POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# CodeKataBattle - RASD v1.0

## Software Engineering 2 Project
## MSc Computer Science and Engineering

Authors: **Michele Bersani, Paolo Chiappini, Andrea Fraschini**
Reference Professor: **Prof. Matteo G. Rossi**

Student IDs: **10707192, 10743051, 10725610**
Academic Year: **2023-24**
Revision: **v1.0 10/12/23**
GitHub repository: **https://github.com/paolo-chiappini/BersaniChiappiniFraschini**

# Contents

# 1. Introduction

## 1.1. Purpose

CodeKataBattle (CKB for short) is a web-based platform designed for students and educators. Its primary objective is to improve students' coding abilities by offering an environment for collaborative training on coding challenges.

Educators can use the CKB platform to challenge students by organizing tournaments and code kata battles, setting registration and submission deadlines, as well as scoring the students' solutions, thus creating a structured learning environment.

A key benefit of utilizing the platform lies in its automatic testing tools, which provide valuable support to both students and educators during the learning journey. These automation tools allow students to receive automated evaluations for their solutions by integrating with GitHub. This streamlined evaluation process also assists educators in their assessment, enhancing efficiency and objectivity.

The platform also offers gamification badges, which are set up by educators and automatically assigned by the platform to students. These badges serve to motivate students using the platform by providing a fun addition to the learning process.

In summary, CKB provides a comprehensive environment for students to practice and improve their coding skills through code challenges enriched with gamification aspects. The platform also simplifies the work of educators by making the organization of educational activities easier.

## 1.2. Definitions, abbreviations, acronyms

### 1.2.1 Definitions

- **Automated Assessment**: the platform conducts static code analysis and performs tests for the provided code through automatic workflows.
- **Static code analysis**: software verification activity that analyzes source code for quality, reliability, and security without executing the code.
- **Software Platform**: set of software and hardware used to provide a certain service or to host an application.
- **Users**: a collective noun that encompasses both student and educator accounts. It refers to the people using the services offered by the platform.
- **Authenticated users**: users that have completed the login process, and therefore have been authenticated in the eyes of the CKB platform.
- **Code kata**: code katas are programming exercises in a language of choice aimed at helping students learn and improve at coding.
- **(Students) group size**: the size of a group of students is defined by the amount (number) of students that form the group.
- **Solution**: refers to any code created with the explicit purpose of addressing a specific problem or challenge. In this specific context, solutions are the pieces of code submitted by students to be evaluated in battles.
- **(Gamification) Badges**: elements in the form of rewards that represent the achievements of individual students during tournaments.
- **Charset:** set of symbols (characters) that are supported by the platform.
- **POST:** software action used to send data to a server to create/update a resource.
- **RESTful API:** API that follows the REST architecture.
- **Authentication token:** an authentication token is a set of characters used to identify and authenticate a user (or set of users).
- **Automatic workflow:** sequence of automated actions.

### 1.2.2 Abbreviations

- **[Gn]:** n-th goal.
- **[Wn]:** n-th world phenomena.
- **[SPn]:** n-th shared phenomena.
- **[Dn]:** n-th domain assumption.

- **[Dn.m]:** m-th sub-domain assumption of domain assumption [Dn].
- **[Rn]:** n-th functional requirements.
- **[Rn.m]:** m-th sub-requirement of functional requirements [Rn].
- **[UCn]:** n-th use case.

- **CKB:** CodeKataBattle platform.
- **S2B:** System to Be, refers to the system to implement.
- **AA**: Automated Assessment.
- **The platform,** or simply **Platform**: refers to the CKB platform.
- **ECA**: External Code Analyzers.
- **ECMAScript:** standard for scripting languages, including, among others, JavaScript.

### 1.2.3 Acronyms

- **API**: Application Programming Interface.
- **GUI**: Graphical User Interface.
- **WCAG**: Web Content Accessibility Guidelines.
- **UTF-8**: Unicode Transformation Format 8-bit.
- **HTTP:** HyperText Transfer Protocol.
- **HTTPS:** HyperText Transfer Protocol Secure.
- **JSON:** JavaScript Object Notation.
- **REST**: Representational State Transfer.

## 1.3. Scope

The platform enables educators to create challenges by first organizing tournaments. Tournaments provide a context for educators to host kata battles, along with colleagues who have been invited to manage the tournament. When creating a tournament, educators provide a deadline for the subscription of students. Throughout tournaments, educators have the opportunity to create and set gamification badges that will be awarded to students upon the tournament's conclusion. Educators can create battles within tournaments by setting up a skeleton project, which includes a description, automation scripts, and tests. They can also define registration and submission deadlines, minimum and maximum sizes for student groups, and scoring parameters. These scoring parameters may include the requirement for manual evaluation on behalf of the educators or the utilization of static analysis tools capable of assessing specific aspects of the code written by students.

Students will be allowed to subscribe to tournaments and form groups to enroll in battles within the given deadlines and parameters set by educators. After the enrollment period concludes, the platform will automatically generate a GitHub repository for the respective battle and send an invitation link to the students who have enrolled. Subscribing to a tournament enables students to receive notifications related to changes in the tournament's status, such as new upcoming battles, rank updates, and tournament conclusion. Students will also be notified when a new tournament is created. The platform provides students with an assessment of their work, assigning points and a ranking for each battle. These points will contribute to the calculation of the overall score for each student and determine their ranking in tournaments.

### 1.3.1 Goals

- **[G1]**: Students want to participate as groups (or by themselves) in coding battles to enhance their software development skills.
- **[G2]:** Students want to receive notifications from tournaments and battles that they are subscribed to, as well as when new tournaments are created.
- **[G3]:** Users want to access information on ongoing tournaments, tournament rankings, battles, and badges earned by students.

- **[G4]:** Educators want to create and manage tournaments and battles along with invited colleagues to challenge students and allow them to demonstrate and improve their programming skills.
- **[G5]:** Educators want to create and customize gamification badges that will be automatically assigned to students.
- **[G6]:** Educators want the platform to provide automated assessments of students' work while also enabling manual evaluations based on the specified evaluation options.
- **[G7]:** CKB owners want users to be registered and authenticated on the platform to know which group they belong to (either "students" or "educators").
- **[G8]:** Platform owners want to integrate with GitHub to allow students to work on a shared repository.

## 1.4. The World and The Machine

The following section provides a view of the world and machine phenomena as described by M. Jackson and P. Zave *[Jackson 95][Jackson-Zave 95]*. The machine represents the system to be (S2B), in our case the CKB platform. The world represents the environment in which the machine is meant to work. Shared phenomena represent the interface between the world and the machine.

### 1.4.2 World phenomena

- **[W1]**: Educators want to improve students' software development skills.
- **[W2]**: Students want to compete against each other in software development battles.
- **[W3]**: Students create their own GitHub Account.
- **[W4]**: ECA analyzes the code.
- **[W5]**: Educators create the base project for each code kata battle.
- **[W6]**: Student forks the GitHub repository of a battle.
- **[W7]**: Student sets up the automated workflow in their forked repository.

### 1.4.3 Shared Phenomena

Phenomena controlled by the world and observed by the machine.

- **[SP1]**: Educator creates a tournament.

- **[SP2]**: Educator closes a tournament.
- **[SP3]**: Educator grants permission to create battles to another educator.
- **[SP4]**: Educator creates a new code kata battle.
- **[SP5]**: Educator defines a new badge for a specific tournament.
- **[SP6]**: Educator assigns their score to a team for a battle of a specific tournament.
- **[SP7]**: User subscribes to the CKB platform.
- **[SP8]**: User logs in to the CKB platform.
- **[SP9]**: Student forms a new team for a battle.
- **[SP10]**: Student invites another student to their group.
- **[SP11]**: Student joins a battle in a group or by themselves.
- **[SP12]**: Student pushes a new commit into the main branch of their repository.
- **[SP13]**: For each kata battle, educators create the skeleton project and tests (e.g. in Java with Gradle).
- **[SP14]**: External code analyzers produce a result.

Phenomena controlled by the machine and observed by the world.

- **[SP15]**: CKB notifies users of successful registration on the platform.
- **[SP16]**: CKB notifies students, who are subscribed to the CKB platform, of the presence of new tournaments.
- **[SP17]**: CKB notifies students, who are enrolled in a specific code kata battle when the final battle rank becomes available.
- **[SP18]**: CKB notifies students, who are subscribed to a specific tournament when the final tournament rank becomes available.
- **[SP19]**: CKB creates a GitHub repository for a battle.
- **[SP20]**: CKB sends the link for the battle's GitHub repository to all students who are members of enrolled teams.
- **[SP21]**: CKB shows scores for each group to create the competition rank.
- **[SP22]**: CKB shows badge ownership.
- **[SP23]**: CKB calls a static analysis tool for the evaluation of a solution.

## 1.5. Document structure

The remainder of the document is organized as follows. In *Section 2* the overall description will be outlined. This includes a top-level description of the main requirements and product characteristics, as well as the assumptions and constraints on the domain of interest. In *Section 3* we will go over the specific requirements, providing more details on

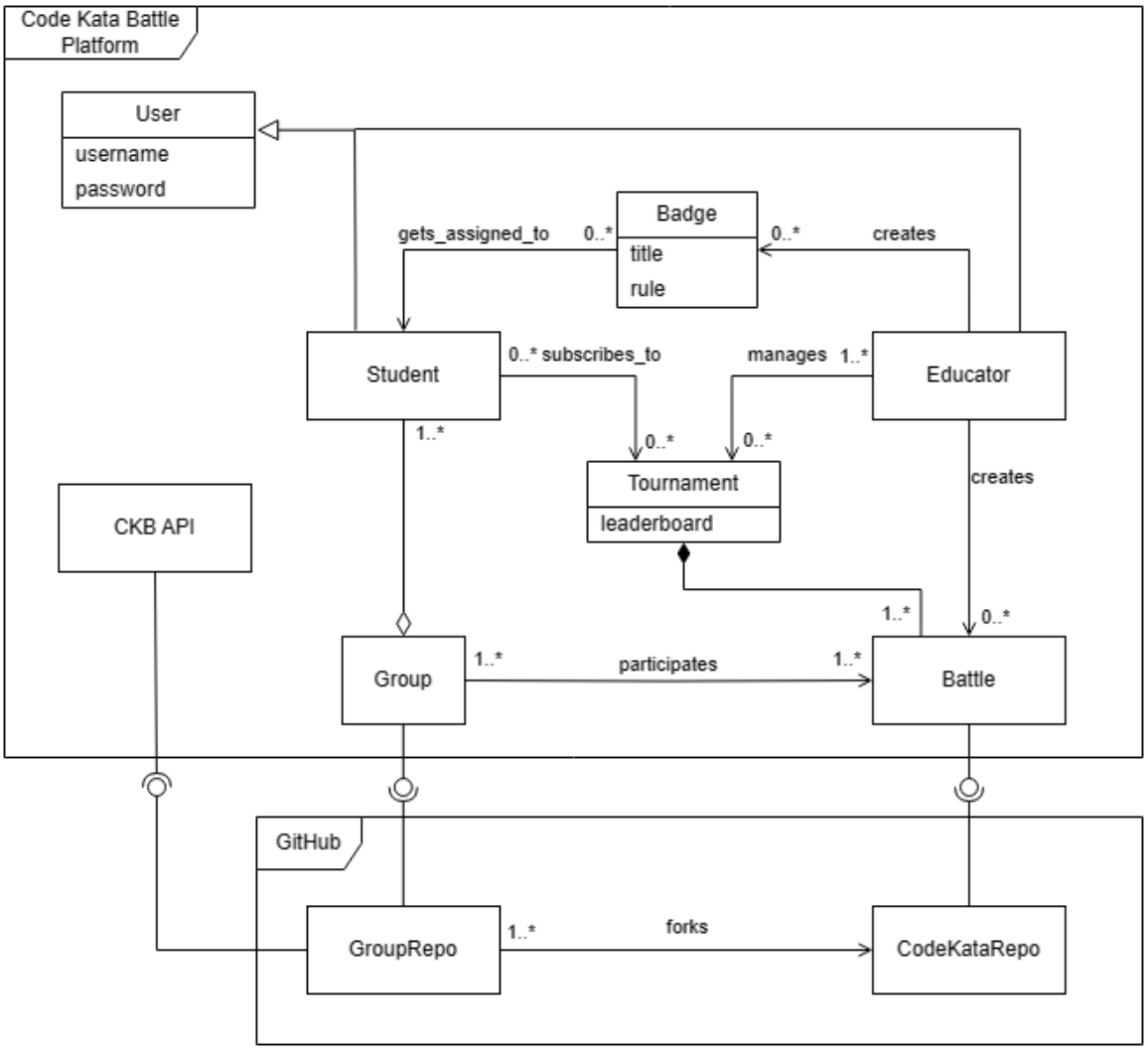
the topics outlined in *Section 2*. This includes both functional and non-functional requirements. Finally, in *Section 4* we will describe a formal model using the Alloy language. This model is used to demonstrate the consistency of the specifications presented in the document.

# 2. Overall Description

In this section are described the main features that the system needs to have to meet the goals. The general characteristics of users are also outlined.

## 2.1. Product perspective

The following diagram represents the main entities of the problem and the relationships between them. The GitHub platform is also shown to specify more explicitly its interaction with CKB's API. In this case, the lollipop notation side with the *cup/hook* indicates the entity that's providing the interface, while the one with the *circle/ball* indicates the entity that's using the interface.



## 2.2. Product functions

The following section will cover the main functions offered by the CKB platform. For clarity reasons, the presented functionalities are grouped based on user class, specifically: common users, educators, and students. It's important to note that when referring to *"common users"*, we encompass student accounts, educator accounts, as well as unauthenticated users, without differentiation.

### 2.2.1 Common user functions

- **Create a new CKB account**: users who want access to the functions offered by the platform must have a registered account. During the account creation process, users need to specify basic information such as name, username, email address, and password, as well as the type of account they want to create: either "Student account" or "Educator account". The type of account to create defines the set of features the user will have access to. Accounts will be bound to the user's email address and username, making it impossible to own multiple accounts with the same email or username. The username, or interchangeably the email, and password make up the user's credentials needed to access the platform when logging in.

- **Logging in a registered CKB account**: users who own a registered CKB account must log in to access the services provided by the platform. The login process requires the user to provide the username (or email) and password that they have used during the account registration process. Providing the correct credentials will result in accessing the user's account and all features related to it.

- **View information about ongoing tournaments and battles**: users who have completed the platform's authentication process via the login system can access information about ongoing tournaments and battles. This information encompasses tournament and battle rankings, students' scores, and badges. Users who are either subscribed to or managing tournaments can also access further details about ongoing battles within those tournaments. For each battle, they can view information such as deadlines, participating groups, group sizes, scores, evaluation results, and specific problem details, including the problem's description and associated tests.

### 2.2.2 Educator functions

- **Creation and management of tournaments**: an educator can create tournaments where they can then organize kata battles. The creation of a tournament is rather simple and only requires the educator to provide a title and student registration deadline. Once the tournament has been created, the educator may decide to invite colleagues (other educator accounts) and permit them to manage the tournament. To be precise, by managing we indicate the following functions:
    - Creation of battles within the tournament;
    - Creation of custom gamification badges for the tournament.
    - Evaluation of students' submissions.

- **Creation and management of battles within tournaments**: an educator with permissions (for a tournament) may decide to create a new kata battle. The creation of a new battle requires educators to provide various pieces of information:
    - Title for the battle (equivalent to the title of the proposed problem);
    - Minimum size for student groups;
    - Maximum size for student groups;
    - Groups registration deadline;
    - Battle deadline (equivalent to students' solutions submission deadline);
    - Problem description;
    - Automation scripts;
    - Tests to run;
    - Evaluation parameters (i.e. allow manual evaluation and run static analysis to assess maintainability and security of the code).

- **Creation and management of gamification badges**: an educator with permission to manage a tournament may decide to define a badge. The creation of a badge requires educators to provide a title and one or more rules that must be fulfilled to achieve the badge. During the creation of a badge, an educator may decide to define new variables to include in the rules.

- **Manual assessment of students' solutions**: when creating a battle, educators have the option to include a manual evaluation at the end of the AA process. This enables educators to individually score each group by manually reviewing their solutions. Upon the completion of the manual assessment, the scores provided by

educators will be consolidated with the scores calculated automatically during the AA process.

### 2.2.3. Student functions

- **Subscription to tournaments**: a student has the option to subscribe to tournaments. By subscribing, they will receive all notifications about that particular tournament. These notifications include updates about newly created battles, changes in battle scores, and updates on rankings, both globally and specific to individual battles. Subscribing to tournaments will also allow students to access and participate in battles for that specific tournament and enable them to be eligible for badges at the end of the tournament.

- **Enrolling in battles in groups (or alone)**: after subscribing to tournaments, students can participate in battles either by forming groups with other students or by enrolling individually. For this document, we will treat students enrolling individually as groups of size 1. To enroll in a battle properly, groups must adhere to the criteria established by educators for that battle, ensuring that their group size neither exceeds the maximum nor falls below the minimum size requirements. Enrollment into battles also has to be done within the deadlines defined by the educators. Once the enrollment deadline for a battle is reached, all enrolled students will receive a link to the GitHub repository for that battle.

## 2.3. User characteristics

In the following section, we will delve into the specific needs of the user classes of interest, namely educators and students.

One primary differentiation between these two classes lies in their use of the platform: educators aim to offer educational content and assess students, whereas students seek to utilize the educational materials provided by educators to learn and improve their coding skills. As it will be stated multiple times throughout the document, it is imperative to note that the platform does not concern itself with quantifying the extent of students' learning or evaluating the quality of the materials provided by educators.

Regarding educators, these users can come from diverse backgrounds, including school teachers, university professors, code camp instructors, influencers (e.g. YouTubers), and any other individuals or groups aiming to produce content aimed at students.

Consequently, different educators may use the platform with different objectives in mind. However, it's crucial to note that the core functions described in Section 2.2 remain consistent for all "types" of educators. This implies that while educators may seek flexibility in their use of the platform, such flexibility can only be attained through the materials they create and offer, rather than through the platform itself.

