A.Y. 2023/2024

POLITECNICO
MILANO 1863

RASD: Requirement Analysis and Specification Document

Davide Grazzani Alessandro Masini

Professor
Matteo Giovanni Rossi

Version 1.0
December 22, 2023

Contents

1 Introduction

CodeKataBattle (CKB) is a new platform that helps students improve their software development skills by training with peers on *code kata*[1]. Educators use the platform to challenge students by creating code kata *battles* in which teams of students can compete against each other, thus *proving & improving* their skills.

1.1 Purpose

Research shows that competing in team-games-tournaments focused on educational topics can improve students motivation[1] as well as their final learning outcome[2], therefore *CodeKataBattle* aims at using this learning model to help students to learn new programming languages along with writing secure, reliable and maintainable code through the use of online tournaments managed by educators

1.1.1 Goals

User Goals

G1: **Have his/her own profile on the application**

G2: **Participate in a coding tournament**

G3: **Cooperate with colleagues**

G4: **Review their own performance score**

Educator goals

G5: **Organize coding tournaments**

G6: **Run coding battles in their tournaments**

G7: **Close an existing coding battle in their tournaments**

G8: **Obtain an automated evaluation over a student solution**

G9: **Manually review students submissions**

[1] A kata is an exercise in karate where you repeat a form many, many times, making little improvements in each iteration.

1.2 Scope

Shared phenomena:

ID	Phenomenom	Controller
S1	User registers through the application	World
S2	User logs into the application	World
S3	Educator creates a tournament	World
S4	Educator set minimum and maximum number of students per group	World
S5	Educator set a registration deadline	World
S6	Educator set a final submission deadline	World
S7	Educator invites another educator to his/her tournament	World
S8	The system notifies the users about a newly created tournament	Machine
S9	User subscribes to a tournament	World
S10	Educator creates a battle	World
S11	The system notifies students about a new battle in a tournament	Machine
S12	Student creates a group	World
S13	Student invites someone into his group	World
S14	Student accepts an invitation to a group	World
S15	Student uploads a solution	World
S16	The system updates the score and rank of a group on the leaderboard	Machine
S17	The educator manually evaluates a solution	World
S18	The system notifies users participating into a battle that the final battle rank is now available	Machine

World phenomena:

ID	Phenomenom
W1	Student develops a possible code kata solution
W2	Students exchange ideas on how to solve a code kata
W3	User has technical problems
W4	Student stops trying to solve a code kata
W5	Educator designs the code kata for the next battle

1.3 Definitions, acronyms, abbreviations

Definitions

- **User**: Anyone who uses the CKB platform.
- **Student**: A user which decides to subscribe to a tournament.
- **Educator**: A user which manages a tournament. A user can become an educator by either creating a tournament or by being invited by the tournament's owner with the intent of helping with its administration.
- **Tournament**: The principal event of the platform, created by an educator, where the students can compete.
- **Battle**: A specific challenge created by an educator in the context of a tournament.
- **Group**: The association of one or more student in order to compete in a tournament.
- **Solution**: The code submitted for a revision by a group within the context of a battle.
- **Tournament's owner**: The educator who has created the specific tournament.
- **Code kata**: A comprehensive description of the battle and it's software, including test cases and build automation scripts.

Acronyms

- **UML**: Unified Modeling Language
- **API**: Application Program Interface

Abbreviations

- **TO**: Tournament's owner
- **CKB**: CodeKataBattle

1.4 Revision history

- Version 0.1: Setup
  - Created first layout
- Version 1.0: First release
  - Added section 1
  - Added section 2
  - Added section 3
  - Added section 4

1.5 Reference documents

- Specification document: "Assignment RDD AY 2023-2024"
- Alloy documentation: https://alloytools.org/documentation.html

1.6 Document structure

- **Section 1**: Introduces the problem, describes the project's goals and gives an analysis of the world and shared phenomena.
- **Section 2**: Gives an overall description of the project and of all the interactions that will occur between the system and its final users, including a list of possible scenarios and a description of all the actors involved. It provides also an UML class diagram that could eventually be used at a later time as a reference point by developers.

- **Section 3**: Includes all the project's requirements and an in-depth description of everything which was presented in Section 2.
- **Section 4**: Shows the Alloy model defined for this project.

2 Overall Description

In this section a description of some *typical scenarios*, the *Product functions* (see 2.2) offered by the CKB platform and the *UML class diagram* (see 2.1.2) are presented. Later on in section also assumption, dependencies and constrains are discussed (see 2.3, 2.4).

2.1 Product perspective

2.1.1 Scenarios

1. Mister Russo has just started a new career in software development and wants to become better at coding. For this reason he wants to create an account on the platform.

    - He opens up the application and clicks on the "Sign Up" button;
    - He inserts all the mandatory information in the provided fields and presses the "Confirm" button;
    - The system checks that e-mail has not been used before. After passing the verification a confirmation notification is sent to mister Russo;
    - He checks his notification to see if he has received the confirmation of successful creation of the account.

2. Mister Ferrari is a well known business owner that wants to organize an awarded competition between his employees. To fulfill his goal mister Ferrari decides to create a tournament. Moreover, since mister Ferrari does not know anything about software development, he decides to invite some coworker with the intent to manage the tournament.

    - He opens up the application and logs-in;
    - He navigates to the tournament section;
    - He clicks on "Create a new Tournament" button;
    - He inserts the tournament's name and deadline in the provided fields and presses the "Continue" button;
    - The system checks that no other tournament has the same name. Upon a successful verification a confirmation notification is sent to mister Ferrari;

- The system generates a notification informing all the users of CKB platform that a new tournament has been created;
- He check his notifications to see if he has received the confirmation of the successful creation of the tournament. At this point he can invite his collaborators;
- He clicks on "Tournament" page and then selects his own tournament;
- He clicks on "Manage educators" button;
- He clicks on the "+" button and search the collaborators he wants to invite using their e-mail or nickname;
- He clicks on "Invite educators" button.

3. Miss Esposito is one of the most skilled programmers in the company where she works. Because of this her boss asked her to create challenges for the company's tournament. The boss has already invited miss Esposito as a tournament's collaborator.

   - She checks her notifications to see if she has received the invite;
   - She clicks on the link on the notification to accept the invite to the tournament. The link redirects her to the platform login page;
   - She logs-in, navigates to the "Tournament" section and selects the tournament where she has been invited as an educator;
   - She clicks on "New Battle" button;
   - She clicks on "Upload" button in order to upload the code kata;
   - She fills up, using the provided fields, all the required information including:
     (a) minimum number of students per group
     (b) maximum number of students per group
     (c) registration deadline
     (d) final submission deadline
     (e) additional configuration for scoring
   - She clicks on "Create new battle" button;
   - The system generates a notification informing all users subscribed to the battle's tournament that a new battle is available.

4. Miss Bianchi is subscribed to the platform. Since she has some free time she wants to subscribe to the tournament she was notified. She has already checked her notification containing all the information about the tournament.

    - She opens up the application and logs-in;
    - She navigates to the tournament section;
    - She clicks on the corresponding tournament name that she has read about on her notification;
    - She clicks on "Subscribe" button.

5. Miss Romano and mister Colombo are close friends that would like to team up and compete in a battle. To do so they decide to create a group. Both of them are already subscribed to the same battle, and they establish a priori that Miss Romano will create the group on the platform.

    - Miss Romano opens up the application and logs-in;
    - She navigates to the tournament section;
    - She clicks on the corresponding tournament name in which both of them are already subscribed;
    - She clicks on the corresponding battle name in which both of them are already subscribed;
    - She clicks on "Create a group" button;
    - She inserts the name of the group;
    - The system checks that no other group with the same name is present in the context of the same battle;
    - She clicks on "Confirm creation" button;
    - She clicks on "Manage group" button;
    - She clicks on "Invite new user" button;
    - She searches the collaborators she wants to invite using their e-mail or nickname;
    - She clicks "Invite" button.
    - Mister Colombo awaits for the notification that contains the invite from Miss Romano;
    - He opens up the notification;

- The system redirects him to the application;
- He logs-in and accept the invite;

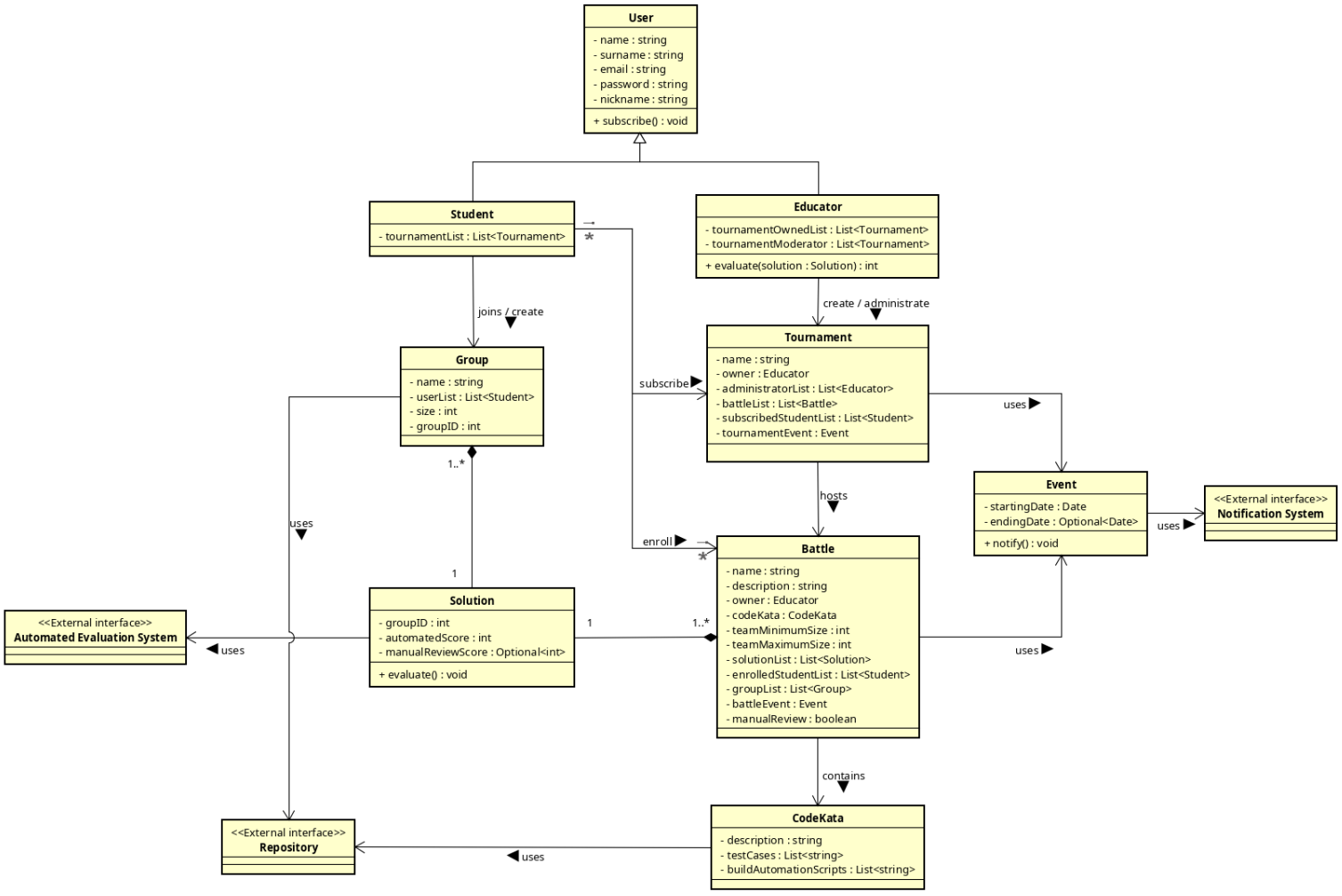
6. The group of students named "PythonMasters" wants to deliver the solution of "C Recursion" battle in which they are competing. PythonMasters has already forked the repository in order to access the code kata.
    - They set up an automated workflow, using the repository service that informs CKB platform when a push action occurs;
    - They push their solution onto the repository;
    - They open up the application and log-in;
    - They navigate to the tournament section and select the tournament in which they are competing;
    - They select "C Recursion" battle;
    - They await for the system to complete automated code evaluation;
    - They can now visualize their score.
7. Miss Bruno, educator of the tournament, wants to personally review the score of a battle. The battle has previously been created such that it needs manual code evaluation.
    - She opens up the application and logs-in;
    - She navigates to the tournament section and selects the tournament in which the battle is being hosted;
    - She clicks on "Review submissions" button;
    - She reviews the solutions.
8. Mister Ricci has created a battle with the wrong code kata and so he decides to remove such battle. He is already logged into the platform.
    - He navigates to the tournament section and selects the tournament in which he has previously created the battle;
    - He navigates to the current battle;
    - He clicks on "Delete battle" button;
    - He clicks on "Yes, I'm sure" button;

9. Miss Marino, given the new schedule, needs to close a tournament before the deadline expires. She is the TO of the aforementioned tournament and she is already logged into the platform.

    - She navigates to the tournament section and selects the tournament she wants to close;
    - She clicks on "Close tournament" button;
    - She clicks on "Yes, I'm sure" button;

2.1.2 Class diagram

In the UML diagram all the main entities of the system are shown. CKB platform is principally composed by two principal entities: *User* and *Tournament*. The *User* entity represents all the users of the platform and models the interaction of the user with the system by a dynamic specialization into *Student* and *Educator* entities. This ensures flexibility for the end user of the platform that can be at the same time a student and an educator. Hence, the user can compete in a tournament and at the same time create its own tournament, all without the need of creating multiple accounts.
The *Tournament* entity represents the tournaments created by the educators. This class models the competitions between students and the interaction of the educators with the system. From here a *Student* can enroll in a *Battle*, created by an *Educator*, and form *Group* with other students.



