is becoming more and more important. Traditionally, developers are trained solely in theory and then thrust into working on a whole complex project without any previous practice. That is what code katas try to tackle. Code katas are short exercises that can be completed in minutes. Some involve programming while others involve thinking about the issues behind programming. In general, code katas are unlikely to have a single correct answer, but the main point of the kata is what you learn along the way. CodeKataBattle (CKB) is a new platform where Educators can create tournaments comprising multiple battles. Each battle consists of one particular code kata and aims to develop several skills. The idea is that students compete in these tournaments in teams to gain hands-on experience and improve their software development skills. The main goals of the platform are the following:

1.1.1. Goals

[G1] Educators create code kata battles

[G2] Students compete in multiple tournaments in teams

[G3] Students receive performance feedback after each battle assessment

1.2. Scope

In this section, we will try to identify CKB plataform domain. There are to main users that interact with the system: Educators and students.

Educators are the one who create the tournaments which consist of several battles grouped by context. For each battle, they will have to provide not only the code kata, but also the number of allowed participants and a deadline for both registration and submission. Another important feature is that they can allow other colleagues to add new battles to the tournament. In addition, to make the tournament a competition, they have to establish the scoring criteria. Lastly, for each tournament Educators will create badges which will be assigned to each students who fulfill some specific rule indicated into the badge itself.

Students will be able to enrol in one of these tournaments in groups. Later, the platform

will notify them when the competition starts and provide them with the repository with all the code necessary for the kata. Students have to fork this repository and then can start working in the solution. They will also have to set a Github action[1] so every push they commit to the repository during the competition is accounted for the computation of their score. Finally, after the deadline, the systems elaborate a rank considering their best score and the manual evaluation performed by the educator.

1.2.1. World Phenomena

[WP1] Educators think of a new kata, code the automation scripts and crate a test suite for the battle

[WP2] Students create a Github account

[WP3] Students fork the repository of the code kata provided

[WP4] Students code the kata in their preferred IDE and commit to their repository

[WP5] Educators think of a proper criteria to evaluate students work

1.2.2. Shared Phenomena

World controlled

[SP1] Students set up an automated workflow through github actions

[SP2] Educators create tournaments, battles and badges.

[SP3] Students invite others to join their team

[SP4] A member of the team register his team to participate in a tournament

[SP5] GitHub informs the CKB platform about new pushes in the students repository

[SP6] Educators assign their personal evaluation once the submission deadline expires

Machine controlled

[SP7] The system creates a GitHub repository and sends the link to the students belonging to teams enroled in a specific battle after the registration deadline.

[SP8] The system updates the score evaluation as soon a new push is performed

[SP9] The system assigns a badge to the students who satisfies the predefined rules and the students are able to see it

[1] We use the term action to match the assigment however we think that it should be WebHook

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

- **Static analysis tool** :Method of debugging that is done by automatically examining the source code without having to execute the program ensuring that is compliant, safe, and secure.
- **Dynamic analysis tool** :Process of testing and evaluating a programme thanks to running a test on the code.
- **OAuth Access Token** :An OAuth Access Token is a string that the OAuth client uses to make requests to the resource server.
- **WebHook** :A webhook is an HTTP-based callback function that allows lightweight, event-driven communication between 2 application programming interfaces (APIs). In this case it lets communication between Code Kata Battle platform and GitHub.
- **Consolidation** : Is a phase that occurs at the end of a tournament where the educator can decide to add a manual evaluation.

1.3.2. Acronyms

- CKB: Code kata battle;
- UI: User Interface;
- UML: Unified Modelling Language.

1.3.3. Abbreviations

- G*: goal
- WP*: world phenomena
- SP*: shared phenomena
- R*: functional requirement
- UC*: use case

1.4. Revision history

- Version 1.0 (20/12/2023)
- Version 2.0 (7/01/2024): integration of the user interfaces

1.5. Reference Documents

The document is based on the following materials:

- The specification of the RASD and DD assignment of the Software Engineering II course a.a. 2023/24
- Slides of the course on WeBeep

1.6. Document Structure

1. **Introduction:** it aims to give a brief description of the project. In particular it's focused on the reasons and the goals that are going to be achieved with its development;
2. **Overall Description:** it is an high-level description of how the system works with a detailed explanation of the phenomena that involve the world,the machine or both,there is also the domain description with its assumptions;
3. **Specific Requirements:** in this section there is a detailed analysis of the requirements needed to achieve the goals.Moreover, it contains more information useful for developers (i.e constraints about HW and SW);
4. **Formal analysis:** it's a formal description of the world phenomena by the means of Alloy;
5. **Effort spent:** it shows the time spent to realize this document organized by section;
6. **References:** it contains the references to any documents and software used to write this paper.

2 | Overall Description

2.1. Product perspective

2.1.1. Scenarios

Educator creates a tournament

The educator A wants to create a new tournament, he enters the KCB with his educator credentials, once on the homepage he can go into the section "create a new tournament" and fill out a form asking name, main context, enrolment deadline. Now he is able to create or add battles for it.

Educator creates a Battle for a specific tournament

The educator B, owner of a tournament 1, wants to create a battle for his tournament. He enters the KCB with his educator credentials, he goes into the tournament section, selects the tournament 1, and he can go into the "create new battle" section. In this section he can fill a form with the data of the battle such as the battle name, the code template, the registration deadline, the final submission deadline, the quality aspects of that battle which are used for the scoring and sets the minimum and the maximum number of students per group that can participate in that battle. The educator defines a set of categories such that :

-functional aspects: measured in terms of the number of test cases that pass out of all test cases;

-timeliness, measured in terms of the time passed between the registration deadline and the last commit;

-quality level:extracted through static analysis tools that consider multiple aspects such as reliability, security, and so on.

The educator gives to each category a weight.

Owner gives permission to another educator to add a Battle

The owner, the educator C, goes to the section dedicated to the tournament and gives permission to other educators to create new battles by pressing a button to share the control of a tournament. As a consequence the system shows the educator a bar that let him to search other educators. The educator can choose the other educators so an educator

D can create a new battle, filling the same form described in the previous scenario.

A student creates a team and invites others to join

Student E, who already has a profile on the platform, will receive a notification as soon as a new tournament is created. Once he is on the platform, he can read the main information about the tournament and decide whether to join it by the given deadline. He can create a team for a battle inviting other students, respecting the minimum number of student imposed for the battle of that tournament. To do that, the student will share with his teammate a link to join the battle.

A member of the team enrols in a battle

Student F wants to join her friend on a team for a battle, she received an email with the link that allows her to participate in the battle. She opens the link and thanks to her platform credentials she can join his friend's team. She knows that students cannot join more than one team for a specific battle. Once on the platform, she will be able to see in the tournament section the tournament she has registered in and the team she just joined.

The battle begins

As soon as the deadline expires, the platform creates the GitHub repository and for each team enrolled sends a notification with a link containing the code kata of the battle. The student G, who has enrolled into this battle, after receiving the notification opens the link and use his credentials to access the source code provided.

A member of the team pushes and commit to the kata repository

Student H who is enrolled in a battle, finishes to code the kata with his teammates and wants to submit it. In order to do that, he should fork the GitHub repository of the kata and set up the automated workflow if no one has done it before. Thanks to this action the platform will be triggered every time any of the team members pushes a new commit into the main branch of the repository. Afterwards he pushes and commits being able to see the new results on the platform.

The battle ends

At the end of the battle the platform updates the tournament score of every student enrolled, by recomputing the scores considering all battles belonging to that tournament.

Educator manually evaluates the code provided by students

Educator L who contributed to the creation of the tournament has the possibility to manually evaluate the work done by a team at the end of each battle. In order to do that, he can access the platform with his credentials, and go to the tournament section, then open the battle information and go through the sources produced by each team. Afterwards, he will assign a score that will be considered by the platform when the final

ranking is computed.

Users see the final rank of the tournament

Educator M closes the tournament thanks to the button 'end tournament' in the tournament section of the KCB platform. As a consequence of that, the platform elaborate the final rank for each student involved to that tournament and finally notify them about it. Student N who participated to the tournament and received the notification via e-mail can consult the final ranking by selecting the tournament into the 'tournament section' of the platform.

Users can see current rank

User J (an educators or a student) involved in a battle is allowed to see the evolving rank during the battles. They can consult the ranking into the dedicated section of the battle and compare students' performances. Every user has the possibility to consult the "ongoing tournament section"in which it is possible to see the list of the ranking for each tournament in progress.

Student receives a badge

When a tournament ends the platform assign the badges to the students that satisfied the requirements to obtain them. As previously mentioned, the criteria for assigning the badge are defined by the educators during the creation of the tournament. Any user subscribed to the platform can view the badges collected by any student so far. So student P who want to see the total number of badges can access the platform with his credentials and see them on his profile. He can also view the badges collected by his friend Student Q, in order to do that he can look for student Q profile and check the number of badges his friend has collected so far.

2.1.2. Domain class diagram

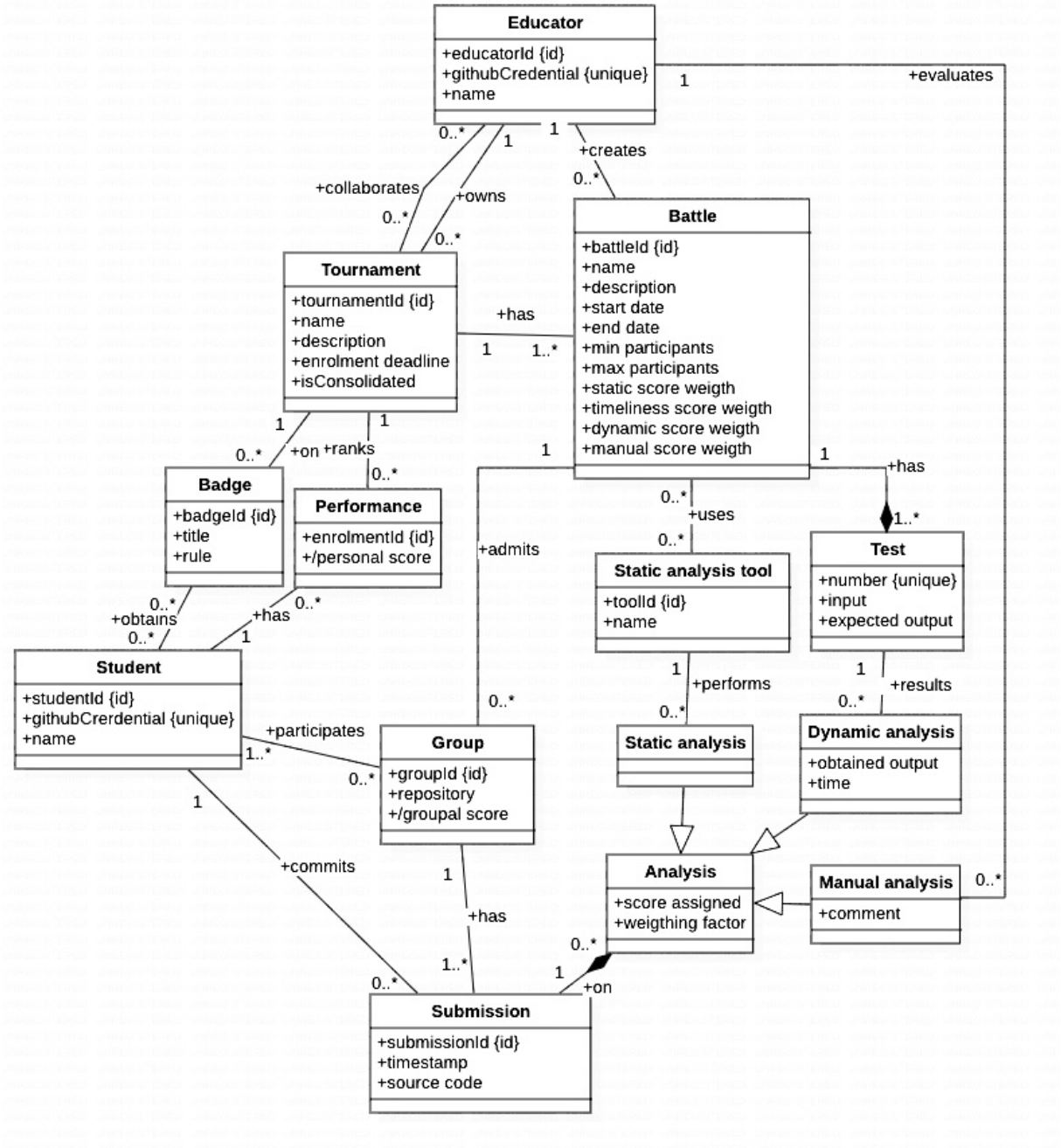


Figure 2.1: Domain class diagram

In Figure 2.3 you can find the domain class diagram. Despite both educators and students being user, we represent them separately in the diagram. As it was mentioned educators can own a tournament or simply collaborate in other's. The tournament has only one owner but can have many collaborators.

Tournaments consists of many battles which are created for that specific tournament. Then for the scoring it is necessary that the battle has the weights for each type of analysis

(static, timeliness, dynamic and manual). As well as the static tools to be used and the tests to evaluate that specific battle.

Afterwards, with all this different types of analysis it will be possible to perform the whole analysis on a particular submission. All the analysis regardless of the type will contribute on the score of the submission similarly. Note that Timeliness is not represented like the other analysis because the submission already contains the timestamp which is sufficient for that purpose. Although it could not be represented in the diagram, the weighing factor of the analysis, the tools used and the executed tests have to be consistent with the ones defined in the battle that submission corresponds. In the same way, the educator who evaluates should be the one responsible of that battle.

In order to participate in a battle students have to make a group this is only correspond to one battle and contains the Github repository from which submissions will be collected each time the source code is modified and committed. Then the score of the group will be taken from the best submission. In the diagram we could not represent that students can only participate only once in a battle, however this is relevant.

Finally, the tournament will have many badges which then will be assigned to the students following the rule criteria. This was not represented neither that students can only have badges for tournaments in which the took part.

2.1.3. State diagrams

In this subsection there is a discussion of the state charts diagram the ones for the management of the battles and the tournaments.Both of them are necessary for having a better understanding of how battles and tournaments works.