Similarly, students can also have diverse backgrounds, encompassing school and university students, code camp participants, and other individuals or groups seeking to utilize the platform to improve their skills in coding. This suggests that various subsets within the student user class may have distinct preferences for the type of materials and learning experiences they seek. However, as previously emphasized, the platform itself does not play a direct role in these variations; rather, it's the educators who shape and provide the materials and learning experiences to cater to the diverse needs of the student users (e.g. battles using different languages or aiming at improving in specific topics).

Because the platform is intended for a broad and diverse audience, it's important to consider accessibility issues that may arise, especially considering that users will have to interact with a GUI. For example, in the context of running tests on a solution, users should be able to differentiate between passing and failing test cases. While a common approach is to use the colors green (pass) and red (fail) for this purpose, it's crucial to be mindful of individuals with color blindness who may have difficulty distinguishing between these colors (see example). Similarly, issues can arise with the interpretation of shape language, such as the use of the symbol "✔" (checkmark), which, for instance, in Japanese examinations may be associated with incorrect answers, in contrast to other cultures where it typically signifies correct answers (see "International Differences" section). Therefore, it's essential to adopt inclusive design principles to address these accessibility concerns and ensure that the platform is usable by a wide range of users, regardless of their individual characteristics or backgrounds. The Web Content Accessibility Guidelines (WCAG) serve as a valuable point of reference for implementing accessibility principles in digital platforms. These guidelines provide a comprehensive framework for making web content more accessible to individuals with disabilities and a diverse range of users. Following the WCAG can help ensure that the platform is designed and developed with inclusivity and accessibility in mind, creating a more user-friendly experience for all.

Among others, one concern users may have involves language support. The platform should, at a minimum, provide support for international characters, including languages

that use non-Latin scripts, such as Chinese characters. Ensuring support for such characters is relatively straightforward by using standard international character encodings, like UTF-8. This approach allows the platform to handle a wide range of languages and character sets, making it more accessible and inclusive for users around the world.

## 2.4. User stories

In this section, the main features of the system are described in an informal way from the perspective of the user. For each story, the structure will be the same and it will contain three main pieces of information:

- *who:* the user that is expressing the need. More precisely, we will utilize the *kind* and *class* of the user to identify them.
- *what:* the need that is being expressed. Since the requirements have been stated from an application driven point of view, we will also describe, when needed, *how* the need is met, that is, some information about the implementation.
- *why:* the main problem that has generated the need. In our case, this detail is useful mainly to delineate non functional requirements.

The stories are listed in no particular order.

1. As an **educator**, a teacher wants to **create a tournament** to have their students practice programming in C.
2. As an **educator**, a teacher wants to be able to **personally evaluate** the code that their students submitted to check that it contains proper documentation.
3. As an **educator**, a teacher wants to be able to **create gamification badges** that are assigned automatically to students once a particular requirement is met.
4. The **students** want to be able to **check every user profile** to compare their badges with the ones of other students.
5. As **users**, both educators and students want to be able to **register and log into their accoun**t to create and participate in tournaments.
6. The **students** want to be able to **join a battle** to win points and gain positions in the tournament's leaderboard.
7. The **students** want to be able to **create a group with other students** to collaborate in battles.

8. As **educators**, professors want to **add** tutors as **educators** for help with **managing a tournament** they have created.
9. The **students** want to be **notified when battles end** to know when they can stop working on it and to go see the results.

## 2.5. Assumptions and dependencies

In this section we will outline all the main assumptions that we made during the writing of this document and everything that we will take for granted during the design and implementation phase.

### 2.5.1 Domain assumptions

- **[D1]**: Users are proficient enough in software development and version control. More precisely:
    - **[D1.1]**: Users are able to use GitHub. More specifically they can create, manage, and fork repositories.
    - **[D1.2]** They should also be able to set up actions workflows to connect to CKB's APIs.
    - **[D1.3]:** Users know about RESTful APIs and how to use them.
    - **[D1.4]:** Users know the basics of unit testing.
- **[D2]:** Educators, in particular, are knowledgeable enough to devise tests for software.
- **[D3]:** Users have an email account.
- **[D4]:** Users have a mostly permanent internet connection.
- **[D5]**: All external services work correctly and respond to requests.
    - **[D5.1]**: GitHub works.
    - **[D5.2]**: ECAs work.
    - **[D5.3]**: Email services work.

### 2.5.2 Dependencies

*Code Kata Battle* uses external services to function, relying primarily on GitHub for retrieving code from the users, External Code Analyzers (ECA) for static analysis of the retrieved code, and email services for sending emails. The main functionalities that are required from these three services are:

- **Repository access:** CKB needs to be able to access repositories to fetch code from students and challenges submitted by educators, as well as data about contributions and commits.
- **ECA's API:** CKB communicates with ECAs via APIs to submit code for analysis and retrieve reports.
- **Email sending:** CKB sends emails to users to notify them about events related to battles and tournaments, as well as to verify the accounts during registration.

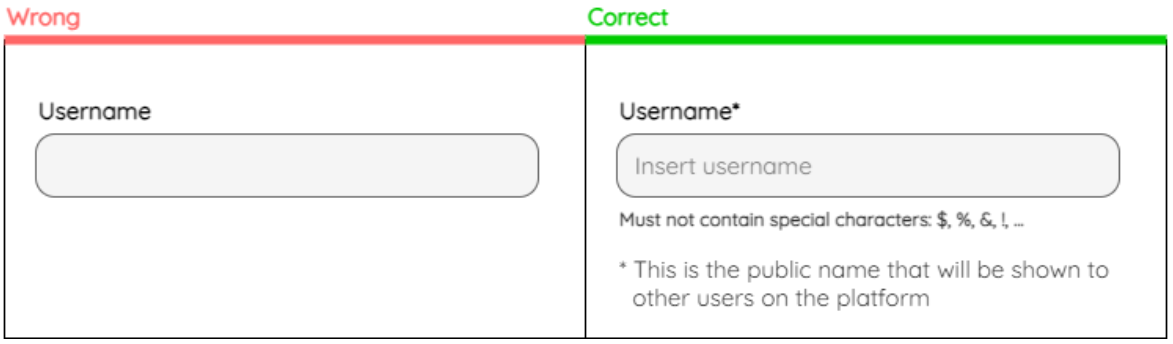
# 3. Specific requirements

## 3.1. External Interface Requirements

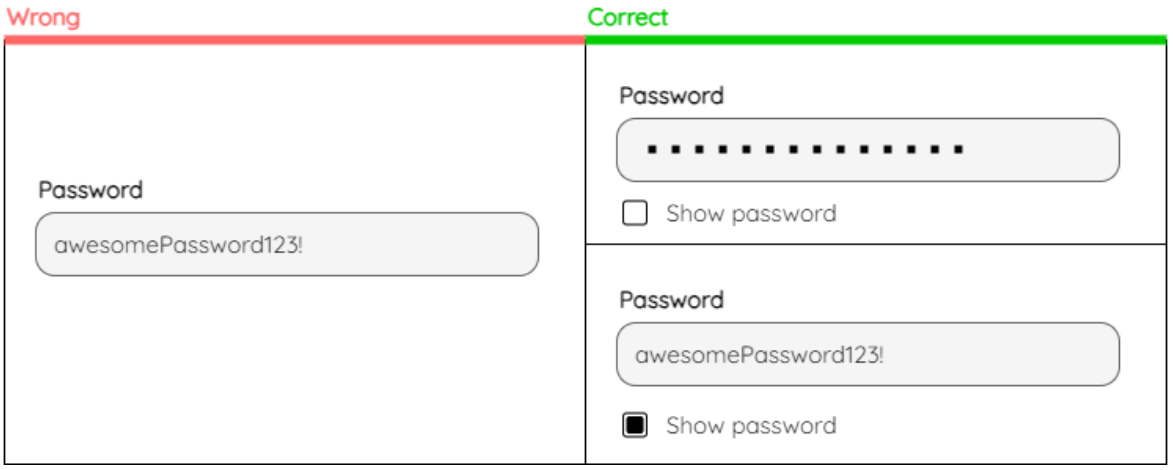
### 3.1.1 User interfaces

The services provided by the platform are available through a GUI. This GUI, in the form of a web page, provides the only user interface to the platform. As previously described in Section 2.3, it is imperative to adopt inclusive design principles such as the ones described in the WCAG. In the following section, we will limit our focus to the most important requirements regarding the GUI.

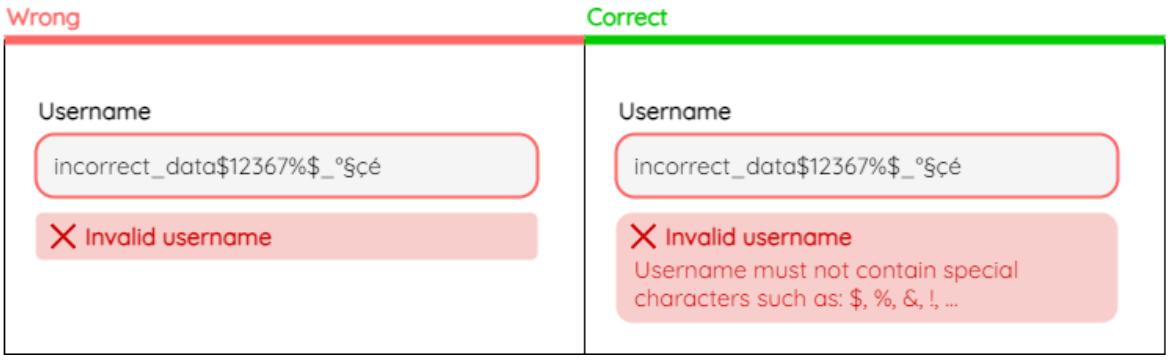
- **Labeled inputs for forms:** input boxes should be labeled or accompanied by a textual description (e.g. tooltip) of the expected input.



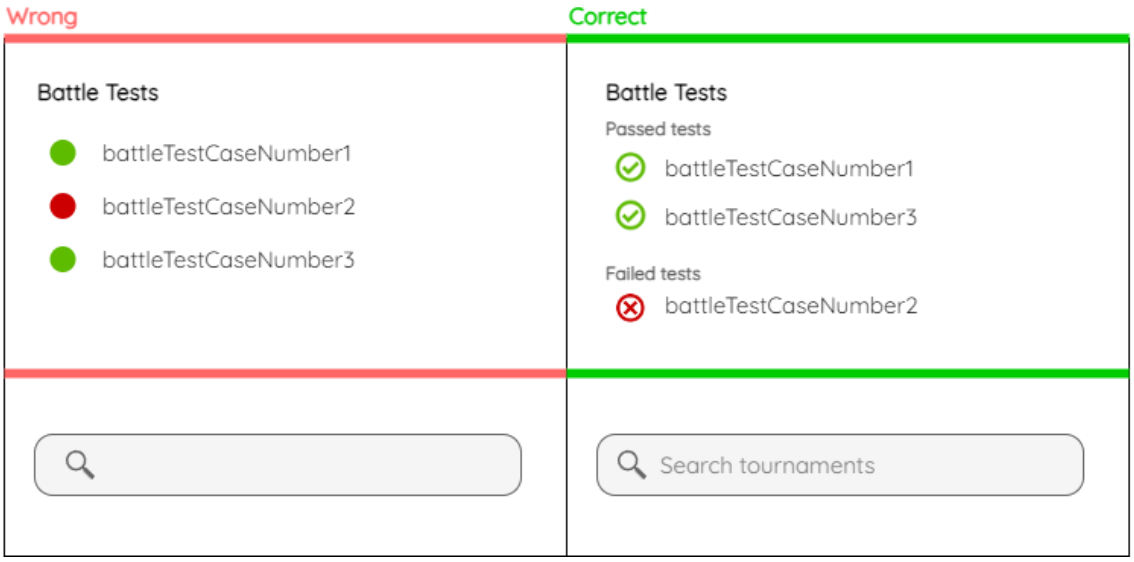

- **Concealment of sensitive inputs:** input boxes that require users to enter sensitive information, like passwords, should obscure the plain text for security reasons. However, for accessibility purposes, secure input boxes should also offer an option for users to reveal the plain text if they wish to do so.




- **Error notifications for inputs (input feedback):** when users input data that the system deems incorrect or invalid, the system should notify the user with an error message that includes a brief explanation of the issue.




- **Visual language accompanied by textual description:** to enhance accessibility, when employing visual elements that could potentially be misinterpreted (such as colors or icons), these elements should be accompanied by a corresponding textual description to ensure clarity and understanding for all users.




- **Responsive design:** the GUI should be responsive to be able to adapt to users' devices whether they use mobile or desktop. Responsiveness of the page is not only important for accommodating different devices but also for accessibility reasons. It enables users, including those with difficulties reading small fonts, to resize text and content for improved readability and usability.
- **International charset support:** as mentioned at the end of Section 2.3, users should be allowed to use international characters to improve inclusivity and accessibility for users around the world.

### 3.1.2 Software interfaces

The platform provides a single API that is intended to be utilized by GitHub to inform when changes have been pushed to the main branch of a repository associated with a battle. This interface operates on an event-driven model: when a push action takes place in a repository, an automatic workflow is triggered, and data is sent to the platform through its interface.

The interface will receive data through the HTTP communication protocol, employing a standard format like JSON. To ensure communication security, the protocol version should be HTTPS and authentication tokens should be included in the requests. The interface shall ignore all requests that lack authentication or that use any other method other than POST. The use of authentication tokens serves a dual purpose: it prevents unauthorized users from accessing the API and safeguards against user impersonation. The data sent to the platform should be in a specific format and at least include information regarding the

origin of the request (i.e. forked repository that triggered the workflow) and the group-specific authentication token.

To prevent stressing the platform, users should also be limited in the number of requests made within a given time frame. The limitation should be imposed with respect to the group-specific token used to make the requests, therefore limiting all members of a group. The rate limit should be defined as no more than 1 request every 10 seconds. The strictness of the restriction is justified considering the nature of the requests: code should not change at a fast pace, and repeated requests may be a sign of improper use on behalf of the students. All the information contained in the discarded requests are not lost as they should be present in the next requests acknowledged by the platform. This is acceptable, since it's not needed that the system is constantly updated in a real-time sense of the state of the repositories.

Regarding response times, the platform should acknowledge the receipt of the request within no more than 1 minute, notifying students of the correctness of the data and the start of the handling process. This timeframe should be achievable, especially considering that the requests are expected to contain a small amount of data (~1KB maximum). This consideration takes into account potential slowdowns during the data transfer process.

In case of any errors within the request (i.e. wrong data format), the platform shall respond with the associated error message detailing the cause of the error and ways to fix it (if any). The response containing the error replaces the one described in the previous paragraph.

Due to the asynchronous nature of events, the API should be capable of managing substantial volumes of concurrent requests originating from different users, or, in this specific scenario, groups of users. This demand is partially mitigated by the implementation of a rate limit and the lightweight nature of the data carried in the requests, as described in previous paragraphs.

## 3.2. Functional Requirements

In the following section are specified all the requirements that the system has to fulfill. In order to work correctly, the system should:

*Please note that whenever the platform sends a notification, it is both on-site and via email.*

- **[R1]:** The system allows the user to sign up.
    - **[R1.1]:** The system does not allow registration if the username is already present in the database.
    - **[R1.2]:** The system does not allow registration if the email is already present in the database.
- **[R2]:** The system allows the registered user to log in.
    - **[R2.1]:** The system does not allow access if at least one of the credentials (username/email or password) is wrong.
- **[R3]** The system allows the student to subscribe to tournaments.
    - **[R3.1]:** The system does not allow non-registered users to subscribe to a tournament.
    - **[R3.2]:** The system should not allow a student to subscribe to a tournament they are already subscribed to.
- **[R4]:** The system allows the student to enroll in a battle.
    - **[R4.1]:** The system should not allow a student to enroll in the same battle by joining different groups.
    - **[R4.2]:** The system allows students to leave the battle before it begins.
    - **[R4.3]:** The system should not allow groups that don't meet battle group size requirements to participate.
    - **[R4.4]:** The system should not allow groups to enroll in battles outside the enrollment deadlines.
- **[R5]:** The system allows the student to invite another student to a battle, creating a group.
    - **[R5.1]:** The system should not allow a student to invite another student to a battle if they are already in a group for that battle.
- **[R6]:** The system allows the educator to create a tournament.
    - **[R6.1]:** Every tournament should have one and only one owner which is the educator who has created it.
    - **[R6.2]:** Every educator who owns a tournament should have permission to manage that tournament.

- **[R7]:** The system allows the educator to create badges within a tournament.
    - **[R7.1]:** The system allows the educator to add a title to the badge.
    - **[R7.2]:** The system allows the educator to add one or more rules to the badge.
    - **[R7.3]:** The system allows the educator to define new variables for the badge.
    - **[R7.4]:** The system allows the educator to associate an icon to the badge.
- **[R8]:** The system allows the educator to create a battle within a tournament.
    - **[R8.1]:** The system allows the educator to upload the code kata.
    - **[R8.2]:** The system allows the educator to set the minimum and maximum number of students per group.
    - **[R8.3]:** The system allows the educator to set a registration deadline.
    - **[R8.4]:** The system allows the educator to set a final submission deadline.
    - **[R8.5]:** The system allows the educator to set additional configurations for scoring.
- **[R9]:** The system allows the educator to grant permissions to another educator.
    - **[R9.1]:** The system allows the educator to grant permissions to another educator to create battles.
    - **[R9.2]:** The system allows the educator to grant permissions to another educator to create badges.
    - **[R9.3]:** The system allows the educator to grant permissions to another educator to give manual evaluations.
- **[R10]:** The system allows the educator to make a manual assessment of the solution provided by the groups if specified in the scoring configurations.
- **[R11]:** The system must create the GitHub repository containing the code kata for the battle.
    - **[R11.1]:** The system must send the GitHub link to all students who are members of subscribed teams.
    - **[R11.2]:** The system must make the GitHub repository link visible on the battle page.
    - **[R11.3]:** The system must provide an authorization token for the group to make API calls.
- **[R12]:** The system must run tests and give an evaluation to the provided solutions.
    - **[R12.1]:** The system must receive messages, through the CKB platform's API, with each new commit made by the students in the main branch.

