Requirements Analysis and Specification Document

Francesco Spangaro - Tosetti Luca - Francesco Riccardi

07 January 2024

POLITECNICO
MILANO 1863

Prof.
Matteo Camilli

Version 1.2
Academic Year 2023 - 2024

Contents

1 Introduction

1.1 Purpose

The purpose of the CodeKataBattle platform is to create a friendly and enjoyable environment for Students, in which to improve and acquire skills in Software Development. This is done by allowing Students to compete with each other in solving problems through software development. All of this is done under the supervision of Educators who can challenge students by creating these competitions.

1.1.1 Goals

G1: ***Allow Educators to create new tournaments:***

Educators have the possibility of creting new tournaments. Educators can specify a time limit within whom Students can subscribe to the tournament. Educators can also specify how big Students groups have to be to subscribe in a tournament. When creating a tournament, Educators have the opportunity to create new badges. Badges have corresponding achievements, called "Rules", which are defined on badge creation. Badges are obtained by Students on achievement completion. Obtained badges will be then displayed on the Student's profile page.

G2: ***Allow Educators to create new battles:***

Educators have the possibility to define new battles within tournaments they have created or in tournaments they have been granted permission to do so. When creating a new battle, Educators have to set different parameters:

- set project description;
- specify the programming language and building tools to utilise, including test cases and build automation scripts;
- set minimum and maximum number of Students per group;
- set a registration deadline;
- set a final submission deadline;
- set additional configurations to evaluate Students' work and compute their score.

G3: ***Allow Educators to administer different tournaments:***

Educators can grant other colleagues permission to create new battles in their tournaments. Educators have the possibility to close their tournaments, thus not letting Students submit new answers to any battle defined in the closed tournament, nor letting their colleagues create new battles in that tournament.

G4: ***Allow Educators to administer different battles:***

Educators have the possibility, once a battle has expired and entered the consolidation phase, to manually evaluate, through the platform, each Student's work, they can then assign a corresponding score to each one of them, ranging from 0 to 100.

G5: ***Allow Students to subscribe in tournaments:***

Students subscribed to the platform have the possibility of subscribing to different tournaments, in which they plan to participate.

G6: ***Allow Students to participate in battles:***

Students can join battles within a set deadline. They can do so by themselves, by inviting somebody else or by accepting someone else's invite.

G6.1: ***Allow Students to form groups to participate in battles with:***

Students have the possibility to send out invitations to other Students, so that they can form a group to participate in a battle with. Groups need to follow the guidelines specified by the battle creator for them to be accepted.

G6.2: ***Allow students to submit their answers to a battle:***

When Students have developed a solution to the battle, they can submit it to the platform. Groups are requested to send only one answer. Students can change their answer as they proceed. When uploading a new solution the older one is overwritten, since there can only be one answer for each battle.

G6.3: ***Allow Students to see their scores and badges:***

After each answer submission, a new score is assigned to the Students. The score can be manually created by an Educator or automatically assigned to the Students by the platform. Students can see the scores obtained in a battle and in a tournament they partecipated in. Finally Students can see the badges they obtained in a tournament in their profile page.

G7: ***Let Students be notified on important events:***

When a new tournament is created, all Students subscribed to the platform are notified of it. A different notification will be sent when a new battle is created in a tournament the Students are subscribed to.

1.2 Scope

CodeKataBattle (CKB) is an easy-to-use platform which aims to let Educators propose homework and/or lessons in a new and fresh way. The goal is to have Students improve and acquire software developing skills. To do so CKB offers Educators the possibility of opening several tournaments. Each tournament is composed by many different battles, in which Students can compete with each other, individually or in groups. In order to offer all of this, CKB relies on the external platform GitHub. GitHub will take the role of "bridge" between the CKB platform and Students, allowing Students to upload their solutions on it. These solutions will then be taken by the CKB platform from GitHub and used to evaluate the Students' scores in the battle for which they uploaded a specific solution.

1.2.1 Phenomena

Events that take place either in the real World, in the Machine World or in both. Used to describe respectively what cannot be observed by the Machine, real World and events that connects the two.

1.2.1.1 World phenomena

Phenomena that take place in the real World and are not observable by the Machine

WP1: Students fork the GitHub repository for which they received a link by the platform.

WP2: A Student writes code on his personal device.

WP3: Students choose which tournament to join.

WP4: Students choose which battle to join from the ones belonging to a tournament he precedently subscribed to.

WP5: Student chooses his teammates for a battle.

WP6: Educator chooses which colleagues to allow access to one of his tournaments.

WP7: Student subscribed to a battle waits for the battle to start (registration deadline expiration).

WP8: Educator decides to close a tournament.

1.2.1.2 Shared phenomena

- Phenomena controlled by the World and observed by the Machine

  ➤ Student related phenomena

  SP1: Student registers to the platform.
  SP2: Student logs in the platform.
  SP3: Student subscribes to a tournament.
  SP4: Student invites other students to form a team.
  SP5: Student accepts an invite from another student and joins his group.
  SP6: Student or a group of Students joins a battle in a tournament they are subscribed in within a deadline.
  SP7: A Student, or a group of Students, upload a new software solution for the battle's problem in which they are partecipating by pushing a new commit on their GitHub repository.
  SP8: Student sees his and others badges, visualizing his or others profile page.

  ➤ Educator related phenomena

  SP9: Educator creates a new tournament.
  SP10: Educator grants his other collegues access to create new battles within a tournament he created.
  SP11: Educator creates a new battle.
  SP12: Educator sets a battle's settings while creating one.
  SP13: Educator manually evaluates the work done by Students in a certain battle of a certain tournament during that battle's consolidation phase.
  SP14: Educator closes a tournament.
  SP15: Educator defines new achievable badges in a tournament while creating it.
  SP16: Educator sees a Student's collected badges by visualizing his profile page.

- Phenomena controlled by the Machine and observed by the World.

  ➤ Student related phenomena

  SP17: The platform, whenever a new tournament is created, notifies all registered Students of its creation.

  SP18: The platform, whenever a new battle is created in a tournament, notifies all Students subscribed to that tournament of the new battle created.

  SP19: The platform, whenever an invite is sent from a Student to another for joining a group, notifies the recipient of the new invite he's received.

  SP20: The platform, when a battle's registration deadline expires, sends every student that joined the battle a link to the GitHub repository created by the platform itself.

  SP21: The platform, at the end of each battle, updates the Students' score in the tournament in which that battle took place, allowing all Students and Educators to see the final scores.

  SP22: The platform, whenever a tournament is closed, notifies all Students.

  ➤ Educator related phenomena

  SP23: The platform, whenever an Educator is granted access to a colleague's tournament, notifies the recipient of the access.

  SP24: The platform, whenever the submission deadline for a battle expires, notifies the battle's owner, then starts the consolidation phase.

1.2.1.3 Machine phenomena

Phenomena that take place in the Machine World and are not observable from the real World

MP1: The platform creates a GitHub repository containing the code kata when a battle's registration deadline expires.

MP2: The platform, when notified by the GitHub API, pulls the latest sources of a battle's repository.

MP3: The platform analyses the sources by running tests on them.

MP4: The platform computes the scores of a team, based on the executables uploaded by the Students for a battle. The score is automatically updated when the platform receives a notification from GitHub about new push actions

MP5: The platform, at the end of each battle, compute the overall tournament rank of each student.

MP6: The platform automatically registers a Student's badge achievement when that Student satisfies the rule to obtain said badge.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

Term	Definition
GitHub Repository	→ A place on the GitHub platform where a user can store code, files and each file's revision history.
Registration deadline	→ Maximum timespan in which a Student can subscribe to a battle or to a tournament.
Submission deadline	→ Maximum timespan in which a Student, or a group of Students, can upload their solution to a battle.
Code Kata	→ The word ”kata” refers to a karate exercise in which a form gets repeated many times, making little improvements each time. In this context it is used to express the fact that the code needs to be developed multiple times to reach an optimal solution.
Consolidation phase	→ Phase started at the end of a battle's submission deadline, used to consolidate the score of each Student in that battle. A manual evaluation of the students' code by an Educator may occur.
Guidelines	→ Rules that must be followed regarding battle, tournament, badges, ... characteristics while creating them. They are imposed by the platform itself.

1.3.2 Acronyms

Acronym	Meaning
API	→ Application Programming Interface: indicates on demand procedure which supplies a specific task.

Acronym	Meaning
CKB	→ CodeKataBattle: the name of the platform described in this document.
IT	→ Used as acronym for Information Technology to identify something, generally a computing or communication hardware, with information storage capability, closely related to the informatic world.
UML	→ Unified Modeling Language: a standard notation for modeling real world objects in an high level diagram representing OO components.
BPMN	→ Stands for Business Process Modeling Notation: a standard notation for representing processes through diagrams.
OO	→ Object-Orientation: a programming paradigm based on the concept of objects that can contain data and code and that usually represent real world objects.
DB	→ Database: a physical container for sets of data held in a computer. The database is accessible, modifiable and queryable in various ways.
GDPR	→ General Data Protection Regulation: an European law that aims at protecting the privacy and security of users' data.

1.3.3 Abbreviations

Abbreviation	Meaning
G#	→ Goal number. #
WP#	→ World phenomena number. #
SP#	→ Shared phenomena number. #
MP#	→ Machine phenomena number. #
D#	→ Domain assumption number. #
R.#	→ Requirement number. #
c.s.	→ Computer science.
e.g.	→ Exempli gratia, latin phrase meaning "for example".

1.4 Revision history

- 22 December 2023: version 1.0
- 07 January 2024: version 1.1
  - Removed every mention of tournament's deadline in the document.
  - Removed every mention of tournament's access method in the document.
  - Reformulated scenario 7 in such a way that it adapts better and more consistently to the decisions made in the RASD and DD.
- 23 January 2024: version 1.2
  - Fixed LaTeX formatting for alloy's signatures and facts.
  - Fixed inconsistencies in alloy's code emerged after DD's file completion.
  - Improved class diagram.

1.5 Reference documents

GitHub references:

- Official documentation to get started with GitHub: → `https://docs.github.com/en/get-started/quickstart`
- Official documentation about fork process → `https://docs.github.com/en/get-started/quickstart/fork-a-repo`
- Official documentation about GitHub actions → `https://docs.github.com/en/actions`

UML official specification → `https://www.omg.org/spec/UML`

BPMN official specification → `https://www.omg.org/spec/BPMN/2.0`

Use case diagrams specification used → `https://it.wikipedia.org/wiki/Use_Case_Diagram`

Sequence diagrams specification → `https://www.uml-diagrams.org/sequence-diagrams.html`

Alloy documentation → `https://alloy.readthedocs.io/en/latest/`

1.6 Document structure

- ***Section 1: Introduction***
  This section introduces the problem and the platform/application that needs to be developed in order to resolve it. It describes the major purpose of the project, every goal of it, the analysis of its domain and every real World only, Machine only and shared phenomena associated with it. In addition, in this section are inserted definitions, acronyms and abbreviations used in this document, including its revision history and refereced documents or web pages.

- ***Section 2: Overall description***
  This section gives an overall description of the project and all interactions that could occur between the platform and the final users (Students and Educators). To do this, this section includes different possibile scenarios that could happen, different actors involved in the platform usage and all of the assumptions, dependencies and eventual constraints that have to be considered in the development of the platform.

- ***Section 3: Specific Requirements***
  This section contains the use cases, that are a more precise description of each scenario. It describes the several functional and performance requirements of the project and their correspondance to the project's goals. It also contains all the design constraints and system attributes that must be followed/guaranteeded while developing the platform.

- ***Section 4: Formal analysis using alloy***
  This section contains a formal description of the platform. The formal description is done using the formal language Alloy (referenced in section 1).

- ***Section 5: Effort spent***
  This section contains all the information about the time spent by each group member in order to complete this document and its division by each section of the document.

2 Overall description

2.1 Product perspective

The following section contains the platform's UML diagram and a list of meaningful scenarios in which the platform can be used by different users.

2.1.1 Scenarios

2.1.1.1 Student signs up to the platform

Peter is an IT's student that wants to improve his software development skills. He learns about the platform CKB one day, when his c.s. professor proposes to his class a software development tournament in substitution to the normal, boring and limited tests organized through the academic year. Peter is extremely intrigued by the idea and the platform, so much that the same day he decides to try and register to the platform. Peter opens his personal browser and goes to the CKB site's homepage. Here he navigates to the student's dedicated page and clicks the "Sign-up" button in order to register. He inserts all the required information in the registration page's mandatory fields(e.g. name, surname, attended school, email, username, password, ...) and clicks on a "Confirm" button. Peter now looks his email homepage for the notification about the correctness of his registration and can now access all the CBK platform's features, after loggin in with his credentials.

2.1.1.2 Educator signs up to the platform

Vittorio is an innovative computer science's teacher. He discovered the CKB platform while searching the web for new testing ideas. Vittorio decides to register to the platform as he's very intrigued with it. To register, Vittorio goes on the CKB's Homepage, then on the page dedicated to educators and clicks the "Sign-up" button. After the registration page shows up, he compiles all the fields in the registration form (e.g. name, surname, school in which he teaches, istitutional email, password, ...), especially the ones related to his profession, then clicks on a "Confirm" button. Vittorio then waits for the registration confirm email and starts using the CKB platform's features, after loggin in with his credentials.

2.1.1.3 Educator creates a new tournament (and badges)

Laura is a c.s. educator who registered to CKB platform. She decides to create a new tournament to let her Students compete with each other and improve their software developing skills. In order to do so, Laura logs in her account and starts the tournament creation procedure. While doing so, she chooses the programming languages to be used when developing the battles' solutions, which will be contained in the tournament, the name of the tournament, its maximum duration and finally she decides if the tournament is going to contain some new or default badges, obtainable by the students by doing some achievements. Laura also wants to create new badges in order to encourage her students to participate more actively to the tournament. To do so, Laura accesses the appopriate section while creating the tournament and starts creating the badges, specifing their title, the rules to obtain them, their icon and their score, that will be added to the Student's score when obtained. At the end of the process Laura confirms her choices and the tournament starts.

2.1.1.4 Educator creates a new battle

Laura is a c.s Educator who registered to the CKB platform. After she created a new tournament, Laura wants to create a new battle within it to let her students compete with each other. To do so Laura logs in her account and accesses the tournament in which she wants to create a new battle. At this point Laura tries to create a new battle within the tournament. While doing so, she decides the battle's name, the programming language allowed in the battle, the dimensions of the students' groups that can partecipate in it, a registration and submission deadline and eventually some personalized rules to elaborate students' scores. Laura then waits for her class' students subscribed to the tournament in which the battle has been created to join the battle. At the end of the registration deadline the partecipants can start competing with each other while Laura supervises their work. At the end of the battle, which is automaticaly closed at submission deadline, the students' work gets evaluated and the battle's score added to their overall tournament score.

2.1.1.5 Educator closes a tournament

Marco is a c.s Educator who is registered to the CKB platform. He has created a new tournament a month ago and today, after this month he decides to close it. In order to do so, Marco logs in his account and accesses the tournament that he wants to close, at this point Marco controls if there are battles still open. Marco notices that Noemi has created a battle in his tournament, and that said battle is still open. Him being the tournament owner, he can control all battles contained in it. He notices that Noemi's battle's submission deadline is in two days, so he decides to wait until the end of her battle before closing his tournament. After three days Marco, after logging in the platform, checks that there aren't open battles or ones that are waiting for manual evaluation. He then proceeds to close the tournament.

2.1.1.6 Educator manually evaluates a battle after its end

Luca is a c.s Educator registered to the CKB platform. He has created a new battle in a tournament 16 days ago, choosing as the submission deadline 14 days after the battle's creation. Luca logs in his account on the platform, accesses the tournament in which he created the battle and sees that said battle is closed and waiting for a manual evaluation. He then decides to examine the solutions that have been submitted by the different groups. The first group he analyses is composed of three students. He sees that they have a score of 90/100 in the automated evaluation. Luca opens the solution proposed by the group, then he checks if the number of commits is equal amongst the students in the group. Since he doesn't notice any large differences between the number of lines of code written and the number of commits made by each student he decides to go to the comment section. By looking at the code written by the students, Luca relises that they have not written any specific explanation to the code, and that the overall documentation is very lacking in quality and detail. Because of this, he decides to modify the previously assigned grade by reducing it from a 90/100 to a 80/100. He then writes an explanation for this change to the group. He then confirms his modifications and goes on evaluating the next group. When there are no more groups to evaluate, he closes the battle.

2.1.1.7 Student forms a group

Francesca signs in the platform with her credentials and sees that, in a tournament in which she is subscribed, a new battle was created. She would like to participate in it and so she decides to form a group with two of her friends to compete in the battle. She clicks on the apposite menu and chooses the Students she would like to invite to join her group. After both her friends accept her invite, the group is correctly formed and now, as group creator, she can subscribe her group to the battle.

2.1.1.8 Student joins a battle

Andres is subscribed to a tournament with his group. One day he is notified that a new battle for the tournament he is subscribed to is available. After reading the battle's description, Andres decides that he wants to try and join in. He autonomously asks his group members if they want to join the battle too (through email or another external messaging platform). After reading their answers, Andres subscribes the group to the new battle. Once the registration deadline for the groups expires, Andres' group forks the automatically created GitHub repository and sets up an automated workflow through GitHub Actions, to notify the platform of each of their commits.