Tournament management

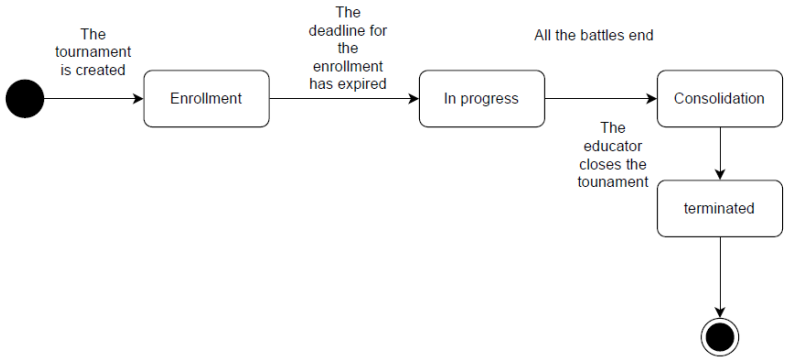


Figure 2.2: Tournament state charts

In this state charts diagram is explained how a tournament works and the different states that it has.In the *Enrollment* state the students enroll in the tournament and is the very first state after the educator has created the tournament. The *In progress* state represents the state in which the deadline for the enrollment has expired and the tournaments begins so the battle are going to be attended.The second-last state is *Consolidation* in which all the battles are ended.The last state is the *Terminated* in which the educator ends the tournament.

Battle management

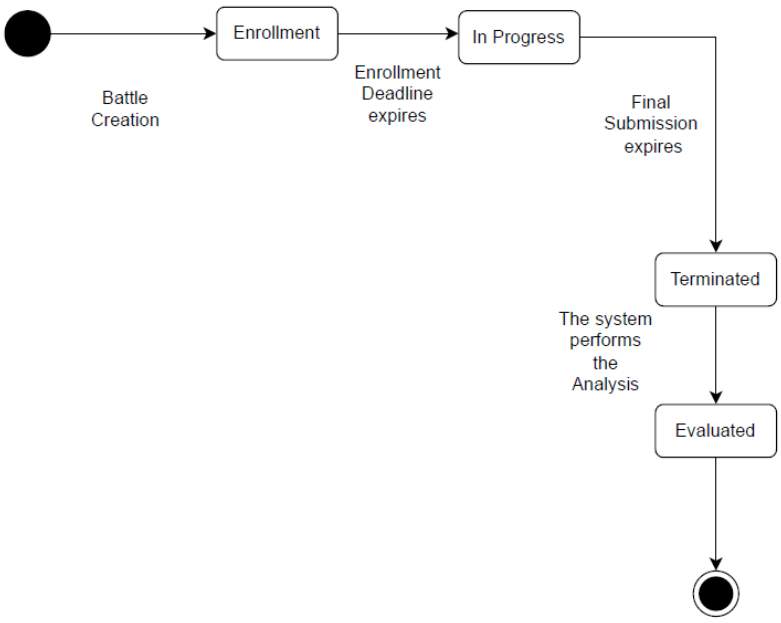


Figure 2.3: Battle state charts

In this state chart diagram is explained how a battle works and the different states that it has. In the *Enrolment* state the students enrol in the battle and it is the very first state after the educator has created the battle. The *In progress* state represents the state in which the deadline for the enrollment has expired and the tournaments begins so the battle is going to be attended.The second-last state is *Terminated* in which the battle is ended.The last state is *Evaluated* in which the system analysis has been performed and an evaluation has given either by the system or the educator.

2.2. Product functions

Sign up and log in

Sign-up is available to all users who want to subscribe to the platform. When a new user opens the platform, he will press the button 'Sign-in'. The user will then be redirected

to GitHub [1] where they are required to log in or create an account in order to link their GitHub account with our platform. Once this operation is completed, the platform will ask the user extra information, such as a username, so that the platform can create the user profile. Once the profile is created, the system allows the user to log into the platform for the first time using his GitHub account.[1]

Tournament and battles management

This function is only available to the educator. Their role is to create a tournament which consist of several battles that are related by the same topic. The first creator of the tournament allows his colleagues to contribute to the creation of the tournament by giving them the permission to add, modify, or delete battles for that tournament. The teacher should provide the students with the necessary materials to do the kata and should also define the minimum and maximum number of students allowed for the battle. Finally, he should specify the deadlines for both registration to the battle and final submission of the code.

Teams creation

This function is available to all student who wants to create a team and join a battle. Once the team leader has signed up for the battle, it has the possibility to share an invitation link with other registered students. The students who received the invitation link can join the team-leader and subscribe to the battle.

Scoring and ranking

This function is available to any educator who wants to score and rank a group of students for their coding abilities. after a battle the educator has the possibility to manually evaluate the team that has uploaded the code based on some criteria.

Badges

This function allows educator to creates badges which are a reward for students achievements. The criteria to assign the badges are decided by the educator during the tournament creation. Thanks to this function, every user can view the badges collected by each student by visiting their profile.

Tournament and battles participation

The students that are registered to the platform will receive a notification. Therefore, they can subscribe to the tournament after registering and will be informed of every incoming battle. For what concerns each battle the students can form a team by using the platform they should create a repository on github if they do not create it, the platform

[1]We decide to require users to use their GitHub account to sign. We already required students to have one to participate in the tournament, and this simplifies the process of checking who submitted the code, as each user is linked to one GitHub account. Also this decision could in the future allow us to seamlessly add more functionalities, such as creating the web-hooks or forking the kata automatically and others that will ease the work of educators as well

will create it after the registration deadline.Then, the students have to fork the repository and they can start working on their project creating a workflow on github.Every push triggers the platform that starts doing the tests

Tournament and battles consolidation

This function allows the system to evaluate the code pushed by each team and assign it a score. This score is updated every time a new push is made into the repository. When the battle ends, the educator who created the battle has the possibility to manually evaluate the code done by the teams for that battle by going through the sources implemented by the students. This phase is called consolidation because right after the manual evaluation the final score of the battle is calculated and all students who participate to it are notified about the ranking.

2.3. User Characteristics

There are mainly two types of user that interact with the platform: student and educator.

2.3.1. Student

The student is able to register and login to the platform and is also allowed to participate to a tournament after the tournament if his performances will be good he will receive a badge based on the rules that he fulfilled

2.3.2. Educator

The educator is a user who can create a tournament and the battle inside of it.This type of user can also evaluate students after the battle and create a badge that will be assigned to students who had the best performances.

2.4. Assumptions, dependencies and constraints

2.4.1. Regulatory policies

The CKB application asks for the GitHub account information. GitHub will not be used for commercial purposes. Personal information will be processed in compliance with the GPDR.

2.4.2. Domain Assumptions

The following assumptions are made for the domain. They are properties or conditions that the system will take for granted.They must be checked to ensure a correct platform behaviour

[D1] User must have a reliable internet connection

[D2] User personal information must be correct

[D3] Educators properly insert information about a tournament

[D4] The Github interaction it's reliable(the user is able to pull and push the code without losing its data)

[D5] Notifications to the user must arrive as soon as the final rank is available

[D6] The Educator properly insert an evaluation manually when it's requested

[D7] The educator correctly adds information about a new badge such as new rules or badge name

3 Specific Requirements

3.1. External Interface Requirements

3.1.1. User Interfaces

In this section is presented the UI of the web platform here are presented two type of user interfaces the one for the educator and the one for the student. The first one has the possibility to create tournaments and battles. The second one has the possibility to compete inside a tournament and to receive some badges that shows the abilities that the student himself has consolidated by using the platform.Both the two types of users need to insert their credentials to have access to the platform.Moreover a mechanism of forgot password it's needed just in case they lose their credentials.

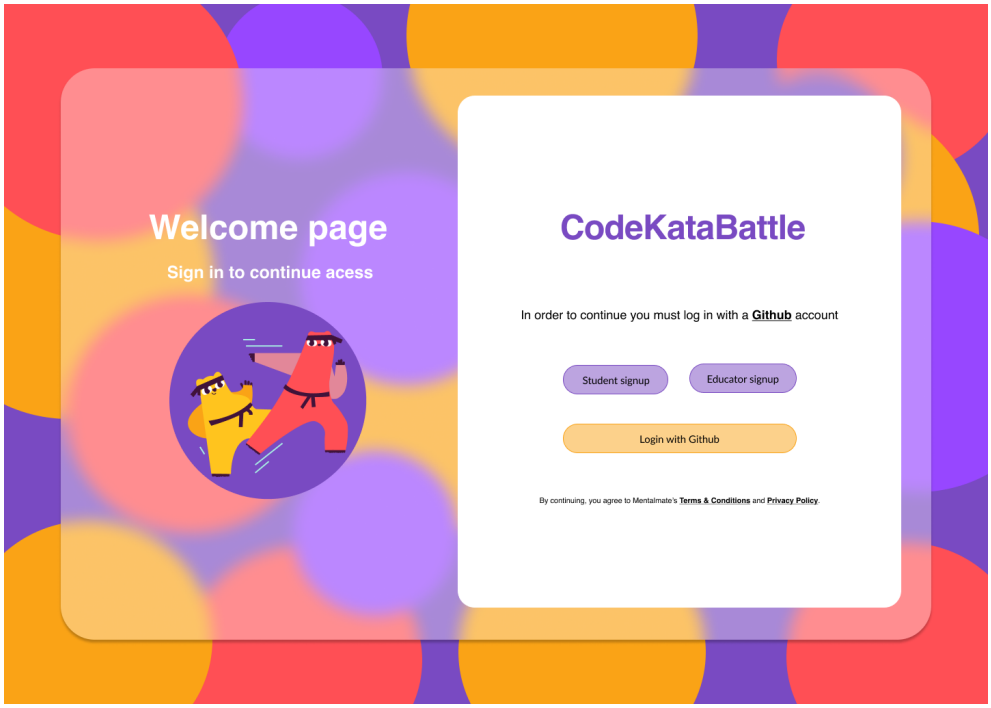


Figure 3.1: User login

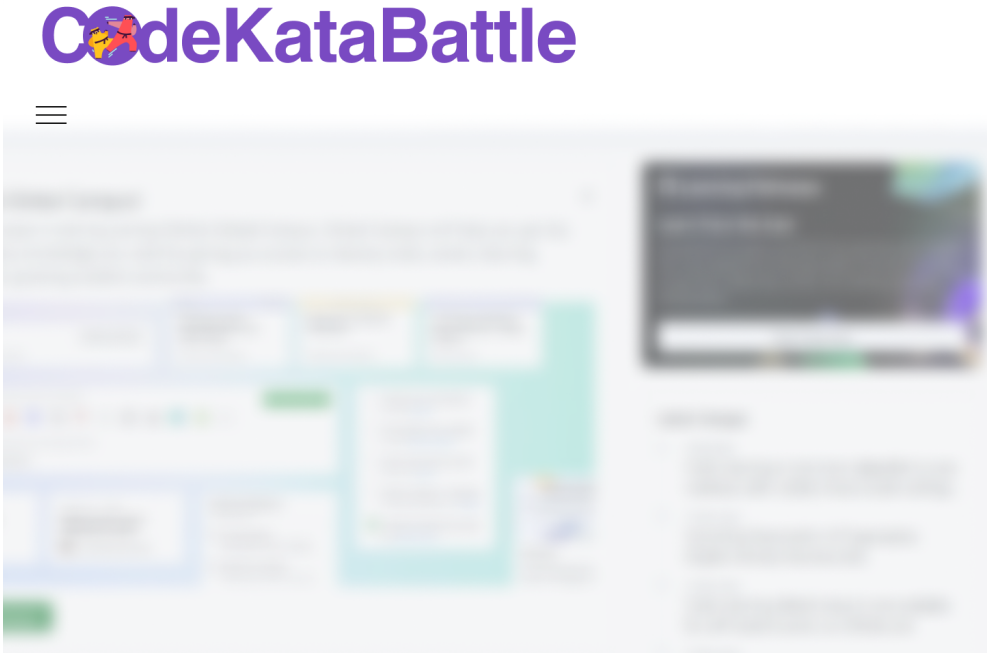


Figure 3.2: selection bar

Figure 3.3: menu

3.1.2. Hardware Interfaces

Our platform is a web app, as a consequence, it does not require any specific hardware interface except for computer and any other device with web browser.

3.1.3. Software Interfaces

In order to work the system needs some software interfaces. Here they are listed in detail:

- Github API:In order to interact with github, for user login and registration as well as to create and manage repositories;
- Static analysis tool API: To evaluate the submitted code, we will send a request to the static tool with the code and then we expect an answer with the assigned score.

3.1.4. Communication Interfaces

The user uses the internet connection to have access to the platform,to communicate with other user inside the platform and for pushing and pulling the code on github.The platform must be HTTPS compliant in order to work on the web properly and to be safe.

3.2. Functional Requirements

Sign up and log in

[R1] The System allows users[1] to register by providing their personal information (Full Name, etc.), a valid email address and a password.

[R2] The System allows registered user to log in

Tournament and battles management

[R3] The System allows Educators to create/modify a battle upload the code kata (description and software project, including test cases and build automation scripts)

[R4] The System allows to create/modify/terminate a tournament by selecting the existing battles, setting the minimum and maximum number of students per group, the registration and final submission deadline.

[R5] The System allows an educator to give or deny permission to his colleagues to modify a tournament.

[R6] The System must notify subscribed user about upcoming battles and deadlines.