2.2 Product functions

- **Sign-up**: let users sign-up through their e-mail and password.

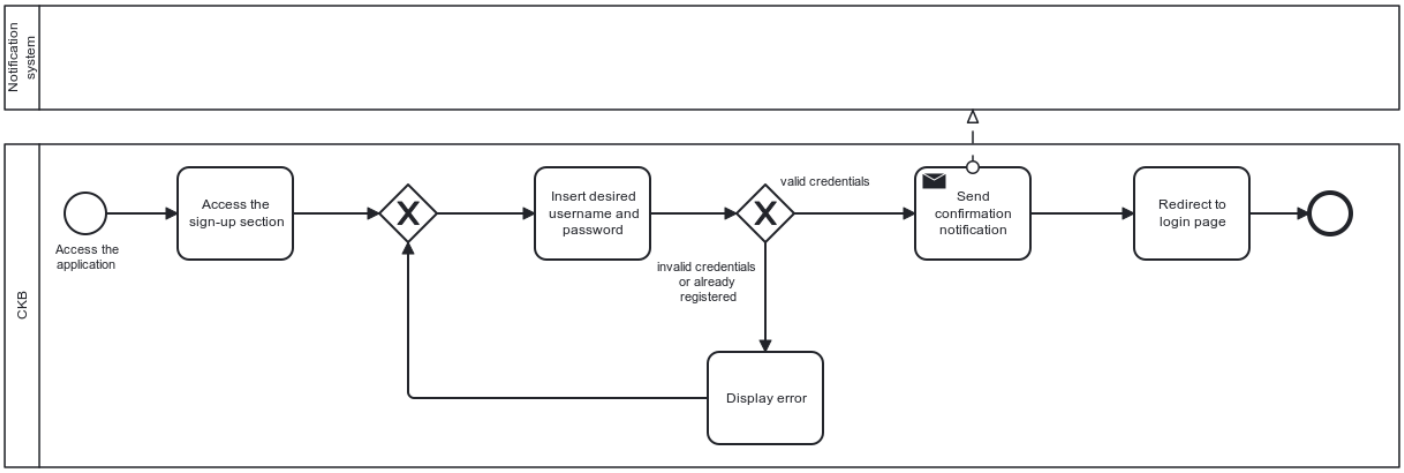


Figure 1: Sign-up to CKB platform.

- **Create a tournament**: let educators create a tournament.

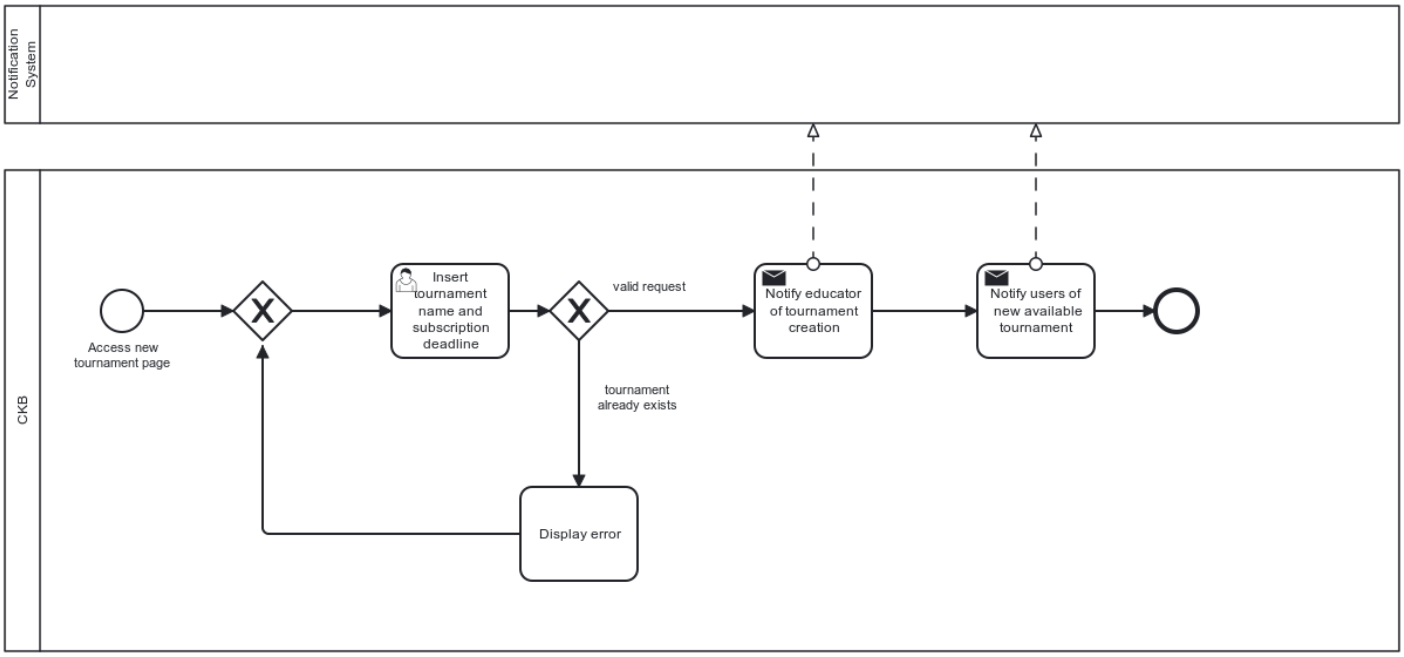


Figure 2: Create a tournament.

- **Invite educator**: let the *TO* invite other educators.

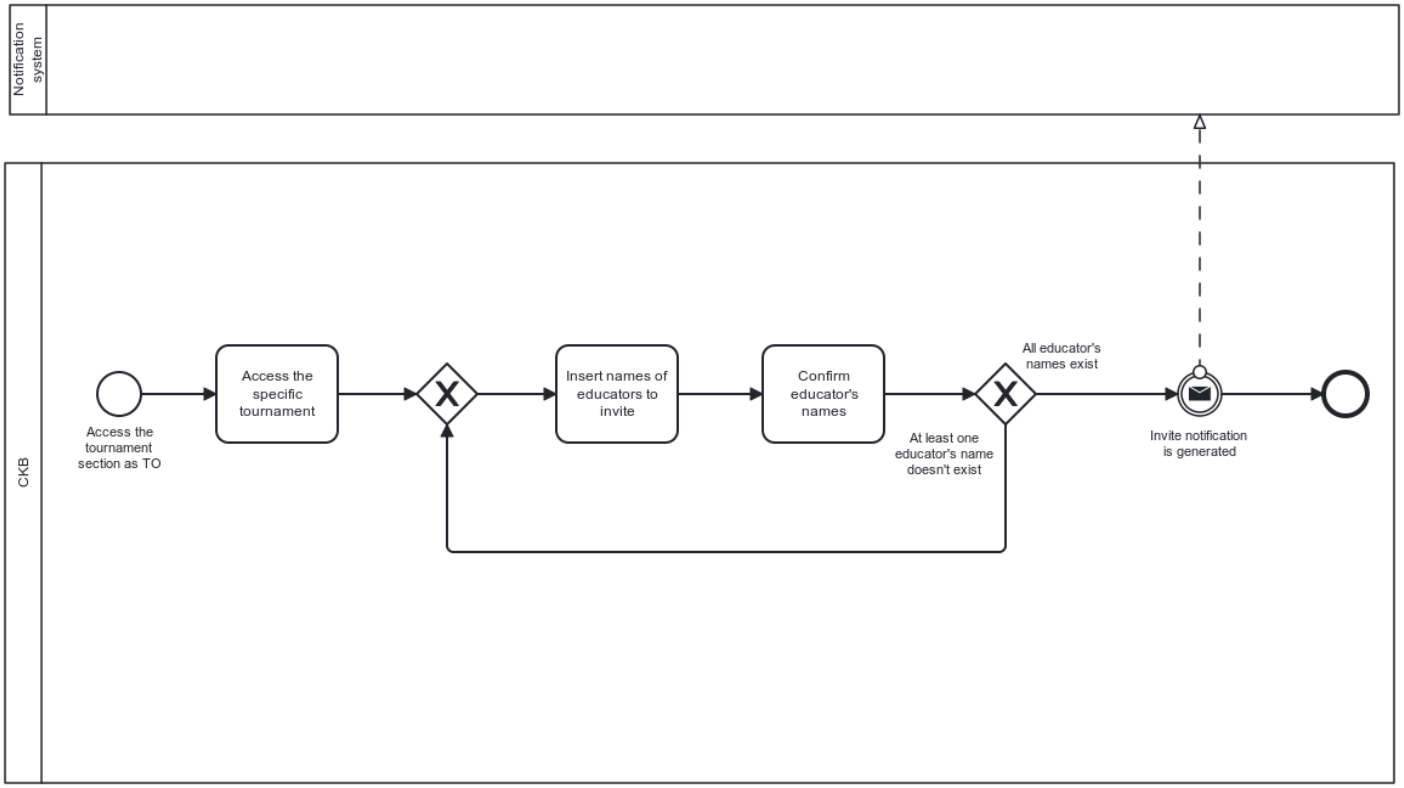


Figure 3: Manage a tournament as *TO*.

- **Create a battle**: let educators create a battle.

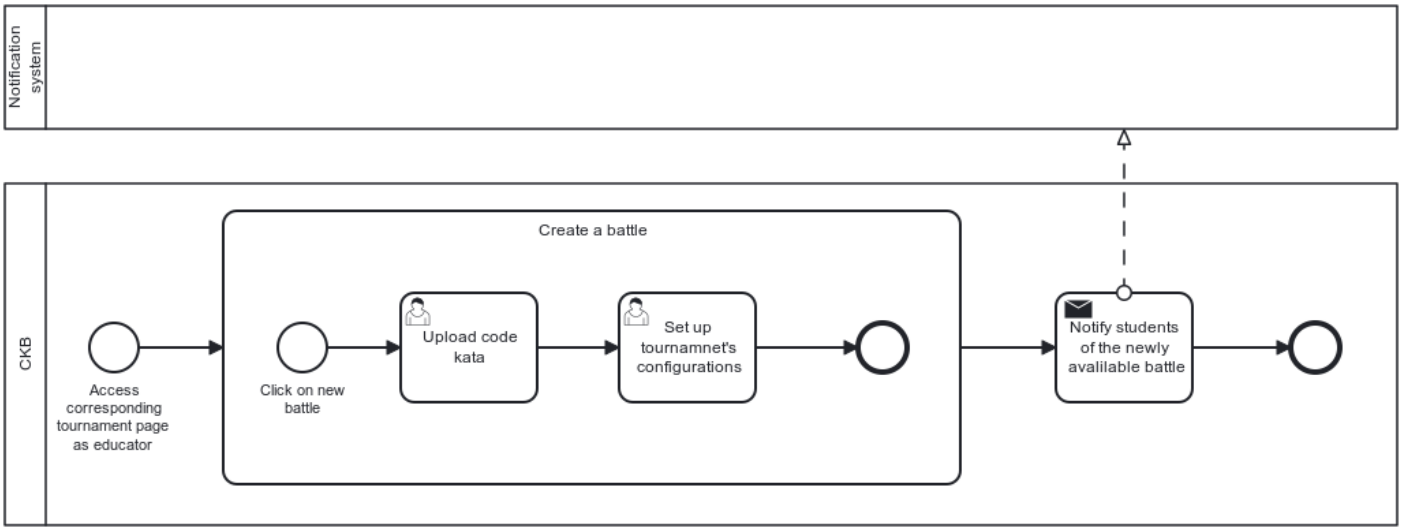


Figure 4: Create a battle.

- **Group creation**: let students enrolled into the same battle to create a group.

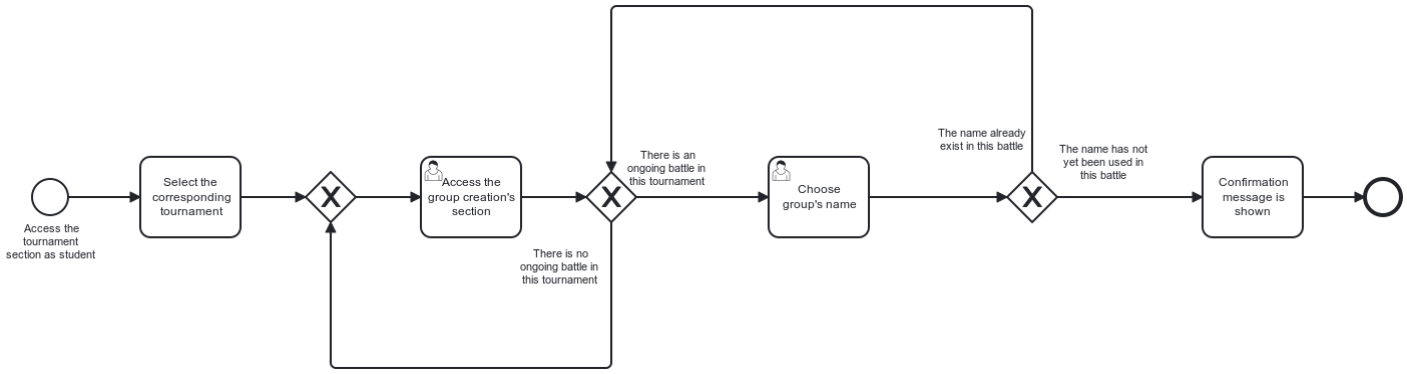


Figure 5: Let students enrolled intto the same battle to create a group.

- **Enroll in a battle**: let a student join a battle in a tournament.