2.1.1.9 Student uploads a solution to a battle

Piero is currently competing with his group in a tournament, they are working on a solution for a battle and they are confident that they are right with what they coded. Piero decides to upload a solution for the battle to the GitHub repository, so that it can be evaluated and they can see how much their score is for this first draft solution. Piero uploads the current solution to the forked

GitHub repository. After the push, the GitHub Actions workflow is started, letting the platform know that Piero has uploaded a new solution for the battle he's currently competing in. The automated evaluation system integrated in the platform now starts, tests are automatically ran and no human evaluation is deemed needed, so the platform gives Piero's solution a score of 75/100. Piero sees the score on his profile on the platform and understands that his solution is found correct, but not the best that can possibly be done. He decides to continue working on it, until his solution will obtain a score of at least 95/100.

2.1.1.10 Student uploads a solution after a battle's submission deadline espired

Frank has been competing in a tournament for a very long time and has finally found a solution for the last battle he struggled to solve. After having uploaded his solution on the forked repository, the GitHub Actions workflow starts and notifies the platform that Frank has uploaded a new solution for his final battle. The CodeKataBattle platform then sees that the deadline for Frank's battle has expired. The platform notifies Frank that his last uploaded solution will not be considered on the final score he will get, and no further action is taken.

2.1.1.11 Students visualise their tournament's results and their badges

Paolo, Lucia e Alessandro are three friends that are subscribed in the CKB platform, one evening they decide to compare their tournament's result on the CKB platform. They want to compare their results in the last tournament, which ended the day before. Paolo, after logging in the site and navigating to his profile page, sees that he achived a score of 70/100. Lucia in last tournament was in the same group as Paolo, so she achived the same score. Alessandro, who was in a different group, sees that he got a score of 90/100, thus becoming the best of the three. Lucia now wants to see the best score they have ever achieved, so after logging in the site and searching for her best result, she discovered that she participated in a tournament two months prior where she achived a score of 95/100. Alessandro wants to see his best result too, and after logging in the site, he sees that in his badges there is the ”perfect score” badge, which is assigned to a student who got a score of 100/100 in at least one battle. Paolo then wants to know who out of the three has the most badges. After accessing his profile page, Paolo sees that he has obtained a total of 57 badges, obtained in all of the tournaments in which he participated. By looking at his badges Paolo notices that he has achieved badges for having the most commits in a battle, for having partecipated in 50 tournaments and for having written the most code in a tournament. Alessandro checks that too, and sees that he obtained only 13 badges total.

2.1.2 Class diagram

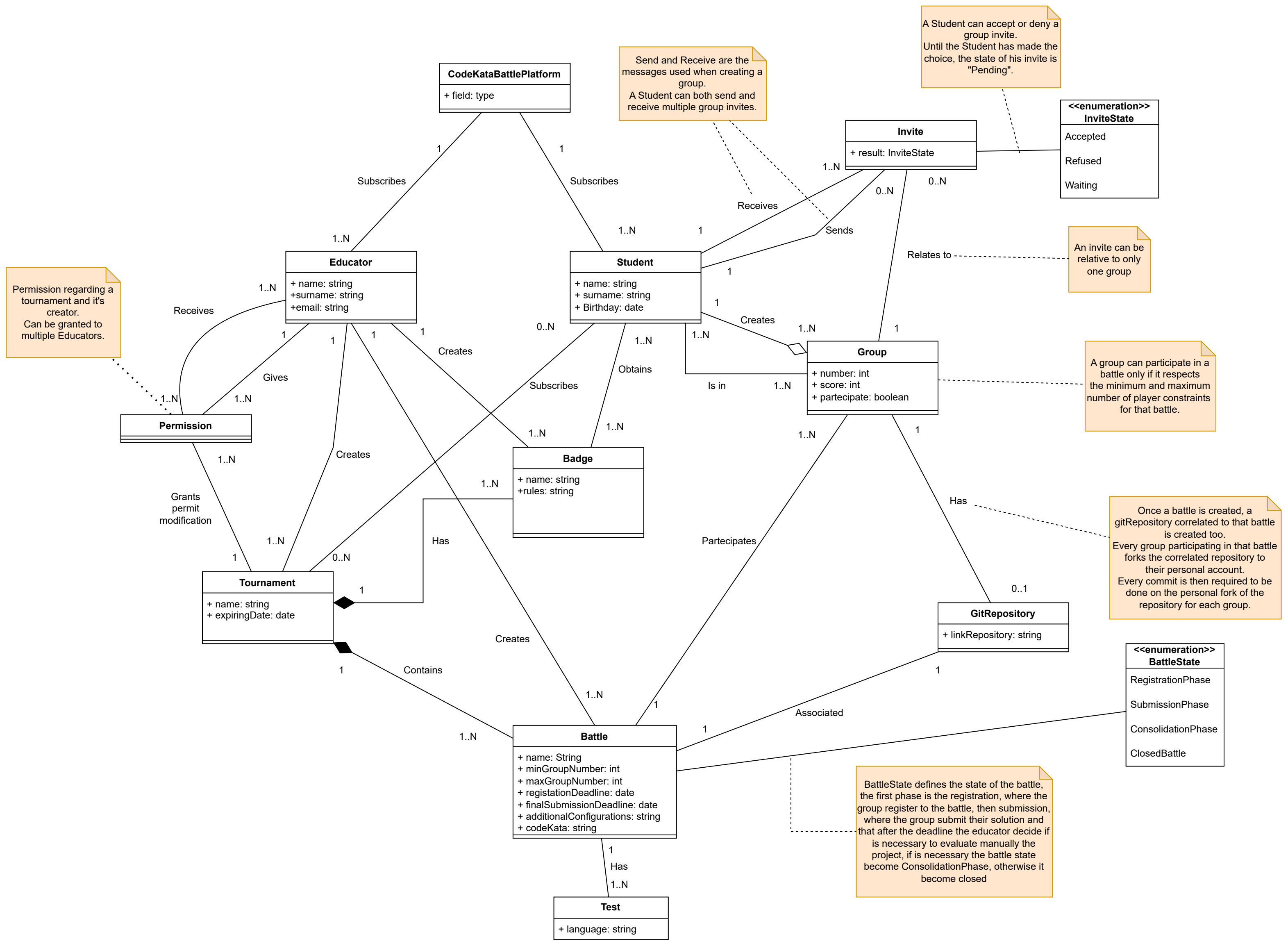

CodeKataBattlePlatform
+ field: type
Educator
+ name: string
+surname: string
+email: string
Student
+ name: string
+ surname: string
+ Birthday: date
Invite
+ result: InviteState
<<enumeration>> InviteState
Accepted
Refused
Waiting
Group
+ number: int
+ score: int
+ partecipate: boolean
Permission
Badge
+ name: string
+rules: string
Tournament
+ name: string
+ expiringDate: date
GitRepository
+ linkRepository: string
<<enumeration>> BattleState
RegistrationPhase
SubmissionPhase
ConsolidationPhase
ClosedBattle
Battle
+ name: String
+ minGroupNumber: int
+ maxGroupNumber: int
+ registationDeadline: date
+ finalSubmissionDeadline: date
+ additionalConfigurations: string
+ codeKata: string
Test
+ language: string
Subscribes
Receives
Sends
Relates to
Creates
Is in
Gives
Grants permit modification
Obtains
Has
Partecipates
Contains
Associated
1
1..N
0..N
0..1
Send and Receive are the messages used when creating a group. A Student can both send and receive multiple group invites.
A Student can accept or deny a group invite. Until the Student has made the choice, the state of his invite is "Pending".
An invite can be relative to only one group
Permission regarding a tournament and it's creator. Can be granted to multiple Educators.
A group can participate in a battle only if it respects the minimum and maximum number of player constraints for that battle.
Once a battle is created, a gitRepository correlated to that battle is created too. Every group participating in that battle forks the correlated repository to their personal account. Every commit is then required to be done on the personal fork of the repository for each group.
BattleState defines the state of the battle, the first phase is the registration, where the group register to the battle, then submission, where the group submit their solution and that after the deadline the educator decide if is necessary to evaluate manually the project, if is necessary the battle state become ConsolidationPhase, otherwise it become closed

Figure 1: Class Diagram

2.2 Product functions

2.2.1 Shared functions

- **Sign-up:** Let the user (Students or Educators) sign-up to the platform.

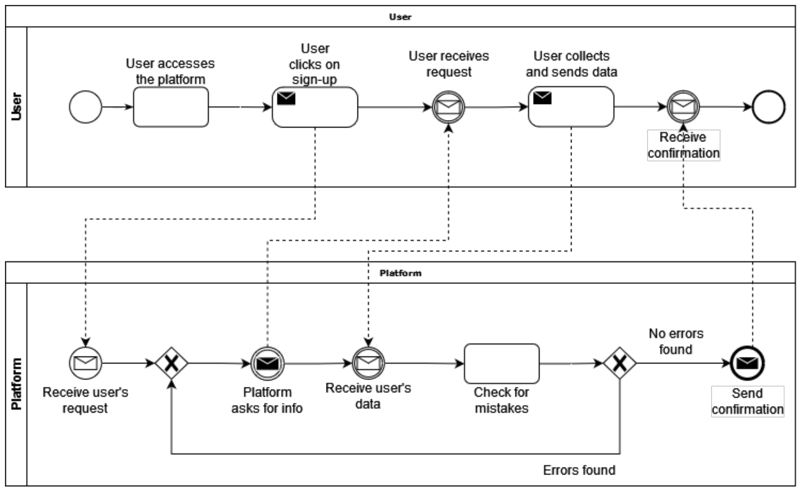


Figure 2: Sign-up BPMN

- **Visualize student's profile:** Let a user (Student or Educator) visualise the page of a specific Student.

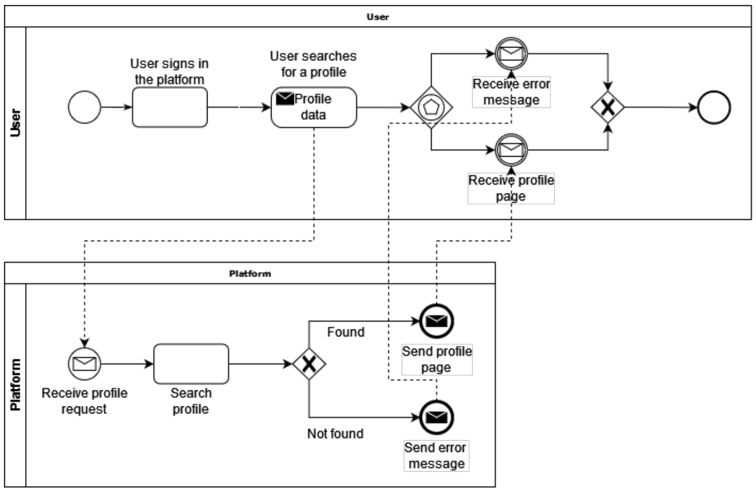


Figure 3: Profile visualization BPMN

2.2.2 Student functions

- **Student subscribes to a tournament:** Let a Student search, according to some parameters (most used programming languages, creation date, number of partecipant, etc...) and subscribe to a tournament, after logging in the platform.

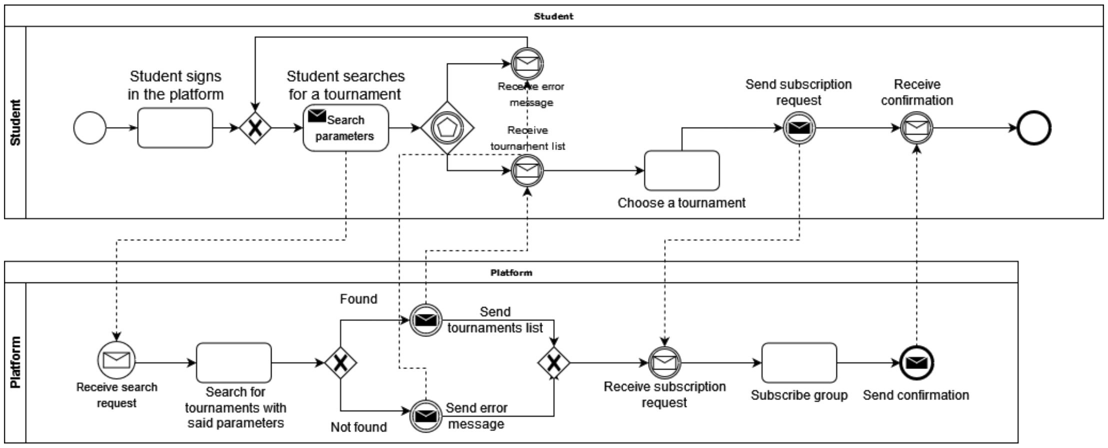


Figure 4: Student's tournament subscription BPMN

- **Student join a battle:** Let a student Subscribed to a tournament search for a tournament's battle according to some parameters (programming language requested, expiration date, difficulty, etc...) and join it alone or in a group.

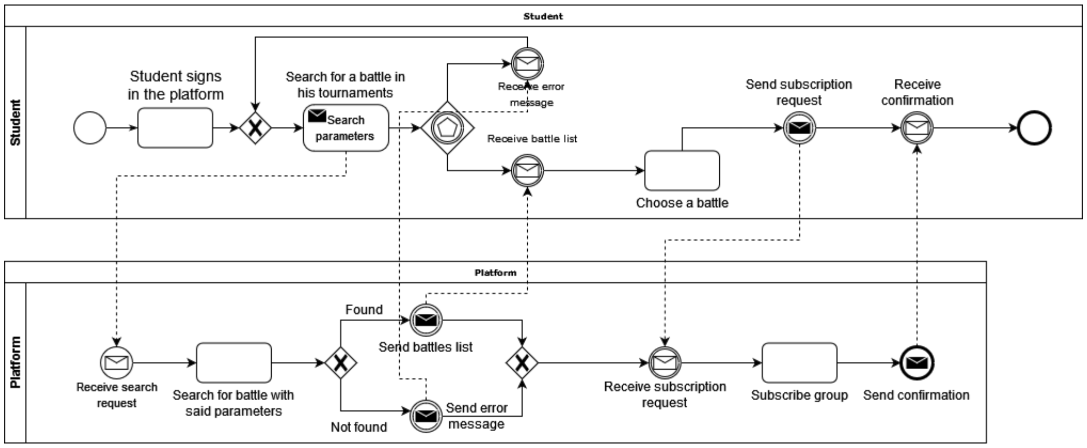


Figure 5: Student's battle joining BPMN

- **Student gets notified of new events:** Notify a Student, if registered to the platform, about new created tournaments and new created battles within the tournaments he is subscribed to.

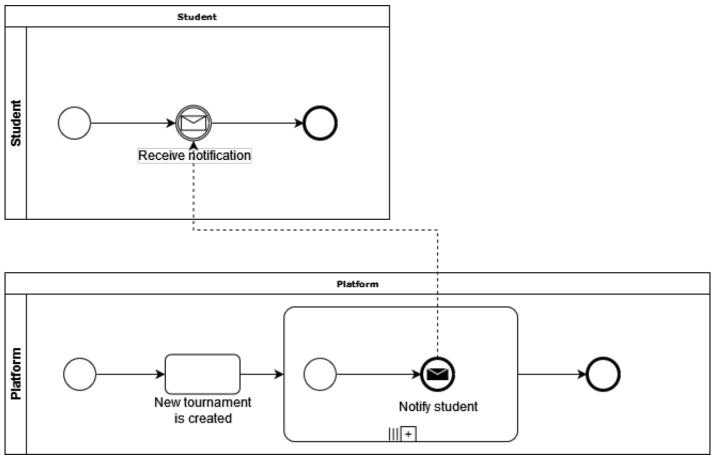


Figure 6: Student notification (Tournament) BPMN

- **Student's solutions get evaluated:** Let a Student, who connects to his forked repository on GitHub, upload his solution and evaluate it according to some parameters.

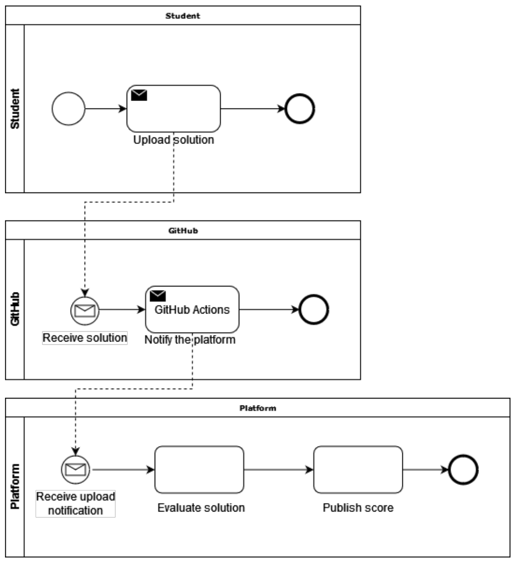


Figure 7: Student's solutions evaluation BPMN

- **Student form a group for a battle:** Let a Student form a group with other Students in order to face a battle, through invite messages.