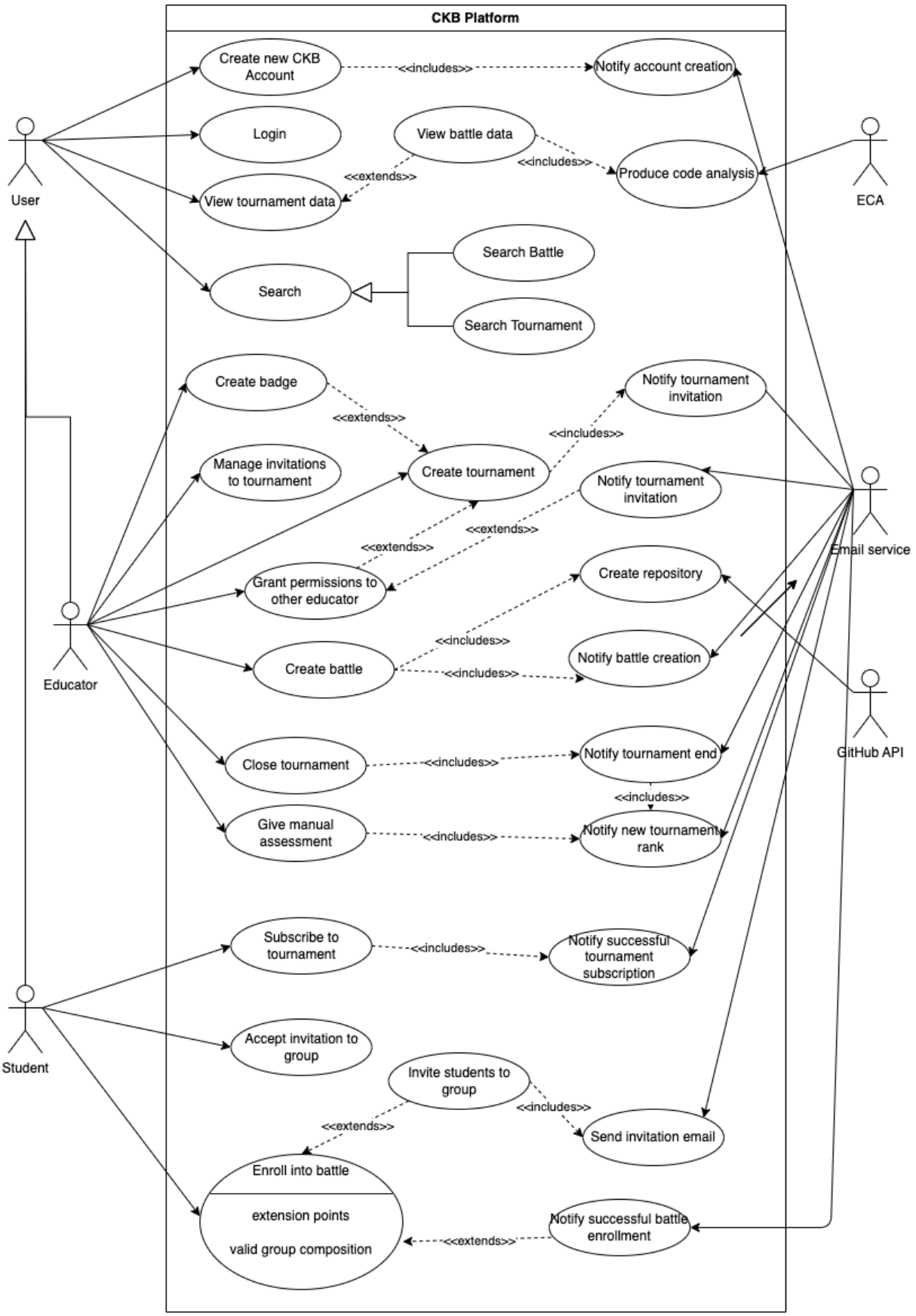
    - **[R12.2]:** When tests have been conducted, the system should show the outcomes of tests to users.
    - **[R12.3]:** The system must analyze the timeliness of the student's solution, measured in terms of time passed between the registration deadline and the last commit.
    - **[R12.4]:** The system must analyze the code by calling ECAs.
    - **[R12.5]:** The system must calculate and update the battle score of the team.
    - **[R12.6]:** The system should not consider solutions outside the submission deadlines.
- **[R13]:** The system allows the user to see the current rank evolving during the battle.
- **[R14]:** The system must update the personal tournament score of each student, that is the sum of all battle scores received in that tournament, at the end of each battle.
- **[R15]:** The system allows the educator who created the tournament to close it.
    - **[R15.1]:** The system must notify all the subscribed students of the closing tournament.
    - **[R15.2]:** The system must notify all the subscribed students when the final battle rank becomes available.
    - **[R15.3]:** The system must notify the students in the tournament who have completed the process to obtain a badge of their achievement.
    - **[R15.4]:** The system should not allow educators who aren't hosts of a tournament to close it.
    - **[R15.5]:** Once a tournament has been closed, it cannot be reopened.
- **[R16]:** The system should assign a badge to one or more students at the end of the tournament if the students have fulfilled the badge's requirements to achieve it.
    - **[R16.1]:** Badges should not be assigned until the tournament is over.
    - **[R16.2]:** Once a badge has been achieved by a student, the student cannot lose that badge.
- **[R17]:** The system allows the user to view another student's badges.
- **[R18]:** The system should automatically close battles after their submission deadline is reached.
    - **[R18.1]:** Once the submission deadline for a battle has been reached, the battle cannot be reopened.
- **[R19]:** The system allows users to search by battle and by tournament.
    - **[R19.1]:** The system allows users to search for specific battles.

    - **[R19.2]:** The system allows users to search for specific tournaments.
- **[R20]:** The system allows invited users to either accept or reject.
    - **[R20.1]:** The system allows educators who have been invited as tournament managers to accept the invite.
    - **[R20.2]:** The system allows educators who have been invited as tournament managers to reject the invite.
    - **[R20.3]:** The system allows students who have been invited in groups for a battle to accept the invite.
    - **[R20.4]:** The system allows students who have been invited in groups for a battle to reject the invite.
    - **[R20.5]:** The system automatically rejects all pending invitations for a battle if the student joins a group.
- **[R21]:** If configured, when the battle ends the system must notify the educators of the tournament that a manual evaluation is required to consolidate scores.

### 3.2.1 Use cases

In the following section are provided the use case diagrams briefly outlined in Section 2.4. This section aims to show the interaction between the actors involved in the use of the system.

User case diagram:


CKB Platform
Create new CKB Account
<<includes>>
Notify account creation
Login
View battle data
<<includes>>
Produce code analysis
<<extends>>
View tournament data
User
ECA
Search
Search Battle
Search Tournament
Create badge
<<extends>>
Notify tournament invitation
<<includes>>
Manage invitations to tournament
Create tournament
Notify tournament invitation
<<extends>>
<<extends>>
Email service
Grant permissions to other educator
Create repository
Educator
<<includes>>
Create battle
Notify battle creation
<<includes>>
GitHub API
Close tournament
<<includes>>
Notify tournament end
<<includes>>
Give manual assessment
<<includes>>
Notify new tournament rank
Subscribe to tournament
<<includes>>
Notify successful tournament subscription
Student
Accept invitation to group
Invite students to group
<<includes>>
<<extends>>
Send invitation email
Enroll into battle
extension points
valid group composition
<<extends>>
Notify successful battle enrollment

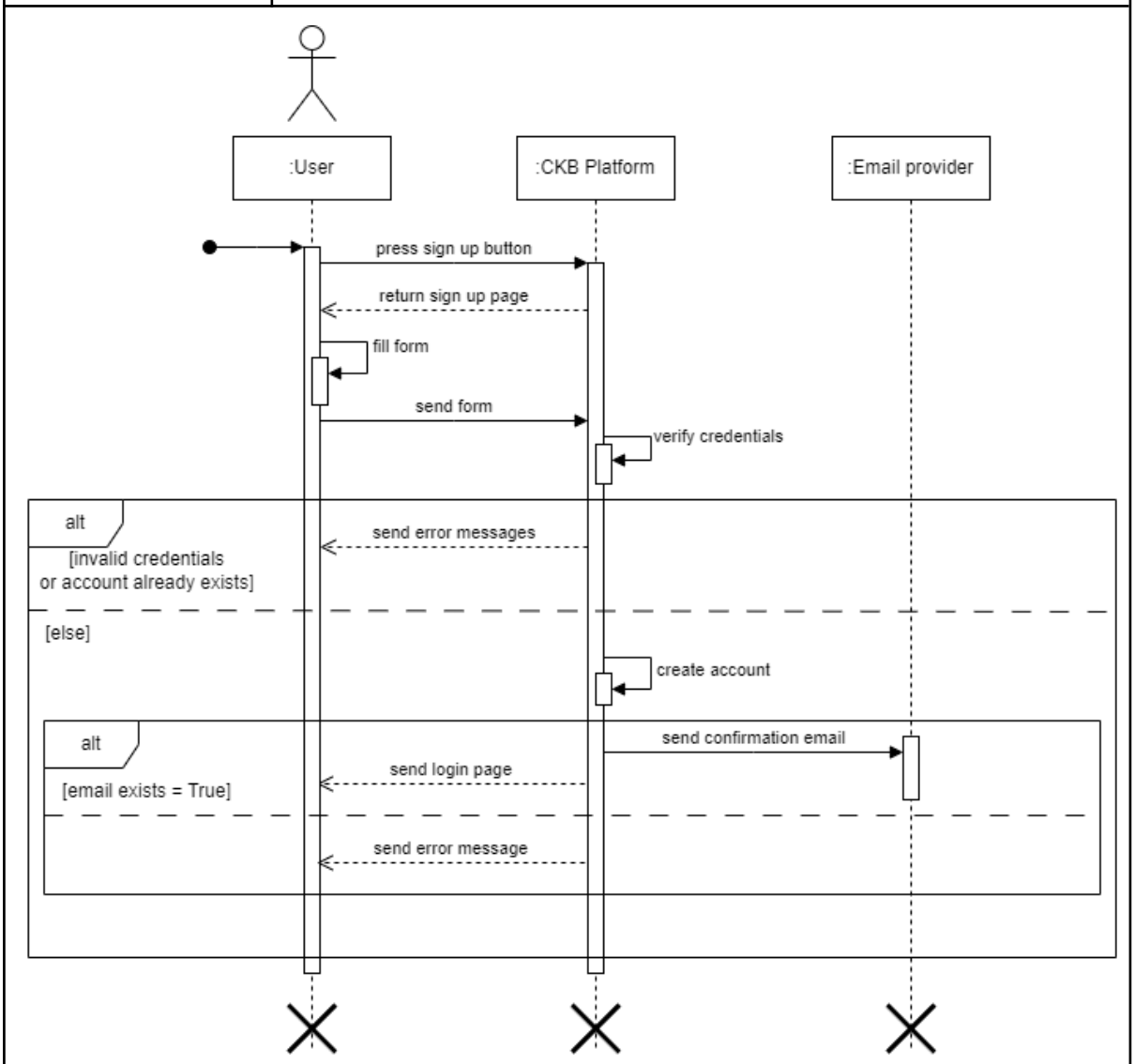
Below are the tables that represent the use cases and, for each of them, we report the corresponding sequence diagram immediately below.

Note, the use cases relating to "view score of students", "view badges" and "view battle ranking" are very similar to view battle data, therefore we don't report the use case table. They differ only in the type of data they load, and in the specific page the data is shown on.

**[UC1]: Creation of new CKB account**

| | |
|---|---|
| **Name** | Creation of new CKB account. |
| **Actors** | User, Email service. |
| **Entry Condition** | The user has opened the browser and is on the homepage of the Platform. |
| **Event flow** | 1. The user clicks on the button "Sign Up".<br>2. The Platform shows the registration form.<br>3. The user inserts the following data:<br>a. name;<br>b. username;<br>c. email address;<br>d. password.<br>4. The user clicks on the specific radio button corresponding to which type of account they want to create:" Student account" or "Educator account".<br>5. The user clicks on the button "Create account" to send the form.<br>6. The system verifies that the username and email are unique.<br>7. The system creates the account.<br>8. The system sends a confirmation email to the user through an email service.<br>9. The system redirects to the login page. |
| **Additional steps** | 10. The user confirms their email address by clicking on the link received in the confirmation email. |
| **Exit condition** | The user accesses the login page. The new account is registered and stored on the Platform. |
| **Exception** | 3. Data is missing or not valid. An error label is shown under |

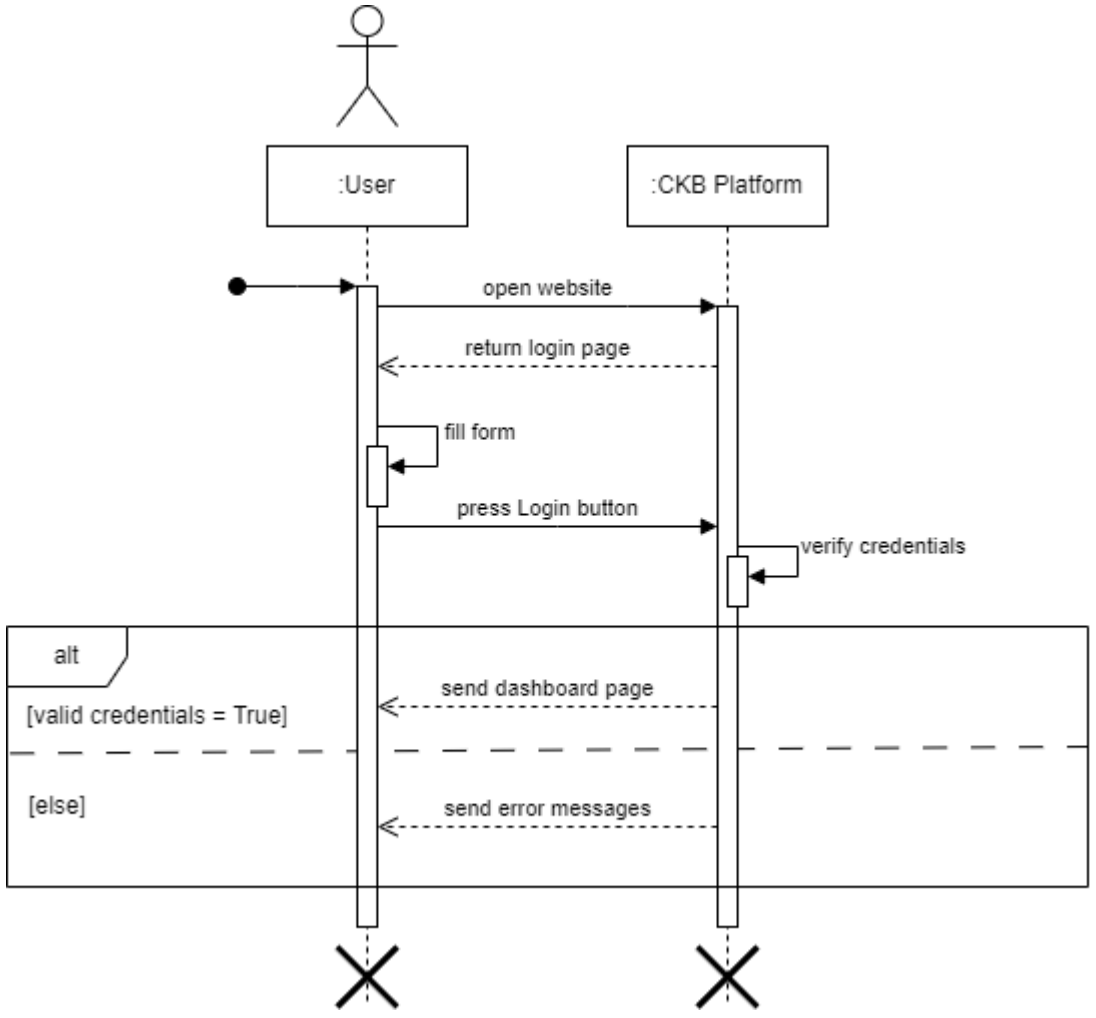
| | |
|---|---|
| | the field.<br>6. A user already exists with the same username or email. An error message is shown.<br>8. The provided email address does not exist. |