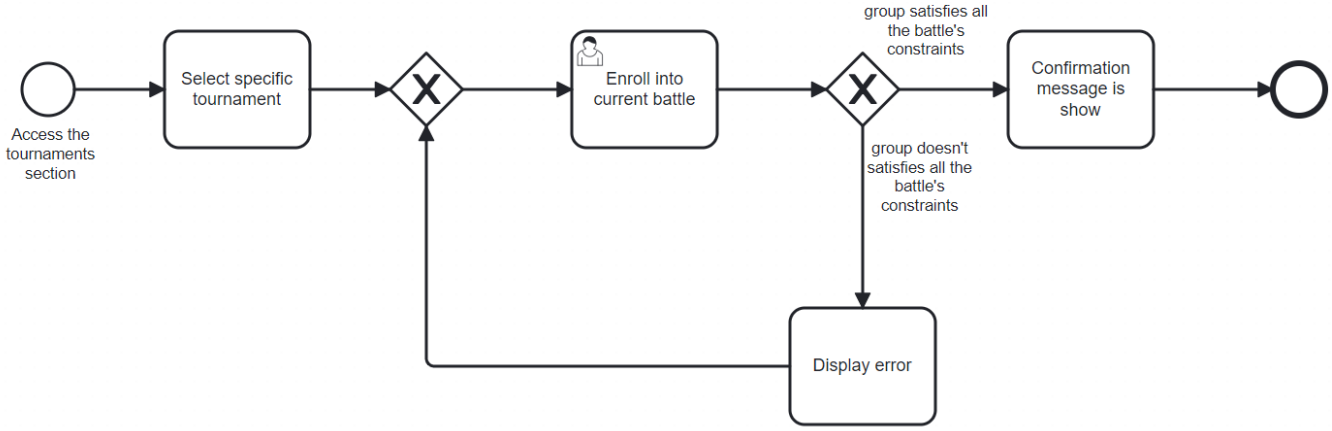


Figure 6: Join a battle in a tournament.

- **Group delivers a solution**: let groups of students upload their solution within the context of a battle and view their personal score.

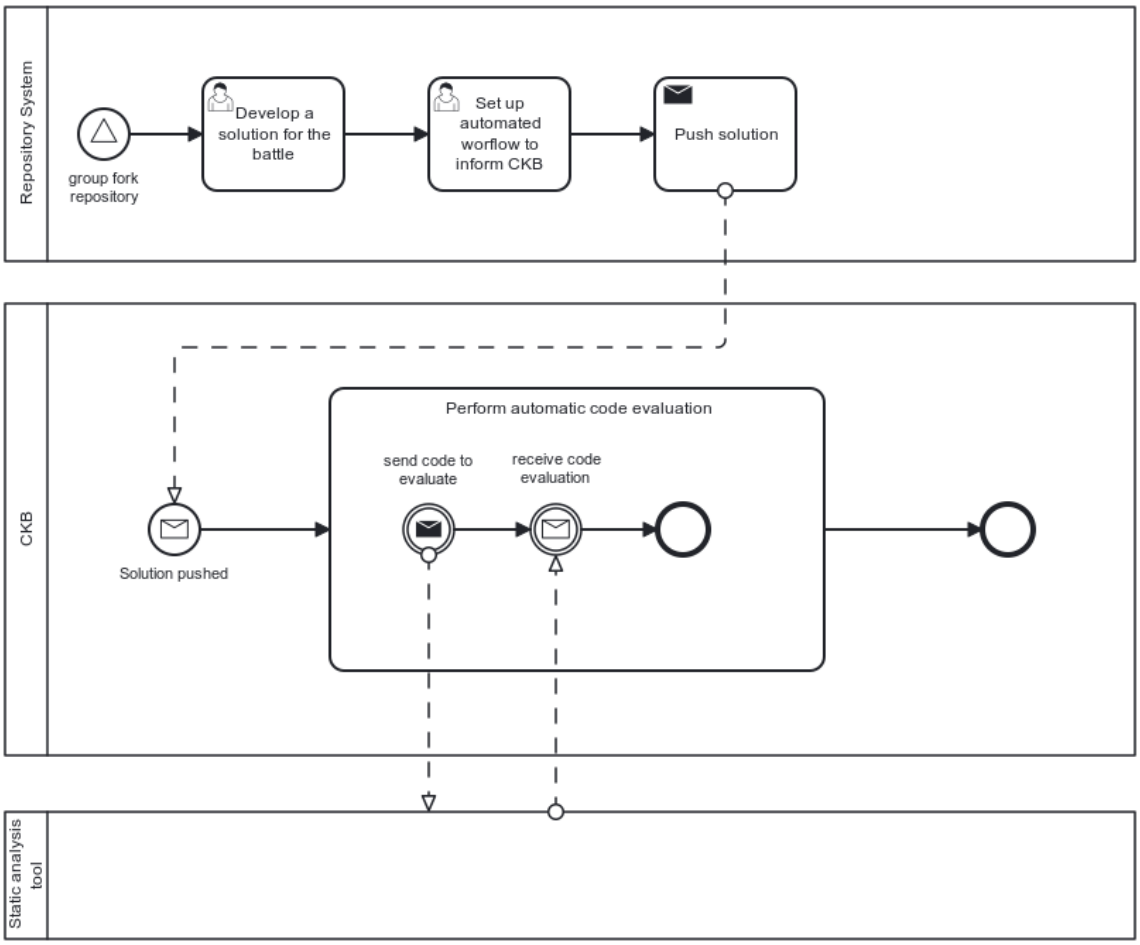


Figure 7: Let gropus upload their solution and see their result.

- **Educator manual review**: let the tournament's educators give a score to a group's solution within the context of a battle.

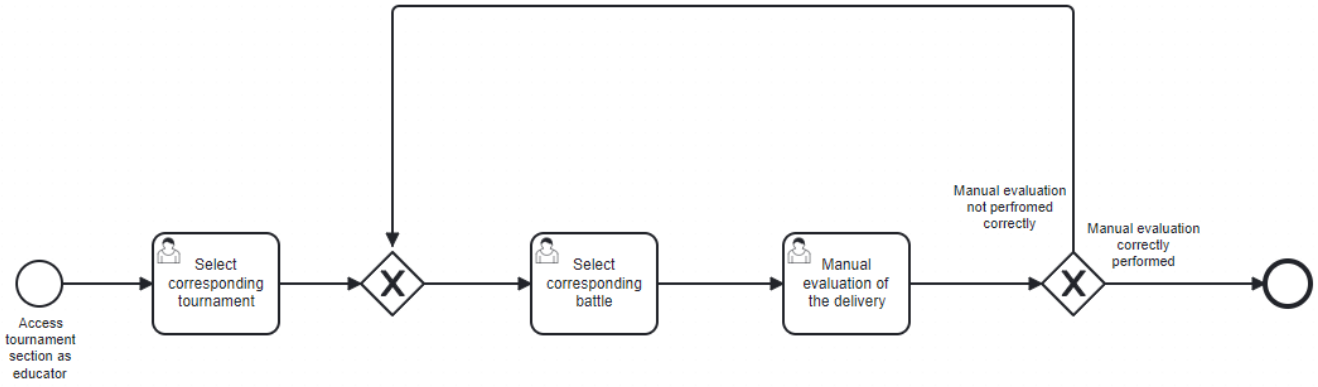


Figure 8: Let tournament's educators give an additional evaluation to a group's score.

2.3 User characteristics

The application has been designed for:

- anyone who wants to become better at coding;
- any organization or physical person who wants to create online, software development related, competitions.

For this reasons, the potential user base includes all the people that have a connection, whether personal or business related[2], with the world of software development. Users should be capable of interacting with the application and be in posses of a valid mailbox.

[2]This aside is intended to emphatize the fact that the platform can be used by users that have personal interest in software development, such as an hobby, but also by users that have professional careers in the field.

2.4 Assumption, dependencies and constrains

- **D1**: Users have internet access while using the application
- **D2**: Users have only one account[3]
- **D3**: Users device always send correct data to the application regarding its specifications[4]
- **D4**: The repository service is reliable and works 99.9% of the time
- **D5**: Users have the ability to interact and correctly use the repository platform
- **D6**: Users notification system is reliable and works 99.9% of the time
- **D7**: Users won't block notifications coming from the application
- **D8**: The automated evaluation system is reliable and works 99.9% of the time
- **D9**: Users who are members of the same group will contribute equally to the delivery
- **D10**: Users won't receive any external help from a person not in their group while competing in a battle
- **D11**: Educator's manual review of the delivery is unbiased

[3]To avoid multiple accounts per user

[4]For example the device type(smartphone, laptop, ...) or the screen width

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

CodeKataBattle's user interface will be used by both students and educators, so it should be easy to navigate and allow the use of all its functionalities to as many users as possible, regardless of the extent of their technical background. Therefore the interface has to be easy to use, intuitive and user-friendly on any kind of device.

3.1.2 Hardware Interfaces

The system revolves around the exchange of information and code solutions between the users and the system itself, so it doesn't require any specific external hardware interface to operate, it just needs to be securely reachable from the highest possible amount of people around the world

3.1.3 Software Interfaces

The system requires the presence of some external software interfaces in order to correctly operate:
First of all, it requires some kind of repository service, to which educators will upload their code kata and which students will fork, in order to implement it with their code. Then each group of students will set up an automated procedure to inform CodeKataBattle of each new solution they will deliver through this system
Once a new solution will be delivered by each group the application needs to have an automated evaluation system with which to automatically evaluate the functional aspects (measured in terms of number of test cases that are passed out of all the test cases, the higher the better), the timeliness (measured in terms of time passed between the registration deadline and the last commit performed by the group which made the delivery, the lower the better) and the quality level (extracted using static analysis tools that will consider aspects such as security, reliability, and maintainability of the code, the higher the better) of the sources obtained from the repository.
Finally, for many important events that will take place while using the application (for example the creation of a tournament or the early closure of a battle), the system will use some kind of notification system to inform the user that such events have taken place (this system can be emails, windows notifications, smartphone notifications, sms or others), those notifications should also often (if possible) have some sort of way to quickly go from the notification itself straight to the application point of interest where that specific action is requested to the user (for example by clicking on the notification itself or by clicking/copying a possible direct link contained in the notification)

Regarding the user instead, to access the system itself no particular software is required, but such software is necessary to develop the solution to be delivered. In order to do that theoretically just a text editor would be enough, but the use of an Integrated Development Environment (IDE) instead is strongly suggested, since it would make the development of such solution much easier and allow a more straightforward quick testing before the delivery

3.1.4 Communication Interfaces

As already mentioned, the system revolves around the exchange of information and code solutions between the users and the system itself, so the system needs to be securely reachable from the highest possible amount of people around the world, to assure that communications will allways be enrypted using TLS 1.3 by the usage of the HTTPS protocol for each communication

3.2 Functional Requirements

3.2.1 List of requirements

Sign up and profile requirements:

ID	Requirement
R1	The system must allow the registration of new users using their e-mail
R2	The system must notify a user about his/her successful registration
R3	The system must allow registered users to log-in using their e-mail.
R4	The system must allow logged-in users to use the application.
R5	The system must respect the GDPR
R6	The system must allow a user to set a nickname for his/her profile
R7	The system must allow a user to set a phone number for his/her profile

Tournaments and battles creation requirements:

ID	Requirement
R8	The system must allow an educator to create a tournament if another tournament with the same name doesn't already exist
R9	The system must reject an educator request to create a tournament if another tournament with the same name already exist
R10	The system must allow the owner of a tournament to invite other educators to manage it
R11	The system must notify an educator when he/she is invited to manage a tournament
R12	The system must allow an educator to accept the invitation of another educator to manage a tournament
R13	The system must allow an educator to set a registration deadline when creating a tournament
R14	The system must allow to set a minimum number of students for each group when creating a new battle
R15	The system must allow to set the maximum number of students for each group when creating a new battle
R16	The system must allow to set a registration deadline when creating a new battle
R17	The system must allow to set a final submission deadline when creating a new battle
R18	The system must allow an educator to choose while creating a new battle which aspects of the quality level of the sources will be automatically evaluated
R19	The system must allow an educator to choose while creating a new battle whether a manual evaluation will be needed after the automated ones
R20	The system must be able to receive a new code kata from an educator when he/she is creating a battle
R21	The system must allow the educators managing a tournament to create new battles in the context of such tournament
R22	The system must notify all users of the platform when a new tournament is created
R23	The system must notify all students subscribed to that tournament when a new battle of such tournament starts

Ongoing tournament requirements:

ID	Requirement
R24	The system must allow users to subscribe to a tournament
R25	The system must allow students to create groups in the context of a battle
R26	The system must allow a student to invite another to his/her own group, if such student is enrolled into the corresponding battle
R27	The system must reject a student request to invite another student to his/her own group, if such student is not enrolled into the corresponding battle
R28	The system must notify a student when he/she is invited to a group
R29	The system must allow a student to join an existing group if invited
R30	The system must allow groups to join a newly created battle if they respect that battle's constraints
R31	The system must allow all students enrolled into a battle, to access its code kata
R32	The system must allow a group to submit his/their solution
R33	The system must update a group battle score after each new solution is delivered
R34	The system must show to each group his/their rank in the current battle
R35	The system must show to each group his/their current battle score

Tournaments and battles concluding requirements:

ID	Requirement
R36	The system must notify all students subscribed to a tournament when the consolidation phase of a battle of such tournament has ended
R37	The system must be able to evaluate functional aspects of a group's delivery
R38	The system must be able to evaluate the timeliness of a group's delivery
R39	The system must be able to evaluate the quality level of the sources of a group's delivery
R40	The system must show the results of its automated analysis to the educators managing the tournament
R41	The system must allow an educator to see the solution provided by each group
R42	The system must allow an educator to manually mark a solution in the consolidation phase, if this was set up in the initial configuration of the battle
R43	The system must update the tournament score of each student participating into that tournament after the consolidation phase has ended
R44	The system must allow a student to see his/her rank in a certain tournament
R45	The system must allow a student to see his/her tournament score
R46	The system must allow all users to see the list of ongoing tournaments
R47	The system must allow all users to see the current tournament score leaderboard for each ongoing tournament
R48	The system must allow an educator to close a battle before its expected conclusion
R49	The system must allow an educator to set up a message explaining his/her motivations when closing a battle ahead of time
R50	The system must notify all students subscribed to a tournament when a battle in such tournament is closed ahead of time
R51	The system must show the educator's motivation message when notifying students about the early closure of a battle
R52	The system must allow an educator to close a tournament, if there is not an ongoing battle in such tournament
R53	The system must reject an educator request to close a tournament, if there is an ongoing battle in such tournament
R54	The system must notify all students subscribed to a tournament when such tournament is closed