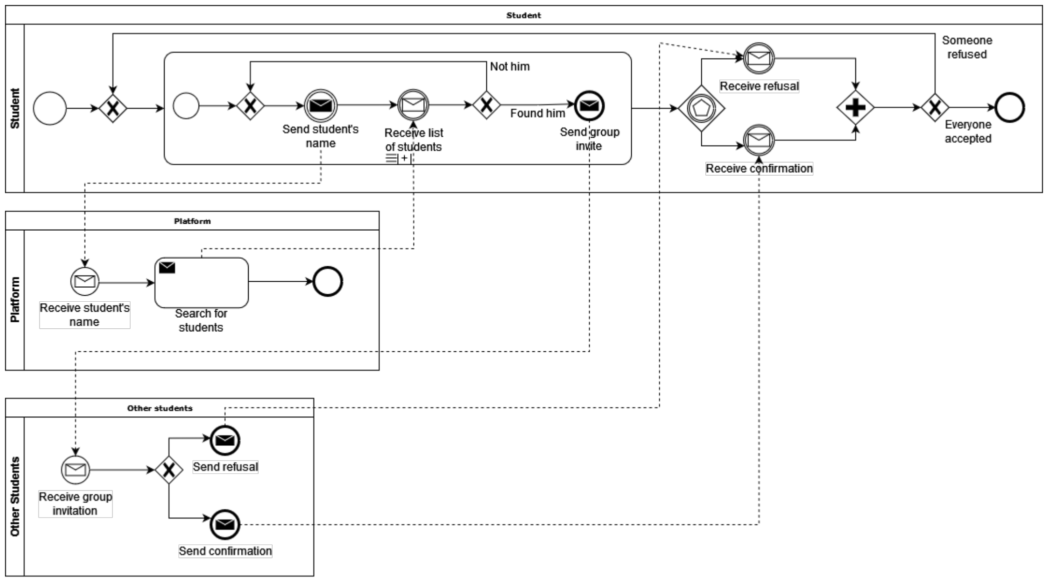


Figure 8: Formation of a group BPMN

2.2.3 Educator functions

- **Educator create a tournament:** Let an Educator create a new tournament and set its parameters.

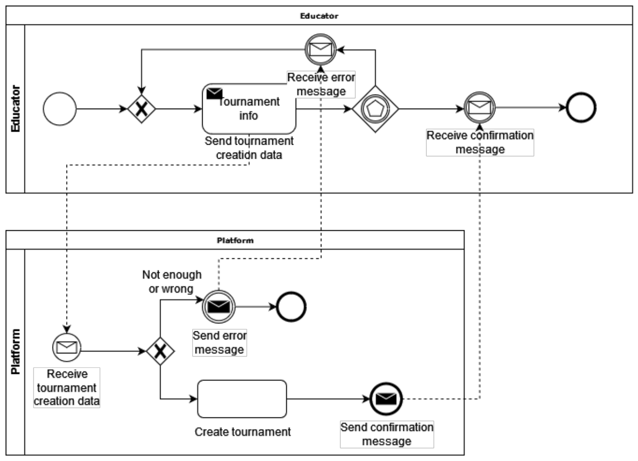


Figure 9: Tournament's creation BPMN

- **Educator grant access to a tournament:** Give an Educator the possibility to grant modification access to one of his tournaments to a colleague.

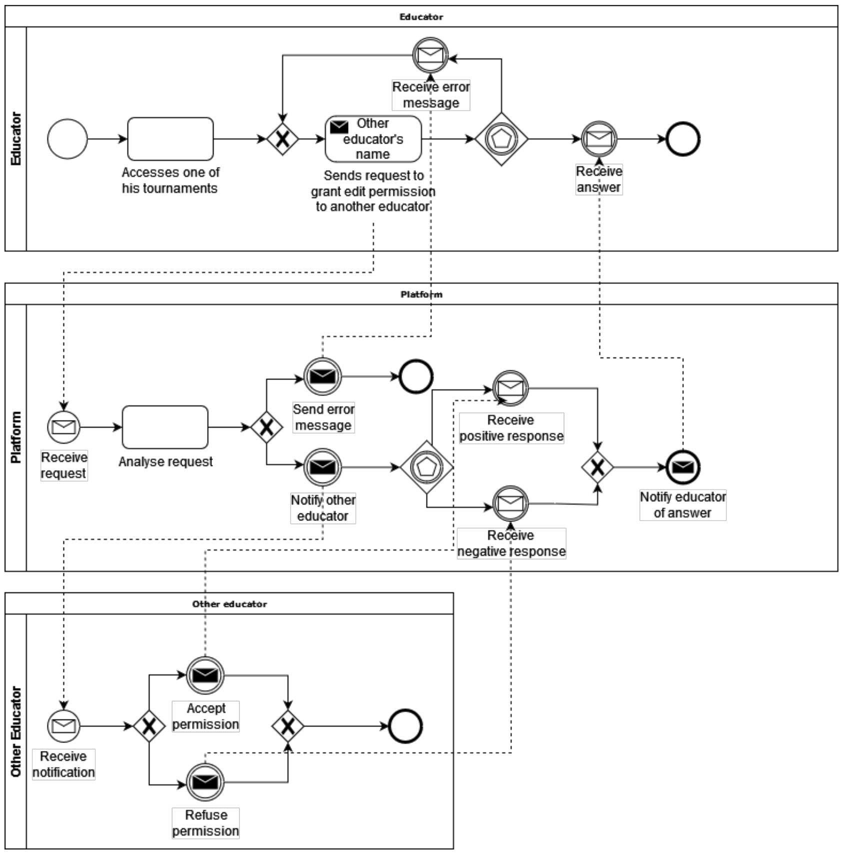


Figure 10: Grant access to tournament BPMN

- **Educator close a tournament:** Let an Educator close one of his tournaments.

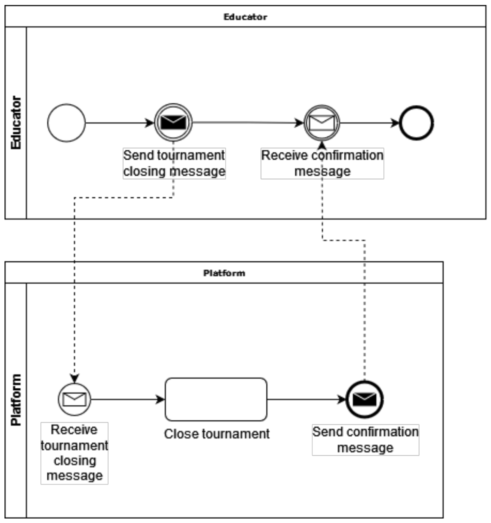

Educator
Educator
Send tournament closing message
Receive confirmation message
Platform
Platform
Receive tournament closing message
Close tournament
Send confirmation message

Figure 11: Tournament's closing BPMN

- **Educator creates a battle:** Grant an Educator the possibility of creating a battle in tournaments where he has permission to.

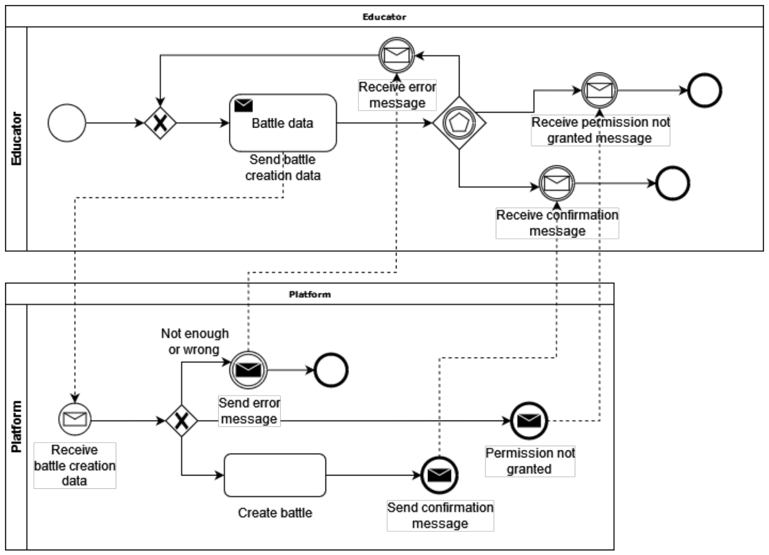

Educator
Educator
Battle data
Send battle creation data
Receive error message
Receive permission not granted message
Receive confirmation message
Platform
Platform
Receive battle creation data
Not enough or wrong
Send error message
Create battle
Send confirmation message
Permission not granted

Figure 12: Create battle BPMN

- **Educator notified of battle's end:** Educator gets notified about the end of a battle in one of his tournaments, then give him the possibility to manually evaluate the students' solutions.

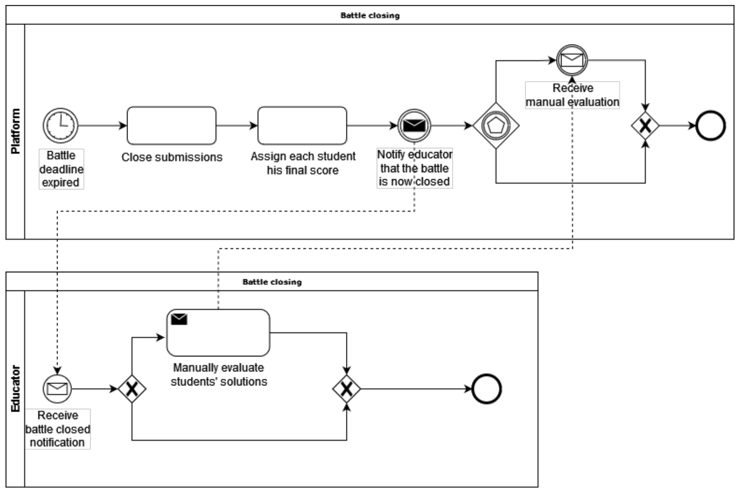

Battle closing
Platform
Battle deadline expired
Close submissions
Assign each student his final score
Notify educator that the battle is now closed
Receive manual evaluation
Battle closing
Educator
Receive battle closed notification
Manually evaluate students' solutions

Figure 13: Ending of a battle BPMN

2.3 User characteristics

CKB platform has 2 different users:

- **Educators:** They are Educators teaching in a school environment. They must be qualified to teach. They need to have medium to high programming skills, or at least enough knowledge of a programming language to create test codes for the battles they want to create. Finally they must be provided with an istitutional email in order to properly register to the platform.

- **Students:** They are Students of a school that want to improve their programming skills. They can be students of any school and they can subscribe to tournaments created by an Educators employed or not by their school. They must have some knowledge about the GitHub platform and how to create GitHub Actions, which are fundamental for the correct functioning of the CKB platform.

2.4 Assumptions, dependencies and constraints

D1: Students and Educators have access to internet while using the platform.

D2: Students and Educators have their own IT device to connect to the application.

D3: Students and Educators have to be subscribed to the platform in order to use its features.

D4: Students know how to use GitHub actions.

D5: Students know how to fork a repository on GitHub.

D6: A Student can join a battle only if subscribed to the tournament in which that battle takes place.

D7: GitHub platform offers reliable services through its API allowing the CKB platform to always get notified when new code is uploaded by Students.

D8: Educator knows how to create new badges and new rules to obtain them.

D9: Time information about registration and submission deadlines for battles are always correct.

D10: Code written by Students cannnot make the platform crash while testing it.

D11: Educators upload, when creating a battle, some correct, meaningful and faultproof test cases and automation scripts.

D12: Students score is always correctly calculated.

D13: Educators can always access to their tournaments and to the ones that has been granted the access to by their creators.

D14: In order to register to a battle, all the group members must be registered in the tournament in which the battle takes place.

D15: Students and Educators must have a functioning browser application installed on their devices.

D16: Only the Student that created the group can invite other Students in it.

3 Specific requirements

3.1 External interface requirements

3.1.1 User Interfaces

The system should interface with the users (both Educator and Students) through their devices, such as: laptops, PC desktops and smartphones, whom must be connected to the internet. Every user, in order to access the platform, has to connect to an existing domain (like "www.codekatabattle.com"), by using an appropriate and functioning browser application in order to navigate on the platform.

3.1.2 Software Interfaces

The system has to use different software interfaces in order to properly function:

- **DB Interfaces:** The system has to interact with a DB, used to store all information necessary for the system to function.
- **GitHub Interfaces:** The system has to interact with the GitHub platform, in order to receive the students' solutions. This is done through "GitHub Actions" APIs made available by GitHub to automate tasks.
- **Testing Interfaces:** The system needs to test the Students' solutions. In order to do this a Testing API must be used.

3.1.3 Hardware Interfaces

The system has to interface with the hardware components containing the DB.

3.1.4 Communication Interfaces

All communication from and to the CKB platform has to be done through the HTTP/HTTPS protocol.

3.2 Functional requirements

3.2.1 Requirements

R.1 The CKB platform should allow an unregistered Student to create a new account.

R.2 The CKB platform should allow an unregistered Educator to create a new account.

R.3 The CKB platform must allow access to its pages only if the used credentials are correct.

R.4 The CKB platform must not allow a Student to register more than once in the system.

R.5 The CKB platform must not allow an Educator to register more than once in the system.

R.6 Educators can access the platform's services only if they are registered to it.

R.7 Students can access the platform's services only if they are registered to it.

R.8 The CKB platform should not allow Students to create tournaments and/or battles.

R.9 The CKB platform should allow Educators to create battles within a tournament only to the tournament creator and to any other Educator that has been granted permission to do so by the tournament creator.

R.10 The CKB platform must allow Educators to personalise the tournaments they create.

R.11 The CKB platform must allow Educators to personalise the battles they create.

R.12 The CKB platform must allow Educators to define new obtainable badges for each tournament they create.

R.13 The CKB platform must allow Educators to manually evaluate the solutions uploaded by the Students for the battles that the Educators created.

R.14 The CKB platform must allow Educators to delete or update badges before finalizing a tournament's creation.

R.15 The CKB platform must allow Educators to define rules to obtain badges in tournaments created by them.

R.16 The CKB platform must ensure that badges' characteristics respect guidelines regarding their name, icon format and rules to obtain them.

R.17 The CKB platform must allow Educators to create new tournaments.

R.18 The CKB platform must ensure that tournaments' characteristics respect guidelines regarding their name, programming language.

R.19 The CKB platform must allow Educators to close tournaments they have created.

R.20 The CKB platform must ensure that when a tournament is closed, Educators cannot create new battles within it.

R.21 The CKB platform must ensure that if a group uploads a solution to a battle after the submission's deadline, that solution will not be considered in the score computation.

R.22 The CKB platform must ensure that the score given to a group in a tournament is coherent with scores given to the same group in the battles they have partecipated in.

R.23 The CKB platform must ensure fair competition between group scores. In the tournament's evaluation, the final group score should be the average score of all the battles in the tournament for each group. Any battle with no solution submitted will count as 0 points.

R.24 The CKB platform must allow Students to subscribe to a tournament.

R.25 The CKB platform must allow Students to subscribe to a tournament's battle within the registration deadline.

R.26 The CKB platform must allow Students to submit solutions to a tournament's battle within the battle's deadline relying on the external GitHub service.

R.27 The CKB platform must allow Students to send and receive group invitations to and from other Students in order to form groups.

R.28 The CKB platform should allow Students to join a battle only if the group composition rules for that battle are complied with.

R.29 The CKB platform must ensure that solutions uploaded by a Student for a battle are evaluated.

R.30 The CKB platform must ensure that only the latest solution uploaded by a Student for a battle he is subscribed to will be taken into consideration for the final score.

R.31 The CKB platform must allow groups partecipating in a battle to change their solution, if the battle's submission deadline hasn't expired yet.

R.32 The CKB platform must allow an Educator to modify the score for a Student's solution.

R.33 The CKB platform must ensure that when a new tournament is created, all Students subscribed to the platform are going to receive a notification.

R.34 The CKB platform must ensure that when a new battle is created in a tournament, all Students subscribed to that tournament are going to receive a notification.

R.35 The CKB platform must allow Students to visualise the score they obtained in a battle they partecipated in.

R.36 The CKB platform must allow Students to visualise the score they obtained in a tournament they partecipated in.

R.37 The CKB platform must allow Students to visualise the badges they obtained.

R.38 The CKB platform must ensures that battles' characteristics respect guidelines regarding their name, deadlines, programming language, number of member per group.

3.2.2 Use case diagrams

- **Student**

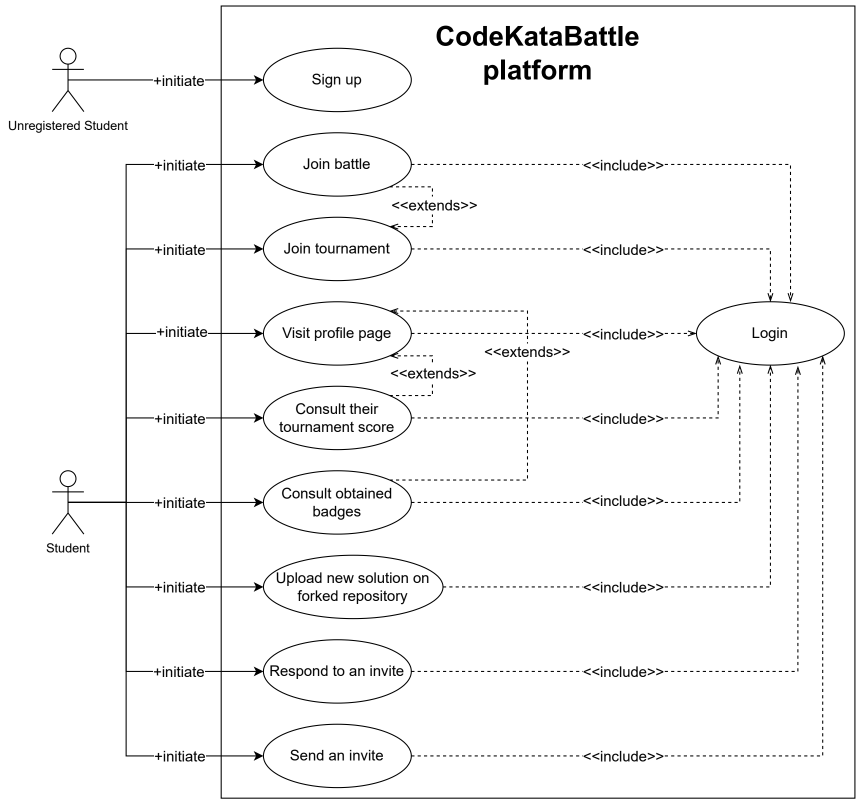


Figure 14: Student use case diagram

N.B: The repository citated in this diagram, on which the student upload their solution, it is not part of the platform. The system, in fact, relies on a third party service (GitHub) to handle the repository. This use case was specified in order to better clarify the tight interaction between the system and the Student through the GitHub service.