**[UC2]: Login**

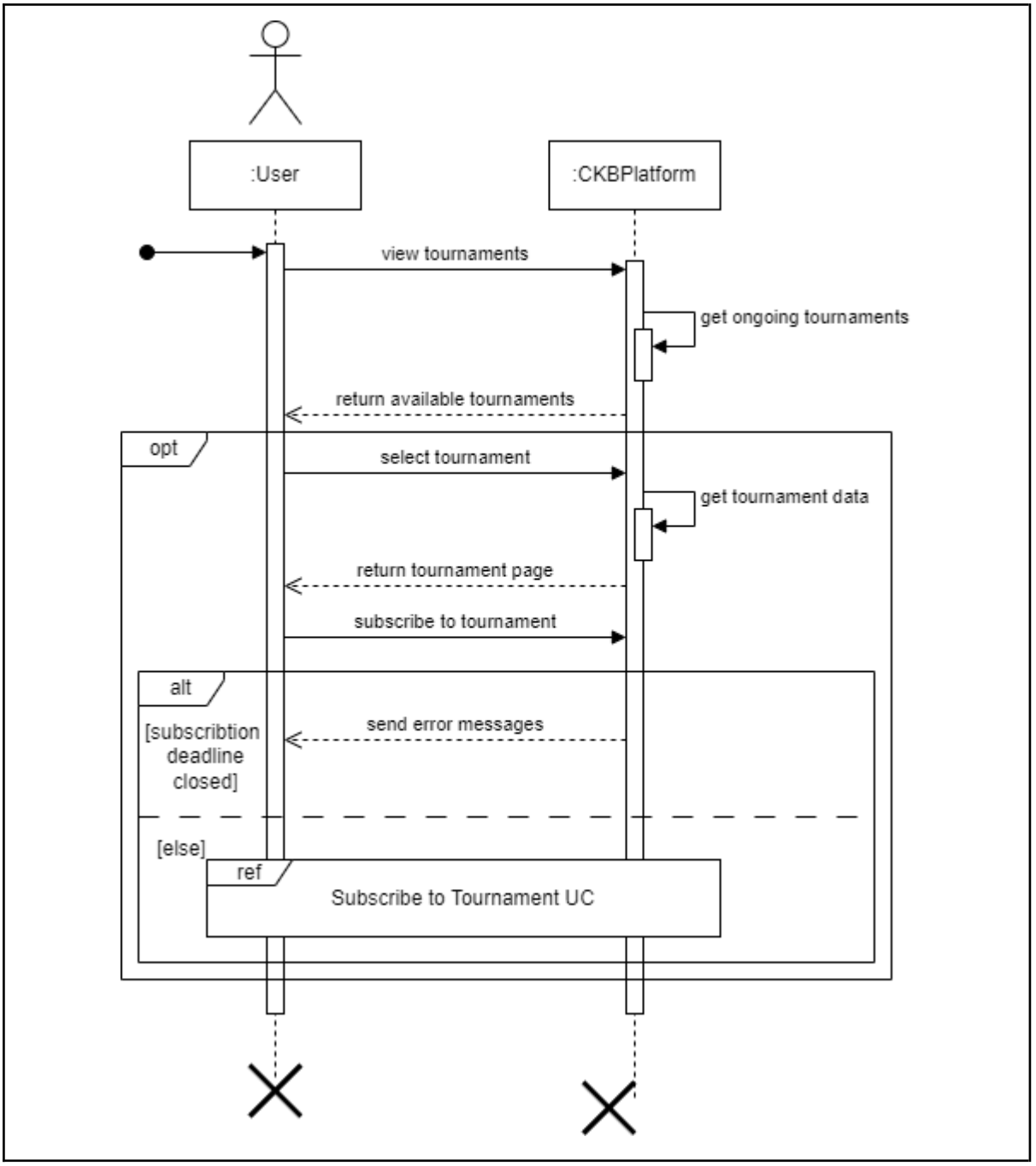
| | |
|---|---|
| **Name** | Login. |
| **Actors** | User. |
| **Entry Condition** | The user has opened the browser and they are on the login page. |
| **Event flow** | 1. The user inserts their username (or email) and password in the form. |

| | |
|---|---|
| | 2. The user clicks on the "Login" button.<br>3. The system checks whether the credentials are correct.<br>4. The Platform shows the user's dashboard. |
| **Exit condition** | The user is authenticated and is able to access their dashboard. |
| **Exception** | 1. Required fields not filled in.<br>3. The data inserted is not valid. The system shows an error message.<br>3. If the user gets the credentials wrong five times in a row, an error message is shown and the login is blocked for 30 minutes. |



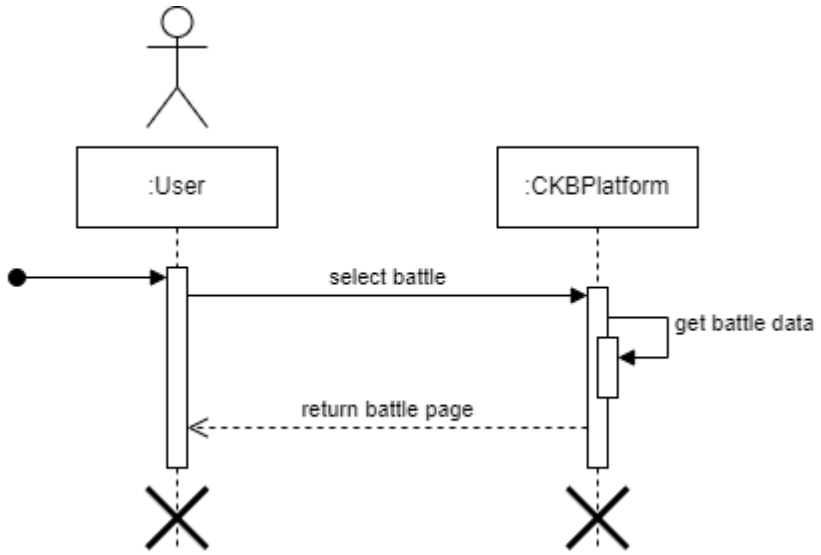
**[UC3]: View tournament data**

| | |
|---|---|
| **Name** | View tournament data. |
| **Actors** | User. |
| **Entry Condition** | The user has logged in to the system and is on the dashboard page. |
| **Event flow** | 1. The user clicks on the button "View tournaments".<br>2. The Platform loads the ongoing tournaments.<br>3. The Platform shows the list of tournaments. |
| **Additional steps** | 4. The user clicks on a specific tournament.<br>5. The user sees the list of the battles that lie within the tournament.<br>6. The user sees the list of badges that are connected to that tournament.<br>7. The user sees the scores of the students.<br>8. The user clicks on the button "Subscribe to Tournament". |
| **Alternative flow** | 8. The tournament's subscription deadline has expired. |
| **Exit condition** | The user can view the tournament data and proceed with further actions. |




**[UC4]: View battle data**

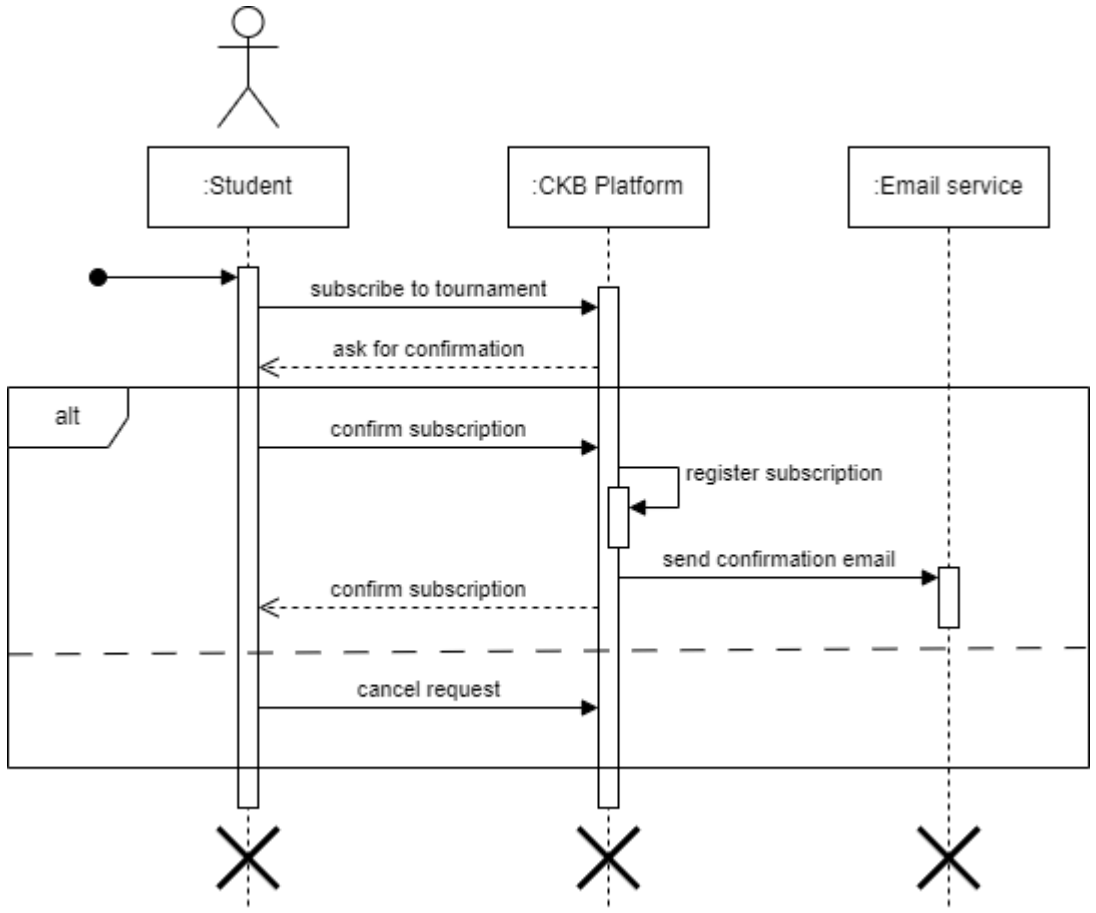
| | |
|---|---|
| **Name** | View battle data. |
| **Actors** | User. |
| **Entry Condition** | The user has opened the browser and they are on a tournament page. |

| | |
|---|---|
| **Event flow** | 1. The user selects and clicks on a specific battle.<br>2. The Platform loads the battle's data:<br>a. title;<br>b. description;<br>c. tests.<br>3. The platform sends battle data to the user. |
| **Exit condition** | The user is able to view the details of the battle. |
|  | |

**[UC5]: Subscribe to tournament**

| | |
|---|---|
| **Name** | Subscribe to tournament. |
| **Actors** | Student, Email service. |
| **Entry Condition** | The student clicks on the button "Subscribe to Tournament" in the tournament page and the student is logged in. |
| **Event flow** | 1. The student sees a pop-up message asking for confirmation that they want to subscribe to the tournament.<br>2. If the student says yes, the system registers them to the tournament.<br>3. The system sends a confirmation email to the student notifying them of the successful subscription to the tournament. |

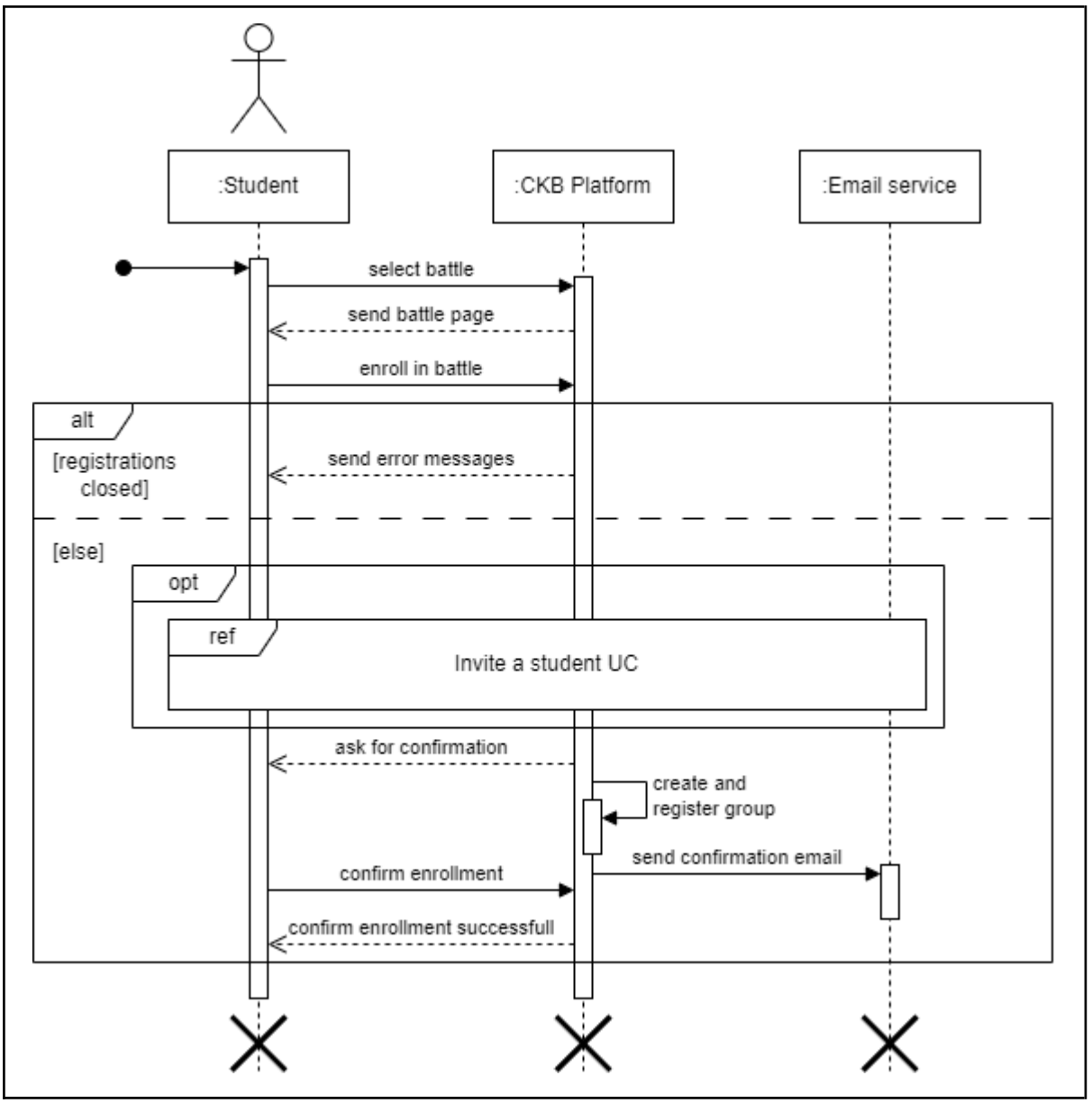
| | |
|---|---|
| **Alternative flow** | 2. If the student says no, then the student returns to the tournament page without being subscribed. |
| **Exit condition** | The system successfully registers the student for the tournament and sends them the confirmation email. |




**[UC6]: Enroll into battle**

| | |
|---|---|
| **Name** | Enroll into battle. |
| **Actors** | Student, Email Provider. |
| **Entry Condition** | The student is logged in and they are on a specific tournament page where they can see the list of the battles. Alternatively, the student clicks on a link received in a mail notifying a new battle has been created. |
| **Event flow** | 1. The student clicks on a specific battle.<br>2. The student clicks on the button "Enroll in Battle". |

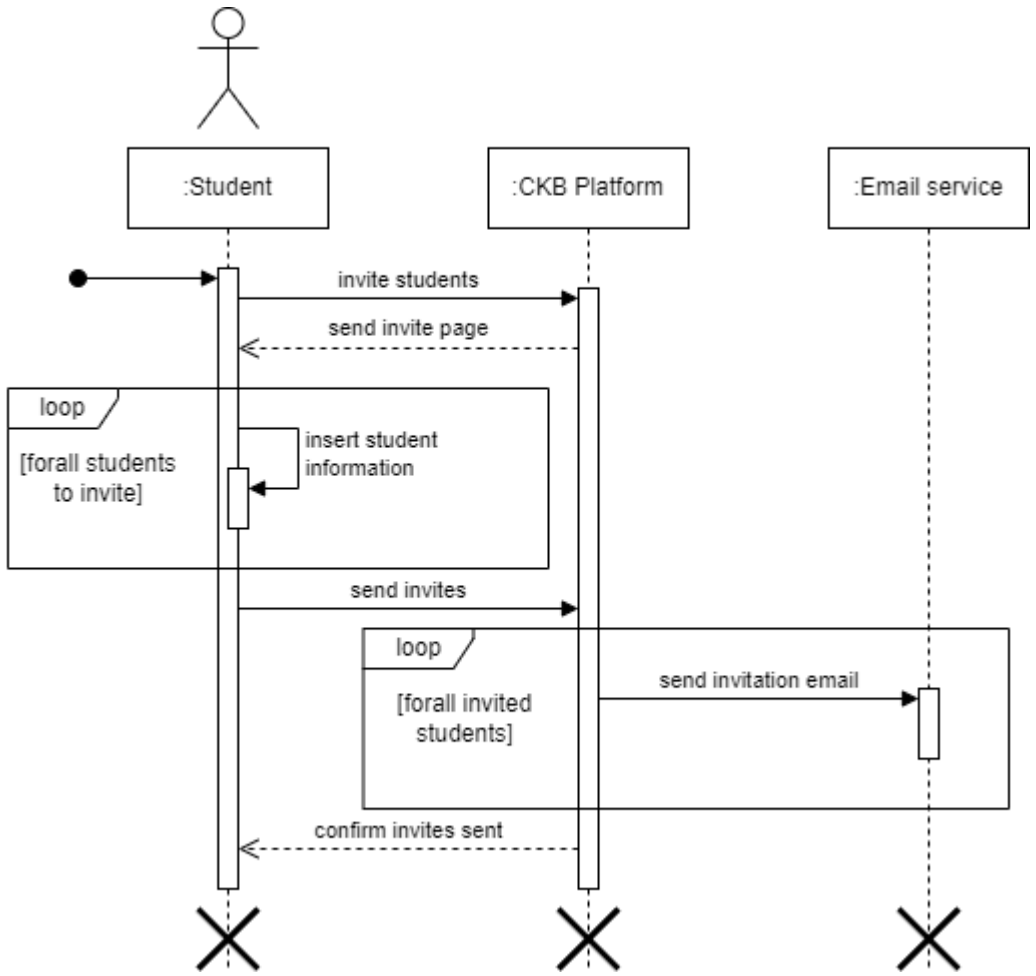
| | |
|---|---|
| | 3. The student can invite another student by adding the username or email in the invite form on the battle enrollment page.<br>4. The student sees a pop-up message asking for confirmation that the student wants to enroll in the battle.<br>5. If the student says yes, then the system creates a new group and enrolls it into the battle.<br>6. The system sends a confirmation email to the student notifying the successful creation of a group for the battle.<br>7. If the student decides to invite others, the system sends an invite email to all invited students. |
| **Alternative flow** | 1. The student clicks on the button "Enroll in the battle" that was in the email sent by Platform.<br>2. The Platform sends the student to the battle page.<br>5. If the student says no, then the student returns to the tournament page without being enrolled. |
| **Exit condition** | The system successfully registers the student's group for the battle and they receive a confirmation email. |
| **Exception** | 2. The student attempts to register for a battle for which registration is closed.<br>5. The group has reached the size limits imposed by the battle. |




**[UC7]: Invite students to group**

| | |
|---|---|
| **Name** | Invite a students to group |
| **Actors** | Student, Email Provider. |
| **Entry Condition** | The user has logged in to the system and they are in a battle page for which they have a group. The battle enrollment deadlines are still open. |
| **Event flow** | 1. The student clicks on the button “Invite students”.<br>2. The system shows a tab where to insert data for the students to invite. |

| | |
|---|---|
| | 3. The student enters the username or email of the students they want to invite.<br>4. The user presses the button “Send invites”.<br>5. The Platform sends an invitation email to all invited students. |
| **Exit condition** | The invitation email is successfully sent to all interested students. |
| **Exception** | 3. The group has reached the size limits imposed by the battle, the input of new students is blocked. |