Student's teams

[R7] The System allows students to create a team

[R8] The System allows students to invite other students into one of their teams

[R9] The System allows students to join a new team which they were invited

Scoring and ranking

[1] users is used to refer to students and educators

[R10] The system allows educators to define the scoring criteria for a specific battle which they have permissions to

[R11] The system maintains and computes the scores of each battle

Badges

[R12] Educators can create a badge and a set of rules associated with that badge

[R13] The system assigns the badges that are created by educators as a reward for the rules they fulfill

[R14] The system shows the badges that are assigned to students

Tournament and battles participation

[R15] The System creates a repository on GitHub containing the code kata right after the registration deadline

[R16] The system sends the link to all the enrolled students after creating the repository with the code kata

[R17] The System receives notifications from GitHub regarding the students registered repositories commits

[R18] The System pulls the repository after receiving a notification for that repository before the deadline of that battle

[R19] The System runs the appropriate test on the new code after every pull of the repository

[R20] The System calculate and update the team's score for that battle after rerunning the tests

Tournament and battles consolidation

[R21] The system updates the personal tournament score for each student enrolled in the tournament right after the battle ends

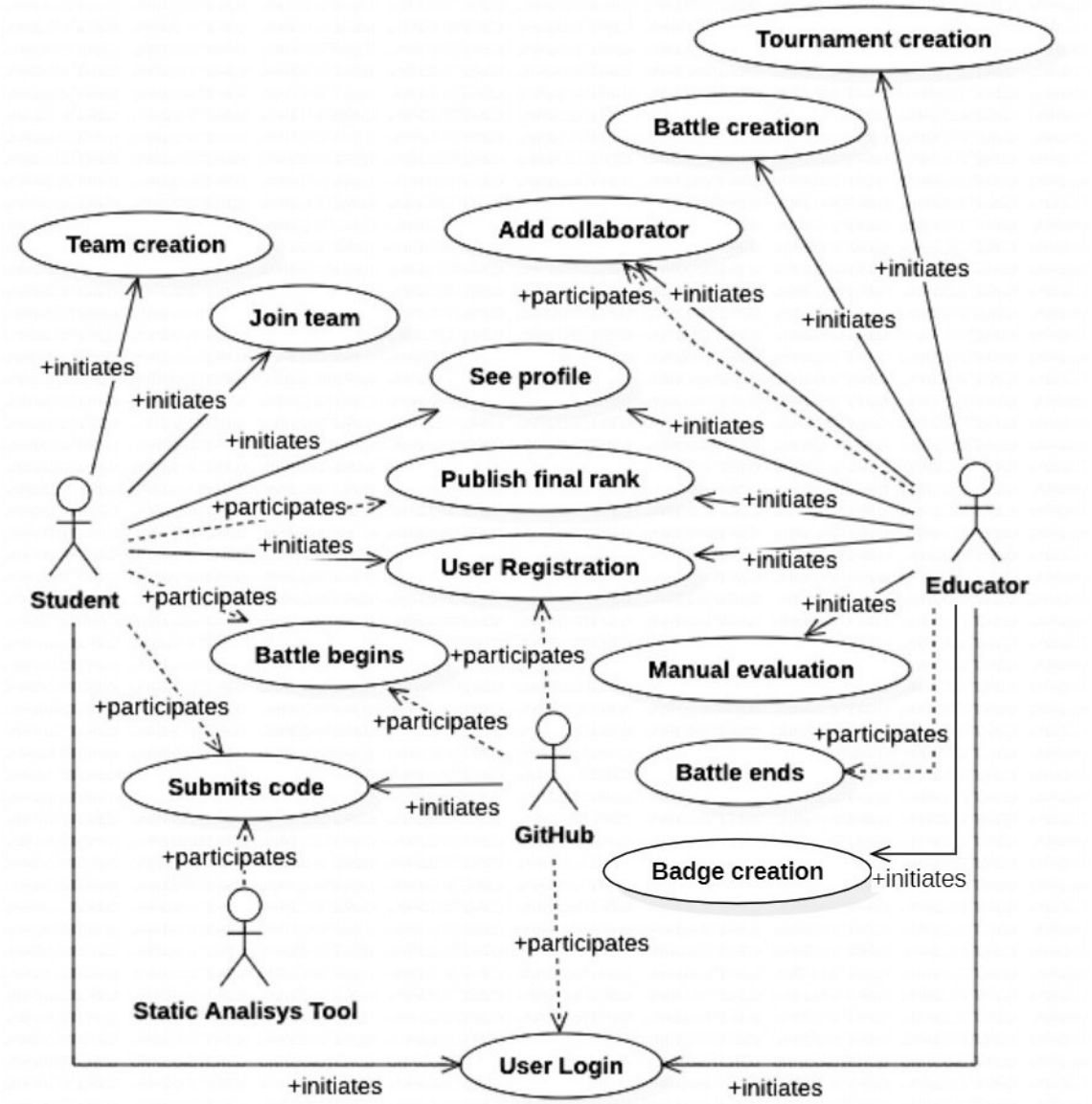
[R22] The system allows educators to manually evaluate the code after the deadline

[R23] The system allows educators to finish the consolidation stage after completely performing the manual evaluation

[R24] The system computes the final ranking of the tournament immediately after consolidation finishes

[R25] The system sends a notification about the tournament's termination to students

3.2.1. Use cases Diagram



3.2.2. Use cases

[UC1] -User Registration

Name	User Registration
Actors	• user • GitHub
Entry Condition	The user has opened the CKB platform
Event flow	(a) The user press the button Sign-in (b) The system redirect the user on GitHub to link their account to the platform (c) The system shows a form to compile (d) The user compile the form by adding a username (e) The user press the button 'Register' to complete the registration (f) The platform shows the login view
Exit condition	The user has successfully registered to the platform
Exception	(d) There already exist a user with those credentials.

[UC2] -User Login

Name	User Login
Actors	• user • GitHub
Entry Condition	The user has opened the CKB platform
Event flow	(a) The user press the button Log-in (b) The System redirect the user to GitHub platform to let him login to it, and checks the credentials (c) The user is redirected to the platform with an access token (d) The system shows the profile information (e) The user presses the button 'Login' (f) The platform shows the dashboard
Exit condition	The user has successfully accessed the service
Exception	(c) The token is not valid. The platform will return to the entry condition

[UC3] -Tournament creation

Name	Tournament creation
Actors	• educator
Entry Condition	The Educator is logged to the CKB platform
Event flow	(a) The educator goes to the tournament section (b) The educator press the button to create a new tournament (c) The system shows the educator the form to compile (d) The educator fills the form with the information about the tournament: name, main context, enrolment deadline (e) The educator press the button to submit the form
Exit condition	The educator has successfully created the tournament and returns to the tournament section
Exception	(d) The educator did not fill all the field of the form or they are not valid. The system will warn the user.

[UC4] -Battle creation

Name	Battle Creation
Actors	• educator
Entry Condition	The Educator is logged to the CKB platform and he already created a tournament
Event flow	(a) The educator goes to the tournament where he want to add the battle (b) The educator press the button to create a new battle (c) The system shows the educator the form to compile (d) The educator fills the form with the information about the battle with the following information: battle name, code template, registration deadline, final submission deadline, sets minimum and maximum number of students per group that can participate in that battle and the scoring criteria for that battle (e) The educator press the button to submit the form
Exit condition	The educator has successfully created the new battle and returns to the tournament main page
Exception	(d) The educator did not fill all the field of the form or they are not valid. The system will warn the user.

[UC5] -Add collaborator

Name	Add collaborator
Actors	• educator
Entry Condition	The educator is logged to the CKB platform and he is in the tournament section
Event flow	(a) The educator select the tournament (b) The educator press the button to share the control of a tournament (c) The system shows the educator a bar where to search other educators (d) The educator can choose the other educators
Exit condition	The educator has successfully shared the access to the tournament control
Exception	(d) The educator did not make the correct choice

[UC6] -Team creation

Name	Team creation
Actors	• student
Entry Condition	The student is logged to the CKB platform and is enrolled into a tournament
Event flow	(a) The student goes to tournament section and selects the tournament (b) The student select the battle for which create the team (c) The student press the button create a team (d) The system shows the student the form to compile (e) The student fills the form with the information about the team: team name (f) The system generate the link of the team (g) The student shares the link of the team with his friends
Exit condition	The link team is successfully generated
Exception	(c) The student has already created a team for that battle

[UC7] -Join team

Name	Join team
Actors	• student
Entry Condition	The student has the link to join a team
Event flow	(a) The student opens the link received (b) The system checks the student's credentials (c) The student press the button 'join the team' (d) The system checks if the team is not full and that the student is not already enrolled in the battle (e) The system redirect the student into the team dashboard
Exit condition	The student has correctly joined the team
Exception	(b) The student is not logged in, he will be redirect to the log in page (d) The system returns an error message

[UC8] -Battle begins

Name	Battle begins
Actors	• student • GitHub
Entry Condition	The enrollment deadline of the battle ended
Event flow	(a) The system creates a GitHub repository by using the appropriate API call for every enrolled team enrolled (b) The system sends a notification to the students enrolled to the battle with the link to the specifically created repository for them to fork
Exit condition	The code is available to all the students participating
Exception	(b) The notification is not correctly sent, the system will retry to send it later.

[UC9] -Submits code

Name	Student pushes the code
Actors	• student • GitHub • Static analysis tool
Entry Condition	The student has forked to the git hub repository provided by the platform and set up the automated workflow. The student has committed the code into their GitHub repository
Event flow	(a) The system is triggered by GitHub about a new commit (b) The system dynamically evaluates the code by running the test on it. (c) The system send the code to the static analysis tool (d) The system receives the evaluation by the tool (e) The system updates the score of the team.
Exit condition	The scoring rank is correctly updated
Exception	(a) The repository is not a fork of a code kata team, the system will ignore it and log it. (a) The system is triggered after the deadline of the battle, it will not evaluate the code and send a notification to the team warning them

[UC10] -The battle ends

Name	The battle ends
Actors	• educator
Entry Condition	the battle deadline has expired
Event flow	(a) The system creates the rank of the battle (b) The system sends a notification the the educators enrolled into the battle about the end of it
Exit condition	The ranking is available on the platform
Exception	(b) The notification is not correctly sent, the system will notify it and retry to send it

[UC11] -Code evaluation

Name	The code is evaluated
Actors	• educator
Entry Condition	The tournament ended, the consolidation phase started and the educator is logged to the CKB platform
Event flow	(a) The educator goes to the tournament section (b) The educator select the tournament which needs a manual evaluation (c) The educator reads the sources produced by the teams (d) The educator assign a score to each student enrolled to the battle
Exit condition	The system stores the score provided by the educator

[UC12] -Publish final rank

Name	Final rank made available
Actors	• student • educator
Entry Condition	The tournament is in consolidation phase, the educator is logged to the CKB platform
Event flow	(a) The educator goes to the tournament section (b) The educator selects the tournament to close (c) The educator press the button 'close the tournament' (d) The system computes the final rank for each student enrolled into the tournament (e) The system notifies the student about the availability of the rank (f) The system checks among the participating students who satisfied the badges requirement (g) The system assign the badge to the students
Exit condition	The final rank is published, and badges has been assigned
Exception	(c) If any student does not have a manual evaluation and it was required then an error message will be shown. (e) The notification is not correctly sent, the system will notify it and retry to send it

[UC13] -See profile

Name	see profile
Actors	• user
Entry Condition	The user is logged to the CKB platform
Event flow	(a) The user search the name of the student (b) The system shows the list (c) The user selects the Student
Exit condition	The platform shows the rank of the selected student
Exception	(a) The person searched does not match with any user in the system. It shows a no student found message

[UC14] -Badge creation

Name	Badge Creation
Actors	• educator
Entry Condition	The educator is logged on to the CKB platform and he already created a tournament
Event flow	(a) The educator goes to tournament section (b) The educator goes to the tournament where he want to add the badge (c) The educator press the button to create a new badge (d) The system shows the educator the form to compile (e) The educator fills the form with the information about the badge with the following information: badge name and the rule which has to comply with the syntax specified [2] (f) The educator press the button to submit the form
Exit condition	The educator has successfully created the new badge and returns to the tournament main page
Exception	(e) The educator did not fill all the field of the form or they are not valid (e.g. the rule is invalid). The system will warn the user.