- **Educator**

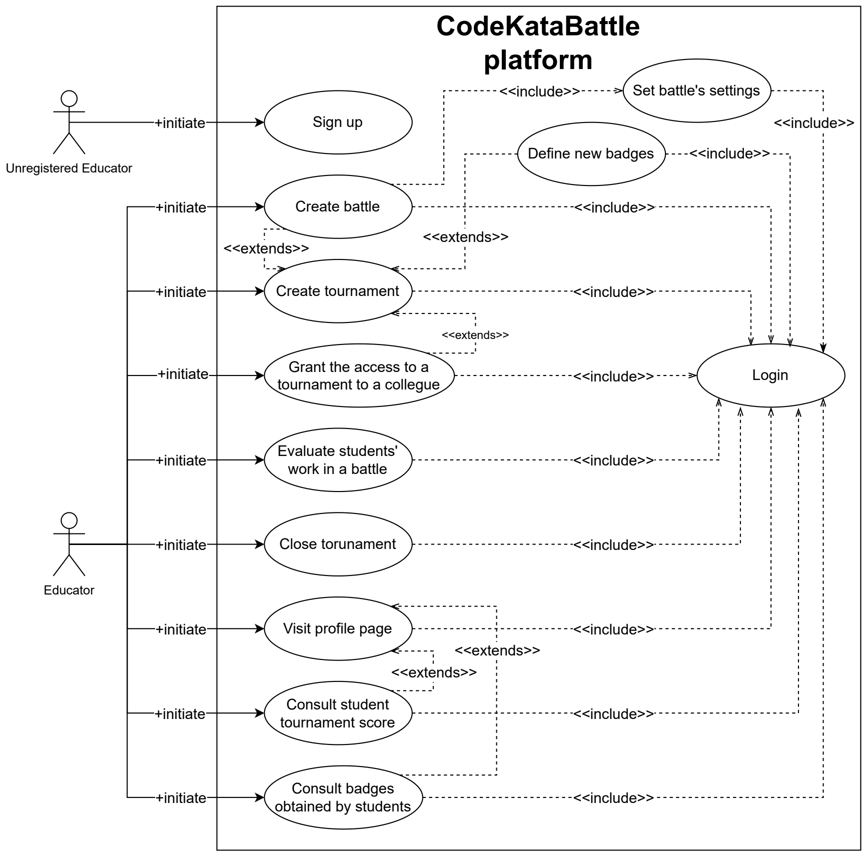


Figure 15: Educator use case diagram

- **GitHub**

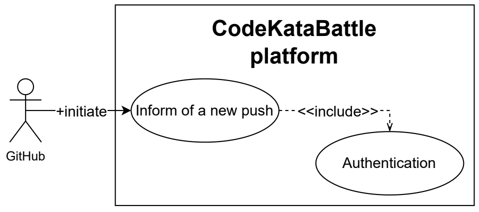


Figure 16: GitHub use case diagram

3.2.3 Use cases w/ sequence diagrams

1. **Student sign-up to the platform**

Name	Student sign-up
ID	UC.1
Actors	Unregistered Student
Entry condition	Student wants to register to the platform
Flow of events	1. Student opens CKB platform. 2. Student presses the sign-up button. 3. Student fills the form with all the required informations (name, surname, username attended school, email, password, ...), accepts the "Terms & Conditions". 4. Student clicks on a "Confirm" button to confirm. 5. CKB platform validates the personal information inserted by the Student. 6. CBK platform displays a confirmation message. 7. CKB platform sends an email notification to the Student regarding the registration outcome.
Exit condition	Student's account is created, and its data are saved into the system.

Exceptions	4.1 Email already used to register another account, or inexisting. 4.2 Inserted attended school non existings. 4.3 Username contains forbidden characters. 4.4 Password doesn't respect security standards. → Unregistered Student gets notified of the registration failure through an error message and flow restarts from point 3.

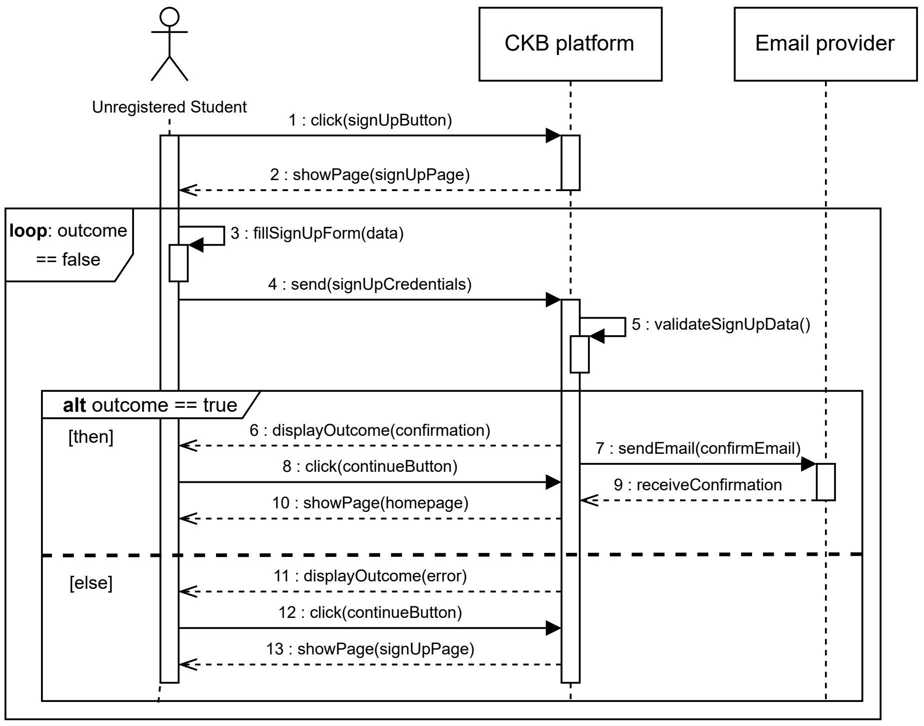


Figure 17: Student sign-up sequence diagram

2. Educator sign-up to the platform

Name	Educator sign-up
ID	UC.2
Actors	Unregistered Educator
Entry condition	Educator wants to register to the platform
Flow of events	1. Educator opens CKB platform. 2. Educator presses the sign-up button. 3. Educator fills the form with all the required informations (name, surname, username school in which teaches, istitutional email, password, ...), accepts the "Terms & Conditions". 4. Educator clicks on a "Confirm" button to confirm. 5. CKB platform validates the personal information inserted by the Educator. 6. CBK platform displays a confirmation message. 7. CKB platform sends an email notification to the Educator regarding the registration outcome.
Exit condition	Educator's account is created, and its data are saved into the system.
Exceptions	4.1 Email already used to register another account, or inexisting. 4.2 Inserted school's details non correct. 4.3 Username contains forbidden characters. 4.4 Password don't respect security standards. → Unregistered Educator gets notified of the registration failure through an error message and flow restart from point 3.

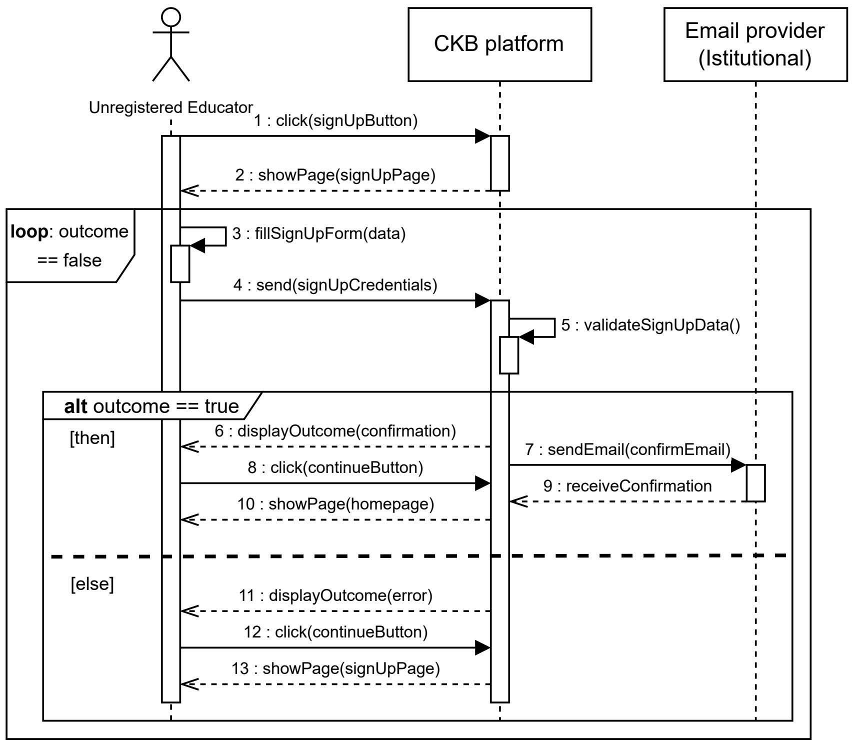


Figure 18: Educator sign-up sequence diagram

3. Educator creates a new tournament

Name	Tournament creation
ID	UC.3
Actors	Educator, Student
Entry condition	Educator has logged in the platform and wants to create a new tournament

Flow of events	1. Educator clicks on a ”Create tournament” button. 2. Educator fills a form on the page that has appeared with tournament details, such as: Allowed programming languages, name, etc. 3. Eventually the Educator defines new badges for the tournament (see UC.4) 4. Educator clicks on a ”Confirmation” button. 5. CKB platform checks the validity of the informations inserted by the Educator. 6. CKB platform displays a message that confirms that the tournament has been created successfully.
Exit condition	Tournament is created by saving its data into the system and the platform sends a notification to all the Students registered to the platform about the new tournament created. Educator is led back to the riepilogative page of the tournament creation.
Exceptions	5.1 Tournament’s name contain forbidden characters. 5.2 The programming language inserted is not recognized by the platform. → Educator gets notified about the tournament’s creation failure through an error message displayed by the platform. The flow restarts from point 2.

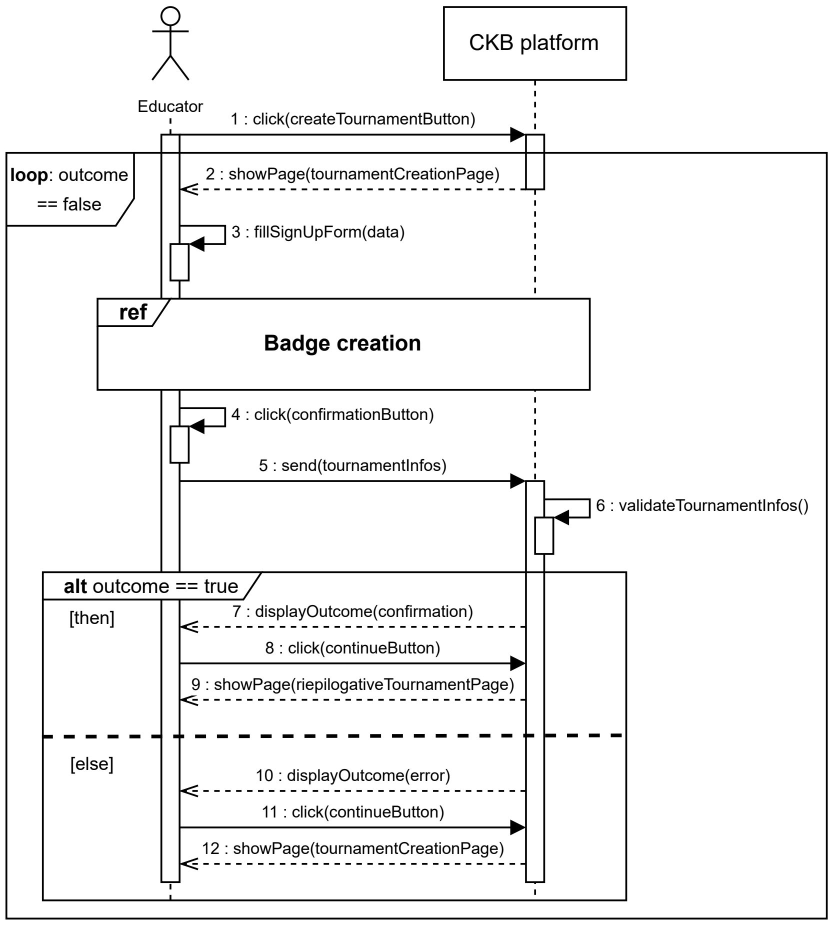

CKB platform
Educator
1 : click(createTournamentButton)
loop: outcome == false
2 : showPage(tournamentCreationPage)
3 : fillSignUpForm(data)
ref
Badge creation
4 : click(confirmationButton)
5 : send(tournamentInfos)
6 : validateTournamentInfos()
alt outcome == true
[then]
7 : displayOutcome(confirmation)
8 : click(continueButton)
9 : showPage(riepilogativeTournamentPage)
[else]
10 : displayOutcome(error)
11 : click(continueButton)
12 : showPage(tournamentCreationPage)

Figure 19: Educator creates a new tournament sequence diagram

4. Educator creates new badges

Name	Badges creation
ID	UC.4
Actors	Educator
Entry condition	Educator has logged in the platform, he's creating a new tournament and wants to define new badges for that tournament
Flow of events	1. Educator clicks on the "Create badge" button on the tournament creation page. 2. On the new page that appeared, the Educator defines badge's characteristics, such as: name, value, icon, rules to obtain it, ... 3. Educator clicks on "Confirm" button in order to confirm its choices. 4. CKB platform checks if all the badge informations are well defined. 5. CKB platform displays a message that confirms the successfull creation of a new badge, and that the badge has been added to the tournament.
Exit condition	Badge is created, its data are saved and associated to the tournament for which is being created by the Educator. The Educator is then led back to the tournament creation page.

Exceptions	4.1 Badge's name contain forbidden characters 4.2 The image uploaded as icon is too big or has an unsupported format. 4.3 Badge's rules are not well defined. 4.4 Some badge's features have been left empty → Educator gets notified about the badge's creation process failure through an error message displayed by the platform. Flow restarts from point 2.

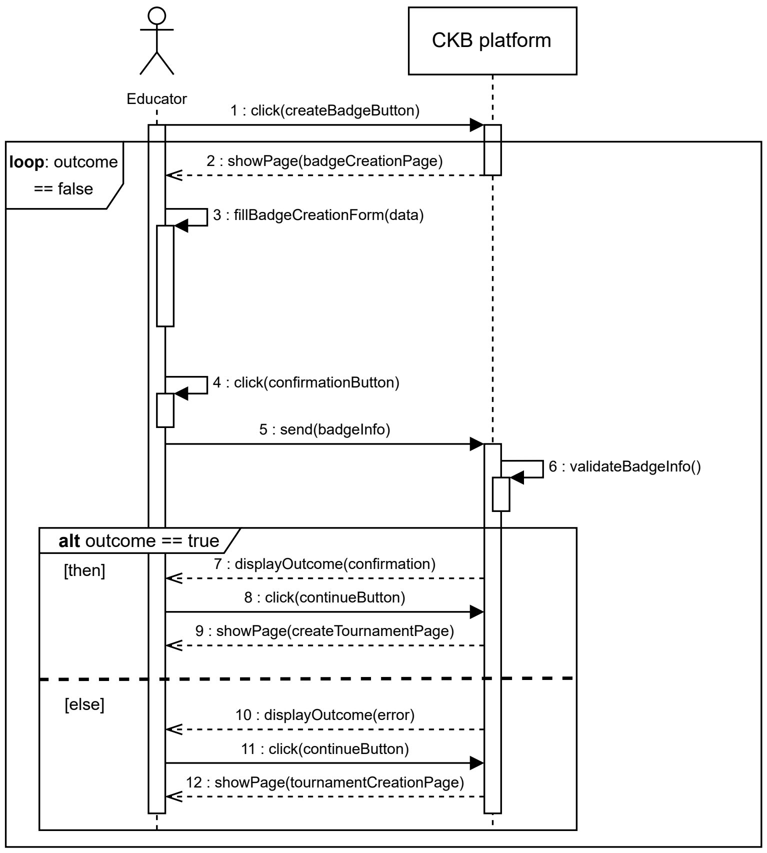

Educator
CKB platform
1 : click(createBadgeButton)
loop: outcome == false
2 : showPage(badgeCreationPage)
3 : fillBadgeCreationForm(data)
4 : click(confirmationButton)
5 : send(badgeInfo)
6 : validateBadgeInfo()
alt outcome == true
[then]
7 : displayOutcome(confirmation)
8 : click(continueButton)
9 : showPage(createTournamentPage)
[else]
10 : displayOutcome(error)
11 : click(continueButton)
12 : showPage(tournamentCreationPage)

Figure 20: Educator creates new badges for a tournament sequence diagram

5. Educator deletes and/or updates a badge

Name	Deletion & update of a badge
ID	UC.5
Actors	Educator
Entry condition	Educator has logged in the platform, has started the creation of a new tournament in which he has already created at least one badge and wants to delete and/or update one of this tournament's badges.
Flow of events	1. Educator clicks on the update button related to the badge and present on the tournament creation page. 2. CKB platform shows a modifiable form, filled with the badge's data that the Educator wants to modify. 3. Educator proceeds to modify the badge's data. 4. CKB platform tries to validate the new data. 5. Update is confirmed with the display of a confirmation message by the CKB platform. 6. Educator clicks on the delete button related to the badge and present on the tournament creation page. 7. CKB platform show a message requiring confirmation for the badge deletion. 8. Educator presses the confirmation button. 9. CKB platform deletes the badge and forwards the Educator back to the previous page.