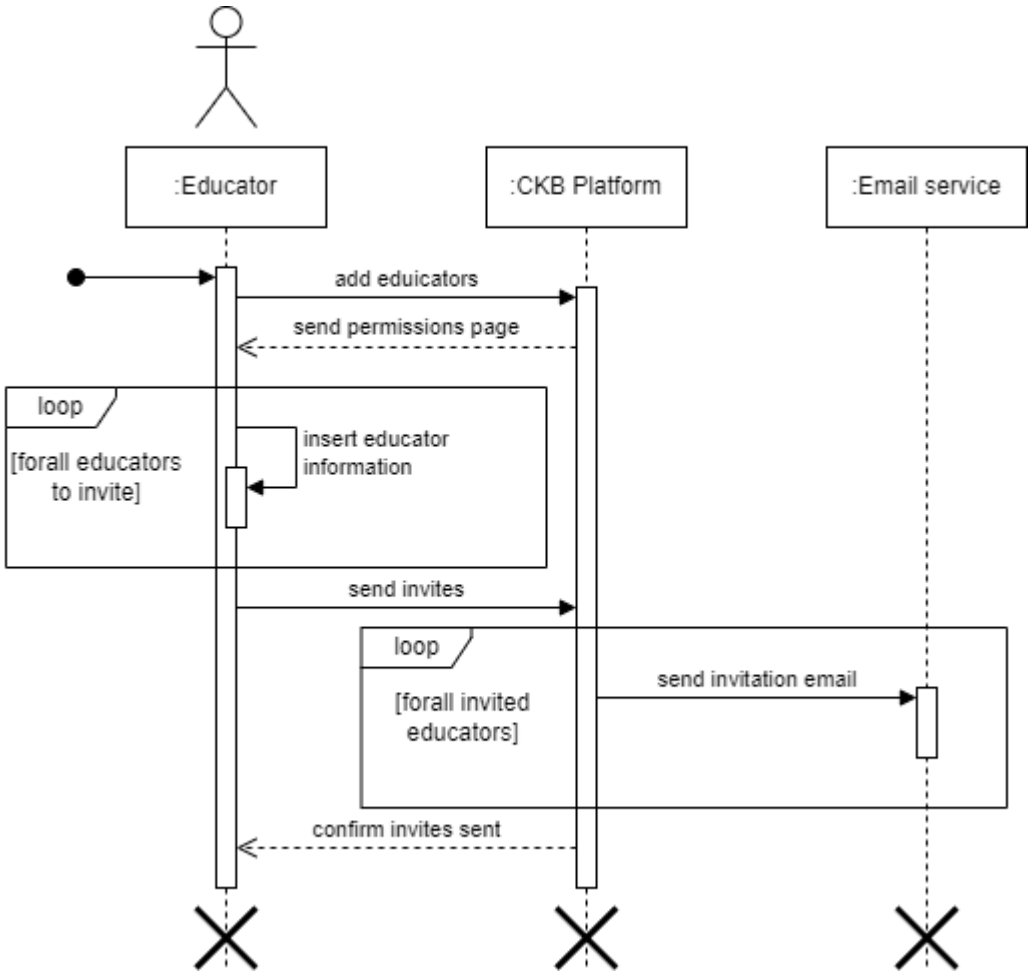

**[UC8]: Creation of badges**

| | |
|---|---|
| **Name** | Creation of badges. |
| **Actors** | Educator. |
| **Entry Condition** | The educator is logged in and they are on the tournament page. |

| | |
|---|---|
| **Event flow** | 1. The educator clicks on the button "Create new badges"<br>2. The Platform shows a form in which to add the badge's information.<br>3. The educator fills out the form by entering:<br>  a. title;<br>  b. one or more rules, creating the variables they need (within the limits imposed by the platform);<br>  c. number of students who can receive the badge;<br>  d. icon.<br>4. The educator clicks on the "Create badge"<br>5. The platform verifies the information of the badge.<br>6. The Platform creates the badge and adds it tot the tournament.<br>7. The Platform redirects the educator to the tournament's creation form. |
| **Alternative flow** | 4. The educator cancels the badge creation by going back to the tournament creation form.<br>6. The platform notifies the presence of errors in the structure of the badge. |
| **Exit condition** | The badge is created and added to the tournament, and the educator returns to the tournament creation form |
| **Exception** | 3. The educator tries to confirm the creation of the badge without having filled in all the mandatory fields.<br>5. A badge with the same structure or name already exists. |

**[UC9]: Grant permissions to other educators**

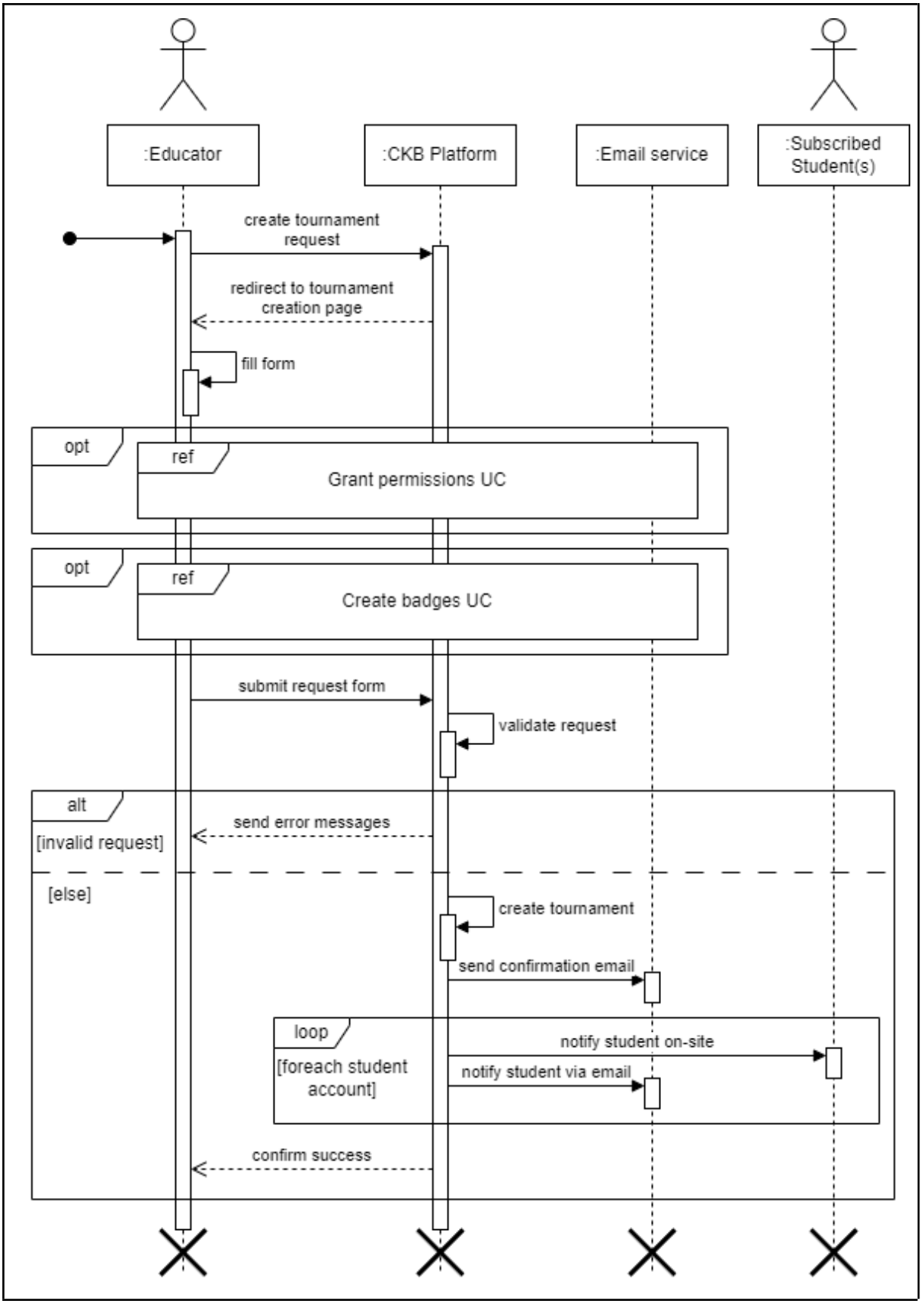
| | |
|---|---|
| **Name** | Grant permissions to other educators. |
| **Actors** | Educator, Email service. |
| **Entry Condition** | The educator is logged in and they are on the page of a tournament that they own. |
| **Event flow** | 1. The educator clicks on the button "Add educators".<br>2. The Platform redirects the educator to the permissions page.<br>3. The educator enters the username or email of the educators |

| | |
|---|---|
| | to whom they give permissions.<br>4. The educator submits the invitations.<br>5. The platform sends an email to all the invited educators notifying the invite to manage a tournament.<br>6. The Platform shows a "success" pop-up message. |
| **Exit condition** | The invited educators received the invitation email and can become managers of the tournament. |
|  | |

**[UC10]: Creation of tournaments**

| | |
|---|---|
| **Name** | Creation of tournaments. |
| **Actors** | Educator. |
| **Entry Condition** | The educator is logged into the system and they are on the homepage for the educator. |

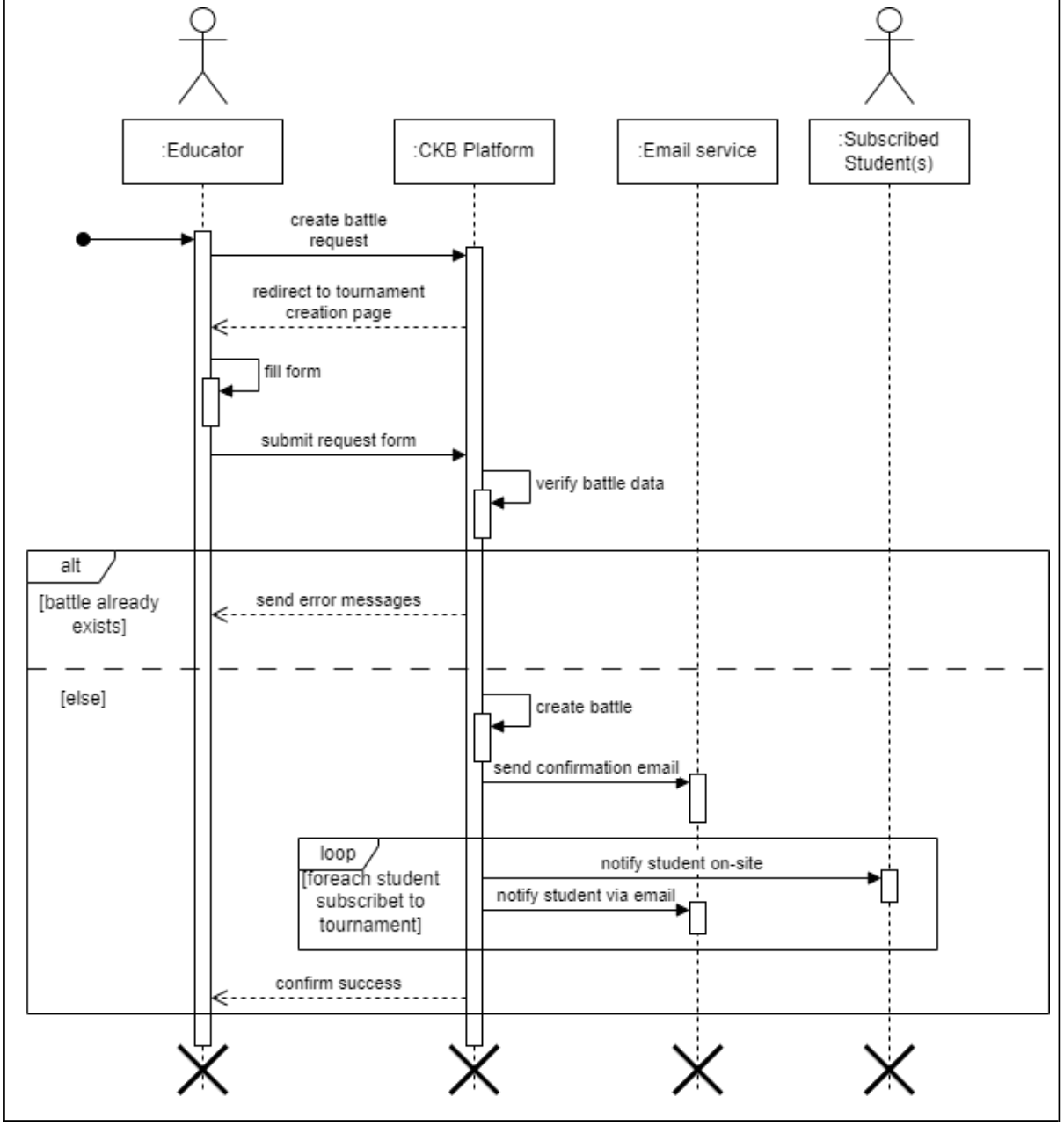
| | |
|---|---|
| **Event flow** | 1. The educator clicks on the button "Create a tournament".<br>2. The system redirects the educator to the page of the tournament's creation.<br>3. The educator fills out the form by entering:<br>  a. title;<br>  b. student registration deadline.<br>4. Here the educator can grant permissions to other educators to manage the tournament.<br>5. The educator can create new badges.<br>6. The educator clicks on the "Create tournament" button, submitting the form.<br>7. The Platform checks if there is no other tournament with the same title.<br>8. if there isn't an equal tournament, then the system creates the tournament.<br>9. The Platform sends an email to all students subscribed to the platform informing them that a new tournament is available.<br>10. The Platform sends a confirmation email to the educator notifying the successful creation of the tournament.<br>11. The Platform shows a "success" pop-up message.<br>12. The Platform redirects the educator into the new tournament page. |
| **Alternative flow** | 8. If not, the system shows an error pop-up message.<br>9. The Platform redirects the educator to the page of the tournament's creation. |
| **Exit condition** | The tournament is created and both the educator and the students subscribed to the platform are notified. |
| **Exception** | 7. The system detects an already existing tournament with the same title. |


:Educator
:CKB Platform
:Email service
:Subscribed Student(s)
create tournament request
redirect to tournament creation page
fill form
opt
ref
Grant permissions UC
opt
ref
Create badges UC
submit request form
validate request
alt
[invalid request]
send error messages
[else]
create tournament
send confirmation email
loop
[foreach student account]
notify student on-site
notify student via email
confirm success

**[UC11]: Creation of battles**

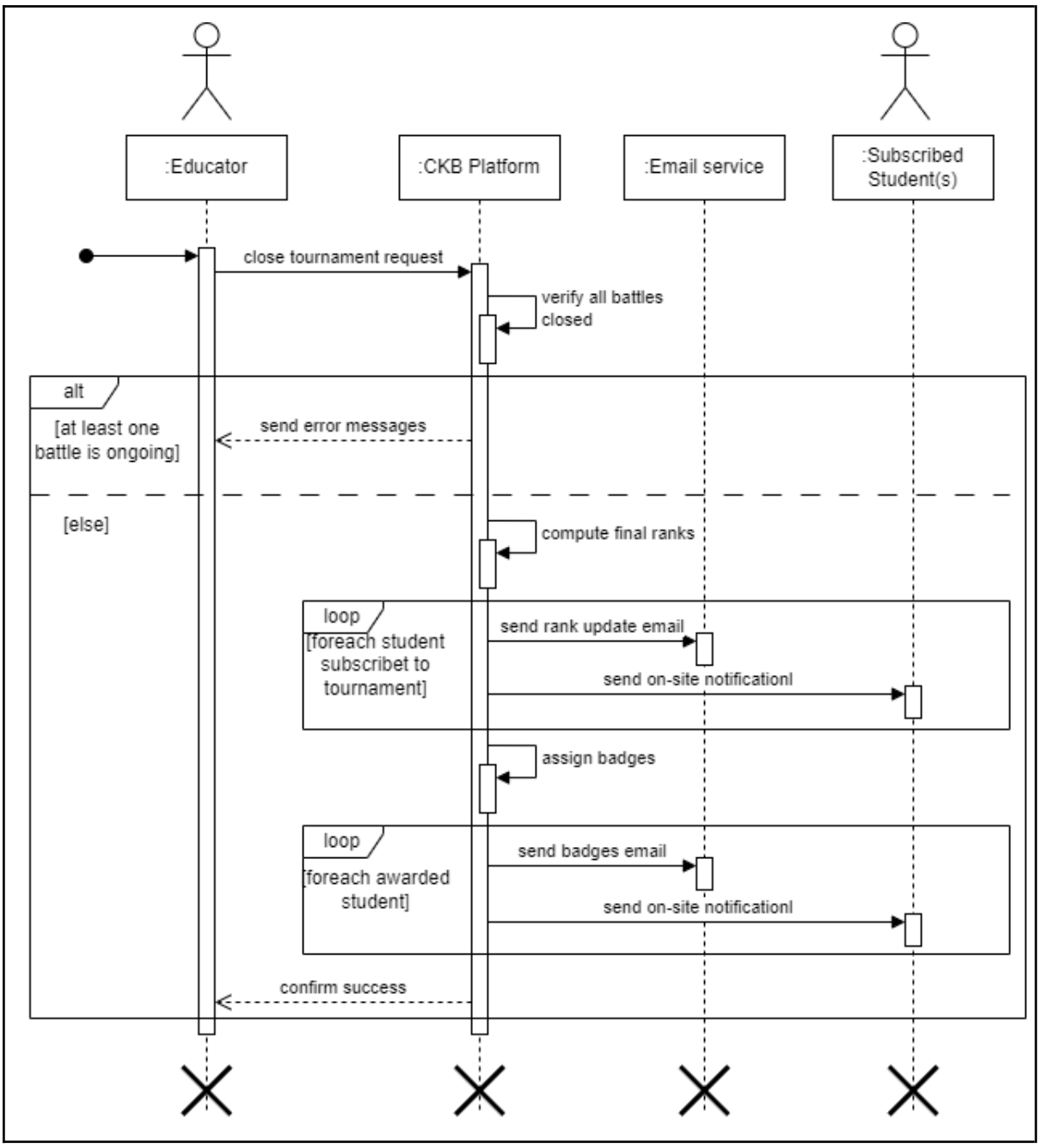
| | |
|---|---|
| **Name** | Creation of battles. |
| **Actors** | Educator, Email service. |
| **Entry Condition** | The educator is logged into the system and they are in a specific tournament page. |
| **Event flow** | 1. The educator clicks on the button "Create a battle".<br>2. The system redirects the educator to the page of the battle's creation.<br>3. The educator fills out the form by entering:<br>a. Title for the battle (equivalent to the title of the proposed problem);<br>b. Minimum size for student groups;<br>c. Maximum size for student groups;<br>d. Groups registration deadline;<br>e. Battle deadline (equivalent to students' solutions submission deadline);<br>f. Problem description;<br>g. Automation scripts;<br>h. Tests to run;<br>i. Evaluation parameters (i.e. allow manual evaluation and run static analysis to assess maintainability and security of the code).<br>4. The educator clicks on the "Create battle" button, submitting the form.<br>5. The Platform checks if there isn't an equal battle within the current tournament.<br>6. If there isn't an equal battle, then the system creates the battle.<br>7. The Platform sends an email to all students subscribed to the tournament in which it says that a new battle is available.<br>8. The Platform sends a confirmation email to the educator notifying the successful creation of the battle.<br>9. The Platform shows a "success" pop-up message.<br>10. The Platform redirects the educator into the new battle |

| | |
|---|---|
| | page. |
| **Alternative flow** | 6. If there is already a battle with the same title, the system shows an error pop-up message.<br>7. The Platform redirects the educator to the page of the battle's creation. |
| **Exit condition** | The battle is created and the educator and all students subscribed to the tournament are notified. |