3.2.2 Mapping

User goals

G1	**Have his/her own profile on the application**
R1	The system must allow the registration of new users using their e-mail
R2	The system must notify a user about his/her successful registration
R3	The system must allow registered users to log-in using their e-mail.
R4	The system must allow logged-in users to use the application.
R5	The system must respect the GDPR
R6	The system must allow a user to set a nickname for his/her profile
R7	The system must allow a user to set a phone number for his/her profile
D1	Users have internet access while using the application
D2	Users have only one account

G2	Participate in a coding tournament
R3	The system must allow registered users to log-in using their e-mail.
R4	The system must allow logged-in users to use the application.
R5	The system must respect the GDPR
R8	The system must allow an educator to create a tournament if another tournament with the same name doesn't already exist
R22	The system must notify all users of the platform when a new tournament is created
R24	The system must allow users to subscribe to a tournament
R31	The system must allow all students enrolled into a battle, to access its code kata
R32	The system must allow a group to submit his/their solution
R46	The system must allow all users to see the list of ongoing tournaments
D1	Users have internet access while using the application
D2	Users have only one account
D3	Users device always send correct data to the application regarding its specifications
D4	The repository service is reliable and works 99.9% of the time
D5	Users have the ability to interact and correctly use the repository platform
D6	Users notification system is reliable and works 99.9% of the time
D7	Users won't block notifications coming from the application

G3	Cooperate with colleagues
R3	The system must allow registered users to log-in using their e-mail.
R4	The system must allow logged-in users to use the application.
R5	The system must respect the GDPR
R25	The system must allow students to create groups in the context of a battle
R26	The system must allow a student to invite another to his/her own group, if such student is enrolled into the corresponding battle
R27	The system must reject a student request to invite another student to his/her own group, if such student is not enrolled into the corresponding battle
R28	The system must notify a student when he/she is invited to a group
R29	The system must allow a student to join an existing group if invited
R30	The system must allow groups to join a newly created battle if they respect that battle's constraints
R31	The system must allow all students enrolled into a battle, to access its code kata
R32	The system must allow a group to submit his/their solution
R33	The system must update a group battle score after each new solution is delivered
R34	The system must show to each group his/their rank in the current battle
R35	The system must show to each group his/their current battle score
D1	Users have internet access while using the application
D2	Users have only one account
D3	Users device always send correct data to the application regarding its specifications
D4	The repository service is reliable and works 99.9% of the time
D5	Users have the ability to interact and correctly use the repository platform
D6	Users notification system is reliable and works 99.9% of the time
D7	Users won't block notifications coming from the application
D9	Users who are members of the same group will contribute equally to the delivery

G4	Review their own performance score
R3	The system must allow registered users to log-in using their e-mail.
R4	The system must allow logged-in users to use the application.
R5	The system must respect the GDPR
R33	The system must update a group battle score after each new solution is delivered
R34	The system must show to each group his/their rank in the current battle
R35	The system must show to each group his/their current battle score
R43	The system must update the tournament score of each student participating into that tournament after the consolidation phase has ended
R44	The system must allow a student to see his/her rank in a certain tournament
R45	The system must allow a student to see his/her tournament score
R47	The system must allow all users to see the current tournament score leaderboard for each ongoing tournament
D1	Users have internet access while using the application
D3	Users device always send correct data to the application regarding its specifications
D8	The automated evaluation system is reliable and works 99.9% of the time
D9	Users who are members of the same group will contribute equally to the delivery
D10	Users won't receive any external help from a person not in their group while competing in a battle
D11	Educator's manual review of the delivery is unbiased

Educator goals

G5	Organize coding tournaments
R3	The system must allow registered users to log-in using their e-mail.
R4	The system must allow logged-in users to use the application.
R5	The system must respect the GDPR
R8	The system must allow an educator to create a tournament if another tournament with the same name doesn't already exist
R9	The system must reject an educator request to create a tournament if another tournament with the same name already exist
R10	The system must allow the owner of a tournament to invite other educators to manage it
R11	The system must notify an educator when he/she is invited to manage a tournament
R12	The system must allow an educator to accept the invitation of another educator to manage a tournament
R13	The system must allow an educator to set a registration deadline when creating a tournament
R21	The system must allow the educators managing a tournament to create new battles in the context of such tournament
R22	The system must notify all users of the platform when a new tournament is created
R24	The system must allow users to subscribe to a tournament
R52	The system must allow an educator to close a tournament, if there is not an ongoing battle in such tournament
R53	The system must reject an educator request to close a tournament, if there is an ongoing battle in such tournament
R54	The system must notify all students subscribed to a tournament when such tournament is closed
D1	Users have internet access while using the application
D3	Users device always send correct data to the application regarding its specifications
D6	Users notification system is reliable and works 99.9% of the time
D7	Users won't block notifications coming from the application

G6	Run coding battles in their tournaments
R3	The system must allow registered users to log-in using their e-mail.
R4	The system must allow logged-in users to use the application.
R5	The system must respect the GDPR
R8	The system must allow an educator to create a tournament if another tournament with the same name doesn't already exist
R14	The system must allow to set a minimum number of students for each group when creating a new battle
R15	The system must allow to set the maximum number of students for each group when creating a new battle
R16	The system must allow to set a registration deadline when creating a new battle
R17	The system must allow to set a final submission deadline when creating a new battle
R18	The system must allow an educator to choose while creating a new battle which aspects of the quality level of the sources will be automatically evaluated
R19	The system must allow an educator to choose while creating a new battle whether a manual evaluation will be needed after the automated ones
R20	The system must be able to receive a new code kata from an educator when he/she is creating a battle
R21	The system must allow the educators managing a tournament to create new battles in the context of such tournament
R23	The system must notify all students subscribed to that tournament when a new battle of such tournament starts
R30	The system must allow groups to join a newly created battle if they respect that battle's constraints
R31	The system must allow all students enrolled into a battle, to access its code kata
R32	The system must allow a group to submit his/their solution
D1	Users have internet access while using the application
D2	Users have only one account
D3	Users device always send correct data to the application regarding its specifications
D4	The repository service is reliable and works 99.9% of the time
D5	Users have the ability to interact and correctly use the repository platform
D6	Users notification system is reliable and works 99.9% of the time
D7	Users won't block notifications coming from the application

G7	Close an existing coding battle in their tournaments
R36	The system must notify all students subscribed to a tournament when the consolidation phase of a battle of such tournament has ended
R48	The system must allow an educator to close a battle before its expected conclusion
R49	The system must allow an educator to set up a message explaining his/her motivations when closing a battle ahead of time
R50	The system must notify all students subscribed to a tournament when a battle in such tournament is closed ahead of time
R51	The system must show the educator's motivation message when notifying students about the early closure of a battle
D1	Users have internet access while using the application
D6	Users notification system is reliable and works 99.9% of the time
D7	Users won't block notifications coming from the application

G8	Obtain an automated evaluation over a student solution
R18	The system must allow an educator to choose while creating a new battle which aspects of the quality level of the sources will be automatically evaluated
R32	The system must allow a group to submit his/their solution
R37	The system must be able to evaluate functional aspects of a group's delivery
R38	The system must be able to evaluate the timeliness of a group's delivery
R39	The system must be able to evaluate the quality level of the sources of a group's delivery
R40	The system must show the results of its automated analysis to the educators managing the tournament
D1	Users have internet access while using the application
D2	Users have only one account
D3	Users device always send correct data to the application regarding its specifications
D4	The repository service is reliable and works 99.9% of the time
D5	Users have the ability to interact and correctly use the repository platform
D8	The automated evaluation system is reliable and works 99.9% of the time
D9	Users who are members of the same group will contribute equally to the delivery
D10	Users won't receive any external help from a person not in their group while competing in a battle

G9	Manually review students submissions
R3	The system must allow registered users to log-in using their e-mail.
R4	The system must allow logged-in users to use the application.
R5	The system must respect the GDPR
R19	The system must allow an educator to choose while creating a new battle whether a manual evaluation will be needed after the automated ones
R32	The system must allow a group to submit his/their solution
R40	The system must show the results of its automated analysis to the educators managing the tournament
R41	The system must allow an educator to see the solution provided by each group
R42	The system must allow an educator to manually mark a solution in the consolidation phase, if this was set up in the initial configuration of the battle
D1	Users have internet access while using the application
D2	Users have only one account
D3	Users device always send correct data to the application regarding its specifications
D4	The repository service is reliable and works 99.9% of the time
D5	Users have the ability to interact and correctly use the repository platform
D8	The automated evaluation system is reliable and works 99.9% of the time
D9	Users who are members of the same group will contribute equally to the delivery
D10	Users won't receive any external help from a person not in their group while competing in a battle
D11	Educator's manual review of the delivery is unbiased

Use cases diagram


Log in
Sign Up
<<include>>
Create a tournament
<<extend>>
<<include>>
Invite other educators to a tournament
Create a battle
Educator
<<include>>
<<extend>>
<<extend>>
Close a battle early
Educator does a manual code review
Close a tournament

3.2.3 Use cases

Sign up

Use Case	Sign up
Actor	User
Entry condition	User wants to register in the system
Flow of events	1. User accesses the application 2. User presses the sign-up button 3. User inserts his email and password 4. User presses the confirmation button 5. System checks that the e-mail has not been used before and that the password satisfy the security constraints 6. System displays a confirmation message 7. System sends a notification to User to confirm the successful registration
Exit condition	User data are saved into the system and the registration ends successfully
Exceptions	Possible exceptions: 1. User email has already been taken 2. User presses the cancel button after having pressed the sign-up button 3. The application is currently unavailable If one of these exceptions takes place, an error message is shown and the flow of events starts again from point 3

Log in

Use Case	Log in
Actor	User
Entry condition	User wants to log into the application
Flow of events	1. User accesses the application 2. User presses the login button 3. User enters his email and password 4. User presses the confirmation button 5. System displays the main page
Exit condition	The system allows the user's to access the desired account and the main page is displayed
Exceptions	Possible exceptions: 1. The inserted email does not correspond to any account 2. The password for the given email is incorrect An error message is shown and the flow of events starts again from point 3

Create a tournament

Use Case	Create a tournament
Actor	Educator
Entry condition	Educator wants to create a new tournament
Flow of events	1. Educator accesses the application 2. Educator logs-in 3. Educator navigates to the tournament section 4. Educator clicks the "Create a new Tournament" button 5. Educator inserts the name of the new tournament 6. Educator inserts the deadline of the new tournament 7. Educator press the "Create tournament" button 8. System checks if a tournament with the same name does already exist 9. System sends a confirmation notification of tournament creation to Educator 10. System sends a notification that a tournament with that name has been created to all CodeKata's users
Exit condition	A new tournament with the required name has been successfully created and all users of the platform have notified about it
Exceptions	Possible exceptions: 1. A tournament with the same name does already exist The system shows an error message and the flow of events starts again from point 5

Invite other educators to manage a tournament

Use Case	Invite other educators to manage a tournament
Actors	• Tournament Owner • Educator(s)
Entry condition	Tournament Owner is logged in and wants to invite one or more educator(s) to manage one of his/her tournaments
Flow of events	1. Tournament Owner clicks on "Tournament" page 2. Tournament Owner clicks on his/her tournament to which he/she wants to invite the educator(s) to 3. Tournament Owner clicks on "Manage educators" button 4. Tournament Owner clicks on the "+" button 5. Tournament Owner writes the name of the educator(s) that he/she wants to invite, after writing each of them he/she press the add button 6. Tournament Owner clicks the "Invite educators" button to confirm his choices 7. System notifies the educators previously selected about the fact that they have been invited to manage a tournament
Exit condition	The educator(s) have been correctly invited to manage the tournament and they have received a notification about it
Exceptions	Possible exceptions: 1. A written educator name does not exist The system shows an error message and the flow of events starts again from point 5

Create a battle

Use Case	Create a battle
Actor	Educator
Entry condition	An educator managing a tournament is logged in and wants to create a battle in such tournament
Flow of events	1. Educator navigates to the "Tournament" section 2. Educator selects the tournament where he/she wants to create a new battle in 3. Educator clicks on the "New Battle" button 4. Educator clicks on "Upload" button in order to upload the code kata 5. Educator uploads the corresponding CodeKata 6. Educator fills up, using the provided fields, all the required information to create a new battle, which are: • minimum number of students per group • maximum number of students per group • registration deadline • final submission deadline • which aspects of the quality level of the sources should be automatically evaluated • whether a manual evaluation will be needed after the automated one 7. Educator clicks on "Create new battle" button 8. System notifies all students subscribed to the tournament that a battle has been created
Exit condition	A new battle has been created with the desired configuration and all students subscribed to its tournament have been notified about it
Exceptions	Possible exceptions: 1. Some configuration fields are left empty 2. Some configuration fields are filled with incorrect values (for example a negative value for the maximum number of students per group) The system shows an error message and the flow of events starts again from point 6

Subscribe to a tournament

Use Case	Subscribe to a tournament
Actor	User
Entry condition	A logged-in user wants to subscribe to a tournament
Flow of events	1. User navigates to the tournament section 2. User clicks on the tournament name of the tournament to which he/she wants to subscribe to 3. User clicks on the "Subscribe" button
Exit condition	The user is subscribed to the tournament he/she wanted to, and now he/she will be informed about every new battle created in that tournament