Exit condition	Deleted badges are removed from the tournament and their data is removed from the platform. The modifications of the updated badges are committed to the platform. The Educator is led back to the tournament creation page.
Exceptions	4.1 Badge's updated name contain forbidden characters. 4.2 The new image uploaded as icon is too big or has an unsupported format. 4.3 New badge's rules are not well defined. 4.4 Some badge's features have been left empty. → Badge is not updated, Educator is led back to the tournament creation page. (point 1.). 6.1 Educator presses the cancel button → Delete procedure is stopped, Educator is led back to the tournament creation page (point 6.).

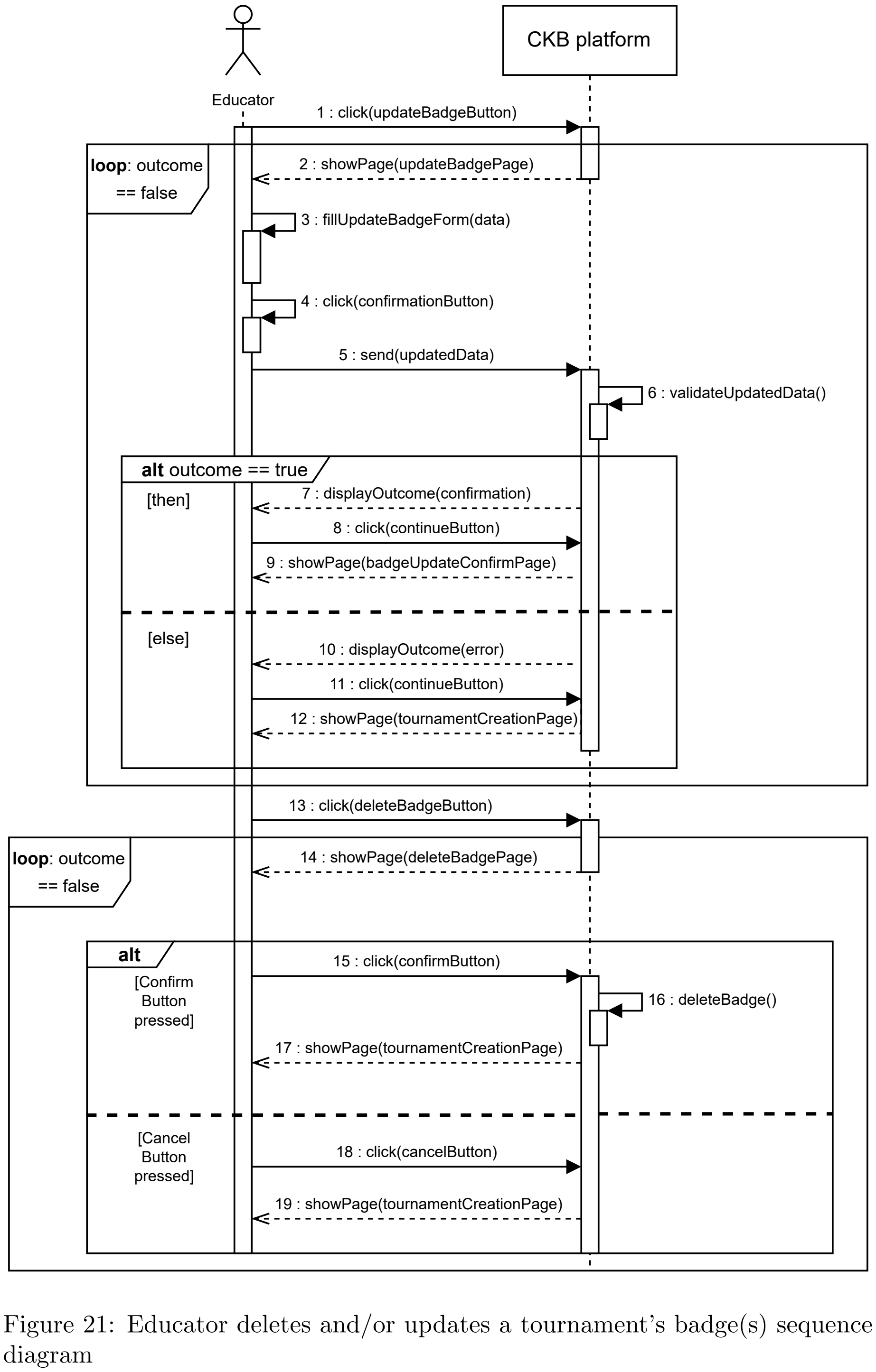
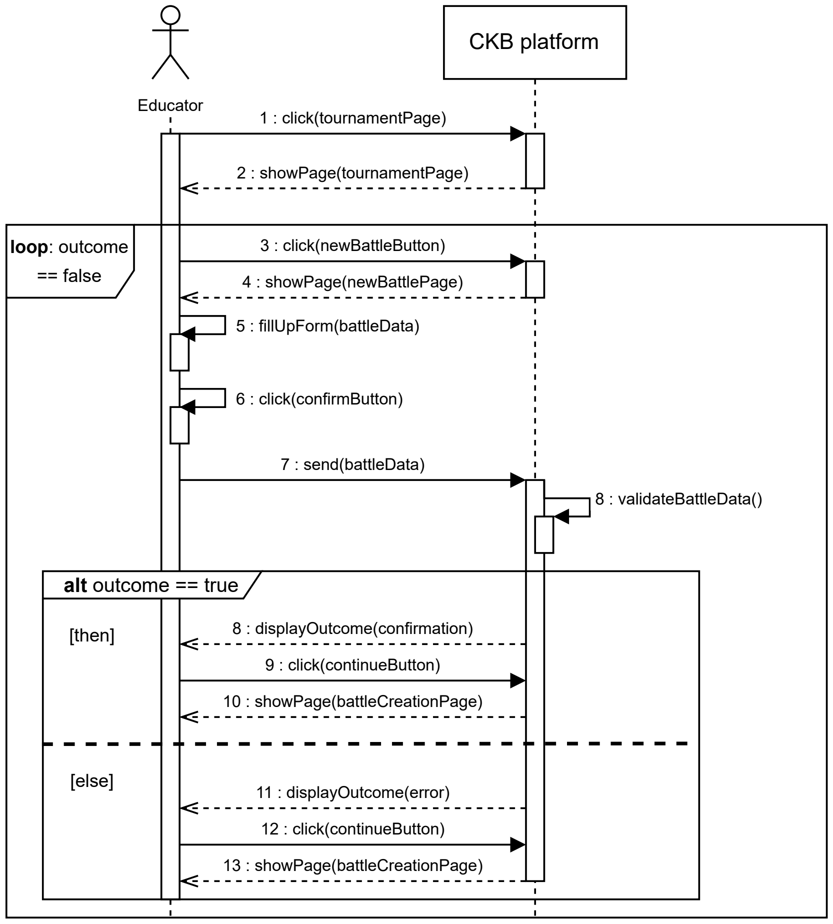

CKB platform
Educator
1 : click(updateBadgeButton)
loop: outcome == false
2 : showPage(updateBadgePage)
3 : fillUpdateBadgeForm(data)
4 : click(confirmationButton)
5 : send(updatedData)
6 : validateUpdatedData()
alt outcome == true
[then]
7 : displayOutcome(confirmation)
8 : click(continueButton)
9 : showPage(badgeUpdateConfirmPage)
[else]
10 : displayOutcome(error)
11 : click(continueButton)
12 : showPage(tournamentCreationPage)
13 : click(deleteBadgeButton)
loop: outcome == false
14 : showPage(deleteBadgePage)
alt
15 : click(confirmButton)
[Confirm Button pressed]
16 : deleteBadge()
17 : showPage(tournamentCreationPage)
[Cancel Button pressed]
18 : click(cancelButton)
19 : showPage(tournamentCreationPage)

Figure 21: Educator deletes and/or updates a tournament's badge(s) sequence diagram

6. Educator creates a new battle

Name	Battle creation
ID	UC.6
Actors	Educator
Entry condition	Educator has logged in the platform, has created at least one tournament and wants to add a battle in a specific tournament.
Flow of events	1. Educator opens the page related to a specific tournament which he has created. 2. Educator clicks on the button used to create a new battle. 3. Educator fills the form shown by the platform with informations related to the battle that he wants to create (name, programming languages allowed, maximum and minimum number of students for each partecipating group, registration and submission deadlines, ...). 4. Educator clicks on a button to confirm. 5. CKB platform validates the battle characteristics. 6. CKB platform displays a confirmation message of the successfull battle's creation.
Exit condition	New battle, with characteristics specified by the Educator, is created within the tournament chosen by the Educator himself. All the Students subscribed to the tournament in which the battle has been created get notified. Educator is then led to the riepilogative page of the tournament's battles.

Exceptions	3.1 Battle's name contains forbidden characters. 3.2 A programming language inserted is not included in the tournament allowed programming languages. 3.3 The specified number of minimum or maximum Students for each group is beyond upper and/or lower limits imposed by the platform. 3.4 Registration deadline inserted for the battle is not valid or bigger than the submission one. 3.5 Submission deadline inserted for the battle is not valid or smaller than the registration one. → Educator gets notified about battle's creation failure by the platform through an error message. The flow restarts from point 3.



Educator
CKB platform
1 : click(tournamentPage)
2 : showPage(tournamentPage)
loop: outcome == false
3 : click(newBattleButton)
4 : showPage(newBattlePage)
5 : fillUpForm(battleData)
6 : click(confirmButton)
7 : send(battleData)
8 : validateBattleData()
alt outcome == true
[then]
8 : displayOutcome(confirmation)
9 : click(continueButton)
10 : showPage(battleCreationPage)
[else]
11 : displayOutcome(error)
12 : click(continueButton)
13 : showPage(battleCreationPage)

Figure 22: Educator creates a new battle sequence diagram

7. Educator closes a tournament

Name	Tournament closure
ID	UC.7
Actors	Educator
Entry condition	Educator has logged in the platform, has created at least one tournament and wants to close one of them in order to end the competition
Flow of events	1. Educator accesses the tournament's page that he wants to close. 2. Educator checks whether there are battles that aren't already closed. 3. Educator clicks on the tournament's close button, then clicks on the confirm button. 4. CKB platform verifies if there are no battles still active within the tournament. 5. CKB platform notifies the Educator through a message that confirms the deletion of the tournament.
Exit condition	The chosen tournament is closed, its data are kept saved in order to create an history of tournaments. Educator is led back to the riepilogative page of the tournaments he created.
Exceptions	2.1 Active battles are found within the tournament. → Educator can prevent other Educators from creating new battles if he's the tournament creator. The flow restarts from point 1. 4.1 Active battles are found within the tournament. → CKB platform interrupts the tournament deletion and displays an error message to the Educator. Flow restarts from point 1.

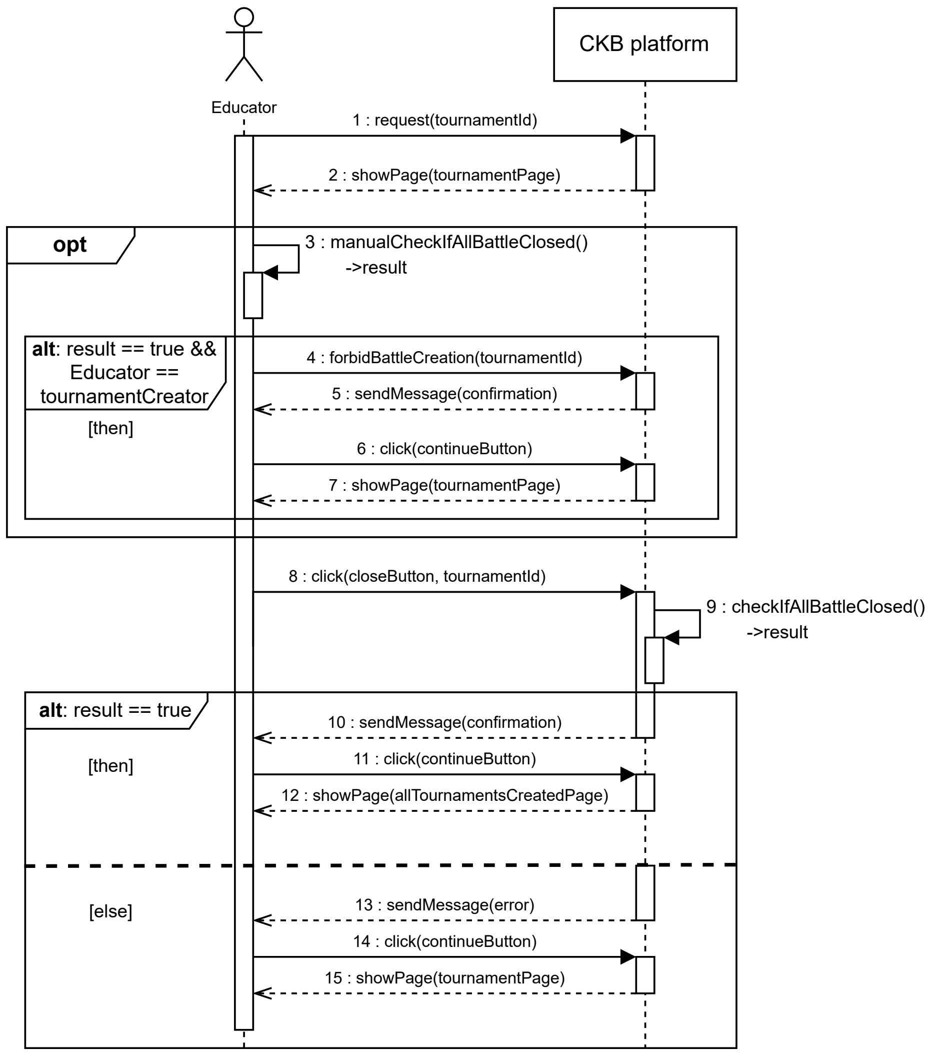

Educator
CKB platform
1 : request(tournamentId)
2 : showPage(tournamentPage)
opt
3 : manualCheckIfAllBattleClosed() ->result
alt: result == true && Educator == tournamentCreator
[then]
4 : forbidBattleCreation(tournamentId)
5 : sendMessage(confirmation)
6 : click(continueButton)
7 : showPage(tournamentPage)
8 : click(closeButton, tournamentId)
9 : checkIfAllBattleClosed() ->result
alt: result == true
[then]
10 : sendMessage(confirmation)
11 : click(continueButton)
12 : showPage(allTournamentsCreatedPage)
[else]
13 : sendMessage(error)
14 : click(continueButton)
15 : showPage(tournamentPage)

Figure 23: Educator closes a tournament sequence diagram

8. Educator evaluates a battle's results

Name	Battle's results evaluation
ID	UC.8
Actors	Educator
Entry condition	Educator has logged in the platform, has created at least one battle within a tournament that has terminated. Educator wants to manually evaluate the results
Flow of events	1. Educator accesses the page related to the tournament he has created. 2. Educator accesses the page related to a terminated battle within that tournament and that is in the consolidation phase. 3. Educator manually evaluates groups' works by decreasing or increasing their scores according to his personal preferences or the ones accorded with the students. 4. Educator after evaluating all the groups proceeds to definitively close the battle.
Exit condition	Groups' scores within the battle get updated by the platform, battle is definitively closed and the Educator is led back to the riepilogative page of the tournament whom the battle belonged.
Exceptions	2.1 The battle cannot be manually evaluated. 2.2 Accessed battle isn't terminated. → CKB platform doesn't allow the Educator to manually evaluate the battle. Flow restarts from point 2.