[2]More details on the rules syntax will be provided in the DD

3.2.3. Sequence diagrams

[UC1] User Registration

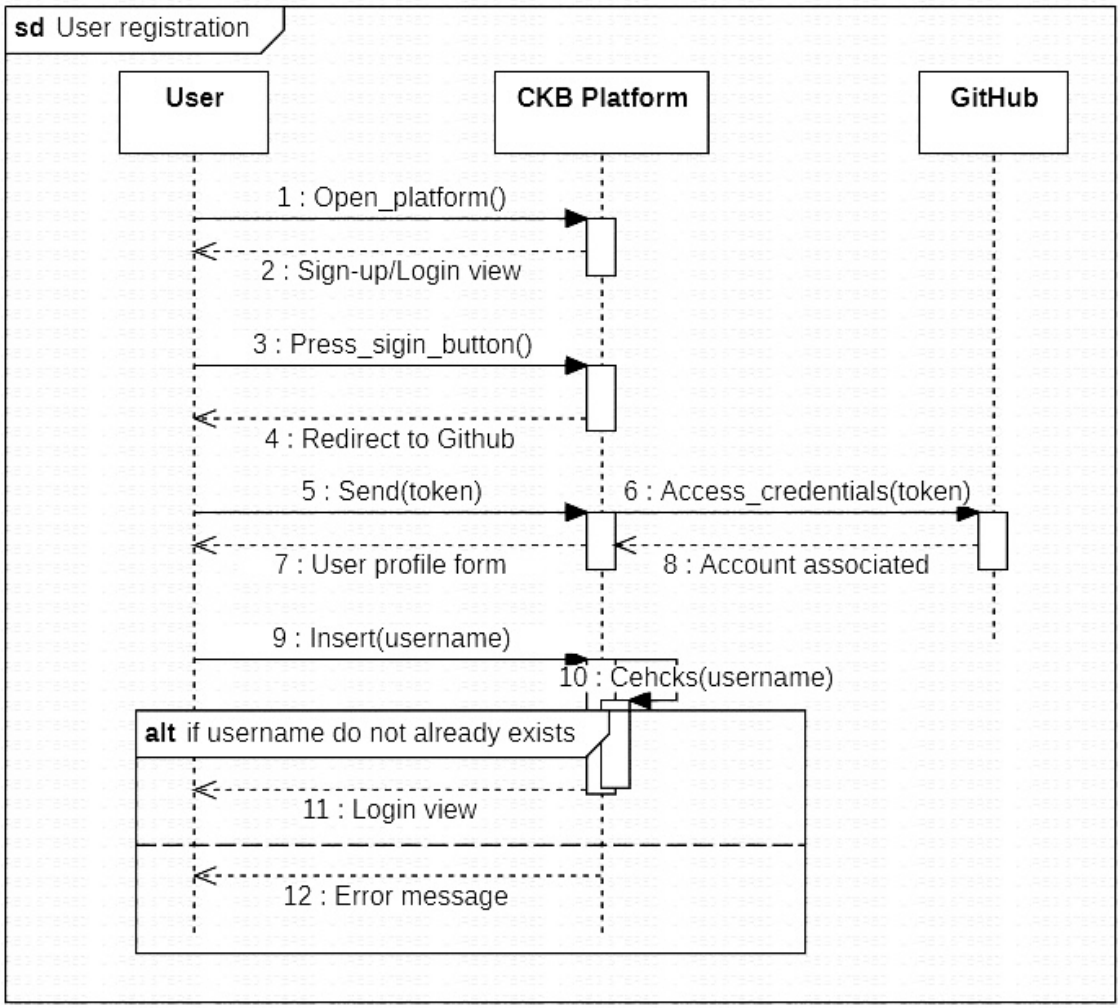


Figure 3.4: User Registration

[UC2] User Login

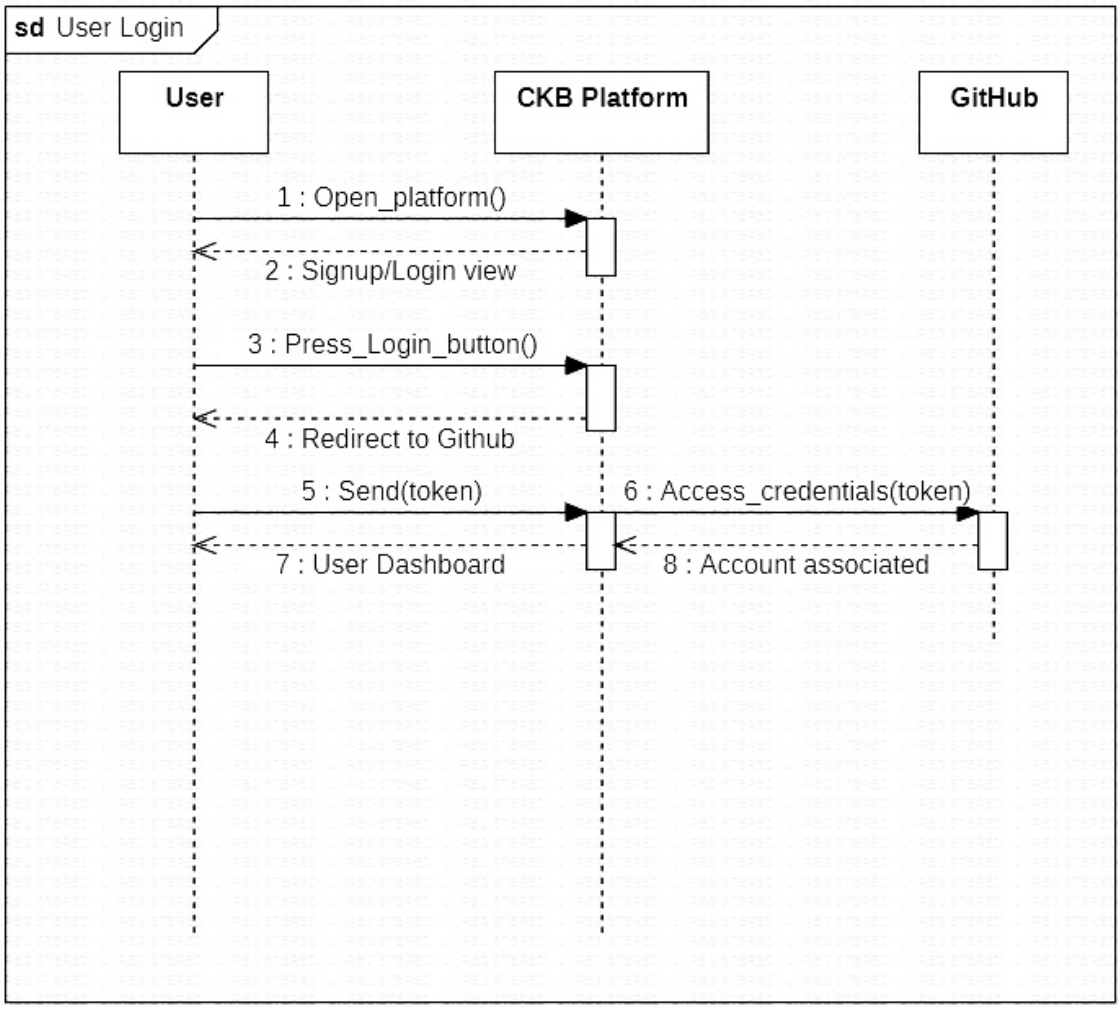


Figure 3.5: User Login

[UC3] Tournament Creation

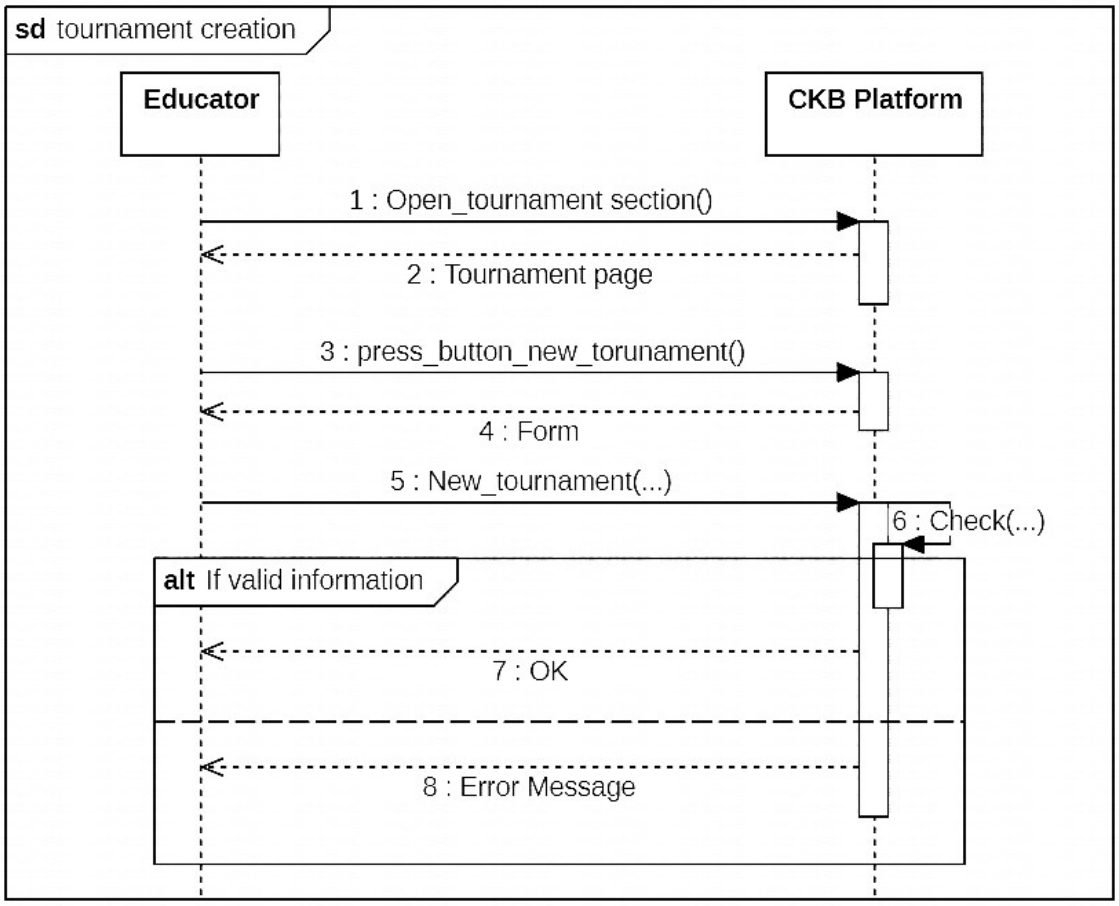


Figure 3.6: Tournament Creation

[UC4] Battle Creation

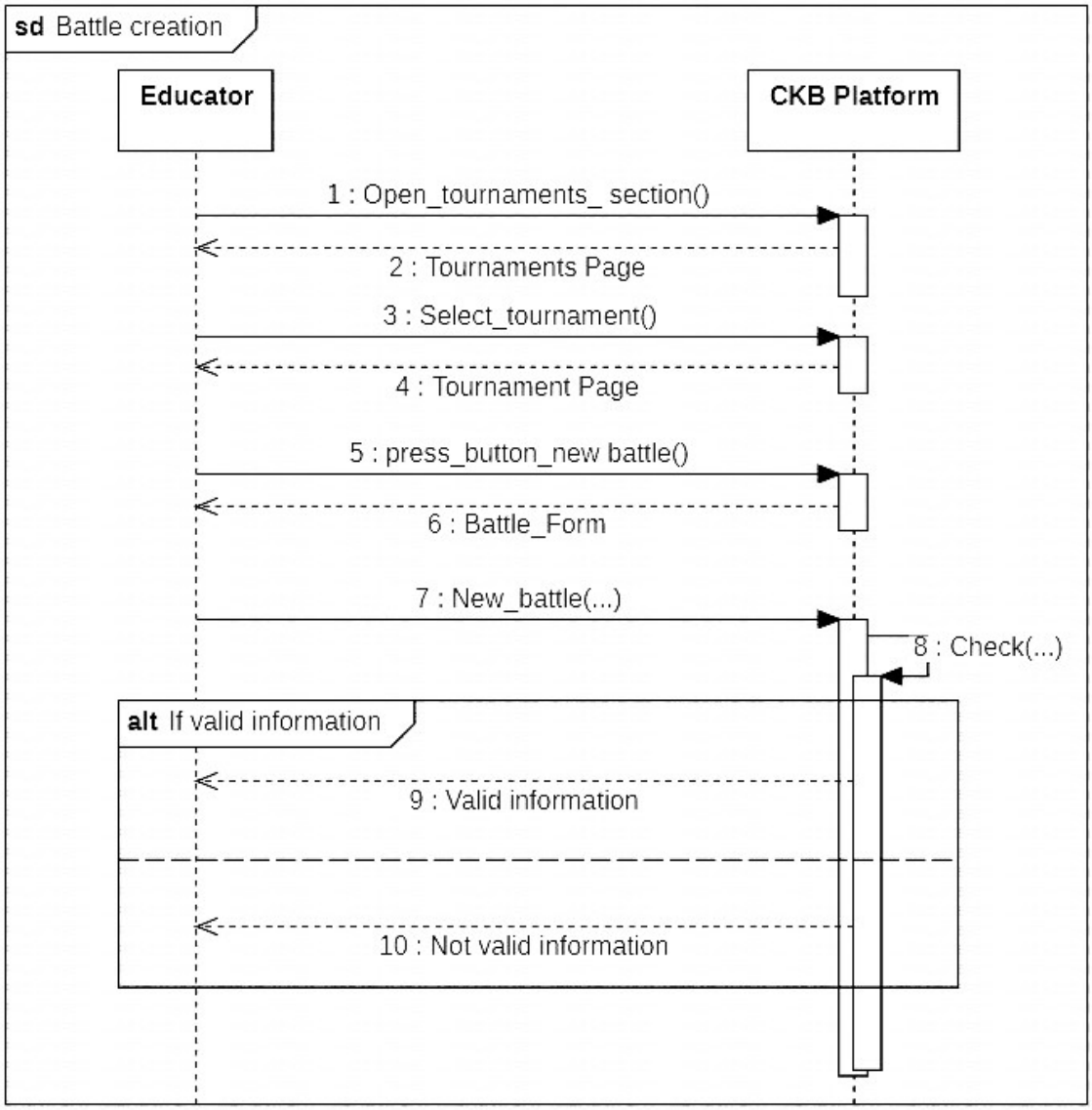


Figure 3.7: Battle Creation

[UC5] Add collaborator

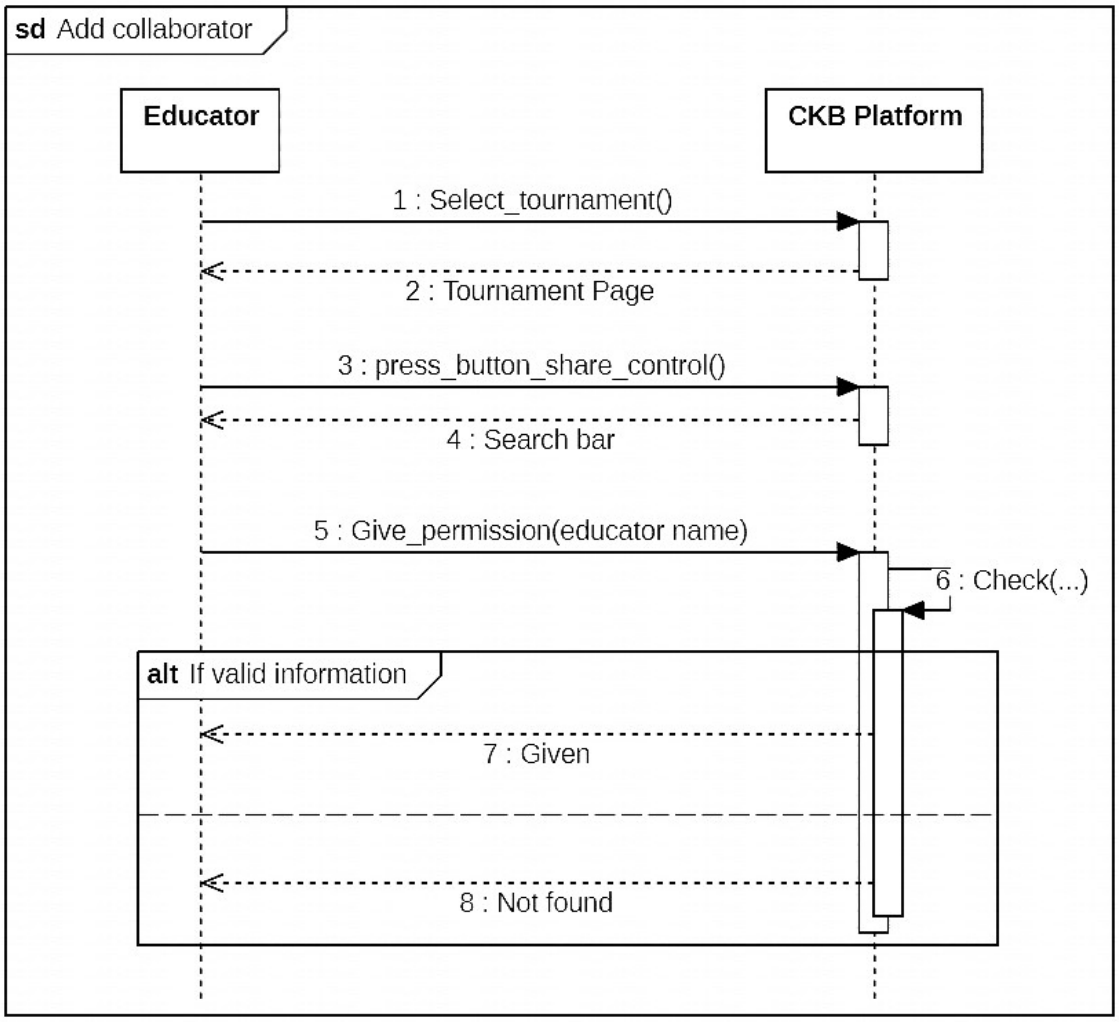


Figure 3.8: Add collaborator

[UC6] Team Creation

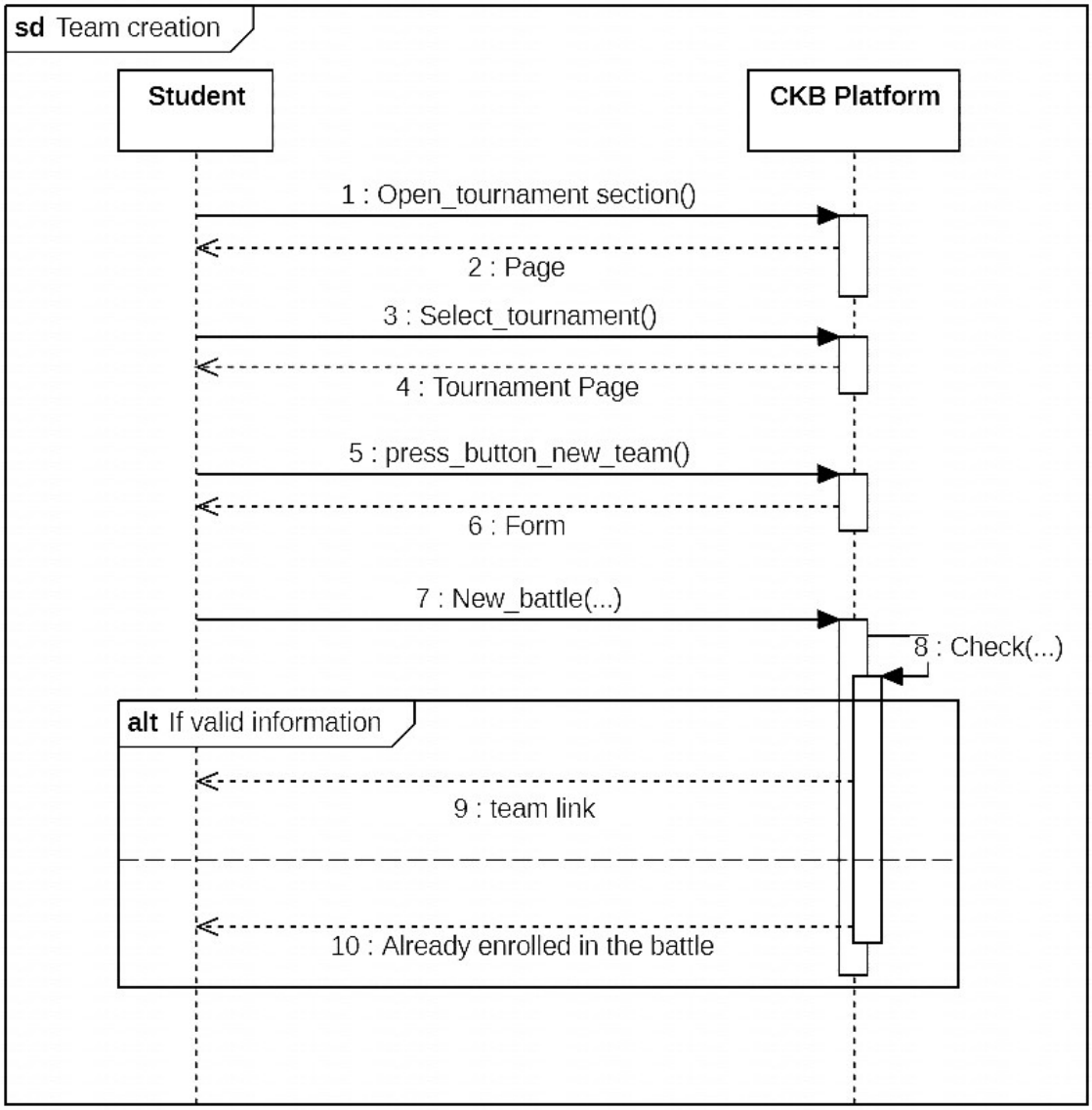


Figure 3.9: Team Creation

[UC7] Join Team

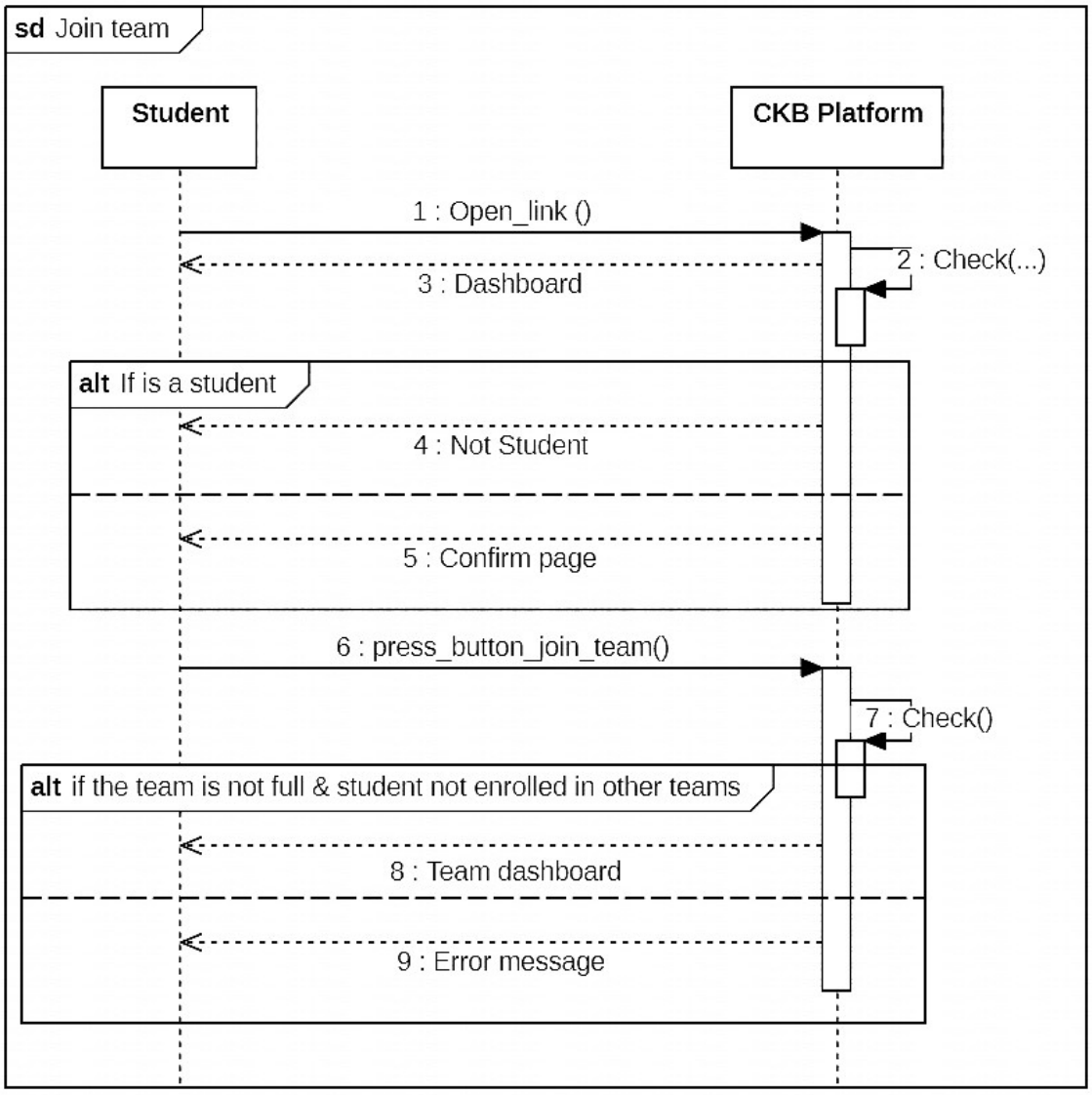


Figure 3.10: Join Team

[UC8] Battle Begins

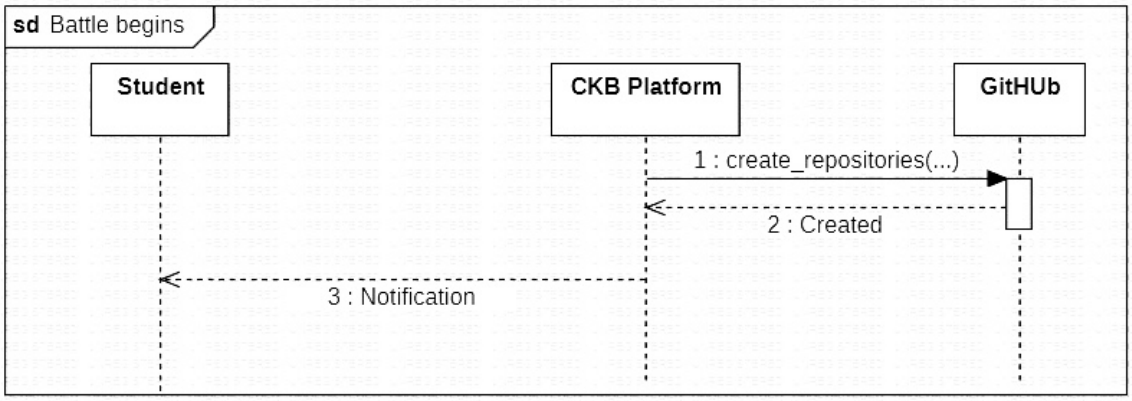


Figure 3.11: Battle Begins

[UC9] Submits code

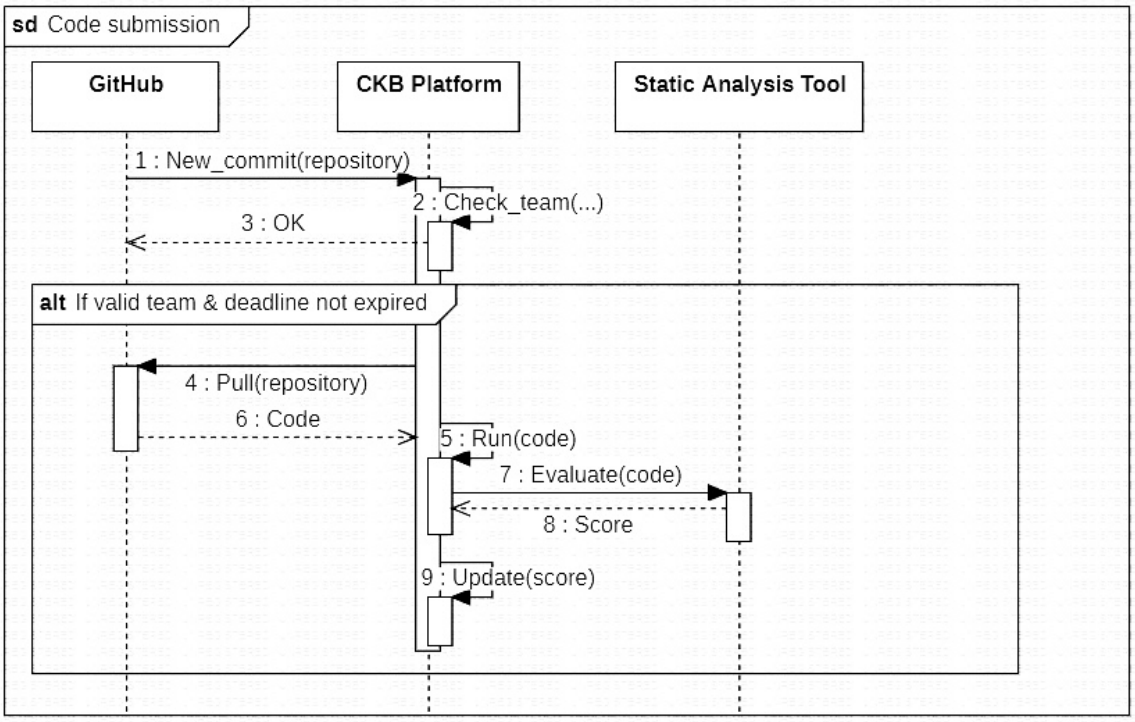


Figure 3.12: Submits Code

[UC10] The battle ends

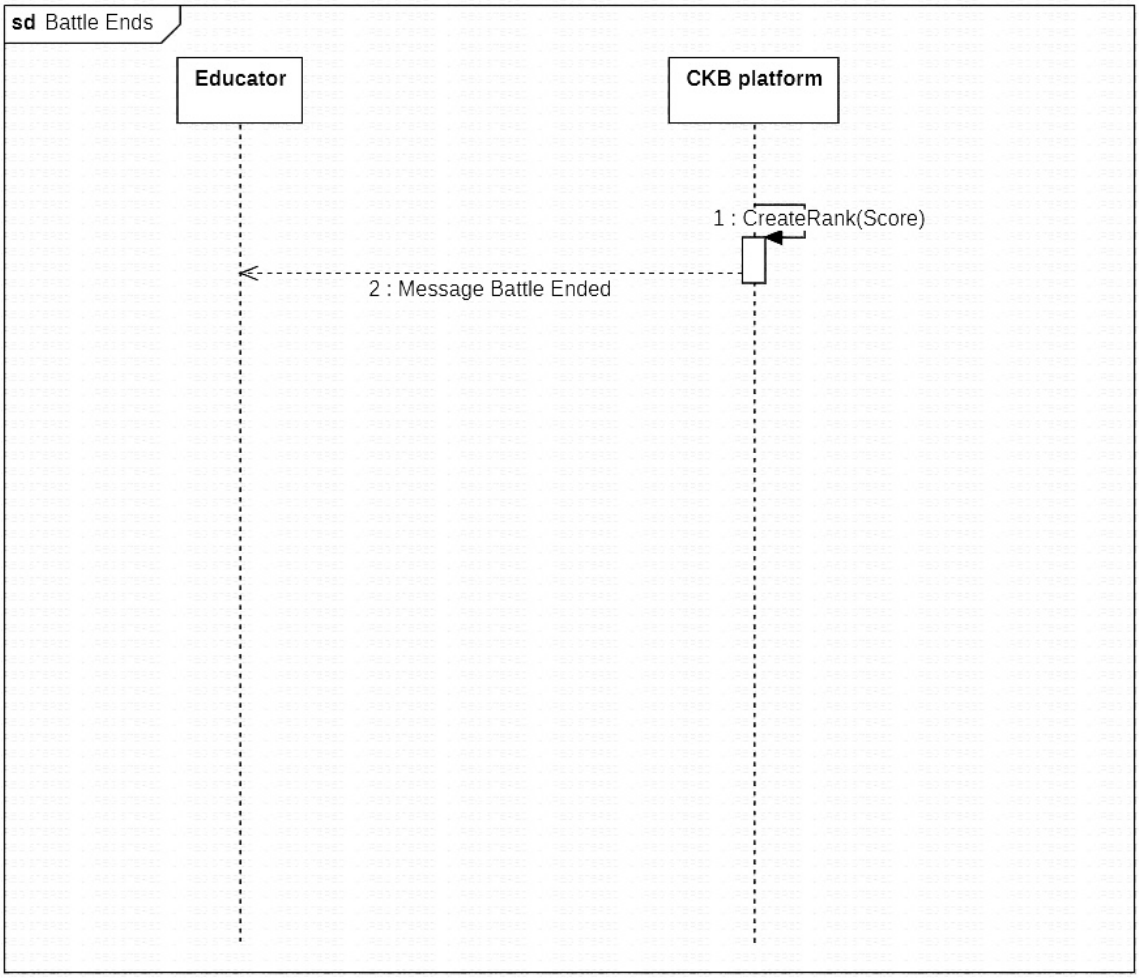


Figure 3.13: The battle ends

[UC11] Code Evaluation

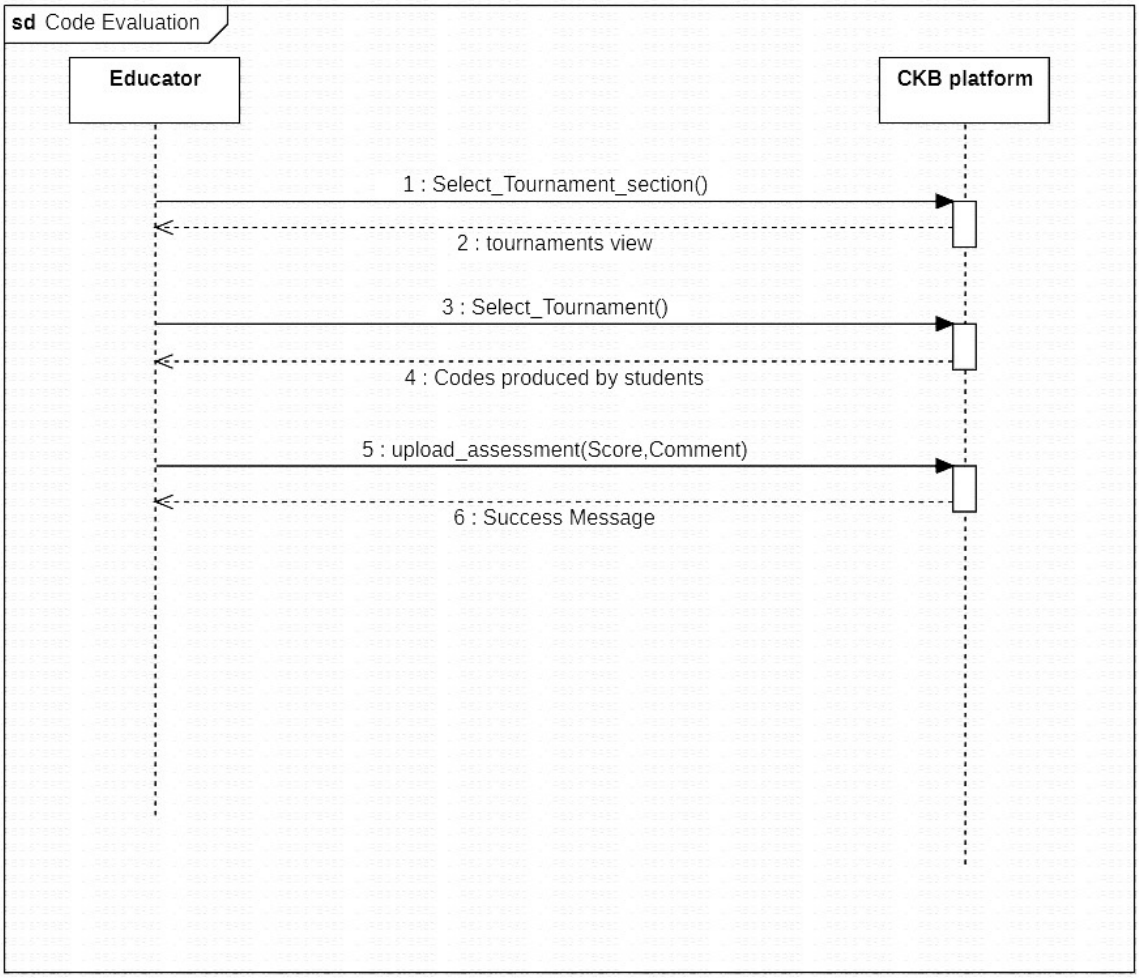


Figure 3.14: Code Evaluation

[UC12] Publish Final Rank

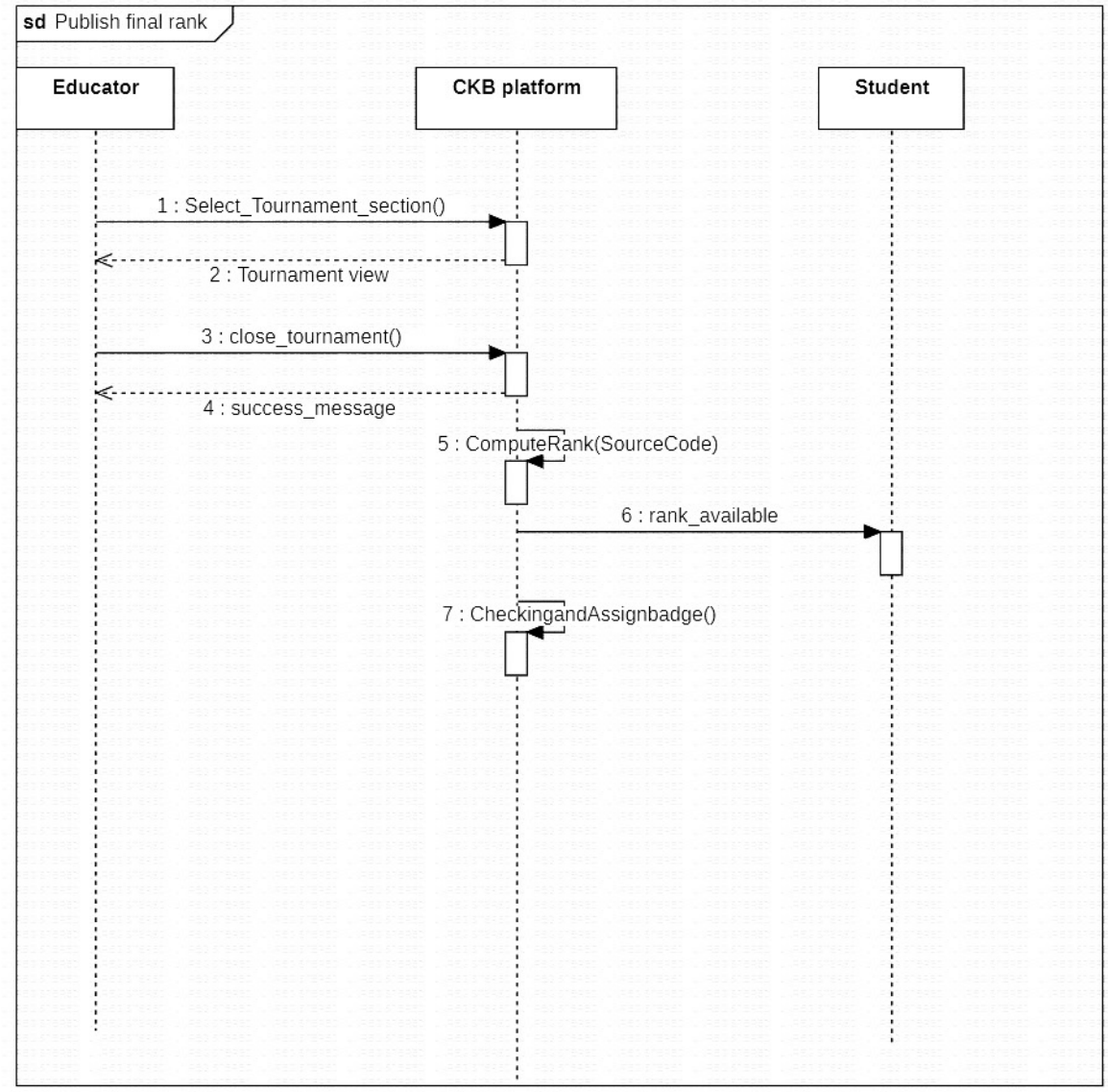


Figure 3.15: Publish Final Rank

[UC13] See Profile

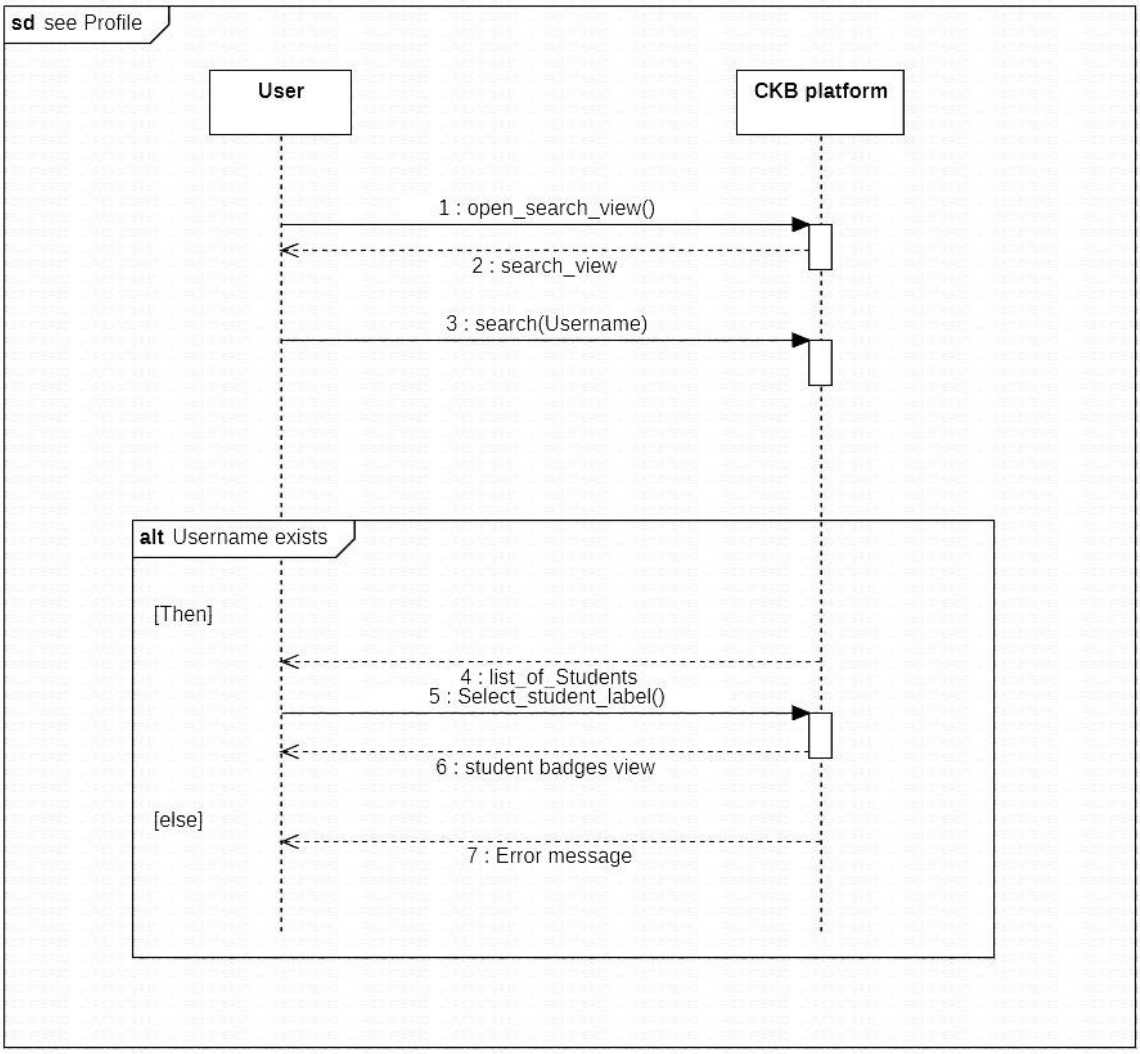


Figure 3.16: See Profile

[UC14] Badge Creation

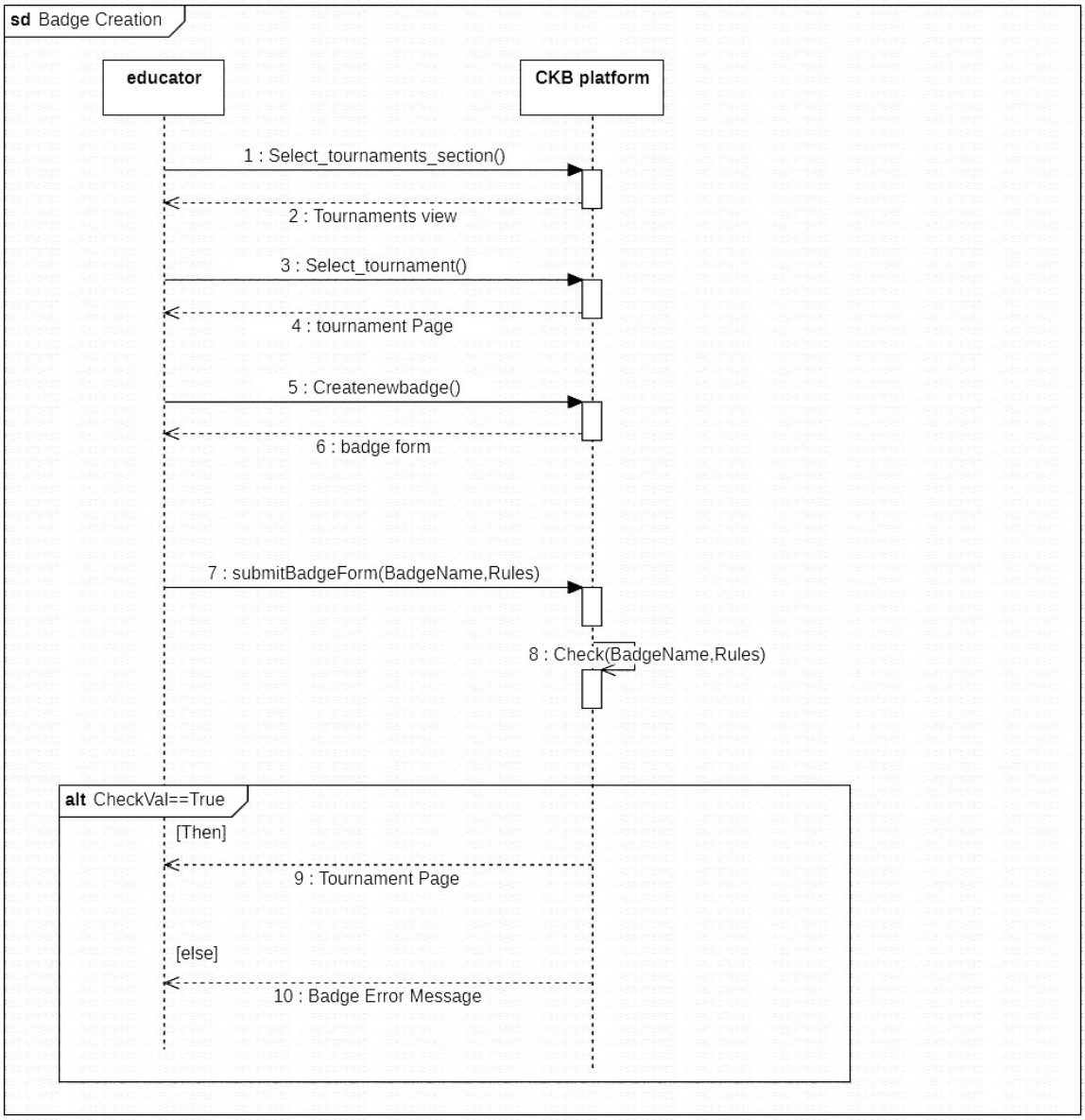


Figure 3.17: Badge Creation

3.2.4. Requirement mapping

[G1] Educators create code kata battles	
R1 The System allows users1 to register by providing their personal information (Full Name, etc.), a valid email address and a password R2 The System allows registered user to log in R3 The System allows Educators to create/modify a battle upload the code kata (description and software project, including test cases and build automation scripts) R4 The System allows to create/-modify/terminate a tournament by selecting the existing battles, setting the minimum and maximum number of students per group, the registration and final submission deadline. R5 The System allows an educator to give or deny permission to his colleagues to modify a tournament. R10 The system allows educators to define the scoring criteria for a specific battle which they have permissions to R11 The system maintains and computes the scores of each battle R12 Educators can create a badge and a set of rules associated with that badge	D1 User must have a reliable internet connection D2 User personal information must be correct D3 Educators properly insert information about a tournament D7 The educator correctly adds information about a new badge such as new rules or badge name

[G2] Students compete in multiple tournaments in teams	
R1 The System allows users to register using a github acccount R2 The System allows registered user to log in R6 The System must notify subscribed user about upcoming battles and deadlines. R7 The System allows students to create a team R8 The System allows students to invite other students into one of their teams R9 The System allows students to join a new team which they were invited R15 The System creates a repository on GitHub containing the code kata right after the registration deadline R16 The