Create Group

Use Case	Create Group
Actor	Student
Entry condition	A logged in student wants to create a group in the context of a battle to which he/she is enrolled
Flow of events	1. Student navigates to the tournament section 2. Student clicks on the tournament name in which he/she wants to create a group 3. Student clicks on the "Create a group" button 4. System checks if there is an ongoing battle and if the user is enrolled into such battle 5. Student inserts in the corresponding field the name of the new group 6. Student clicks on the "Confirm Creation" button 7. System check if the name of the group is already used in the context of the same battle 8. System shows a confirmation message about the successful group creation
Exit condition	The new group has been created with the corresponding name
Exceptions	Possible exceptions: 1. In this tournament there is no current ongoing battle 2. There already exist a group with the same name in the context of that battle The system shows an error message and the flow of events starts again from point 2 if exception 1 has occurred and from point 5 if exception 2 has occurred

Invite a student to an existing group

Use Case	Invite a student to an existing group
Actors	• Group Student • New Student
Entry condition	A logged in student which has already created a group for a tournament he/she is subscribed to (Group Student), wants to invite another student which is also subscribed to such tournament to his/her group (New Student)
Flow of events	1. Group Student navigates to the tournament section 2. Group Student clicks on the tournament name in which such group has been created 3. Group Student clicks on the "Manage my group" button 4. Group Student clicks on the "Invite new student" button 5. Group Student write the nickname or the email of New Student 6. Group Student clicks the "Invite" button 7. System sends a notification to New Student about his invitation to such group
Exit condition	New Student is successfully invited to the existing group and has been notified about it
Exceptions	Possible exceptions: 1. The nickname/email written doesn't correspond to any user 2. The invited user is not subscribed to such tournament The system shows an error message and the flow of events starts again from point 4

Enroll in a battle

Use Case	Enroll in a battle
Actor	Student
Entry condition	A logged in student wants to enroll in a battle
Flow of events	1. Student navigates to the tournament section 2. Student clicks on the tournament name of the tournament in which there is the battle to which he/she wants to enroll into 3. Student clicks on the current ongoing battle 4. Student clicks on the "Enroll into battle" button 5. System checks that the group of the student satisfy all the constraints required by the specific battle 6. System shows a confirmation message about the successful enrollment into that battle
Exit condition	Student and possibly his/her group are enrolled into the battle
Exceptions	Possible exceptions: 1. The student's group doesn't satisfy that battle's constraints The system shows an error message and the flow of events starts again from point 3

Student delivers a solution

Use Case	Student delivers a solution
Actor	Student
Entry condition	A logged in student wants to deliver their shared solution or his/her personal solution of a certain active battle
Flow of events	1. Student set up an automated procedure, using the repository system that informs the CKB platform when a push action occurs 2. Student commits the change that he/she has made to the source code 3. Student push their shared solution to the repository
Exit condition	The solution made by such group has been delivered and CKB has been informed about this delivery

Educator does a manual code review

Use Case	Educator does a manual code review
Actor	Educator
Entry condition	A battle has ended and the educator when creating this battle set up that a manual evaluation was needed after the automated ones
Flow of events	1. Educator navigates to the tournament section 2. Educator selects the tournament in which the battle awaiting a manual delivery has been created 3. Educator selects the battle requiring a manual evaluation 4. Educator clicks on the "Review submissions" button 5. The system presents to the educator the first delivery that has been submitted, by alphabetical order of the name of the group that submitted it along with the result of its automated evaluations 6. Educator read the whole code and gives his/her personal evaluation to the delivery 7. Educator presses the "Next One" button
Exit condition	The delivery has received a manual evaluation
Exceptions	Possible exceptions: 1. Educator presses the "Next one" button before having given his/her personal evaluation The system shows an error message and the flow of events starts again from point 4

Close a battle early

Use Case	Close a battle early
Actor	Educator
Entry condition	A logged in educator wants to close one of his/her battles before its expected conclusion
Flow of events	1. Educator navigates to the tournament section 2. Educator selects the tournament in which he has previously created such battle 3. Educator navigates to the current battle 4. Educator clicks on the "Delete battle" button 5. Educator clicks on the "Yes, I'm sure" button 6. System closes the corresponding battle
Exit condition	The battle has been successfully removed early

Close a tournament

Use Case	Close a tournament
Actor	Educator
Entry condition	A logged in educator wants to close one of his/her tournament
Flow of events	1. Educator navigates to the tournament section 2. Educator selects the tournament he/she wants to close 3. Educator clicks on the "Close tournament" button 4. Educator clicks on the "Yes, I'm sure" button 5. System closes the corresponding tournament
Exit condition	The educator's tournament has been closed
Exceptions	Possible exceptions: 1. The tournament has an ongoing battle The system shows an error message and the flow of events starts again from point 2

Traceability Matrix

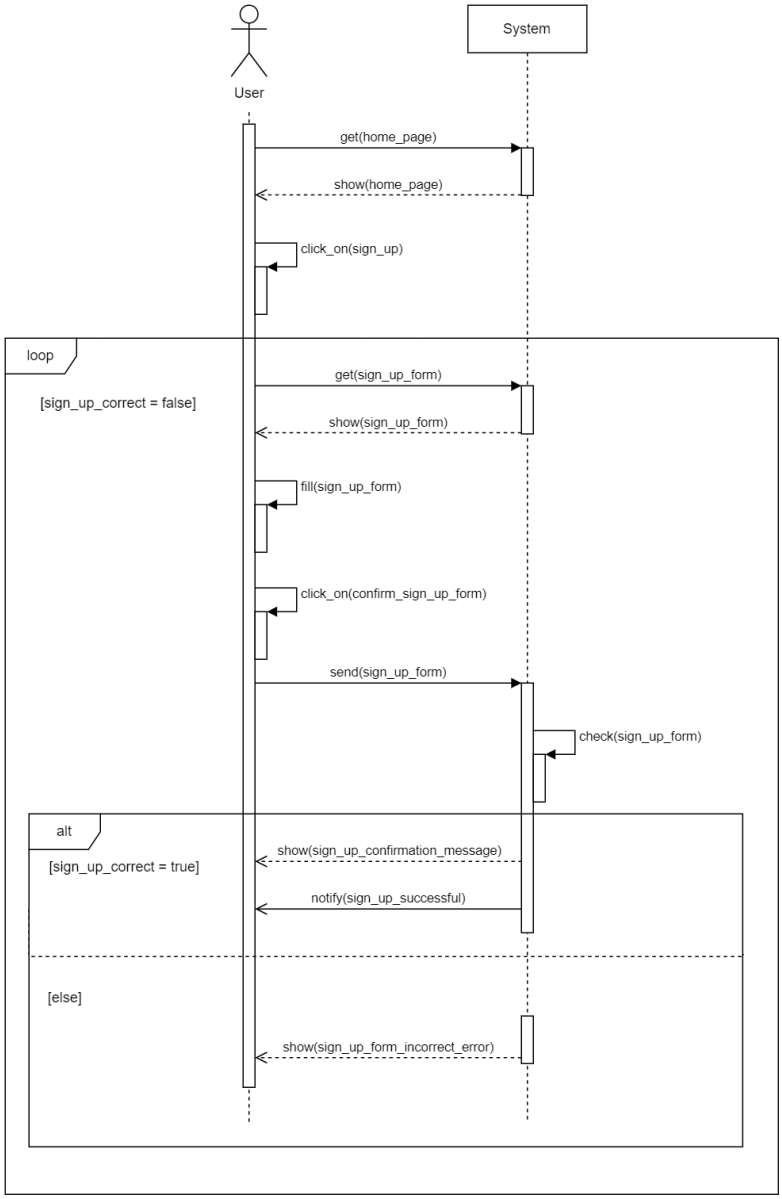
Use case	Requirement ID
Sign up	**R1:** The system must allow the registration of new users using their e-mail **R2:** The system must notify a user about his/her successful registration **R5:** The system must respect the GDPR
Log in	**R3:** The system must allow registered users to log-in using their e-mail **R4:** The system must allow logged-in users to use the application
Create a tournament	**R8:** The system must allow an educator to create a tournament if another tournament with the same name doesn't already exist **R9:** The system must reject an educator request to create a tournament if another tournament with the same name already exist **R13:** The system must allow an educator to set a registration deadline when creating a tournament
Invite other educators to manage a tournament	**R10:** The system must allow the owner of a tournament to invite other educators to manage it **R11:** The system must notify an educator when he/she is invited to manage a tournament **R12:** The system must allow an educator to accept the invitation of another educator to manage a tournament
Create a battle	**R14:** The system must allow to set a minimum number of students for each group when creating a new battle **R15:** The system must allow to set the maximum number of students for each group when creating a new battle **R16:** The system must allow to set a registration deadline when creating a new battle **R17:** The system must allow to set a final submission deadline when creating a new battle **R18:** The system must allow an educator to choose while creating a new battle which aspects of the quality level of the sources will be automatically evaluated **R19:** The system must allow an educator to choose while creating a new battle whether a manual evaluation will be needed after the automated ones **R20:** The system must be able to receive a new code kata from an educator when he/she is creating a battle **R23:** The system must notify all students subscribed to that tournament when a new battle of such tournament starts

Use case	Requirement ID
Subscribe to a tournament	**R24:** The system must allow users to subscribe to a tournament
Create group	**R25:** The system must allow students to create groups in the context of a battle
Enroll in a battle	**R30:** The system must allow groups to join a newly created battle if they respect that battle's constraints
Invite a student to an existing group	**R26:** The system must allow a student to invite another to his/her own group, if such student is enrolled into the corresponding battle **R27:** The system must reject a user request to invite another student to his/her own group, if such student is not enrolled into the corresponding battle **R28:** The system must notify a student when he/she is invited to a group **R29:** The system must allow a student to join an existing group if invited
Student delivers a solution	**R32:** The system must allow a group to submit his/their solution
Educator does a manual code review	**R40:** The system must show the results of its automated analysis to the educators managing the tournament **R41:** The system must allow an educator to see the solution provided by each group **R42:** The system must allow an educator to manually mark a solution in the consolidation phase, if this was set up in the initial configuration of the battle

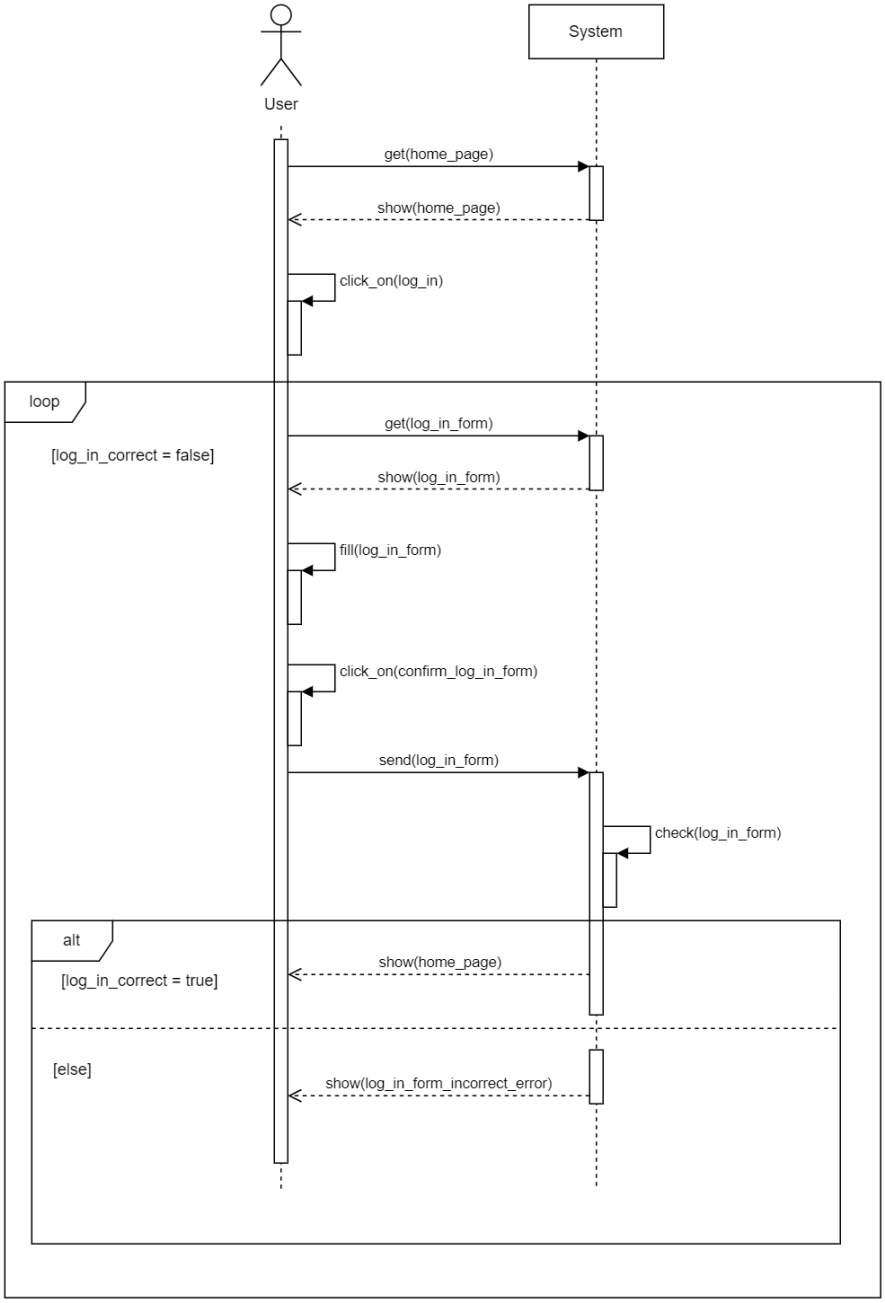
Use case	Requirement ID
Close a battle early	**R48:** The system must allow an educator to close a battle before its expected conclusion **R49:** The system must allow an educator to set up a message explaining his/her motivations when closing a battle ahead of time **R50:** The system must notify all users subscribed to its tournament when a battle is closed ahead of time **R51:** The system must show the educator's motivation message when notifying students about the early closure of a battle
Close a tournament	**R52:** The system must allow an educator to close a tournament, if there is not an ongoing battle in such tournament **R53:** The system must reject an educator request to close a tournament, if there is an ongoing battle in such tournament **R54:** The system must notify all students subscribed to a tournament when such tournament is closed