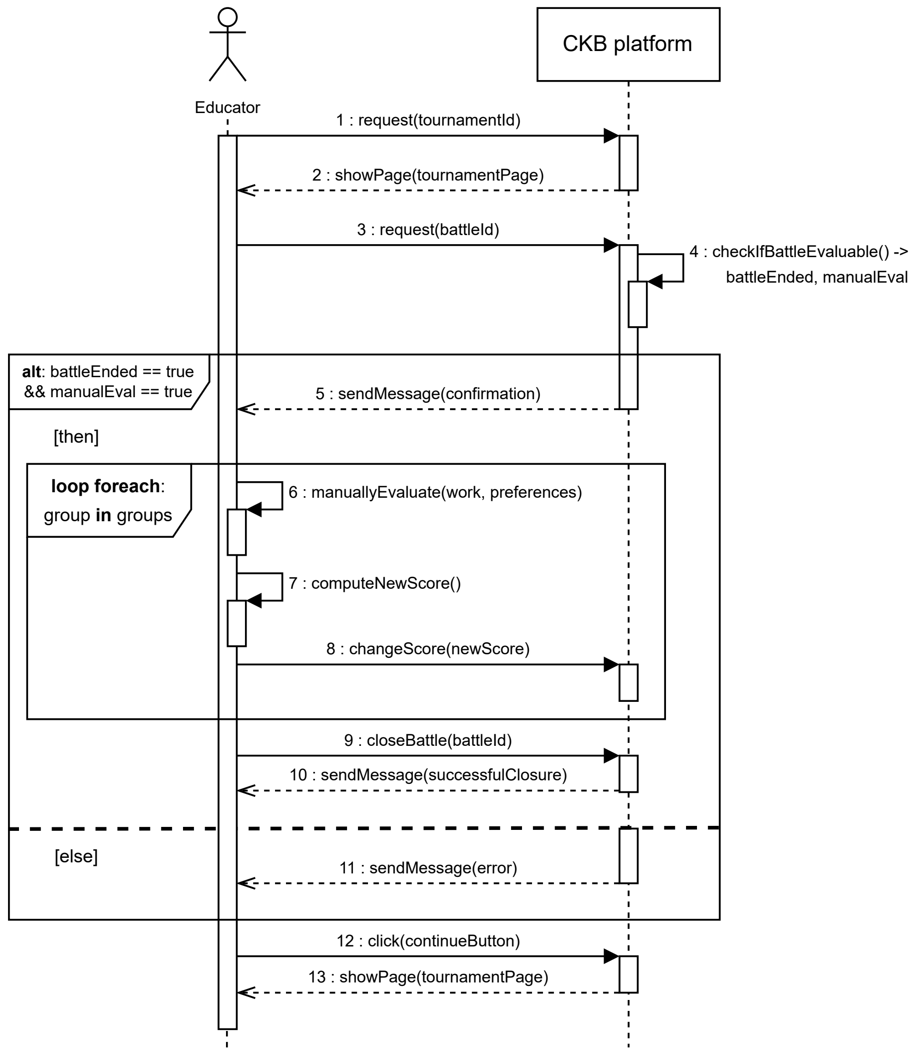

Educator
CKB platform
1 : request(tournamentId)
2 : showPage(tournamentPage)
3 : request(battleId)
4 : checkIfBattleEvaluable() -> battleEnded, manualEval
alt: battleEnded == true && manualEval == true
5 : sendMessage(confirmation)
[then]
loop foreach: group in groups
6 : manuallyEvaluate(work, preferences)
7 : computeNewScore()
8 : changeScore(newScore)
9 : closeBattle(battleId)
10 : sendMessage(successfulClosure)
[else]
11 : sendMessage(error)
12 : click(continueButton)
13 : showPage(tournamentPage)

Figure 24: Educator evaluates manually a battle's score sequence diagram

9. Student forms a group

Name	Group formation
ID	UC.9
Actors	Student
Entry condition	Student has logged in the platform, has subscribed to a tournament and wants to form a group in order to later join a battle within that tournament.
Flow of events	1. Student clicks on a button that leads to an appropriate page to form a group between Students. 2. Student choose the filters to search one or more other Students to invite. 3. CKB platform computes all the filters and returns to the Student the available Students, according to the filters he inserted. 4. Student chooses the other Students to whom send the invite from the showed page. 5. Student clicks on confirm button to send the invitations. 6. CKB platform proceeds to send the invitations to all Students specified and sends a notification with it in order to notify the receivers. 7. Invited Students accept the invitation. 8. CKB platform sends notification to the Student.
Exit condition	The group is formed and registered by the CKB platform. The Student is led back to the tournament's riepilogative page.

Exceptions	3.1 The set of Students returned applying the filters doesn't contain any available Student. → CKB platform shows an empty list, with a message explaining the absence of Students in the list. Flow restarts from point 2. 3.2 The set of Students returned contains only Students already in a group. → CKB platform shows the Students returned from the computation but doesn't allow the Student to select them. Flow restart from point 2. 7.1 One or more invited Students don't accept the invitation → CKB platform notifies the Student that sent the invitation of the received refusal. Flow restarts from point 1.

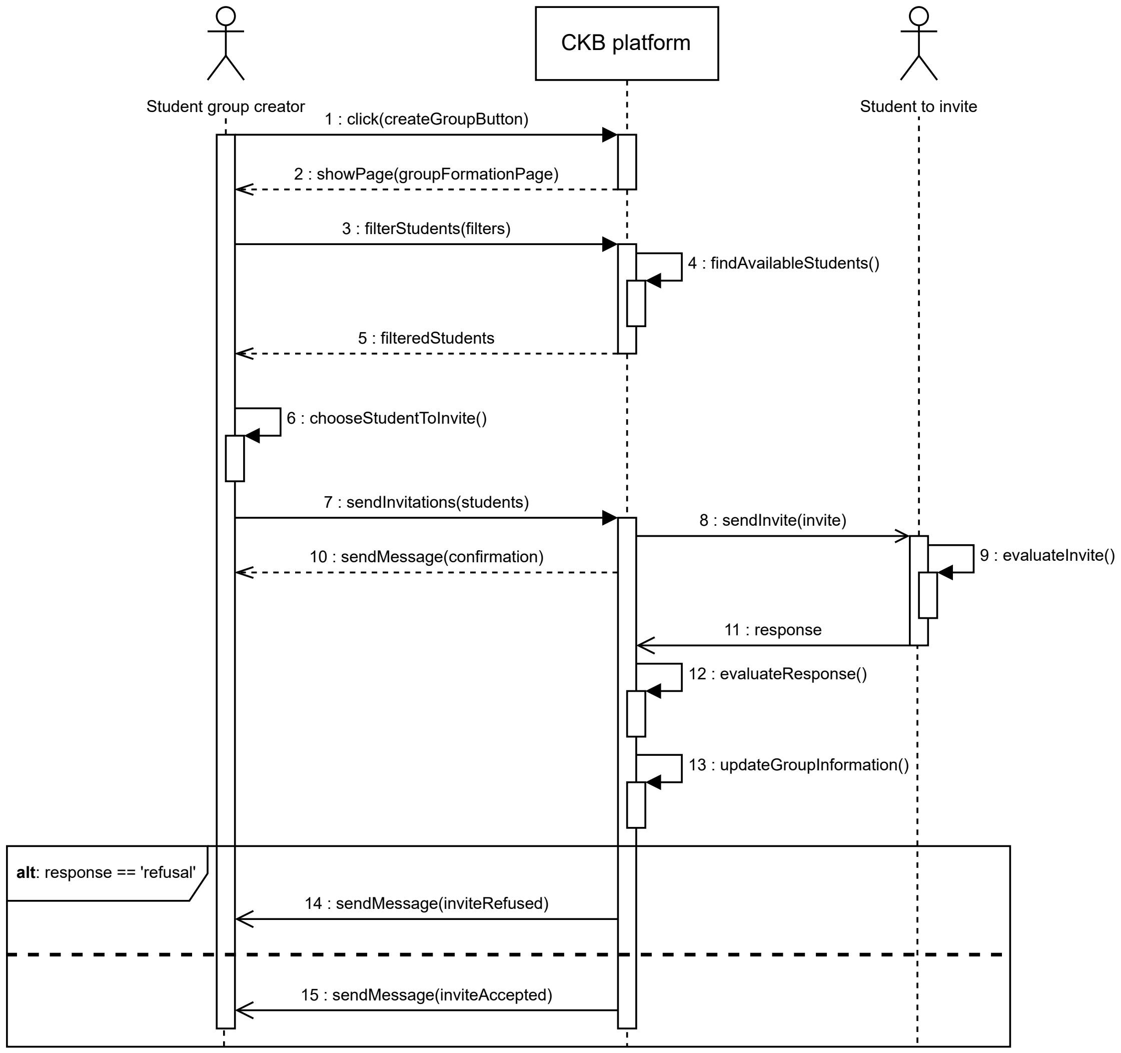

CKB platform
Student group creator
Student to invite
1 : click(createGroupButton)
2 : showPage(groupFormationPage)
3 : filterStudents(filters)
4 : findAvailableStudents()
5 : filteredStudents
6 : chooseStudentToInvite()
7 : sendInvitations(students)
8 : sendInvite(invite)
9 : evaluateInvite()
10 : sendMessage(confirmation)
11 : response
12 : evaluateResponse()
13 : updateGroupInformation()
alt: response == 'refusal'
14 : sendMessage(inviteRefused)
15 : sendMessage(inviteAccepted)

Figure 25: Students form a new group sequence diagram

10. Student joins a battle

Name	Battle joining
ID	UC.10
Actors	Student
Entry condition	Student has logged in the platform, has subscribed to a tournament and wants to join a battle within that tournament.
Flow of events	1. Student enters the tournament's page and subsequently the battle's page. 2. Student clicks on the button to join the battle. 3. CKB platform shows a riepilogative page of the battle. 4. Student clicks on the button to subscribe his group to the battle. 5. CKB platform checks whether the group respects all of the battle requirements. 5. CKB platform returns a confirmation message to all the group members. 6. CKB platform sends a notification to the group members when the subscription deadline expires and battle starts.
Exit condition	The Student and his group have joined the battle. CKB platform starts saving all the solutions and statistics regarding the group's work. The Student is led to the riepilogative page of the battle.
Exceptions	5.1 The group doesn't respect the battle's requirements (too many or too few members). 5.2 The group contains members not subscribed to the tournament in which the battle is held. → CKB platform shows an error message.

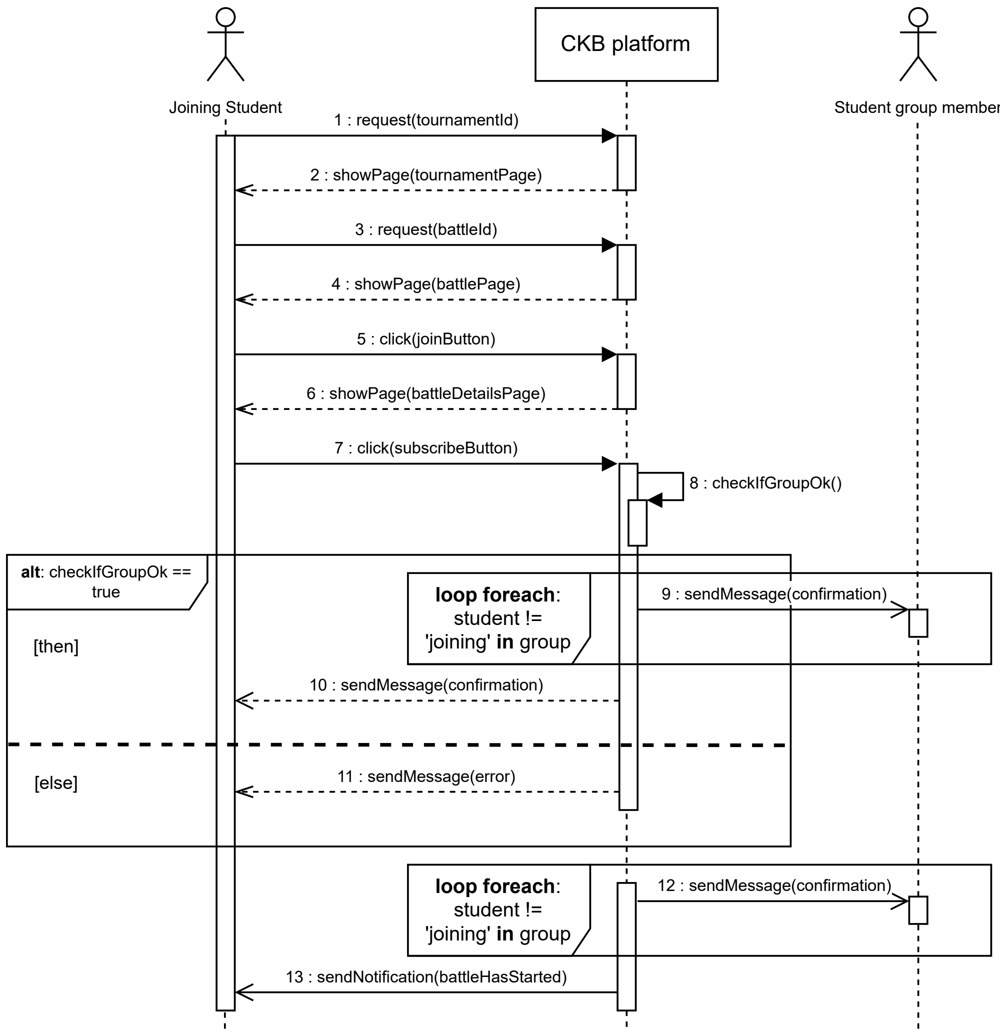

Joining Student
CKB platform
Student group member
1 : request(tournamentId)
2 : showPage(tournamentPage)
3 : request(battleId)
4 : showPage(battlePage)
5 : click(joinButton)
6 : showPage(battleDetailsPage)
7 : click(subscribeButton)
8 : checkIfGroupOk()
alt: checkIfGroupOk == true
[then]
loop foreach: student != 'joining' in group
9 : sendMessage(confirmation)
10 : sendMessage(confirmation)
[else]
11 : sendMessage(error)
loop foreach: student != 'joining' in group
12 : sendMessage(confirmation)
13 : sendNotification(battleHasStarted)

Figure 26: Student joins a battle within a tournament, sequence diagram

11. Student uploads a new solution

Name	Upload of a solution within deadline
ID	UC.11
Actors	Student, GitHub
Entry condition	Student has a GitHub account, has registered to at least one tournament and one battle in it, and wants to upload a new solution to that battle.
Flow of events	1. Student accesses his GitHub account. 2. Student uploads group's solution to his GitHub repository, precedently forked. 3. GitHub platform, through its "GitHub Actions" notifies the CKB platform of the upload of the new solution by the Student. 4. CKB platform receives the new solution uploaded and starts testing it. 5. CKB platform gives a score to the solution based on the tests' results. 6. CKB platform updates the group's score in the battle and tournament. 7. CKB platform notifies the group's members of the successful testing.
Exit condition	Student's (or his group's) score is updated based on the uploaded solution.
Exceptions	2.1 Generic upload error. → Handled by GitHub, flow restarts from point 2. 4.1 Tested code generated major issues, such as infinite loops, huge resources consumption, ... → CKB platform terminates the testing phase and notifies the Student's group with an error message instead of the solution score, flow ends.

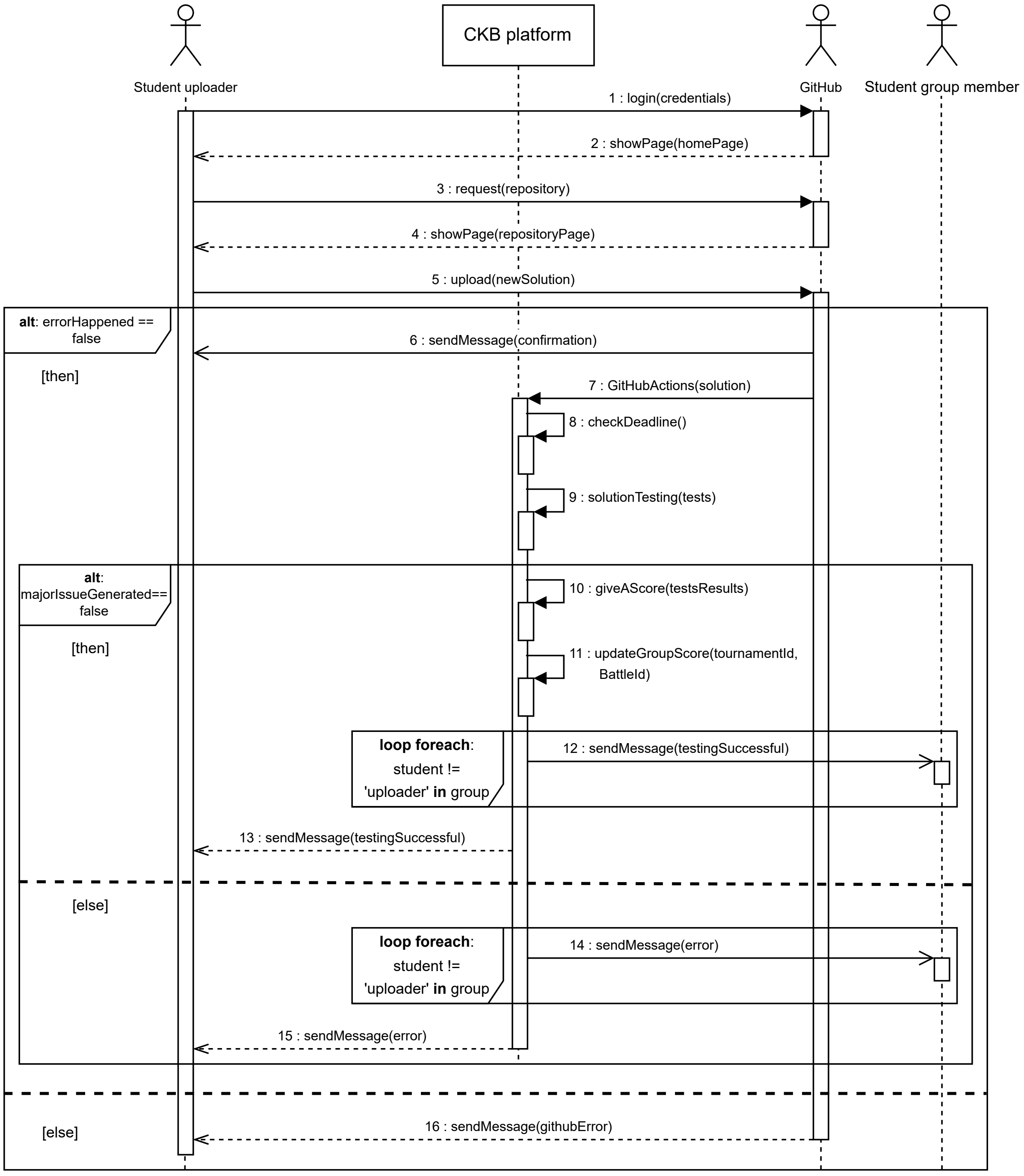

CKB platform
Student uploader
GitHub
Student group member
1 : login(credentials)
2 : showPage(homePage)
3 : request(repository)
4 : showPage(repositoryPage)
5 : upload(newSolution)
alt: errorHappened == false
6 : sendMessage(confirmation)
[then]
7 : GitHubActions(solution)
8 : checkDeadline()
9 : solutionTesting(tests)
alt: majorIssueGenerated== false
10 : giveAScore(testsResults)
[then]
11 : updateGroupScore(tournamentId, BattleId)
loop foreach: student != 'uploader' in group
12 : sendMessage(testingSuccessful)
13 : sendMessage(testingSuccessful)
[else]
loop foreach: student != 'uploader' in group
14 : sendMessage(error)
15 : sendMessage(error)
[else]
16 : sendMessage(githubError)

Figure 27: Student uploads a new solution of the battle's problem within the deadline, sequence diagram

12. Student uploads a solution after submission deadline

Name	Solution upload exceeding deadline
ID	UC.12
Actors	Student
Entry condition	Student has logged in the platform, has registered to at least one tournament and one battle within it, and wants to upload a new solution to that battle, but the submission deadline expired.
Flow of events	1. Student accesses his GitHub account. 2. Student uploads his solution to the group's GitHub repository, precedently forked. 3. GitHub platform, through his "GitHub Actions" notifies the CKB platform of the Student's new solution's upload. 4. CKB platform receives the new solution uploaded and sees that the submission deadline of the battle for which the solution was uploaded has expired. 5. CKB platform notifies the group's members of the error by sending an error message.
Exit condition	Student's (or his group's) receives a notification by the CKB platform stating that their last uploaded solution will not be considered on the final score since the submission deadline has expired.
Exceptions	2.1 Generic upload error. → Handled by GitHub, flow restarts from point 2.