system sends the link to all the enrolled students after creating the repository with the code kata R17 The System receives notifications from GitHub regarding the students registered repositories commits R18 The System pulls the repository after receiving a notification for that repository before the deadline of that battle R19 The System runs the appropriate test on the new code after every pull of the repository R20 The System calculate and update the team's score for that battle after rerunning the tests R25 The system sends a notification about the tournament's termination to students	D1 User must have a reliable internet connection D2 User personal information must be correct D4 The Github interaction it's reliable(the user is able to pull and push the code without losing its data) D5 Notifications to the user must arrive as soon as the final rank is available

[G3] Students receive performance feedback after each battle assessment	
R10 The system allows educators to define the scoring criteria for a specific battle which they have permissions to R11 The system maintains and computes the scores of each battle R12 Educators can create a badge and a set of rules associated with that badge R13 The system assigns the badges that are created by educators as a reward for the rules they fulfill R14 The system shows the badges that are assigned to students R20 The System calculate and update the team's score for that battle after rerunning the tests R21 The system updates the personal tournament score for each student enrolled in the tournament right after the battle ends R22 The system allows educators to manually evaluate the code after the deadline R23 The system allows educators to finish the consolidation stage after completely performing the manual evaluation R24 The system computes the final ranking of the tournament immediately after consolidation finishes	D4 The Github interaction it's reliable(the user is able to pull and push the code without losing its data) D5 Notifications to the user must arrive as soon as the final rank is available D6 The Educator properly insert an evaluation manually when it's requested D7 The educator correctly adds information about a new badge such as new rules or badge name

3.3. Performance Requirements

The system has to guarantee good performances in order to serve a great number of users (educators and students).To achieve this goal the user experience must be as good as possible in order to fulfill this requirement the response time must be low no more than a second.If the user's internet connection is slow the response time can increase enormously

3.4. Design Constraints

3.4.1. Standards compliance

The CKB platform pays a great attention for what concerns users privacy cause of this the CKB project is in compliance with the General Data Protection Regulation (GPDR) a regulation in EU law on data protection and privacy for all individuals within the European Union (EU) and the European Economic Area (EEA). Moreover, the platform has to use the international format of date and time to adequate to the newer standards.

3.4.2. Hardware limitations

Here is presented a summary of the hardware features that a user should have to use the platform properly:

- The user must have a device with a good internet connection for providing this the device should be compatible with at least one of this standards 3G, 4G, 5G, IEEE 802.11 and IEEE 802.3. Both for the wired and wireless communications it must be connected to a device able to guarantee an internet connection such as a modem or an access point and so on;
- The user must have a device with good hardware features such as a processor with high performance as an example intel i5 or i7 and a display with high resolution at least full hd and a fair amount of ram at least 8 GB.

3.4.3. Any other constraint

The UI should be user-friendly because the CKB platform is developed for educators and students that are learning to code. Furthermore, they must be able to use the platform in a simple and efficient way.

3.5. Software System Attributes

Here are explained some software side attributes that the system should provide.

3.5.1. Reliability

The system has to be reliable because it will have to run continuously for a long period of time. To ensure this feature the platform must have some sort of replication and consistency policy to avoid system crash. Moreover, as best practice, it is important to have offline backups of the system for recovering information after data loss.

3.5.2. Availability

The most important attribute that the system has to provide is the availability. The system should have an availability of 99%.Since the platform has to achieve this goal some

replication policies must be implemented and a single point of failure should be avoided. Also it has to be specially prepared for a possibly large amount of submissions when deadline is close.

3.5.3. Security

The system will store the users personal data so the security aspect must be carefully considered. Passwords stored in the central database must be encrypted. The data store must be protected with all possible security measures to avoid internal and external attacks. The CKB platform must ensure integrity, consistency and confidentiality by using appropriate cyber-risk avoidance policies. Additionally, because students code will be ran on the system for the dynamic analysis, a proper way to do these has to be explored making sure no malicious code can damage the platform.

3.5.4. Maintainability

The system must guarantee a good level of maintainability. The code has to be well documented. A testing routine has to be provided and it has to cover at least 75% of the entire code excluding the UI code.

3.5.5. Portability

The system is a web application so it must be compatible with different web browser (Firefox,Google Chrome and so on) and devices (smartphones, computers, etc).

4 | Formal Analysis Using Alloy

For the alloy part we focused on modelling the ranking of a tournament (T) and see how it evolves after groups submit their code. We also analyse different properties of the ranking ($R_{t,T}$) a relation that establishes which student ($s \in S$) ranks better or equal depending on their score and their current submissions($R_{t,T} \subseteq S \times S$).

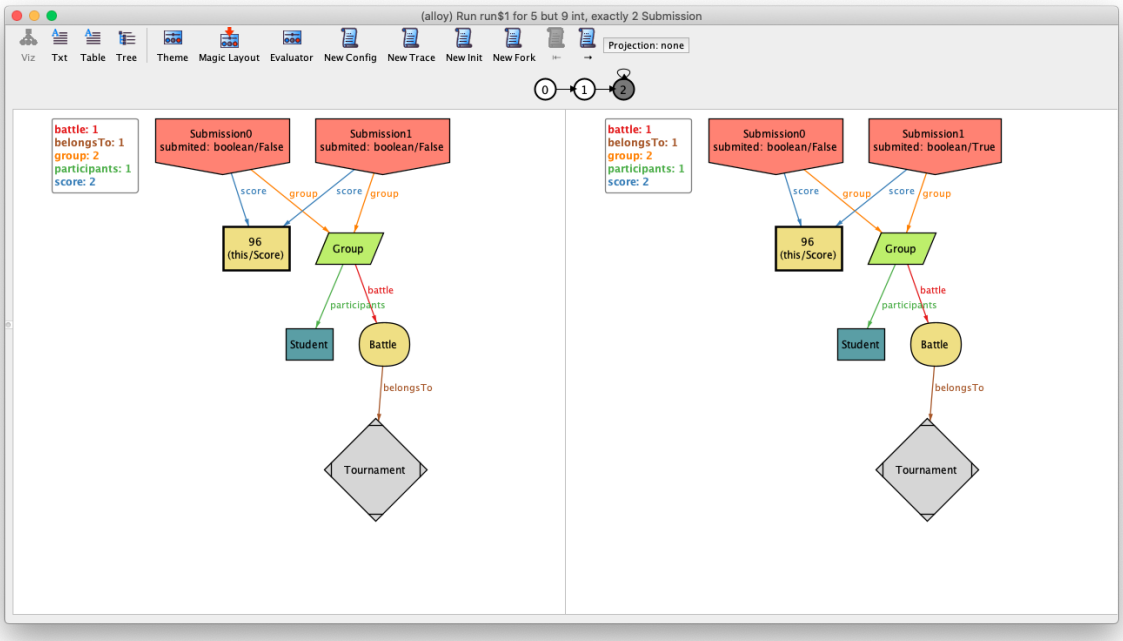
```
open util/relation