3.2.4 Sequence Diagrams

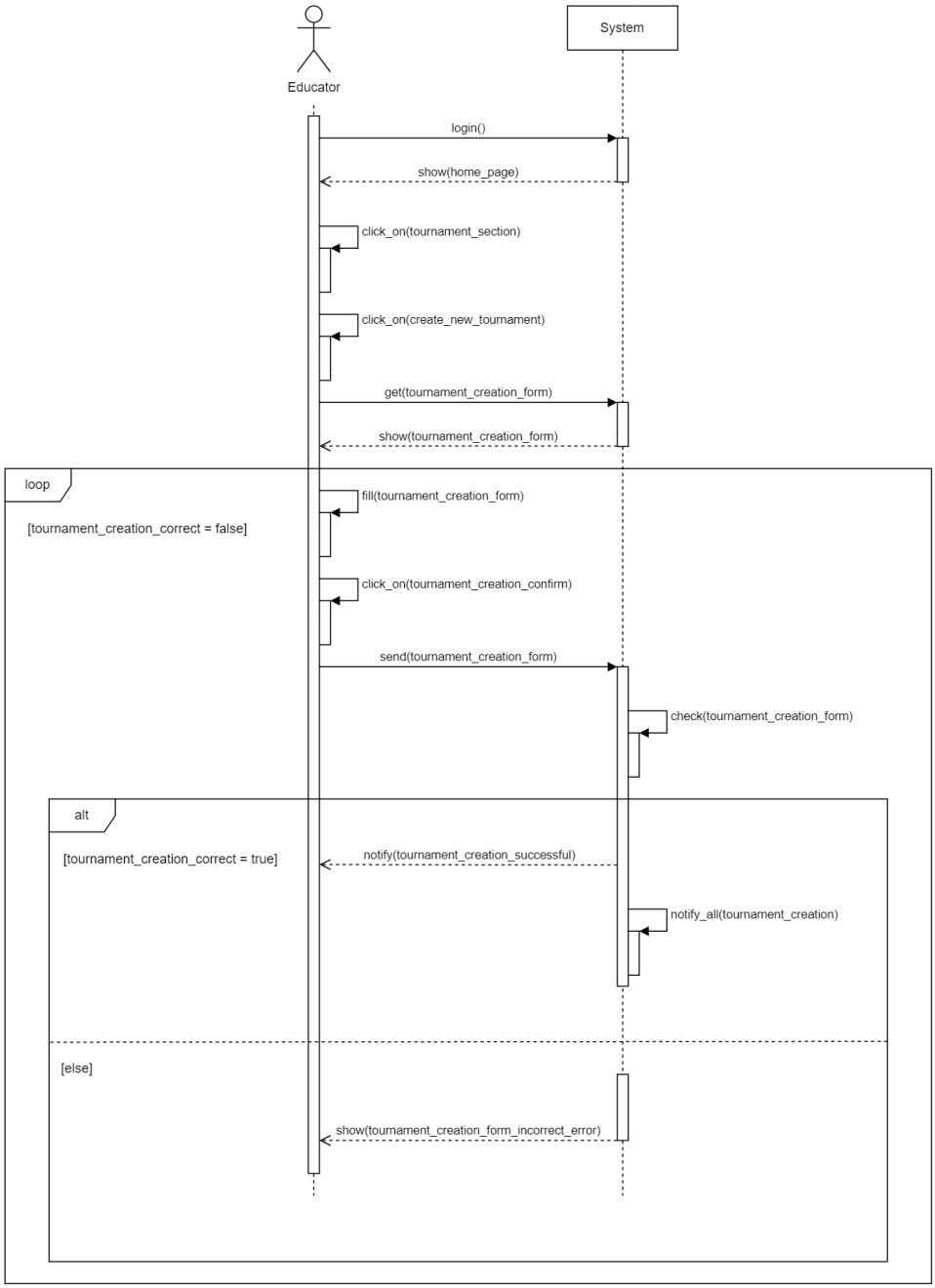
Sign up



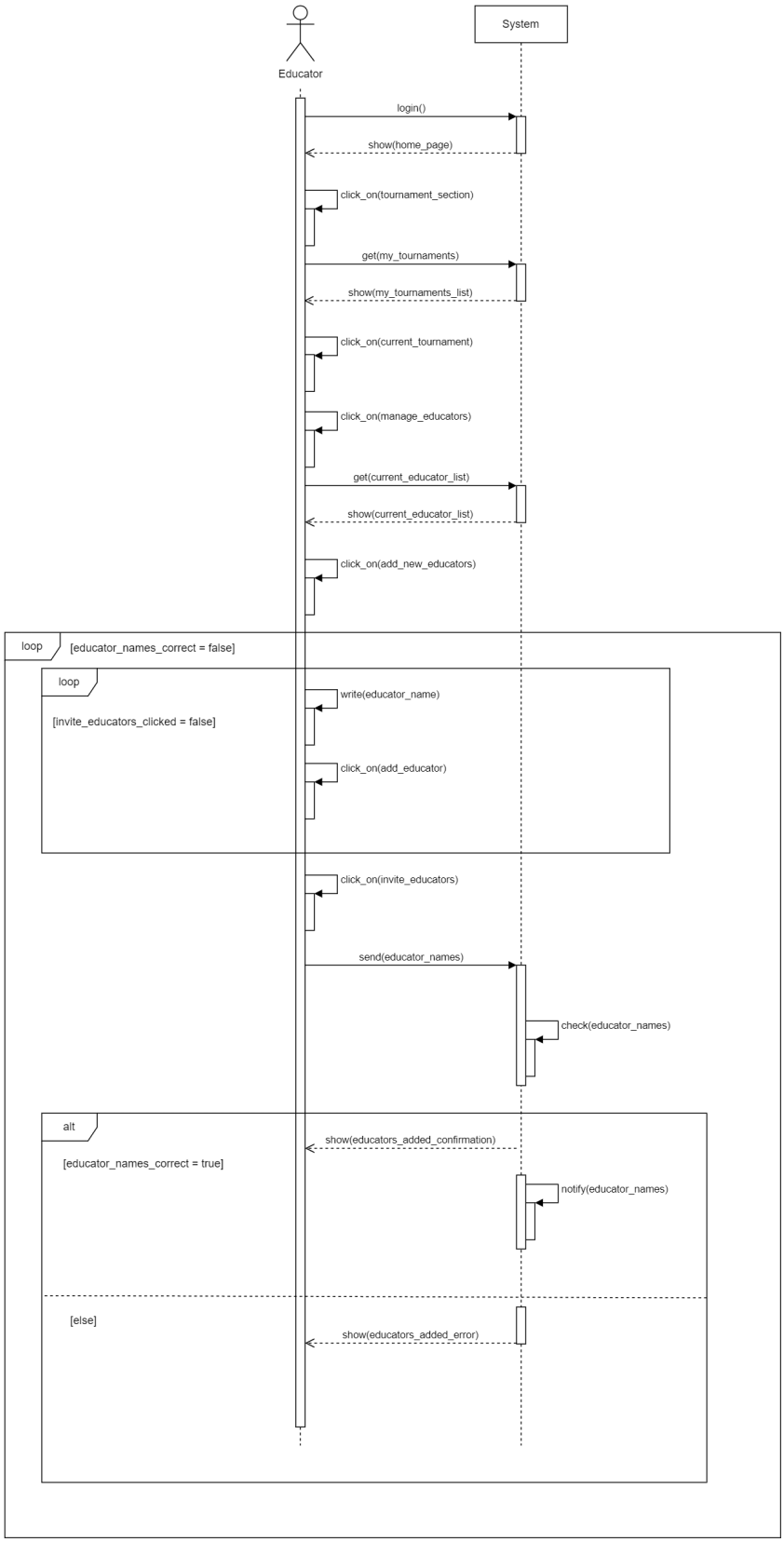
Log in


System
User
get(home_page)
show(home_page)
click_on(log_in)
loop
[log_in_correct = false]
get(log_in_form)
show(log_in_form)
fill(log_in_form)
click_on(confirm_log_in_form)
send(log_in_form)
check(log_in_form)
alt
[log_in_correct = true]
show(home_page)
[else]
show(log_in_form_incorrect_error)

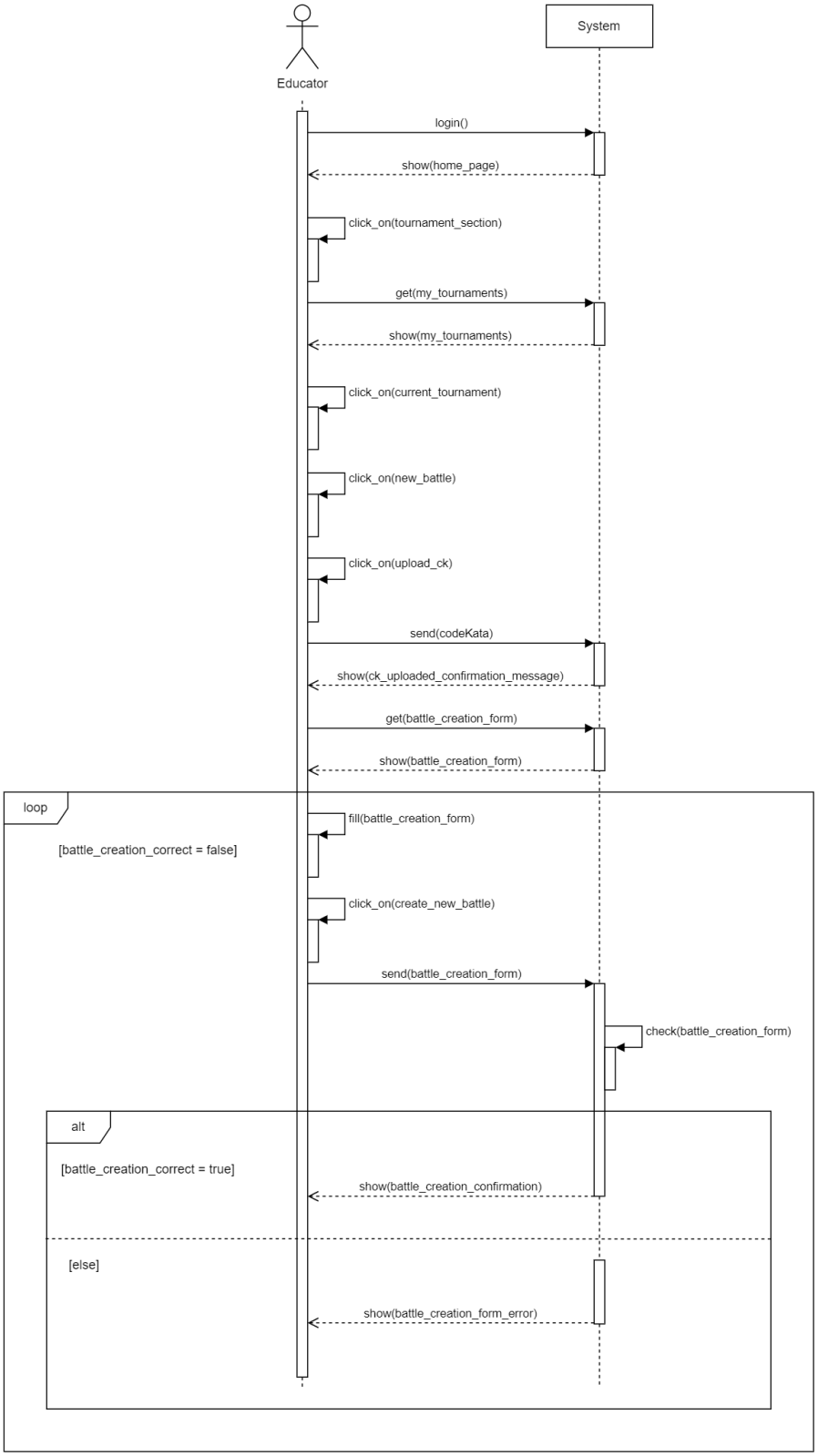
Create a tournament



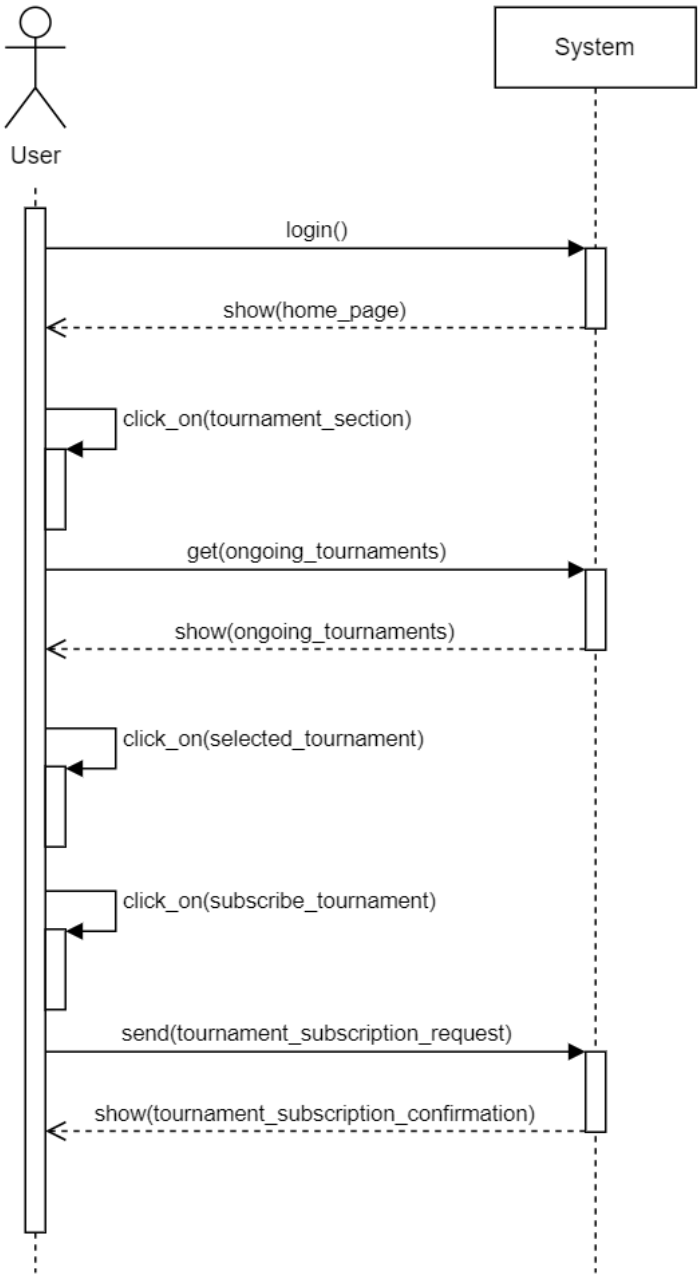
Invite other educators to mange a tournament