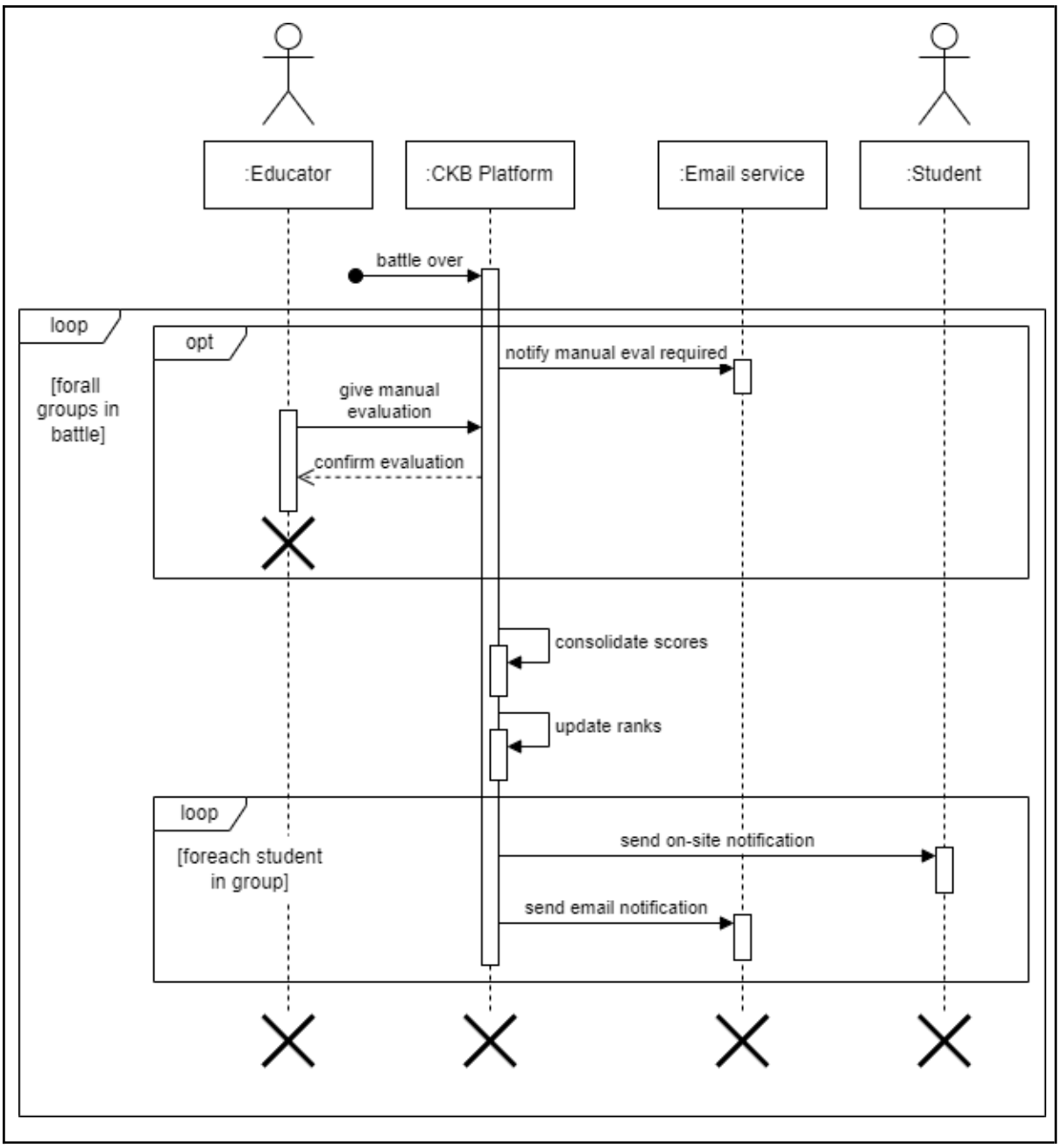
**[UC12]: Closing of a tournament**

| | |
|---|---|
| **Name** | Closing of a tournament. |
| **Actors** | Educator. |
| **Entry Condition** | The educator is on the tournament page and all the battles are over. |
| **Event flow** | 1. The educator clicks on the button "Close the tournament."<br>2. The Platform checks if all the battles are over.<br>3. The Platform computes the final tournament ranking for students.<br>4. The Platform awards badges to students who have fulfilled their requirements.<br>5. The platform notifies all students in the tournament of the updates to the ranks and badges.<br>6. The Platform closes the tournament. |
| **Exit condition** | The tournament is successfully closed. |
| **Exception** | 2. There is still an ongoing battle. |


:Educator
:CKB Platform
:Email service
:Subscribed Student(s)
close tournament request
verify all battles closed
alt
[at least one battle is ongoing]
send error messages
[else]
compute final ranks
loop
[foreach student subscribet to tournament]
send rank update email
send on-site notificationl
assign badges
loop
[foreach awarded student]
send badges email
send on-site notificationl
confirm success

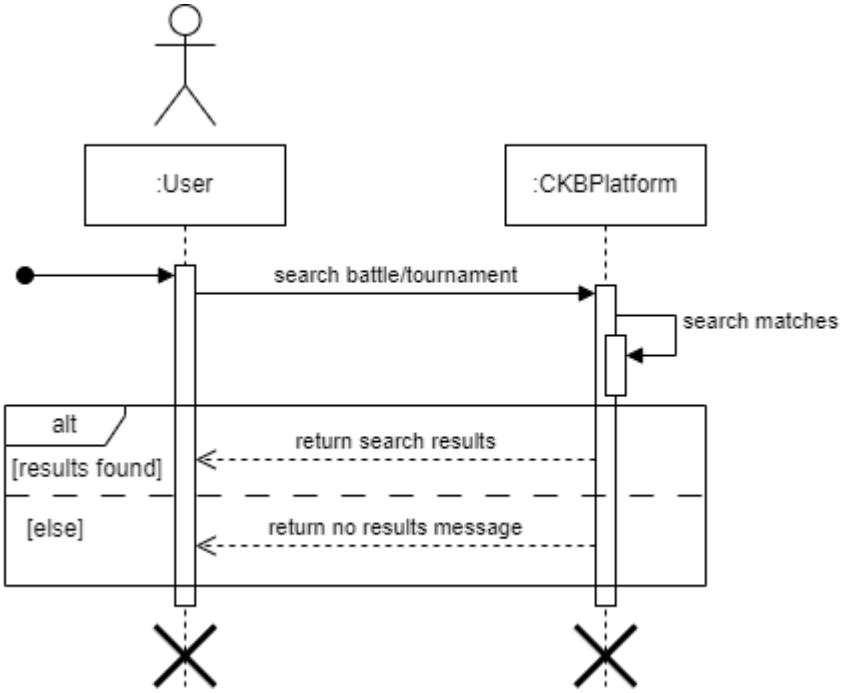
**[UC13]: Give manual assessment**

| | |
|---|---|
| **Name** | Give manual assessment. |
| **Actors** | Educator, Email service. |
| **Entry Condition** | The educator is logged in and the submission deadline for the battle has expired, also the battle requires a manual evaluation. |
| **Event flow** | 1. The educator is on the battle page.<br>2. The Platform loads the list of the groups that have released a solution.<br>3. The educator clicks on a specific group.<br>4. The educator clicks on the button "Give manual assessment".<br>5. The platform redirects the educator to the manual assessment page.<br>6. The educator releases their manual reviewing entering the score for the group.<br>7. The educator submits their manual reviewing.<br>8. The Platform consolidates the educator's score with the scores calculated automatically during the time of the battle.<br>9. The Platform updates the battle's rank and the personal student's rank, for each student in the group.<br>10. The Platform sends an email to all students of the group.<br>11. The Platform redirects the educator to the battle page. |
| **Alternative flow** | 6. The educator chooses to cancel the manual assessment process, returning to the battle page. |
| **Exit condition** | Manual evaluation is completed successfully and the system updates battle ranking with scores provided by the educator, and notifies students. |


:Educator
:CKB Platform
:Email service
:Student
battle over
loop
[forall groups in battle]
opt
notify manual eval required
give manual evaluation
confirm evaluation
consolidate scores
update ranks
loop
[foreach student in group]
send on-site notification
send email notification

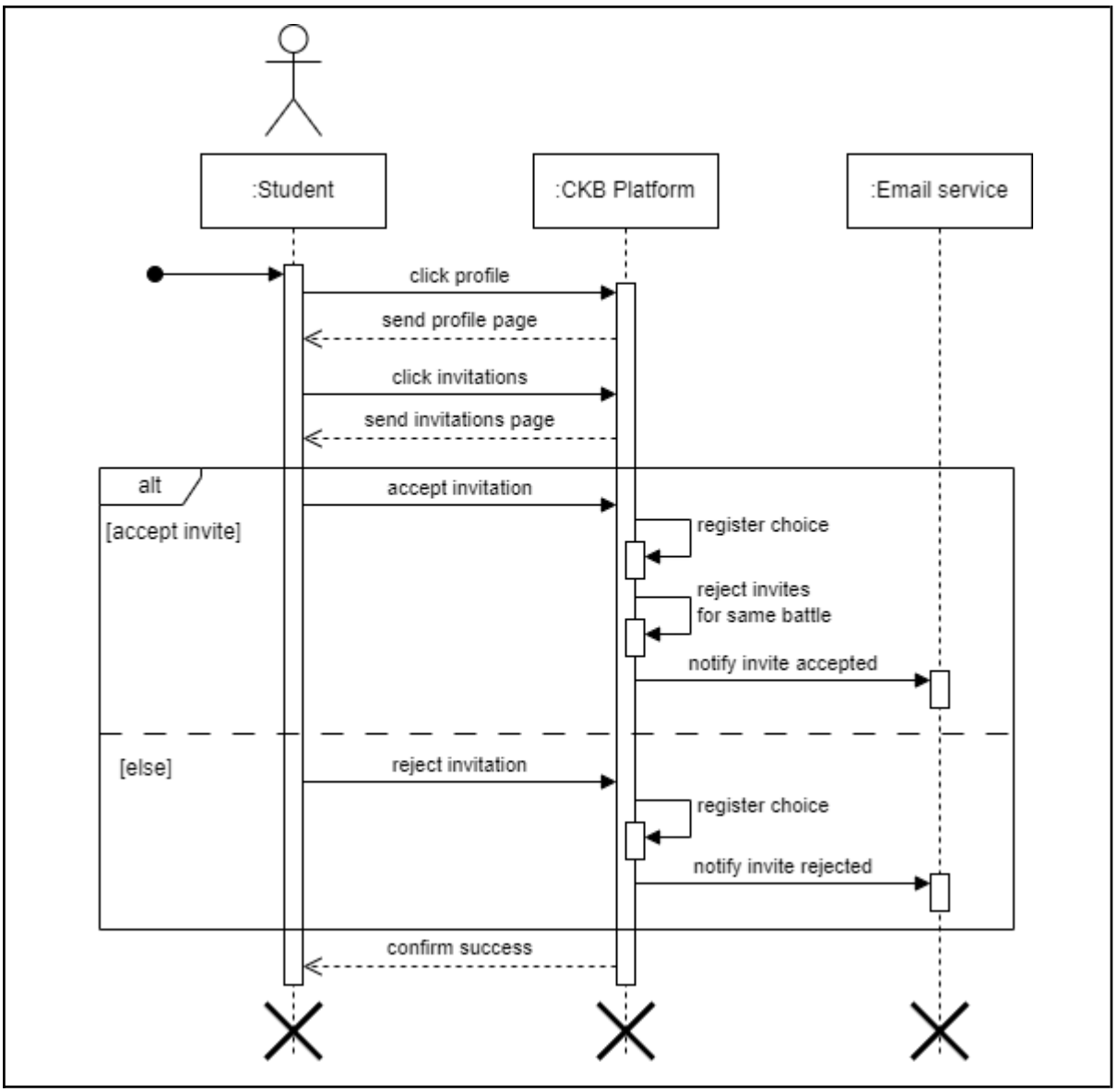
**[UC14]: Search for battles and tournaments**

| | |
|---|---|
| **Name** | Search for battles and tournaments. |
| **Actors** | User. |
| **Entry Condition** | The user is logged in and is on the home page. |
| **Event flow** | 1. The user inserts the name of a tournament or of a battle in the search bar.<br>2. The platform looks for the requested battle/tournament in the list of ongoing tournaments and battles.<br>3. The platform returns the search results to the user. |
| **Alternative flow** | 3. The platform could not find any results.<br>4. The platform notifies the user that the search has produced no results. |
| **Exit condition** | The user views a list of results that correspond to the search criteria in input. |



**[UC15]:Manage invitations to groups**

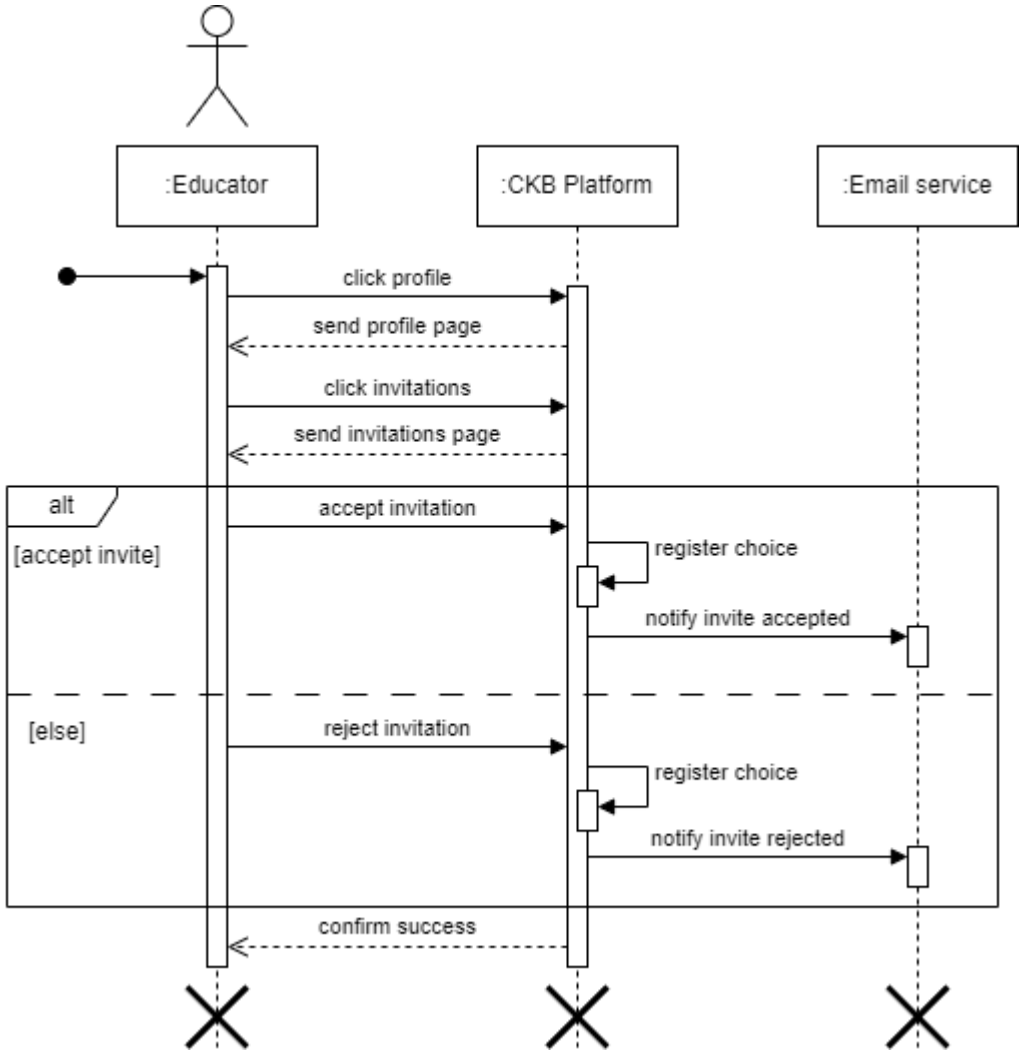
| | |
|---|---|
| **Name** | Manage invitations. |
| **Actors** | Student, Email service. |
| **Entry Condition** | The student is logged in and has received an invitation to a group. |
| **Event flow** | 1. The student goes to their profile page.<br>2. The student clicks on “Invitations”.<br>3. The student clicks on “Accept”.<br>4. The platform registers the user’s choice by adding them to the group.<br>5. The platform automatically rejects all other pending invites for that battle.<br>6. The platform notifies the members of the group that the student has accepted the invite.<br>7. The platform notifies the student of the success of the action. |
| **Alternative flow** | 3. The student clicks on “Reject”.<br>4. The platform registers the user’s choice.<br>6. The platform notifies the members of the group that the student has rejected the invite. |
| **Exit condition** | The platform registers the student’s choice and, if they accept they are added to the group, otherwise the invite will be canceled. |