open util/boolean


sig Tournament {}

//A battle has to belong to exactly one tournament
sig Battle {
        belongsTo: one Tournament
}

sig Student {}

//A group is a team of students for a specific battle
sig Group {
        participants: set Student,
        battle: Battle
} { participants ≠none and cannotEnrollTwiceInABattle }

//We rename it for clarity
sig Score in Int {}

//The submission it is of a specific group and has associated a score
//To represent that submissions are submitted over time we added the submited attribute
sig Submission {
        group: one Group,
        score: one Score,
        var submited: one Bool
}{ score≥0 and score≤100 }
//Here we capture the fact that student can enroll only once to any battle if not it
    ↪ would be unfair
pred cannotEnrollTwiceInABattle {
        all disj g,h:Group | g.participants&h.participants=none or g.battle≠h.battle
}


//Auxiliary functions used to compute the score:

//Retrieves all the submissions of a group
fun submissions [g:Group]:set Submission{
        (group:>g).dom
}

//Retrieves only the submited ones of a group
fun effectiveSubmissions [g:Group]:set Submission{
        {s:g.submissions | s.submited.isTrue}
}

//Calculates all the obtained scores of a group until that moment
```

```
fun scores [g: Group]: set Score{
        g.effectiveSubmissions.score
}

//The best score of a group if it has not submitted it is 0
fun bestScore [g: Group]: Score {
        max[g.scores + {0}]
}

//The participants of a tournament in this case it is not necessary to have submitted jus
    ↪  to be enrolled (be part of a group in any of the battles belonging to it)
fun participants[t: Tournament]: Student {
        {s: Student | (some g: Group | t=g.battle.belongsTo and s in g.participants)}
}

//Score of the student only considering the scores obtained in the battles belonging to
    ↪ it
fun scoreInTournament[s: Student, t: Tournament]: Int{
        sum g: s.groupsForATournament[t] | g.bestScore
}

//For the prevoius we need to retrieve the groups in which the student participates
    ↪ related to the tournament
fun groupsForATournament[s: Student, t: Tournament]: set Group {
        {g: Group | s in g.participants and t=g.battle.belongsTo }
}

//Returns the ranking relation for a tournament as previously explained
fun ranking[t: Tournament]: Student -> Student {
        {b,w: t.participants | int b.scoreInTournament[t]≥ int w.scoreInTournament[t]}
}


//To model correctly the evolution of the competition we add the following facts:

//The competition begins with 0 effective submissions
fact noSubmissionsAtStart {
                all s: Submission| s.submited.isFalse
}

//It ends after all the generated submissions become effective
pred allTournamentsEnd {
        eventually all s: Submission| s.submited.isTrue
}
fact {allTournamentsEnd}

//Each step can only submit one submission
pred submit [s: Submission]{
        submited' = submited ++ s->True
}
fact oneSubmissionPerStep {
        always some s: Submission| s.submit
}
```

4.1. Examples

Figure 4.1: A simple example with two submissions to see if the behavior is the expected one

```
run {} for 5 but exactly 2 Submission, 9 int
```