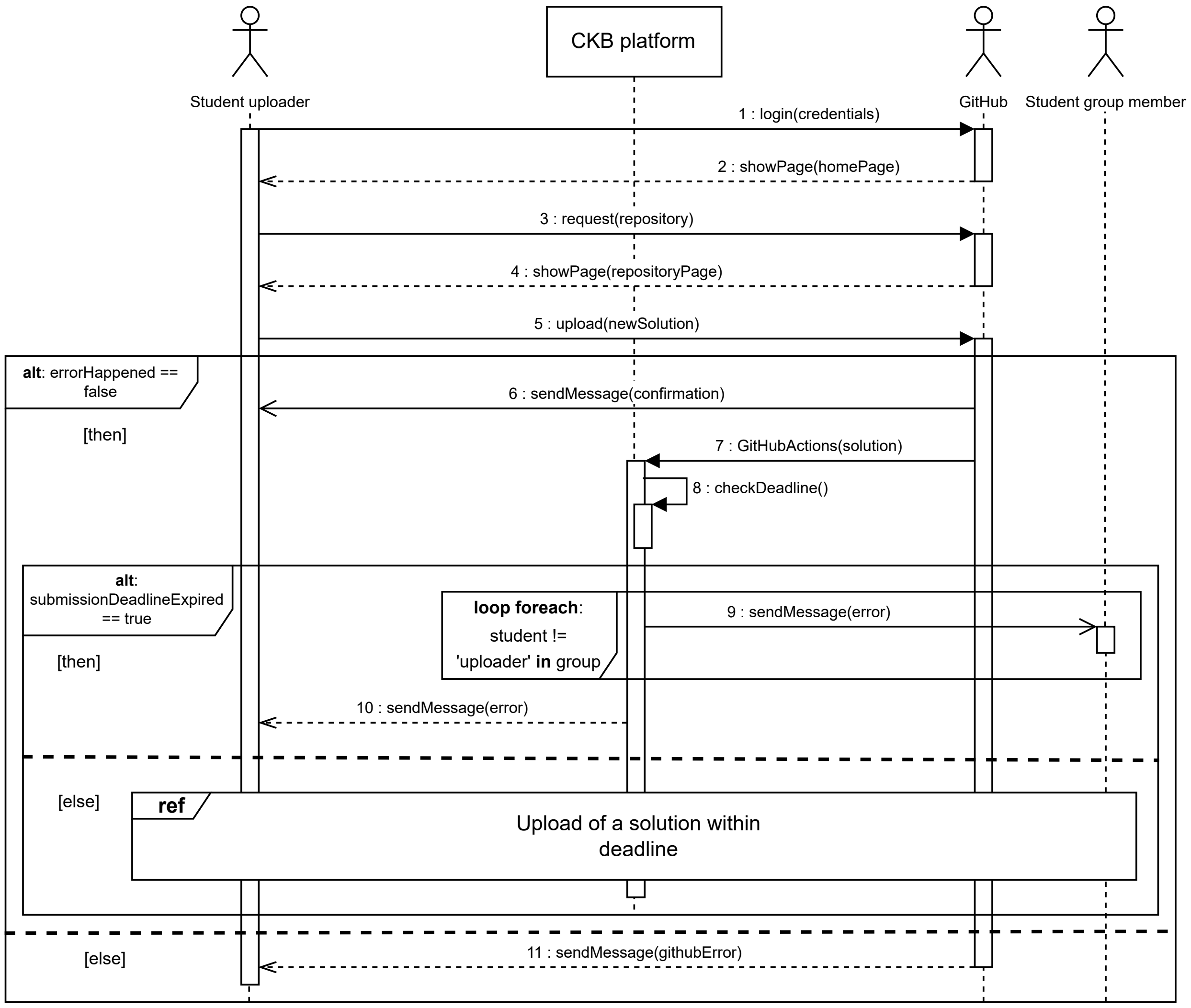

CKB platform
Student uploader
GitHub
Student group member
1 : login(credentials)
2 : showPage(homePage)
3 : request(repository)
4 : showPage(repositoryPage)
5 : upload(newSolution)
alt: errorHappened == false
6 : sendMessage(confirmation)
[then]
7 : GitHubActions(solution)
8 : checkDeadline()
alt: submissionDeadlineExpired == true
loop foreach: student != 'uploader' in group
9 : sendMessage(error)
[then]
10 : sendMessage(error)
[else]
ref
Upload of a solution within deadline
[else]
11 : sendMessage(githubError)

Figure 28: Student uploads a solution after submission deadline sequence diagram

13. Student visualizes his tournament's results and badges

Name	Viewing results and badges
ID	UC.13
Actors	Student
Entry condition	Student has logged in the platform and wants to see his or others' tournament's scores and badges.
Flow of events	1. Student enters in his profile page or in the profile page of the Student he wants to see the tournament scores and badges. 2. Student chooses the tournament of which he wants to see the details from the appropriate section. 3. Student consults his tournament scores and badges
Exit condition	Student has consulted his tournament score and badges obtained during it. The Student can see several details about all the tournaments in which he partecipated. Flow ends when the Student goes back to a precedent page.
Exceptions	2.1 The Student has not yet participated in any tournament. → CKB platform still allows the Student to access the profile page of the Student that hasn't partecipated to any tournament, but doesn't show anything regarding tournaments. Flow ends.

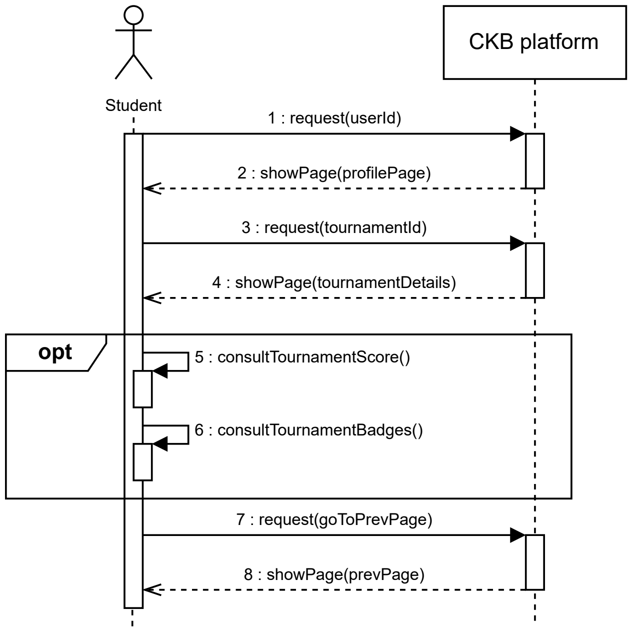

Student
CKB platform
1 : request(userId)
2 : showPage(profilePage)
3 : request(tournamentId)
4 : showPage(tournamentDetails)
opt
5 : consultTournamentScore()
6 : consultTournamentBadges()
7 : request(goToPrevPage)
8 : showPage(prevPage)

Figure 29: Student visualises his/other score(s) and badge(s), sequence diagram

3.2.4 Traceability matrix

Goal ID	Req. ID	Use case ID	Assump. ID
G.1	R.2, R.3, R.5, R.6, R.8, R.10, R.12, R.14, R.15, R.16, R.17, R.18	UC.2, UC.3, UC.4, UC.5	D.1, D.2, D.3, D.8, D.9, D.15
G.2	R.2, R.3, R.5, R.6, R.8, R.9, R.11, R.38	UC.2, UC.6	D.1, D.2, D.3, D.9, D.11, D.15
G.3	R.2, R.3, R.5, R.6, R.19, R.20	UC.2, UC.7	D.1, D.2, D.3, D.13, D.15
G.4	R.2, R.3, R.5, R.6, R.13, R.32	UC.2, UC.8	D.1, D.2, D.3, D.12, D.15
G.5	R.1, R.3, R.4, R.7, R.24	UC.1	D.1, D.2, D.3, D.9, D.15
G.6	R.1, R.3, R.4, R.7, R.25, R.27, R.28	UC.1, UC.10	D.1, D.2, D.3, D.4, D.5, D.6, D.9, D.14, D.15
G.6.1	R.1, R.3, R.4, R.7, R.27	UC.1, UC.9	D.1, D.2, D.3, D.14, D.15, D.16
G.6.2	R.1, R.3, R.4, R.7, R.21, R.26, R.29, R.30, R.31	UC.1, UC.11, UC.12	D.1, D.2, D.3, D.4, D.5, D.7, D.9, D.10, D.15
G.6.3	R.1, R.3, R.4, R.7, R.22, R.23, R.35, R.36, R.37	UC.1, UC.13	D.1, D.2, D.3, D.12, D.15
G.7	R.1, R.3, R.4, R.7, R.33, R.34	UC.1, UC.3, UC.6	D.1, D.2, D.3, D.15

3.3 Performance requirements

We expect thousands of registrations to the platform, due to the possible subscription of several private and public academic environments (especially high schools). In the subsequent years we expect to have way less subscriptions, since the first year every school and university student will have the need to subscribe to the platform. After this first boom of subscriptions, in the subsequent years only new students will have to subscribe, hence there is no need to have such a high tolerance in scalability. There is also to consider the fact that not many people will access the platform at the same time, since once a Student has subscribed to a battle, he can work locally and only interact with the GitHub platform. We expect the CKB platform to be accessed only when a new tournament or battle is created, and when a tournament or a battle is closed, so that Students can access the platform and check their results. Response time and reliability are our main focuses. We need to guarantee a low response time in order to let Students and Educators interact actively with the platform without high delays that could compromise Students' works. We have to take into consideration reliability since with a low reliability platform when Students upload their solutions on the forked GitHub repository, triggering GitHub Actions to upload the solutions to the CKB platform, if the CKB platform is not reliable, it could cause delays in the solution's uploading to the platform. A low reliable platform can also mean that Student's solutions may not be correctly graded, since the platform may incur in a byzantine failure type and erroneusly attributes points to some solutions. Educators would also not be able to access the platform and manually evaluate Students' solutions, and would be a bad experience overall. For data storing, we expect we'll need a high capacity DB. It's necessary for storing the user's data, tournaments and battles' data and statistics, informations and code about groups' commits. We'll need a high capacity DB, since being the platform built around students and schools, we expect to have lots of users subscribing together at the same time, when school starts. These times are differenciated throught the year, for example in Italy schools starts in Semptember, while in New Zealand it starts in July. The platform being based around schools means that we'll have lots of user's data related to old students that are now not using the platform anymore.

3.4 Design constraints

3.4.1 Hardware constraints

The users, both Educators and Students, to have access to the platform need to have a functioning internet access and a device connected to the net. The Educators must have at least a smartphone, so that they can perform all of the different functions the platform offers them, like creating new battles and tournament, badges and manually evaluate groups' solutions. All of this can be done via browser installed on a smartphone. The Students are required to have a personal computer, either a desktop or a laptop, such that they can write and submit their solutions to the platform. Students can access the platform via smartphone too, for consulting their profile and looking at new tournaments and battles, but it would be best if everything that is concerning to coding and submitting a solution is done via personal computer.

3.4.2 Privacy constraints

Since the main application area is the european one, the platform must be compliant to the GDPR's law. Data must be encrypted before it is saved on the DB, no data must be accessed by any user other than the data's owner. For preventing cyber attacks, data must pass through secure connections and channels. In this case, no *Man in the Middle* attacks can occur. Autentication and login will be provided by the CKB platform, so all data must be treated accordingly to the platform's privacy policy. When a new User, both Educator and Student, wants to register to the platform, the privacy policy must be shown and accepted, before the new user is saved in the DB. If the privacy policy is not accepted, the user registration procedure is not completed and the new user is not saved in the DB, not granting access to the person that was trying to register to the platform.

3.5 Software system attributes

3.5.1 Usability

The platform must be easy to use. Users must not have doubts on which page to go to perform a precise task. Each button and link must be named accordingly, to limit time waste and useless interaction with the platform.

3.5.2 Reliability

The platform must prevent downtime in order to notify Students of new tournaments or battles as soon as they are created, in order to let Educators create new tournaments and battles whenever they want and for correctly publish new evaluations and grades on the platform for Students to see and access. We expect the highest number of simultaneous access to be whenever schools start and end, since more students will register on school starting, and more will rush to finish battles and projects as school is ending.

3.5.3 Availability

The platform must be available as much as possible, at least 99% of the time. Both Students and Educators must have access to the platform whenever they need to.

3.5.4 Security

Communication between Users and the Platform must pass through secure channels, and it must be encrypted. With a web certificate for the platform we can ensure that the SSL protocol is followed, and can ensure security for the platform and its Users. The DB guarantees that performed operations are all authorized, and permitted. *A Student cannot create new battles, for example.*

3.5.5 Web Browsing

The platform must be usable on any major web browser, since access to it is fully online.

3.5.6 Maintainability

The platform must be designed in a way that eases the future adding of new features with the least possible effort. Maintainability is also key for having a functioning platform, since the easier it is to maintain, the easier errors, reliability and availability failures can be corrected.

4 Formal analysis using Alloy

4.1 Signatures

```
//Base signatures used to identify an Educator and a Student
sig Name {}
sig Email {}
sig Password {}

//Used to represent an Educator subscribed to the CBK platform
sig Educator {
        name : one Name,
        email : one Email,
        password : one Password
}

//A Permit is given by an Educator to a set of other Educators, and regards
only one tournament created by the permit's sender
sig Permit {
        tournamentCreator : one Educator,
        battleCreators : some Educator,
        tournament : one Tournament
} {
        tournamentCreator not in battleCreators and
        tournament.creator = tournamentCreator }

//A tournament has a creator and a set of Educators that can create battles in
it
sig Tournament {
        name : one Name,
        creator : one Educator
}

//A battle has a creator and a tournament to which it belongs, the creator must
be the tournament's creator or another Educator that can create battles within
that tournament
sig Battle {
        name : one Name,
        tournament : one Tournament,
        creator : one Educator,
        tests : set Test,
        battleState : one BattleState
}
//Test represents a battle's test
sig Test {}

//Students subscribed to the CKB platform can have a set of badges they ob-
tained and a set of tournaments they are subscribed to
sig Student{
        name : one Name,
        email : one Email,
        password : one Password,
        badges : set Badge,
        tournament : set Tournament
}

//A forked repository is assigned to a group and is relative to one battle
sig ForkedRepository{
        group : one Group,
        battle : one Battle
}{
        group.battle = battle
}

//A group is formed by some Students and partecipates to one battle
sig Group{
        groupCreator : one Student,
        students : set Student,
        battle : lone Battle
}{
        groupCreator not in students
}

//A badge is created by an Educator, is relative to one tournament and has
some rules that needs to be checked before is obtained by any Student
sig Badge{
        creator : one Educator,
        tournament : one Tournament,
        rules : some Rule
}{
        creator = tournament.creator
}
```

```
//A rule needs to be defined for a badge to be created
sig Rule{}

//An invite is made by a Student of a group and directed to another Student
sig Invite{
        sender : one Student,
        receiver : one Student,
        group : one Group,
        inviteState : one InviteState
}{
        sender != receiver and
        sender = group.groupCreator
}

//Signature that defines the battle's state
abstract sig BattleState{}

one sig RegistrationPhase extends BattleState{}
one sig SubmissionPhase extends BattleState{}
one sig ConsolidationPhase extends BattleState{}
one sig ClosedBattle extends BattleState{}

//Signature that defines the invite's state
abstract sig InviteState{}

one sig Accepted extends InviteState{}
one sig Refused extends InviteState{}
one sig Waiting extends InviteState{}
```