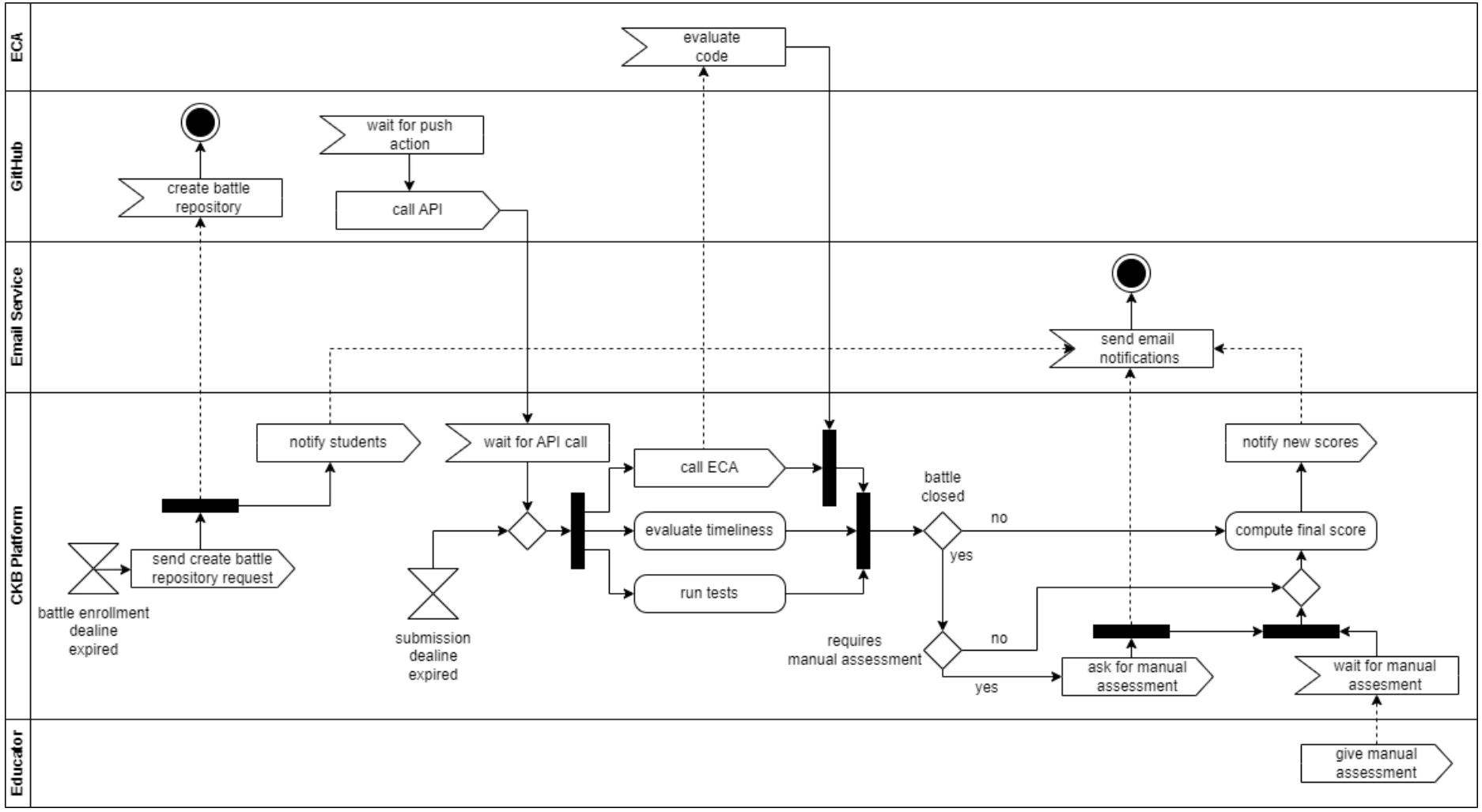



**[UC16]:Manage invitations to tournament**

| | |
|---|---|
| **Name** | Manage invitations. |
| **Actors** | Educator, Email service. |
| **Entry Condition** | The educators logged in and has received an invitation to manage a tournament. |
| **Event flow** | 1. The educator goes to their profile page.<br>2. The educator clicks on “Invitations”.<br>3. The educator clicks on “Accept”.<br>4. The platform registers the user’s choice by adding them to the tournament managers. |

| | |
|---|---|
| | 5. The platform notifies the owner of the tournament that the educator has accepted the invite.<br>6. The platform notifies the educator of the success of the action. |
| **Alternative flow** | 3. The educator clicks on “Reject”.<br>4. The platform registers the user’s choice.<br>5. The platform notifies the owner of the tournament that the educator has rejected the invite. |
| **Exit condition** | The platform registers the student’s choice and, if they accept they are added to the group, otherwise the invite will be canceled. |






*Activity diagram showing the repository creation and the evaluation process through the ECA.*

### 3.2.2 Requirements mapping

| Goals | Requirements | Use Case | Domain Assumptions |
|---|---|---|---|
| G1 | R1, R2, R3, R4, R5, R20.3, R20.4, R20.5 | UC1, UC2, UC5, UC6, UC7, UC15 | D1, D3, D4, D5.3 |
| G2 | R1, R2, R3, R4, R11.1, R15.1, R15.2, R15.3 | UC1, UC2, UC5, UC6 | D3, D5.3 |
| G3 | R1, R2, R3, R11.2, R12.5, R13, R14, R17, R19 | UC1, UC2, UC3, UC4, UC5, UC6, UC14 | D3, D4 |
| G4 | R1, R2, R6, R7, R8, R9, R16, R15.4, R15.5, R18, R20.1, R20.2 | UC1, UC2, UC8, UC9, UC10, UC11, UC12, UC16 | D1, D2, D3, D4, D5.1, D5.3 |
| G5 | R1, R2, R7, R15.3, R16, | UC1, UC2, UC8 | D3, D4, D5.3 |
| G6 | R1, R2, R9, R10, R12, R21 | UC1, UC2, UC9, Uc13 | D2, D3, D4, D5 |
| G7 | R1, R2 | UC1, UC2 | D3, D4, D5.3 |
| G8 | R1, R2, R11, R12.1 | UC1, Uc2 | D1, D4, D5.1 |

## 3.3. Non-functional requirements

### 3.3.1 Security

This is the category in which the most important non-functional requirements fall into.

The system must adopt robust security measures for the storage of user data that is collected during the authentication process. All user-related information, including login credentials and personal data, must be securely stored using industry-standard hashing and encryption algorithms so that information (such as passwords) is not directly stored in plain text. The authentication process should adhere to best practices, including but not limited to, strong password policies and secure session management.

The system must also implement measures to prevent common security threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

In addition, the system must comply with the General Data Protection Regulation (GDPR) of the European Union. This includes obtaining explicit consent from users for data processing, providing mechanisms for users to access and manage their data, and ensuring the right to be forgotten. More details will be provided in the next sections.

### 3.3.2 Performance and Usability

The system should ensure a seamless user experience both in terms of performance and usability of the interface.

Page load times for commonly accessed features, such as login, leaderboard visualization, and in general all kinds of data visualization should not exceed three seconds under standard network conditions.

The system should be capable of handling a concurrent user load of at least 5000 users without a significant degradation in performance.

As stated in previous sections of the document, the user interface (UI) must be intuitive and user-friendly, catering to users of varying technical proficiency. The design should follow established usability principles, and user interactions should be straightforward and efficient.

The application must be responsive and compatible with a range of devices and screen sizes, ensuring a consistent user experience across different platforms.

### 3.3.3 Accessibility

As previously explained in the document, accessibility standards, such as those outlined in the Web Content Accessibility Guidelines (WCAG), must be adhered to, ensuring the application is usable by individuals with disabilities.

The following principles from the WCAG should be incorporated into the design and development process:

- ***Perceivable***: Information and user interface components must be presented in a way that is easily perceptible to all users, regardless of their abilities; for example by providing text alternatives for non-text content.
- ***Operable***: The interface should be navigable and operable using various input methods, such as keyboard, mouse, or touch. All functionality must be available through a keyboard interface, and users should have sufficient time to read and complete tasks.
- ***Understandable***: The information and operation of the user interface must be clear and straightforward. The application should use consistent navigation and functionality, avoiding confusion. Input assistance, error prevention, and error recovery mechanisms should be in place to help users understand and correct any mistakes.

## 3.4. Design Constraints

### 3.4.1 Standards compliance

The main constraint on the design and implementation of Code Kata Battle is the compliance with the *General Data Protection Regulation* (GDPR) enacted by the European Union (EU).

To ensure conformity with the GDPR, CKB must implement the following features and practices that have been partially explained in section 3.3. Non-functional requirements:

1. **User consent:** CKB must obtain explicit and informed consent from users before collecting any data.

2. **Data minimization:** CKB only collects the necessary personal data required for its purposes.
3. **Data security:** CKB employs robust security measures to protect user data from unauthorized access and manipulation.
4. **Right to access and erasure:** CKB allows users to access their personal data and request its deletion.
5. **Data protection by default:** CKB incorporates data protection principles into its design and development process.
6. **Privacy policy:** CKB provides users with a clear and up-to-date privacy policy that outlines the types of data collected, the purposes of processing, and user's rights.
7. **Data processing records:** CKB keeps records of its data processing activities to ensure accountability.

### 3.4.2 User device requirements

The platform is accessed by the user through a web application, therefore the user's device must be able to handle running a modern web application on a browser. The recommended hardware requirements are the following:

- Processor (CPU):
    - Dual-core processor (or higher);
    - Clock speed of 2.0 GHz (or higher).
- Memory (RAM):
    - 4 GB RAM (or higher).
- Graphics:
    - Integrated graphics or dedicated graphics card.
- Display:
    - A display with at least 1366 x 768 resolution is recommended for laptops and monitors;
    - A display with at least 360×800 resolution is recommended for mobile devices.
- Operating System (OS):
    - Windows 10, macOS, or recent Linux distribution.
- Browser:
    - Latest versions of Mozilla Firefox, Chrome, Safari, Opera, Microsoft Edge, or any other browser supporting HTML 5 and ECMAScript.

# 4. Formal Analysis

In the following section, we present the Alloy model used to perform the formal analysis for a portion of the statements in previous sections. To be more precise, the model aims to describe and verify the following:

- Educators host (and manage) tournaments and tournaments host battles.
- Students can participate in battles both in groups or alone (while remaining within the confines defined for those battles).
- Tournaments can be associated with badges which can be automatically assigned to students who fulfill the requirements.
- Solutions provided by groups are automatically evaluated. If the battle has the option for manual evaluation, the solution will also require an educator with managing permissions for the tournament to assess the solution at the end of the battle.
- Students who have provided a solution for a given battle receive a score for the battle. Once the tournament closes, students also receive a tournament score.

*Note, For a clearer view, the full model can be found on the GitHub page for the project at the following link: https://github.com/paolo-chiappini/BersaniChiappiniFraschini/tree/main/Alloy. We report the model in the document to add a few comments.*

To achieve the goals described above, we define the following signatures:

```
open util/boolean

abstract sig User {}

enum Status { Open, Closed }

sig Student extends User {
      var awardedBadges: set Badge,
      var satisfiedBadges: set Badge,
      var tournamentScore: Tournament -> Int
} {
      Tournament.tournamentScore >= 0
}

sig Educator extends User {
      owns: disj set Tournament,
      hasPermissionsFor: set Tournament
```

```
}

sig Battle {
      groupsMinSize: one Int,
      groupsMaxSize: one Int,
      requiresManualEvaluation: Bool,
      var status: one Status,
      enrolledGroups: disj set Group
} {
      groupsMinSize > 0 and
      groupsMaxSize >= groupsMinSize and
      groupsMaxSize <= 2 // for simulation purposes
}

sig Tournament {
      battles: disj set Battle,
      badges: disj set Badge,
      var status: one Status
}

sig Badge {}

var sig Solution {
      var evaluated: Bool,
      var evaluatedBy: lone Educator
}

sig Group {
      owner: one Student,
      members: some Student,
      var battleScore: one Int,
      var currentSolution: disj lone Solution
} {
      battleScore >= 0 and
      battleScore <= 5 // for simulation purposes
}
```

Note that the signatures above represent only a partial, simplified system view. Thus, signatures are to be considered instrumental in describing the desired scenarios and not as an accurate description of their real counterparts. The signatures don't take into account, for instance, deadlines, user data, as well as badge data, among other things.

For the purpose of aiding in the description of the model, we define the following functions:

```
fun getBattleByGroup[g : Group]: one Battle {
    enrolledGroups.g
}

fun getStudentsInTournament[t : Tournament]: set Student {
    ((t.battles).enrolledGroups).members
}

fun getSolutionsByBattle[b : Battle]: set Solution {
    (b.enrolledGroups).currentSolution
}

fun getBattleBySolution[s : Solution]: one Battle {
    enrolledGroups.(currentSolution.s)
}

fun getTournamentManagersByBattle[b : Battle]: set Educator {
    hasPermissionsFor.(battles.b)
}

fun getAwardableBadges[t : Tournament, s : Student]: set Badge {
    t.badges & s.satisfiedBadges
}

fun getGroupsByStudentAndTournament[s : Student, t : Tournament]: set
Group {
    (members.s) & (t.battles).enrolledGroups
}

fun sumBattleScoresForGroups[g : Group]: Int {
    sum score : g.battleScore | score
}
```

We now introduce the model. For clarity, we report the description of the requirements from previous sections that have been modeled.

```
// General facts

// Solutions are created by groups
fact groupSolutions {
    always all sl : Solution | some g : Group | sl in g.currentSolution
}

// Consider only students in groups
fact studentsInGroups {
    always all s : Student | some g : Group | s in g.members
}

// After a badge is assigned, the student can't be eligible for that
badge again
```

```
fact noDoubleBadges {
    always all s : Student | s.awardedBadges & s.satisfiedBadges = none
}

// Students cannot become eligible for badges of already closed
tournaments
fact achieveOpenTournamentBadges {
    always all b : Badge | all s : Student | let t = badges.b |
        b in s.satisfiedBadges implies t.status = Open
}

// Student cannot achieve badges fo tournament they didn't participate in
fact mustPartecipateToTournamentForBadge {
    always all s : Student | all t : Tournament |
        (s.satisfiedBadges & t.badges) != none implies s in
getStudentsInTournament[t]
}

// Once an educator has manually evaluated a solution, no other educator
can evaluate
fact evaluatorRemainsTheSame {
    always all s : Solution | s.evaluatedBy != none implies
        s.evaluatedBy = s.evaluatedBy'
}

// Manual evaluation implies that the solution has been evaluated
fact evaluationConsistency {
    always all s : Solution |
(getBattleBySolution[s].requiresManualEvaluation = True and
        s.evaluatedBy != none) implies s.evaluated = True
}

// If a tournament is closed, there can't be any open battle belonging to
that tournament
fact noOpenBattlesForClosedTournament {
    always all t : Tournament | t.status = Closed implies (
        let b = t.battles | b.status = Closed
    )
}

// If a battle is closed, all current solutions must be evaluated
fact allSolutionsEvaluatedForClosedBattle {
    always all b : Battle | b.status = Closed implies
        (let s = getSolutionsByBattle[b] | s.evaluated = True)
}

// If a group hasn't uploaded a solution or the current one hasn't been
evaluated,
// they should have 0 points
fact noSolutionOrEvaluationNoScore {
    always all g : Group | (g.currentSolution = none or
(g.currentSolution).evaluated = False)
```

```
        iff g.battleScore = 0
}

// The score assigned to a specific solution is always the same
fact sameSolutionSameScore {
    always all g : Group |  g.currentSolution = g.currentSolution'
implies
        g.battleScore = g.battleScore'
}

// -------- REQUIREMENTS ----------

// [R4.1]: The system should not allow a student to enroll in multiple
battles within
// the same tournament.
fact studentOneEnrollmentPerBattle {
    all disj g1, g2 : Group | (g1.members & g2.members) != none iff
        getBattleByGroup[g1] != getBattleByGroup[g2]
}

// [R7]: The system allows an educator to create badges within a
tournament.
fact badgeInTournament {
    all bg : Badge | some t : Tournament | bg in t.badges
}

// [R8]: The system allows an educator to create a battle within a
tournament.
fact battleInTournament {
    all b : Battle | some t : Tournament | b in t.battles
}

// [R10]: The system allows the educator to make a manual assessment of
the students'
// solution, if specified in the scoring configurations.

// The educator giving a manual assessment must have permissions to
manage battle
fact evaluatorIsTournamentManager {
    always all s : Solution | s.evaluatedBy != none implies
        s.evaluatedBy in
getTournamentManagersByBattle[getBattleBySolution[s]]
}

// Solutions should only be evaluated at the end of a battle
fact manualEvaluationOnlyAfterBattleCloses {
    always all s : Solution | s.evaluatedBy != none iff (
        let b = getBattleBySolution[s] | b.requiresManualEvaluation =
True and b.status = Closed
    )
}
```

```
// [R14]: The system must update the personal tournament score of each
student, that is
// the sum of all battle scores received in that tournament, at the end
of each battle.
fact tournamentScoreEqualToSumOfBattles {
    always all s : Student, t : Tournament | s in
getStudentsInTournament[t] implies
        t.(s.tournamentScore) =
sumBattleScoresForGroups[getGroupsByStudentAndTournament[s, t]]
}

// [R15]: The system allows the educator to close a tournament.
pred closeTournament[t : Tournament] {
    t.status = Open and t.status' = Closed
}

// [R16]: The system should assign a badge to one or more students at the
end of the
// tournament if the students have fulfilled the badge's requirements to
achieve it.
fact assignOnlySatisfiedBadges {
    always all t : Tournament | closeTournament[t] implies
        (all s : Student | s.awardedBadges' = s.awardedBadges +
getAwardableBadges[t, s])
}

// [R16.1]: Badges should not be assigned until the tournament is over.
fact noBadgesForOpenTournaments {
    always all t : Tournament | no s : Student | t.status = Open and
        (s.awardedBadges & t.badges) != none
}

// [R12.6]: The system should not take into considerations solutions
outside the
// submission deadlines (aka after a battle ends)
fact noUploadsAfterBattleEnd {
    always all b : Battle | b.status' = Closed implies
        getSolutionsByBattle[b] = (b.enrolledGroups).currentSolution'
}

// [R4.3]: The system should not allow groups that don't meet battle
group size
// requirements to participate.
fact correctGroupSize {
    all g : Group | let b = getBattleByGroup[g] |
        #g.members >= b.groupsMinSize and #g.members <= b.groupsMaxSize
}

// [R6.1]: Every tournament should have one and only one owner.
fact singleTournamentOwner {
    all t : Tournament | one e : Educator | t in e.owns
}
```

```
// [R6.2]: Every educator that owns a tournament should have permission
to manage that
// tournament.
fact tournamentOwnership {
    all e : Educator | e.owns in e.hasPermissionsFor
}

// Group owner is part of the group members.
fact groupOwnership {
    all s : Student | all g : Group | s in g.owner implies s in g.members
}

// [R15.5]: Once a tournament has been closed, it cannot be reopened
fact noReopeningTournaments {
    always all t : Tournament | t.status = Closed implies t.status' =
Closed
}

// [R16.2]: Once a badge has been assigned to a student, the student
cannot lose that badge.
fact badgesCannotBeLost {
    always all s : Student | s.awardedBadges in s.awardedBadges'
}

// [R18.1]: Once the submission deadline for a battle has been reached,
it cannot be reopened
fact noReopeningBattles {
    always all b : Battle | b.status = Closed implies b.status' = Closed
}
```