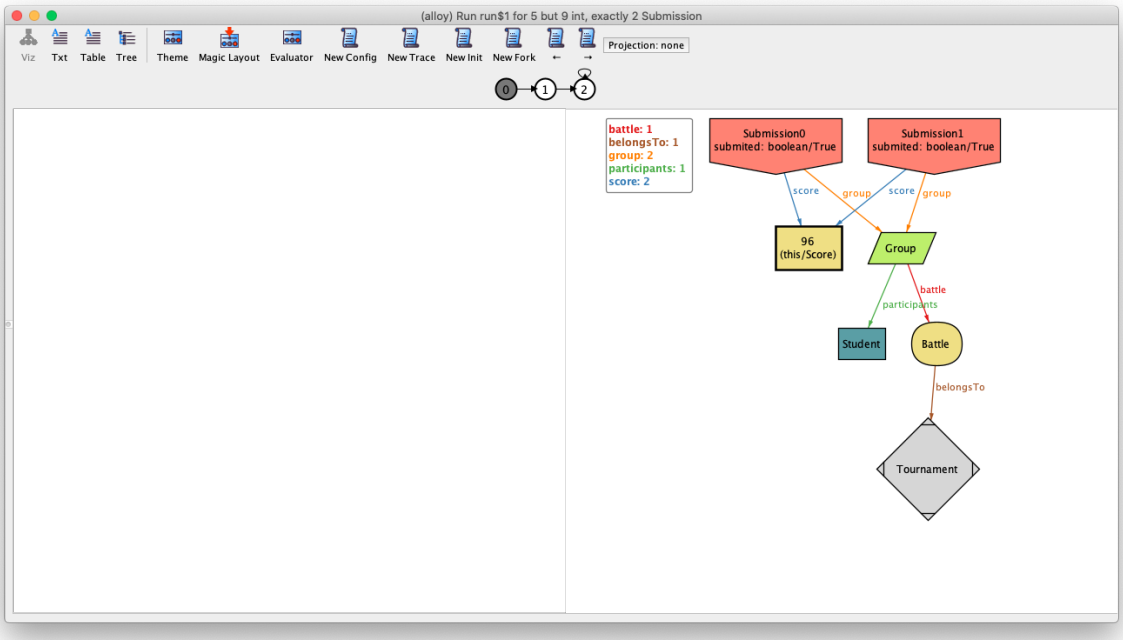
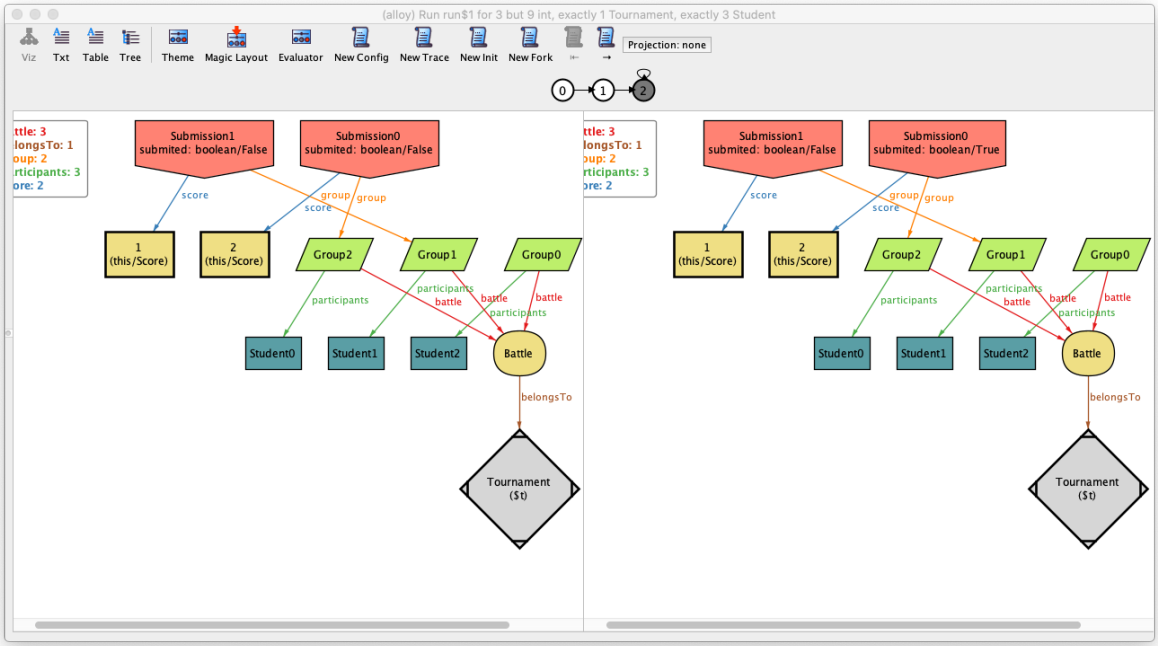
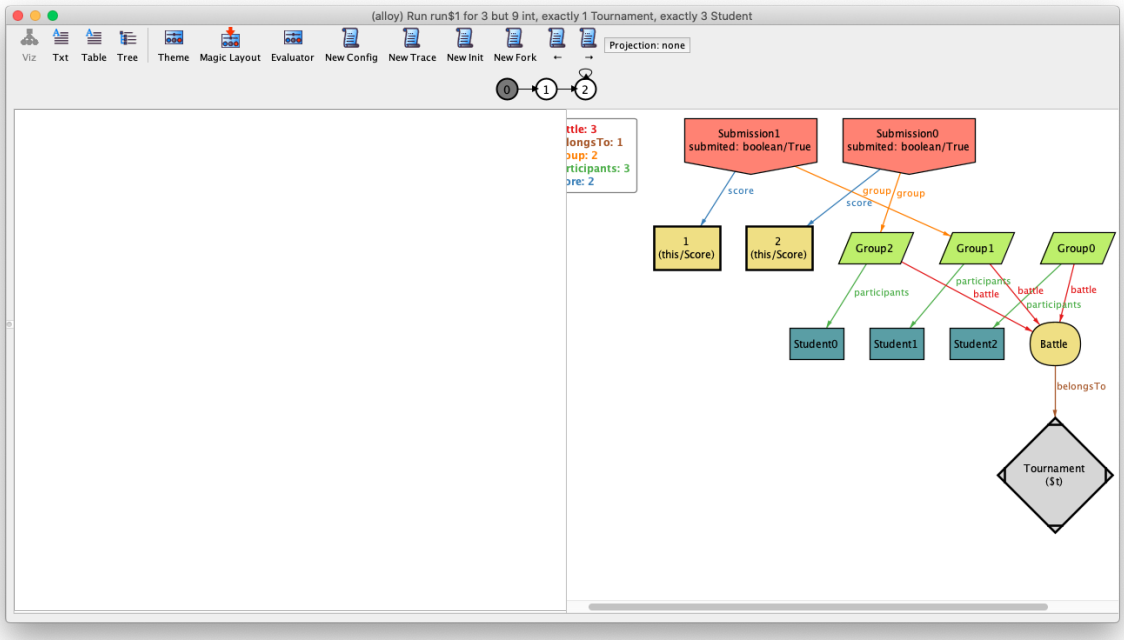


Figure 4.2: An example where the final ranking is a relation of total order (i.e. with no ties and everyone participates)

```
run {
        eventually some t: Tournament |  totalOrder[t.ranking,Student]
} for 3 but exactly 1 Tournament, exactly 3 Student, 9 int
```



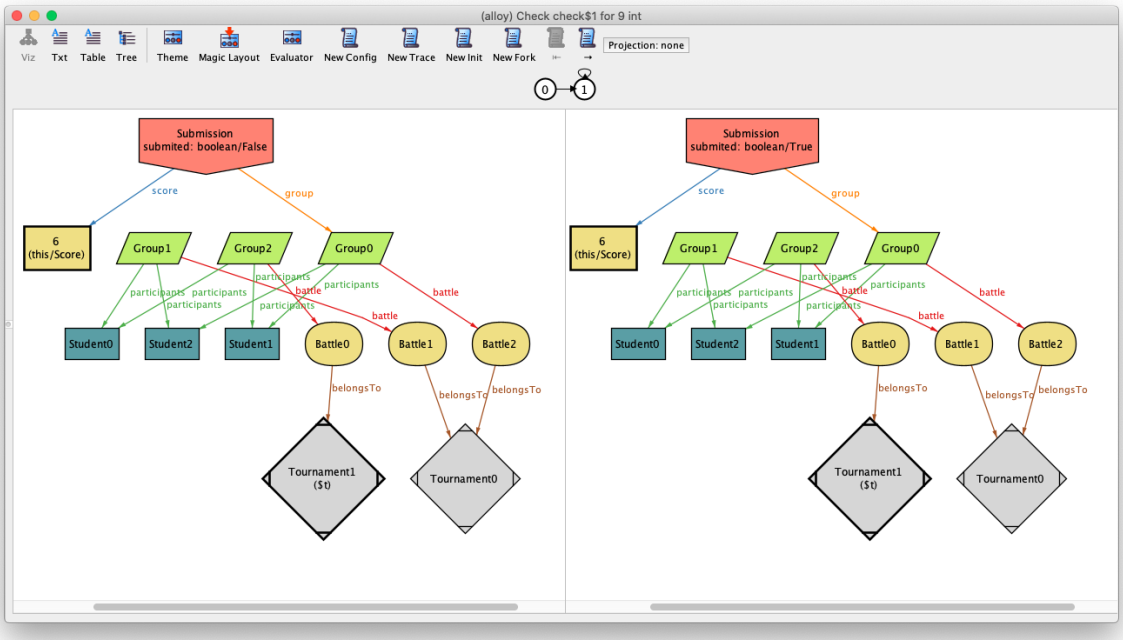


Final ranking:

Student	Score
0	2
1	1
2	0

Figure 4.3: Counterexample to prove that ranking is not always a partial order (neither a total order) because of ties.

```
check {
        after  all t: Tournament | partialOrder[t.ranking,t.participants]
}  for 9 Int
```



Final ranking for $t:

Student	Score
0	0
1	0

Figure 4.4: Check that ranking is a preorder

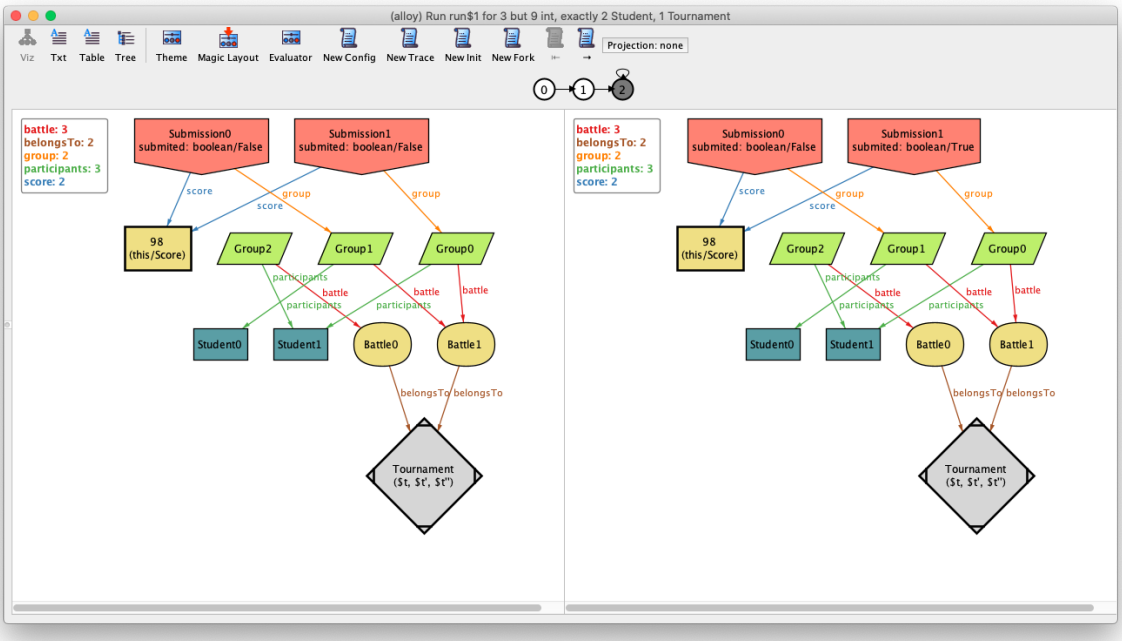
```
check {
        after all  t: Tournament |  preorder[t.ranking,t.participants]
} for 9 Int
```

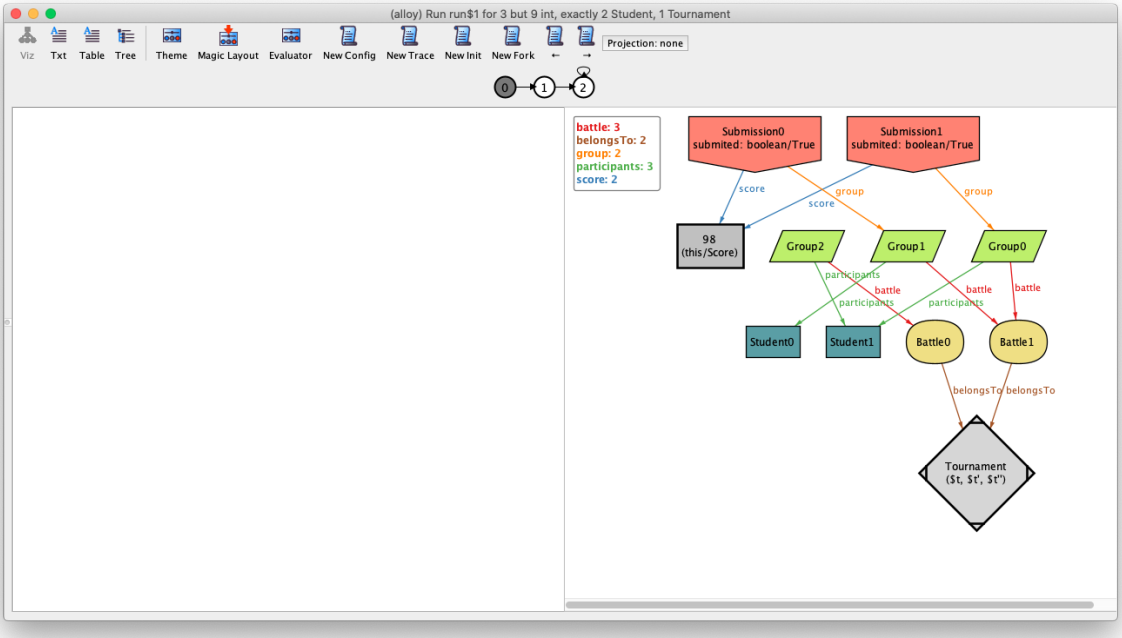
```
Executing "Check check$1 for 9 int"
  Solver=sat4j Steps=1..10 Bitwidth=9 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
  1..10 steps. 646809 vars. 21625 primary vars. 3125768 clauses. 416349ms.
  No counterexample found. Assertion may be valid. 31092ms.
```

Figure 4.5: Simple example to show that a student can improve his score and tie the best one

Because the relation has only 2 participants: the ranking relation is symmetric $\leftrightarrow$ there is a tie

```
run {
        eventually some  t: Tournament | #t.participants=2 and symmetric[t.ranking];
        some  t: Tournament | antisymmetric[t.ranking];
        some  t: Tournament | symmetric[t.ranking]
} for 3 but exactly 2 Student, 1 Tournament, 9 int
```





Ranking evolution:

Student	Score	Score'	Score"
1	0	98	98
0	0	0	98

5 | Effort Spent

In this part there is an overview of the time effort spent by each member of this team. Everyone have spent some time writing each section of this document and here its visible the amount of time.

- **Andaloro Emanuele**

chapter	Effort(In hours)
1	5
2	10
3	25
4	3

- **Deveali Lew Simon**

chapter	Effort(In hours)
1	5
2	10
3	10
4	18

- **Galantino Claudia**

chapter	Effort(In hours)
1	5
2	16
3	20
4	2

Bibliography

[1] Github. Oauth, 2017. URL `https://docs.github.com/en/apps/oauth-apps`. Last accessed 16 Dec 2023.

List of Figures

List of Tables