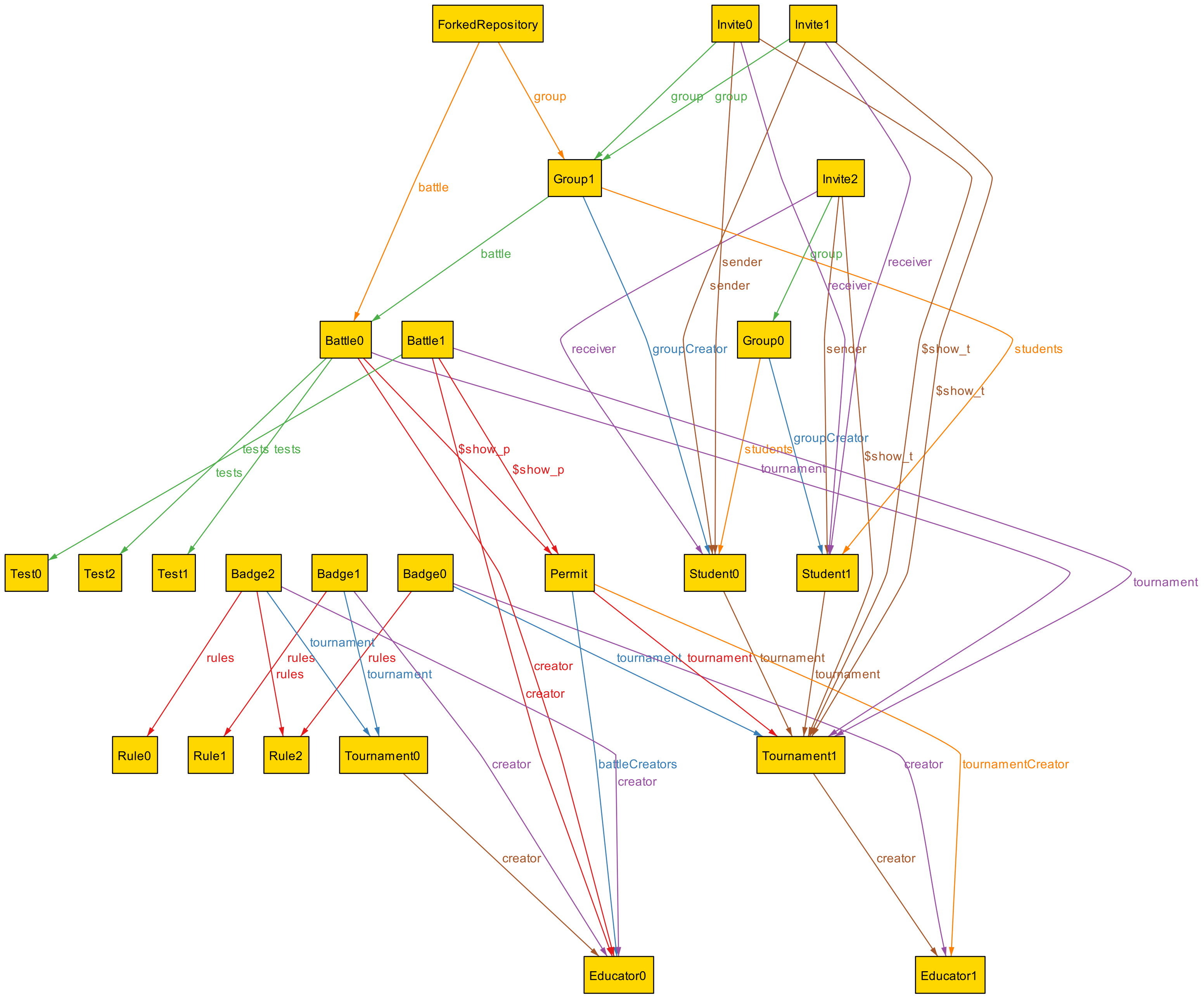
4.2 Facts and Predicates

```
//A tournament's creator can grant to (only once per Educator) another Edu-
cator the permission to create battles in his tournament
pred NewGrantedPermit[ t : Tournament, e : Educator, p, p' : Permit ]{
//Preconditions
        e != t.creator
        e not in p.battleCreators
        t = p.tournament
//Postconditions
        p'.battleCreators = p.battleCreators + e
        p'.tournamentCreator = p.tournamentCreator
        p'.tournament = p.tournament
}
```

```
//A group can have at most one repository for each battle
fact onlyOneRepository {
        all disj r1, r2 : ForkedRepository |
        r1.battle != r2.battle or
        r1.group != r2.group
}

//For each battle, a Student can participate in it while being in only one group
fact noStudentsInMoreGroupInABattle{
         all s : Student | all disj g1, g2 : Group |
        ((s in g1.students and s in g2.students ) or
        (s = g1.groupCreator and s = g2.groupCreator ) or
        (s in g1.students and s = g2.groupCreator ) or
        (s = g1.groupCreator and s in g2.students )) iff
        (( g1.battle != g2.battle ) or (g1.battle = none and g2.battle = none))
}

//The sender and the receiver of a group invitation must have at least one
tournament subscription in common
fact SenderAndReceiverInSameTournament{
         all i : Invite | some t : i.receiver.tournament |
         t in i.sender.tournament
}

//Students must be subscribed to at least one battle to receive badges from
the tournament that contains that battle
fact StudentReceiveBadges{
         all s : Student | all b : s.badges | all g : Group |
         ((s in g.students) or (s = g.groupCreator)) and
         g.battle.tournament = b.tournament
}

//Students must be subscribed in a tournament to compete in that tournament's
battles
fact StudentsGroupBattleInTournament{
         all g : Group | all s : g.students |
         g.battle.tournament in s.tournament and
         g.battle.tournament in g.groupCreator.tournament
}
```

```
//A Student needs to receive at least one invite from the leader of a group to
be a member of that group
fact StudentInAGroup{
        all g : Group | all s : g.students | some i : Invite |
        i.sender = g.groupCreator and
        i.receiver = s and
        i.group = g
}

//Educator must has the permission to create battle in a tournament or be
the tournament's creator
fact battleCreatorHasPermit{
        all b : Battle | some p : Permit |
        (b.creator in p.battleCreators or
        b.creator = p.tournamentCreator) and
        p.tournament = b.tournament
}
```

4.3 Examples of instances



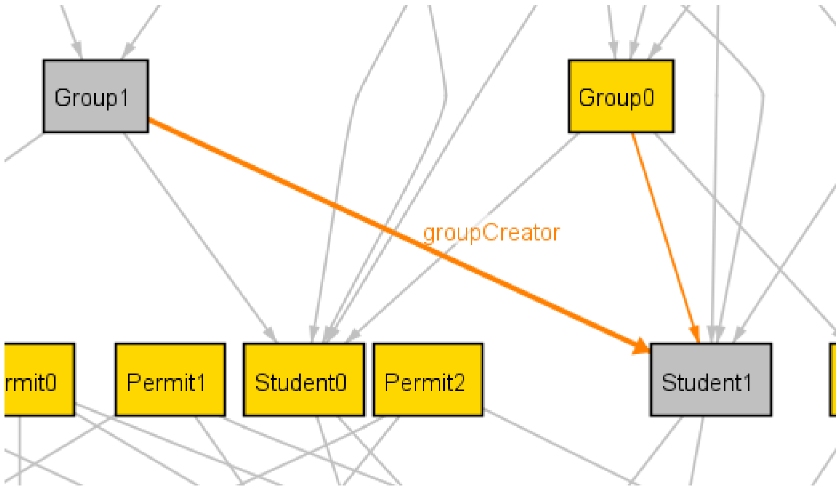


Figure 31: Student1 have created Group0 and Group1

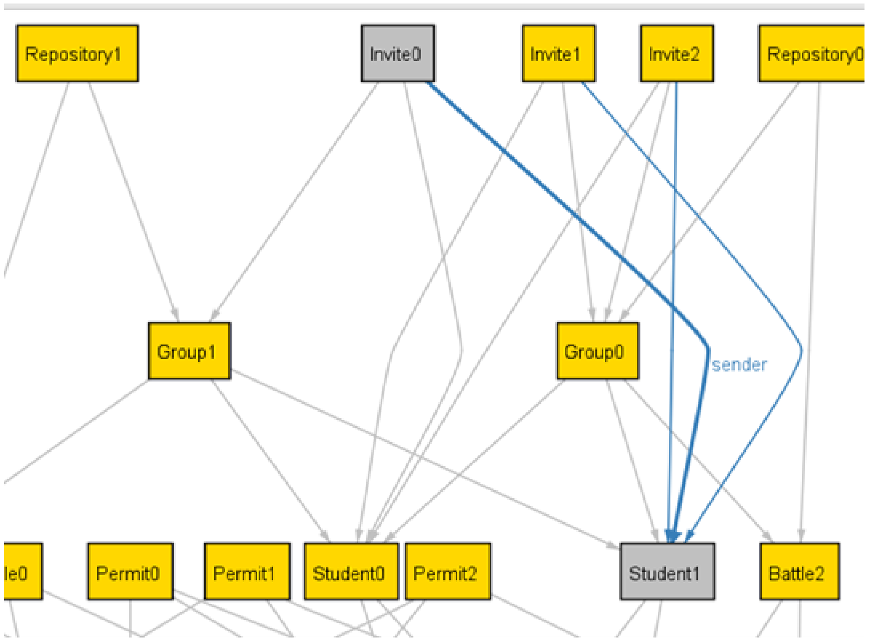


Figure 32: Student1 send 3 invite: Invite0, Invite1, Invite2

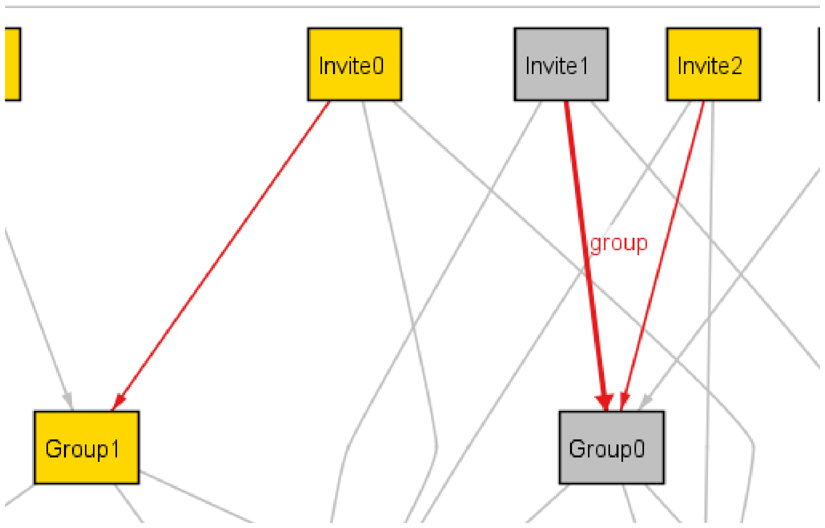


Figure 33: Each invite is relative to one group

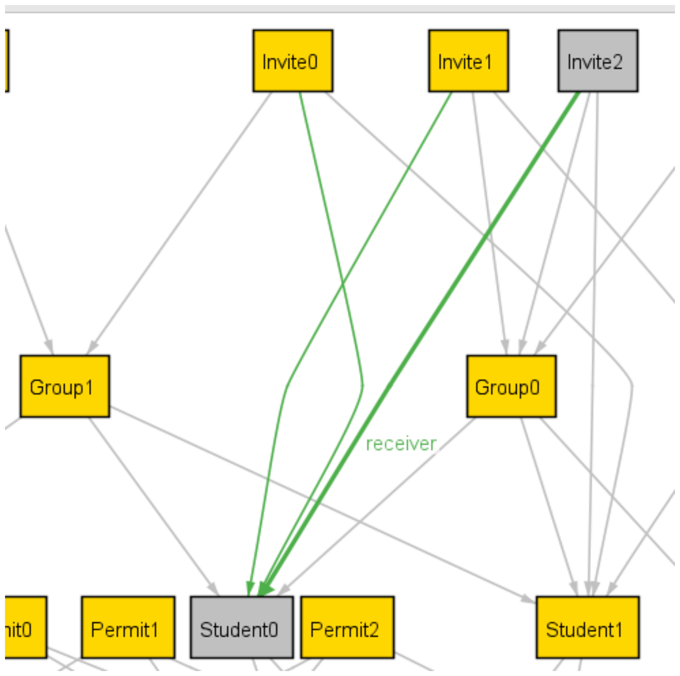


Figure 34: All invites are being sent to Student0

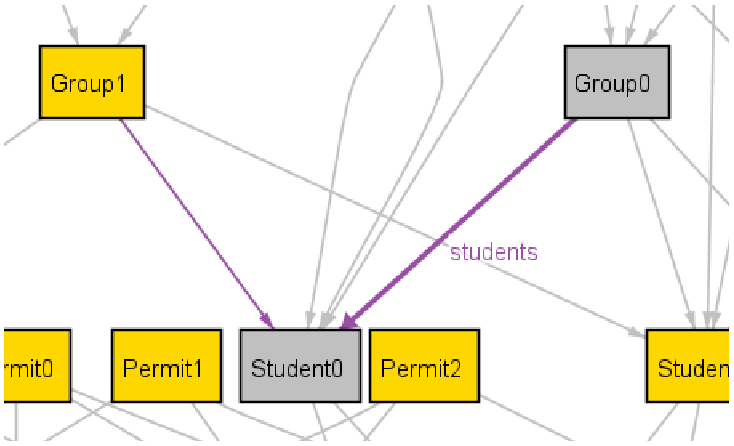


Figure 35: Here we can see Student0 in group1 and group0, since he has received two invites feom group0's creator, Student0 has rejected the first invite, but has accepted the second one

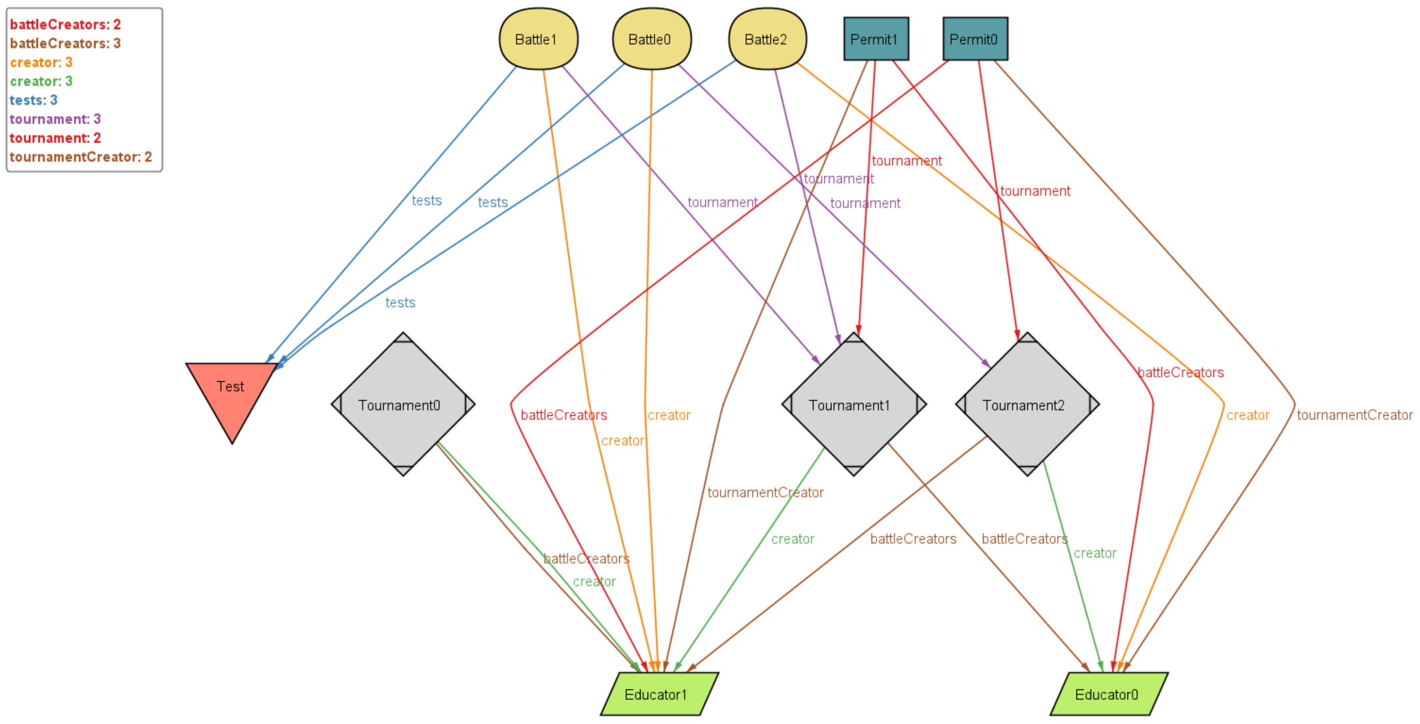


Figure 36: tournament, educator and battle

Every permit gives an Educator who isn't the creator, the possibility of creating new battles inside another Educator's tournament. The tournament's creator must grant permission to the other Educator, via a permit. (battleCreators)
Every battle is created by an Educator, he can be the tournament's creator or an Educator that has received modification permit by the tournament's creator (creator) [Ex. Battle0 is created by Educator1 in Tournament2, that is created by Educator0. Educator0 grants Educator1 Permit0 so that Educator1 can create Battle0 in Tournament2]
Every tournament can be created by only one Educator (creator)
Tests may be reused amonst more battles, while a test has to be relative to at least one battle, it can be utilized by more than one (tests)
A battle is only in one tournament (tournament)
A permission is relative to one tournament created by an Educator (tournament)
A permission can only be granted by an Educator that created at least one tournament (tournamentCreator)

5 Effort spent

Group member		**Effort spent**
Francesco Spangaro	Introduction	*6h*
	Overall description	*14h*
	Specific Requirements	*8h*
	Formal analysis using Alloy	*2h*
Luca Tosetti	Introduction	*6h*
	Overall description	*7h*
	Specific Requirements	*19h*
	Formal analysis using Alloy	*2h*
Francesco Riccardi	Introduction	*6h*
	Overall description	*10h*
	Specific Requirements	*2h*
	Formal analysis using Alloy	*13h*

6 References

6.1 Paper references

- Specification document Assignment RDD AY 2023-2024.pdf
- Alloy course's slides

6.2 Used tools

- GitHub for project versioning.
- Diagrams.net for UML,Use case,Sequence and BPMN diagrams.
- Visual Studio Code as LaTeX editor.
- Alloy analyzer for alloy code's formal analysis.