In order to verify the satisfaction of the chosen goals, we introduce the following assertions:

```
// [G1]: Students want to participate as groups (or by themselves) in
coding battles
assert studentsParticipateToBattles {
    some s : Student | some g : Group | some b : Battle |
        g in b.enrolledGroups and
        s in g.members and #g.members >= 1
}

check studentsParticipateToBattles

// [G5] Educators want to create and customize gamification badges that
will be
// automatically assigned to students.
assert badgesCanBeAssignedToStudents {
    eventually some s : Student | some b : Badge |
        b in s.awardedBadges
```

```
}

check badgesCanBeAssignedToStudents

// [G6]: Educators want the platform to provide automated assessments of
students'
// work while also enabling manual evaluations.
assert computePointsForBattles {
    eventually all g : Group |
        g.battleScore != 0 iff (g.currentSolution != none and
g.currentSolution.evaluated = True)
}

check computePointsForBattles

assert pointsAssignedForTorunaments {
    eventually all s : Student | all t : Tournament |
        let g = getGroupsByStudentAndTournament[s, t] |
            s in getStudentsInTournament[t] and t.status = Closed implies
            t.(s.tournamentScore) = sumBattleScoresForGroups[g]
}

check pointsAssignedForTorunaments

assert automatedAssessment {
    eventually some s : Solution |
        s.evaluated = True
}

check automatedAssessment

assert allowManualAssessment {
    eventually all s : Solution | let b = getBattleBySolution[s] |
        b.requiresManualEvaluation = True and b.status = Closed implies
some e : Educator |
            s.evaluatedBy = e and e in getTournamentManagersByBattle[b]
}

check allowManualAssessment
```

From running the model, and by checking the assertions, we can see that the Alloy Analyzer is unable to find counterexamples for the described statements. Thus, we conclude that the model is potentially consistent.

We also show one of the potential outputs for the model, run for 3 steps in time:

*Note, the following representation is a transposed version of the actual Alloy output. The output has been cleared and re-ordered for clarity and readability. Similar outputs can be obtained by running the model from the GitHub repository of the project.*

*Fig. 1 Alloy generated scenario at time step 0.*

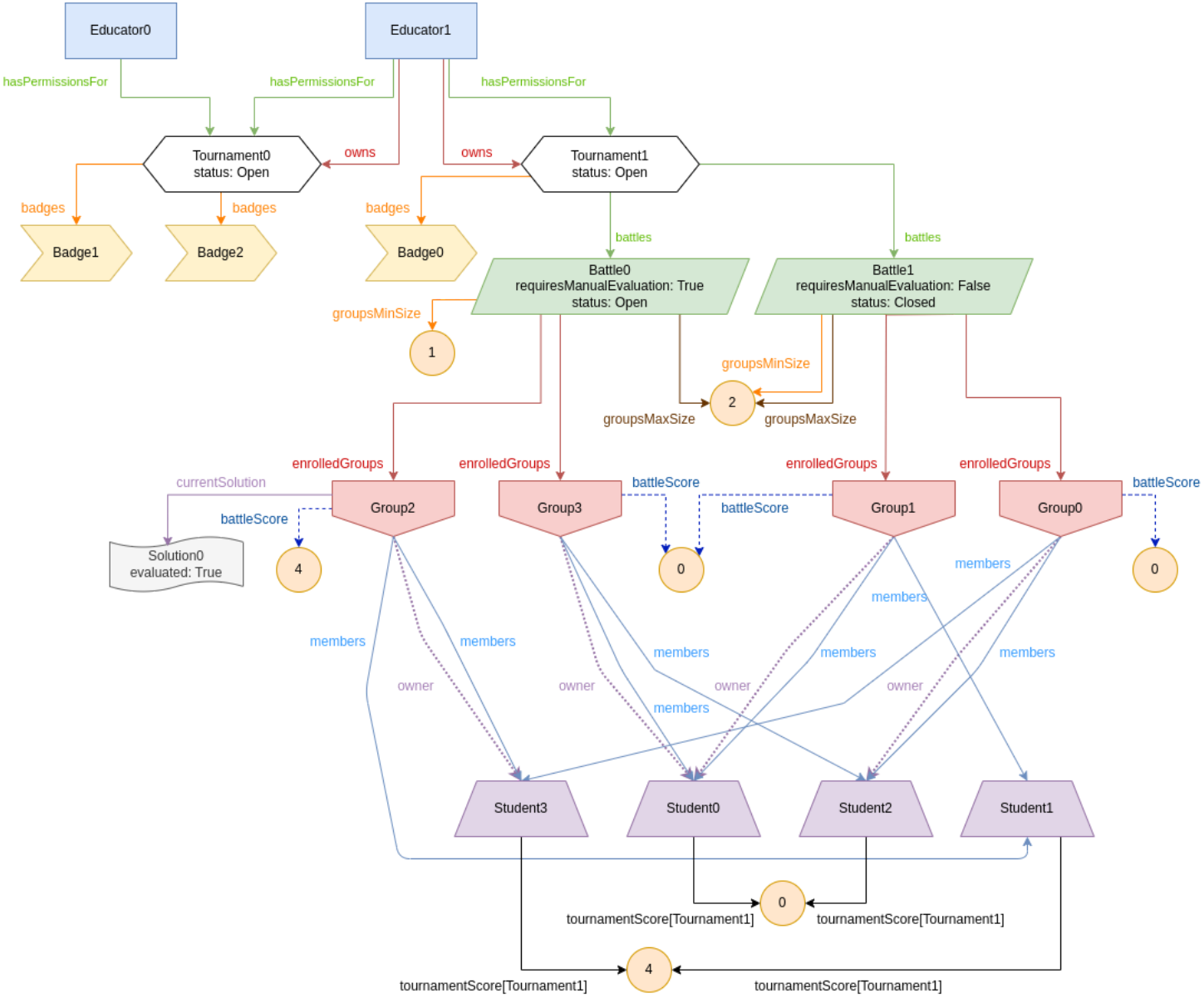


*Fig. 2 Alloy generated scenario at time step 1*

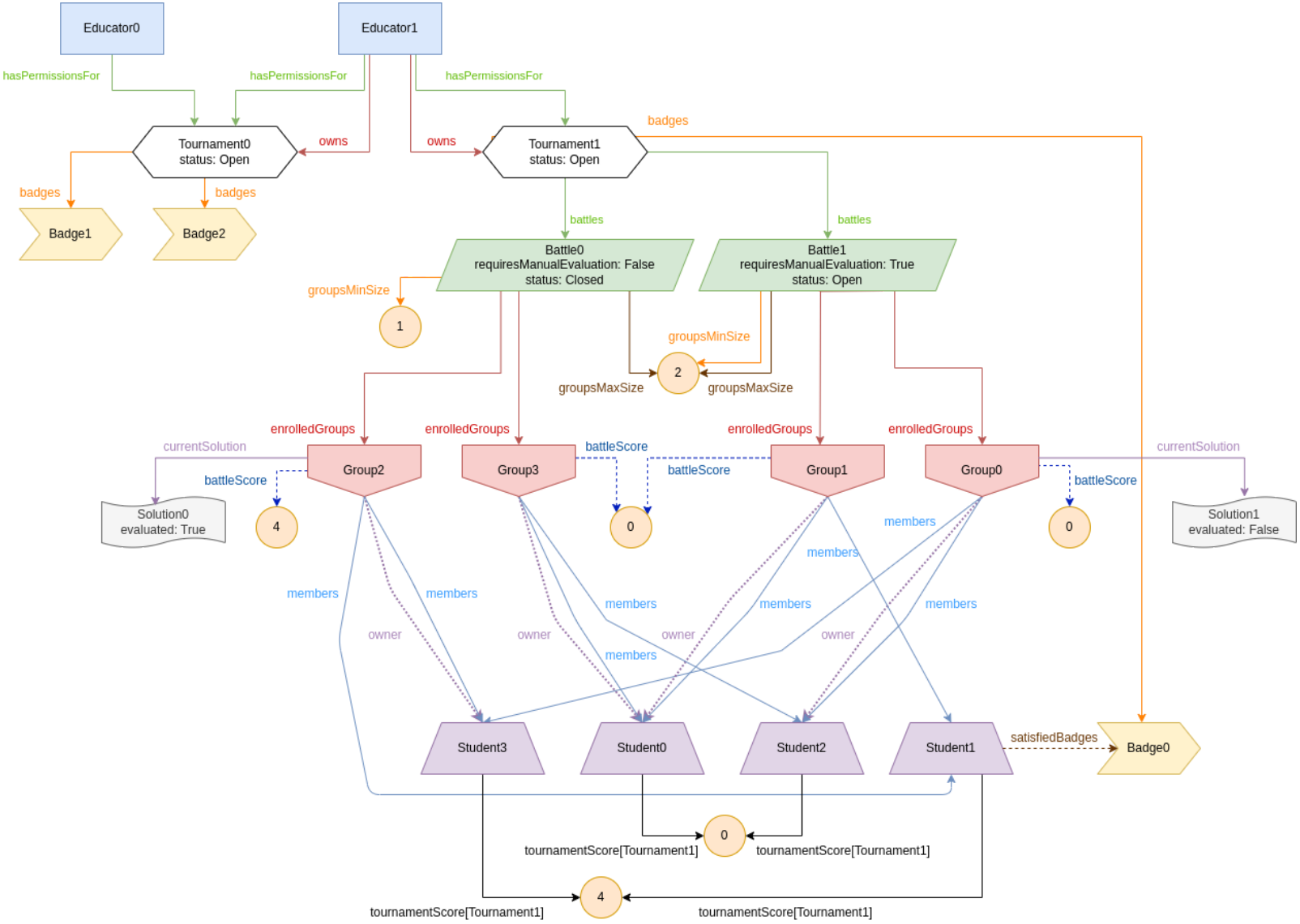
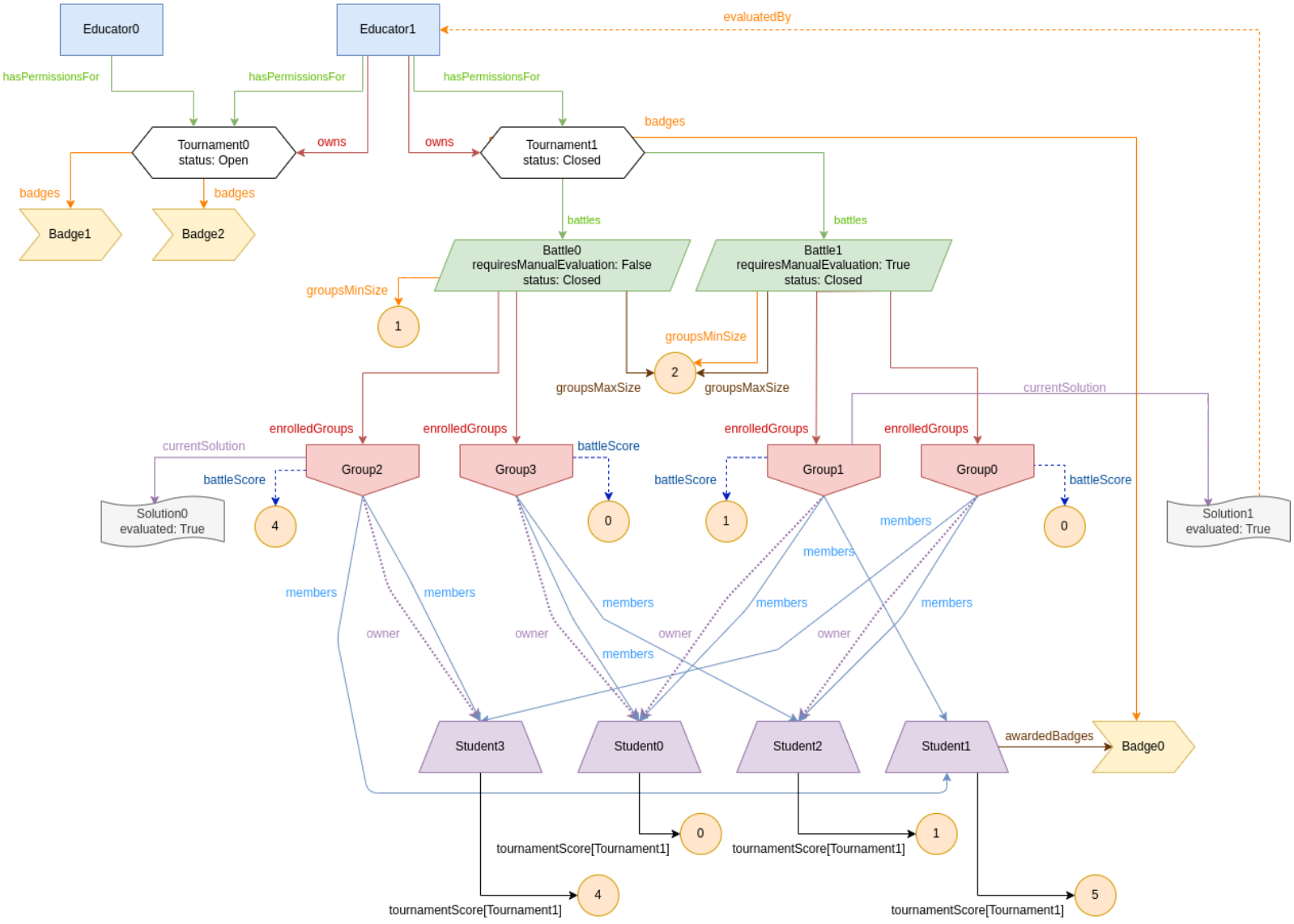


*Fig. 3 Alloy generated scenario at time step 2*



# 5. Effort spent on document

## 5.1. Shared effort

This is the effort spent by all the members of the group during various meetings and/or group discussions.

| **Activity** | **Effort spent** |
|---|---|
| Initial setup and task allocation | 2h |
| Meeting, review of goals and phenomena, and task allocation | 1h |
| Meeting, review of requirements and use cases | 1h |
| Meeting, review of the final details | 1h |
| Other meetings and discussions | 2h |
| **Total shared effort** | **7h** |

## 5.2. Individual Effort

| **Bersani Michele** | |
|---|---|
| **Activity** | **Effort spent** |
| The world and the machine phenomena | 2h |
| Beginning of use cases | 1h |
| Use cases (continuation) and functional requirements | 2h |
| Revision of use cases and requirements, use case tables | 2h |
| Use case tables | 3h |
| Requirement mappings | 1h |
| Use cases update | 1h |
| Revisions | 2h |
| **Total individual effort** | **14h** |

| **Chiappini Paolo** | |
|---|---|
| **Activity** | **Effort spent** |
| Paper reading | 1h |
| Purpose and Scope sections | 2h |
| Product functions | 2h |
| User characterization and accessibility concerns | 2h |
| User interfaces requirements | 1.5h |
| Software interfaces requirements | 1.5h |
| Diagrams (sequence and activity) | 5h |
| Alloy Model | 10h |
| Revisions | 8h |

| **Total individual effort** | **33h** |
|---|---|

| **Fraschini Andrea** | |
|---|---|
| **Activity** | **Effort spent** |
| Product perspective and diagram | 2h |
| Non functional requirements | 3h |
| User stories | 2h |
| Assumptions and dependencies | 2h |
| Revisions | 8h |
| **Total individual effort** | **17h** |

** Please note that all the above are estimates and may not accurately reflect the actual effort spent.*

# 6. References


- [Jackson 95] Michael Jackson. 1995. "The world and the machine". *Proceedings of the 17th international conference on Software engineering (ICSE '95). Association for Computing Machinery, New York, NY, USA, 283–292.* DOI:https://doi.org/10.1145/225014.225041.
- [Jackson-Zave 95] M. Jackson and P. Zave, "Deriving Specifications from Requirements: an Example," *1995 17th International Conference on Software Engineering, 1995, pp. 15-15,* DOI:https://doi.org/10.1145/225014.225016.
- Web Content Accessibility Guidelines 2.1 (WCAG) https://www.w3.org/TR/WCAG21.
- General Data Protection Regulation (GDPR) https://gdpr-info.eu/.
- Alloy Analyzer documentations: https://alloytools.org.
- Shape language in UX/UI design: https://bootcamp.uxdesign.cc/why-focus-on-cross-cultural-design-in-ui-ux-design-a81ecbf84bfb.
- Internalization standards: https://www.w3.org/standards/.