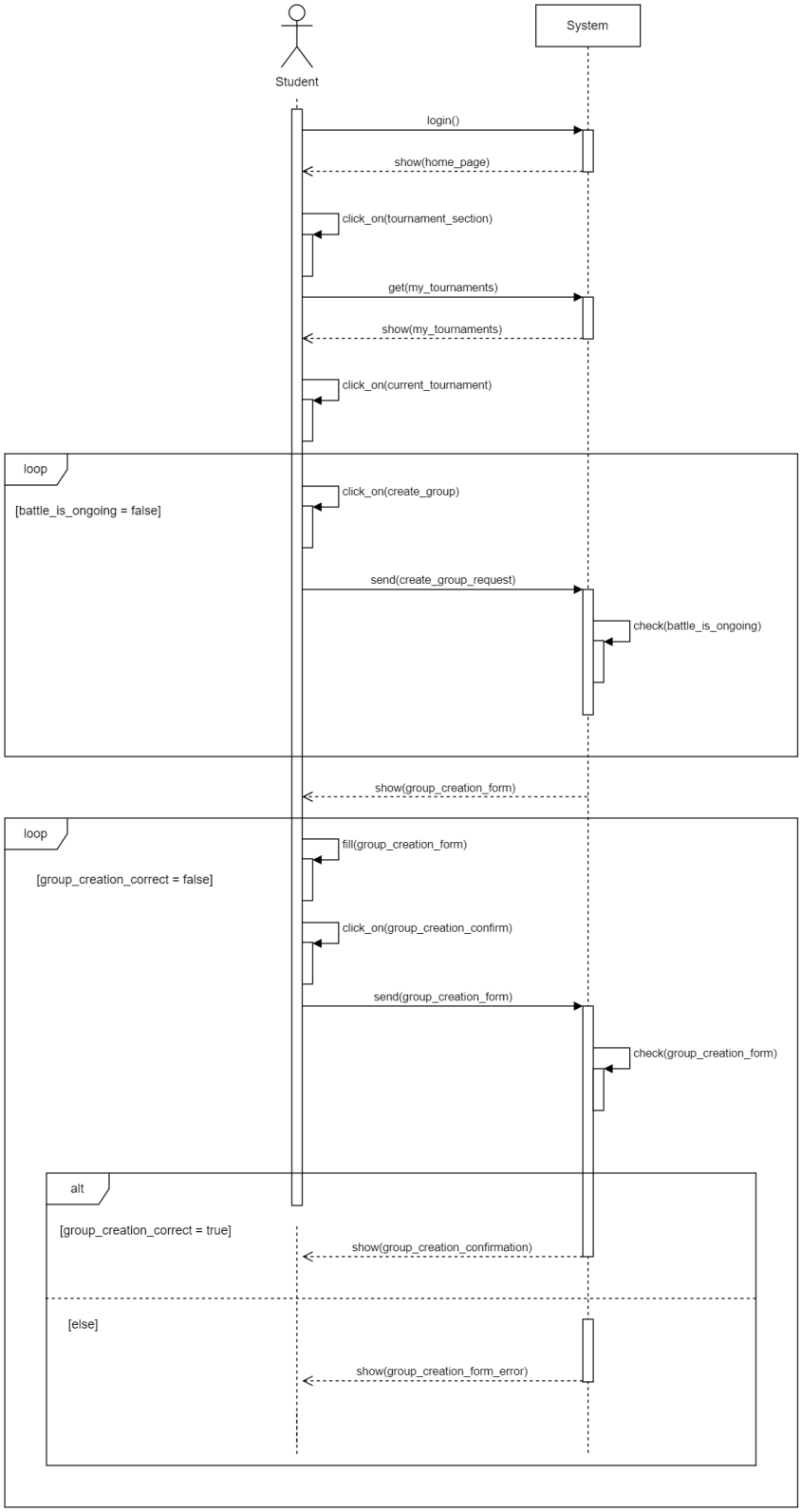
Create a battle



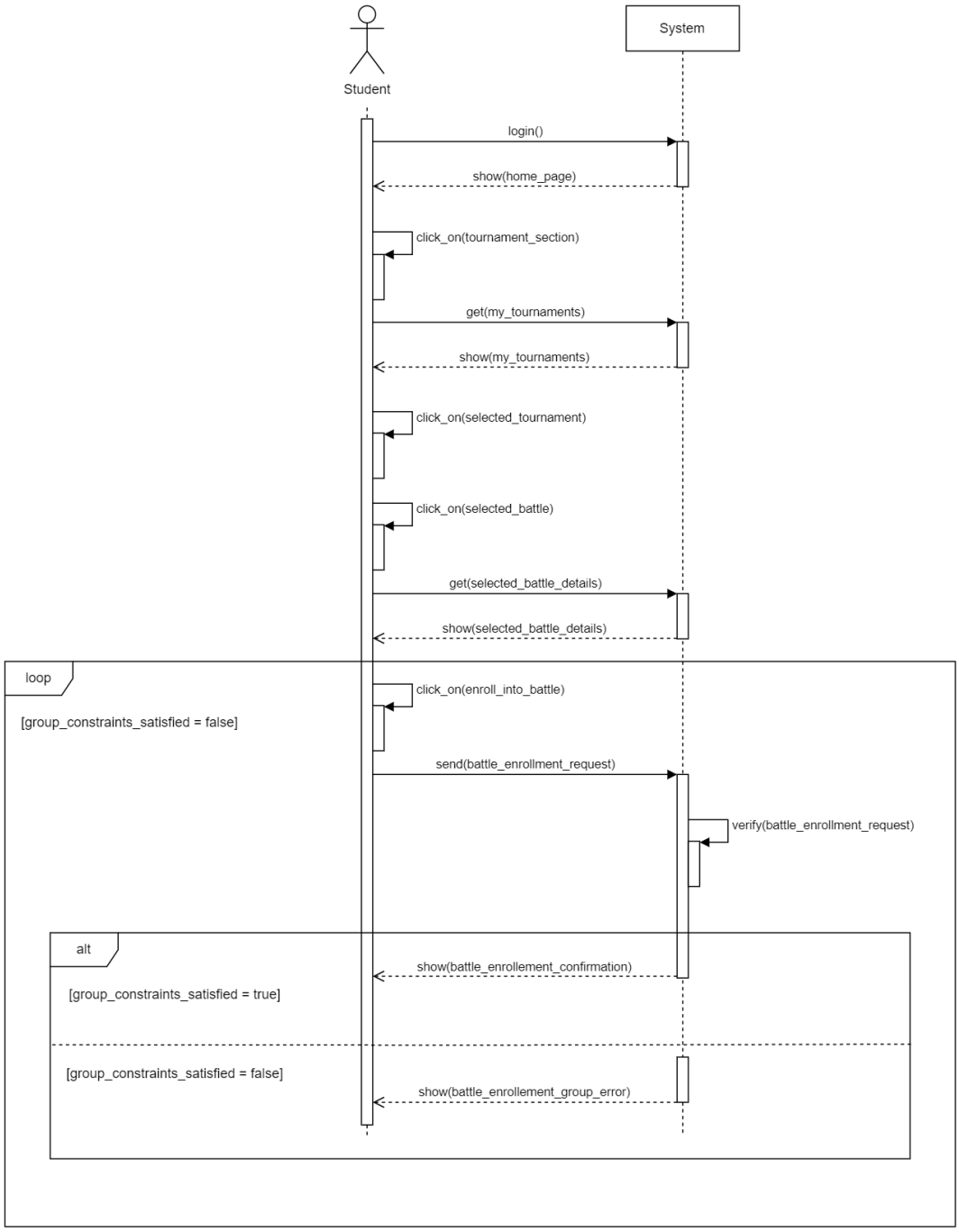
Subscribe to a tournament



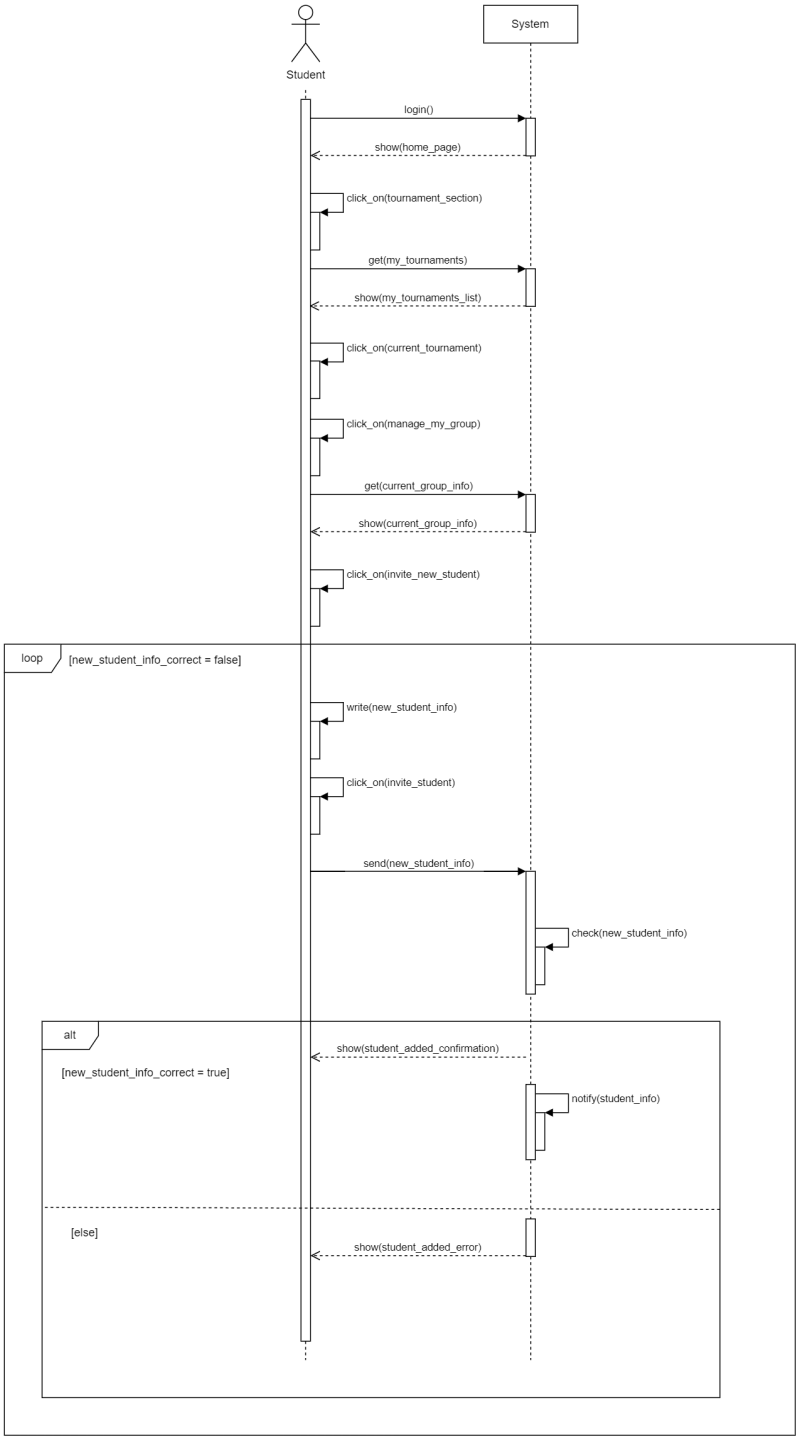
Create a group



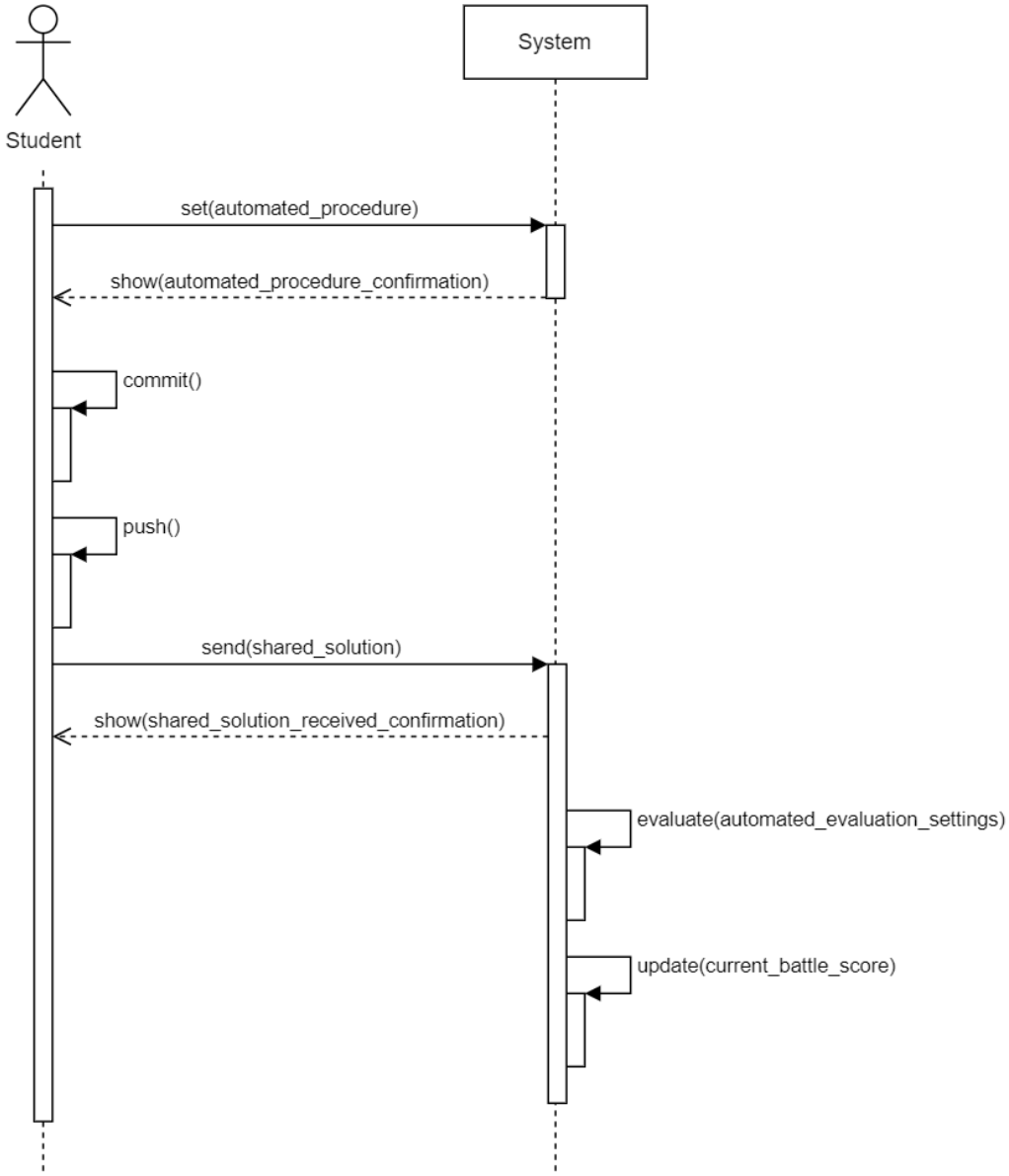
Enroll in a battle



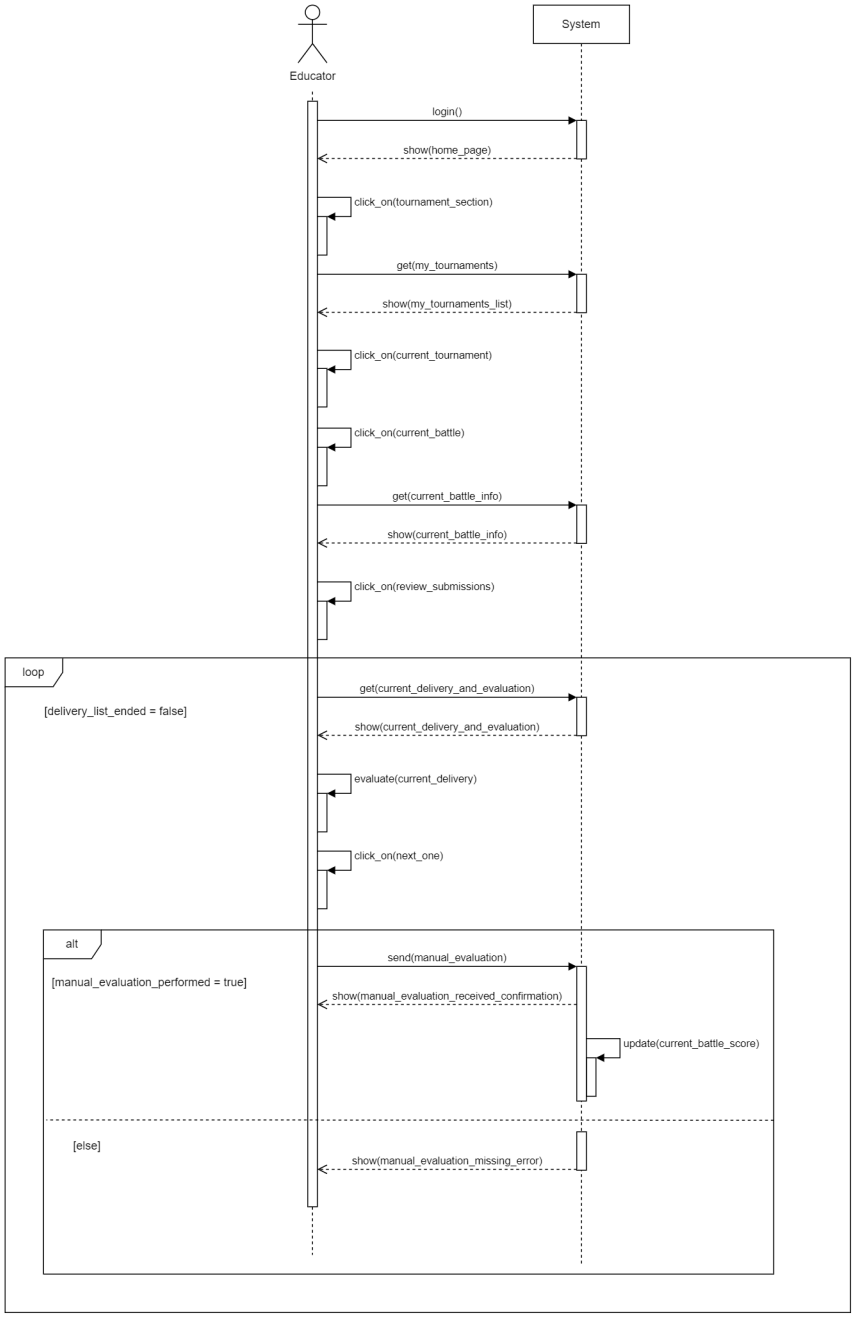
Invite a student to an existing group



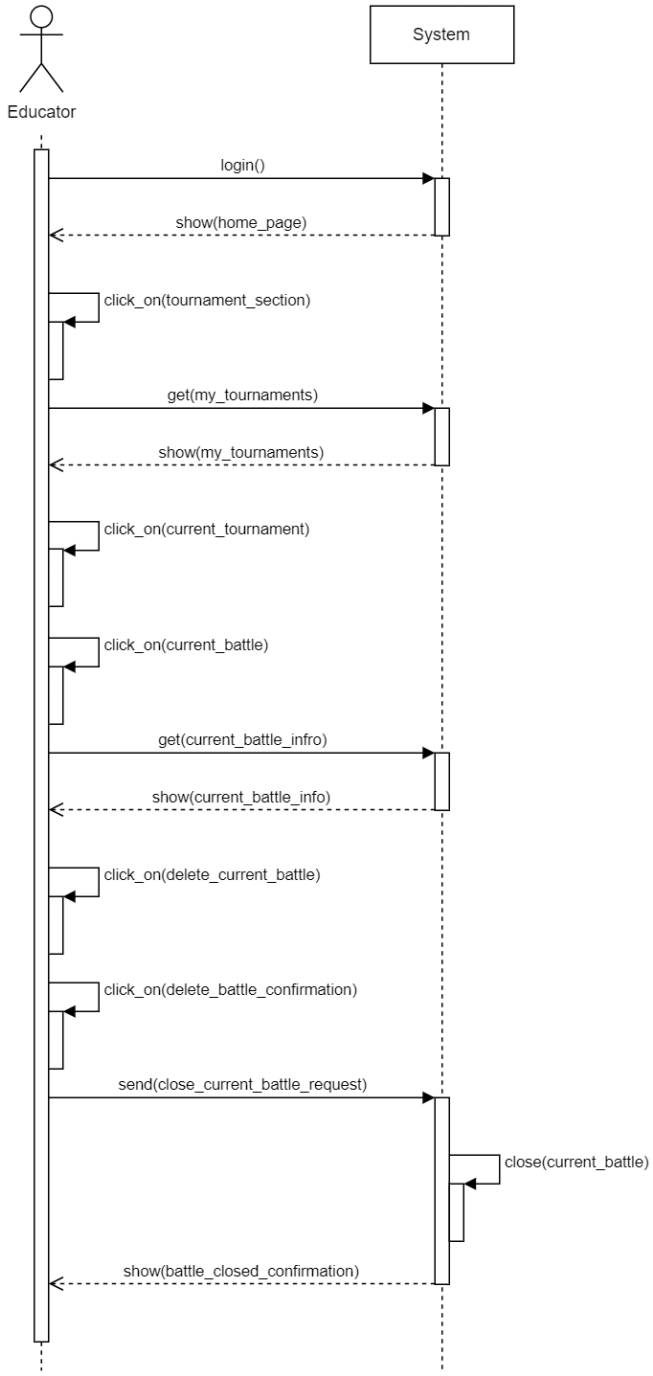
Student delivers a solution



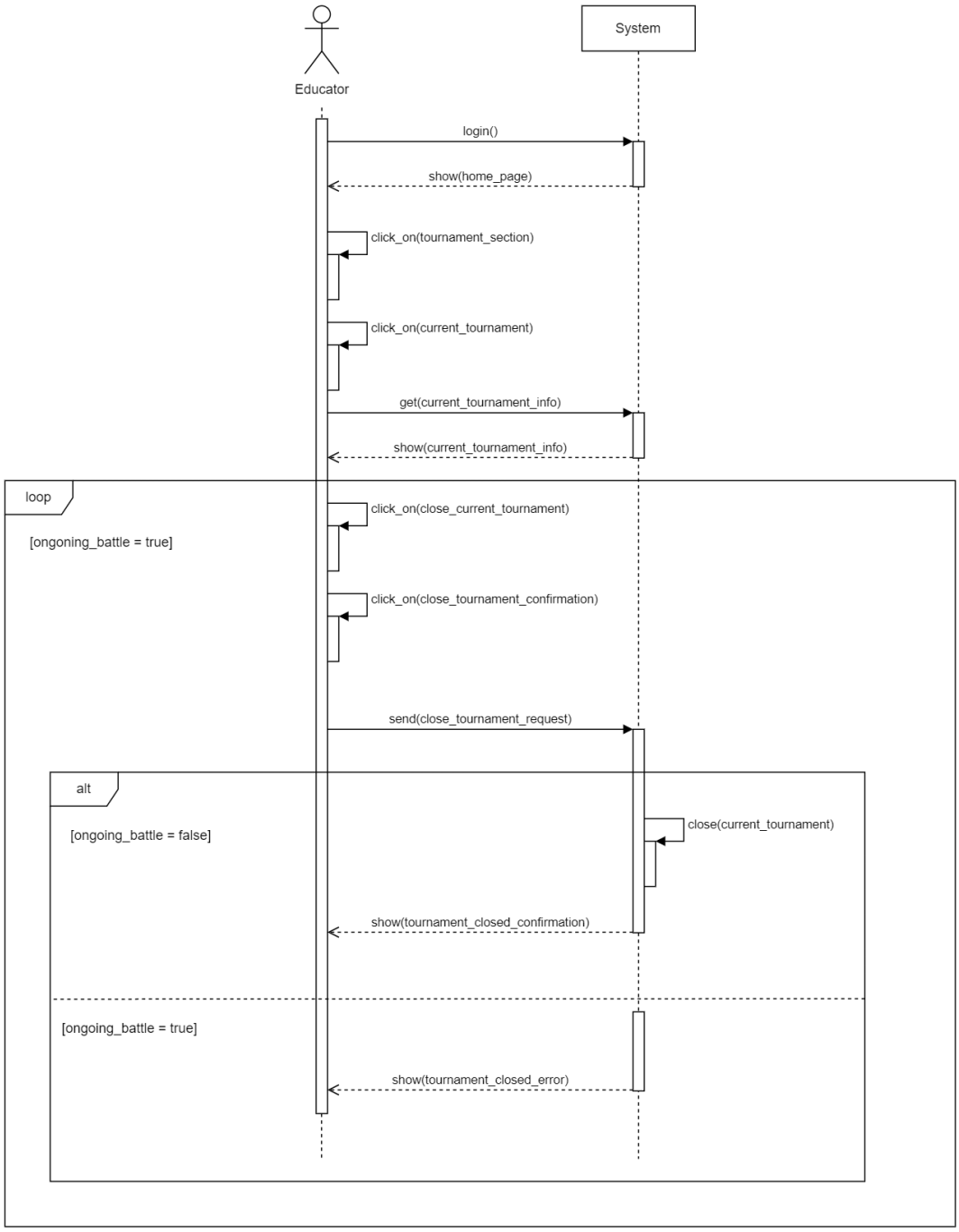
Educator does a manual code review



Close a battle early



Close a tournament



3.3 Performance Requirements

Given the fact that in each of our battles it will be mandatory for the system to evaluate the timeliness of the delivery, it is of paramount importance for us to guarantee that as less time as possible is lost on our side to evaluate the solutions and to send back our results, (consisting in the new student's rank and battle score) to give each student participating in the battle as much time as possible to fix his/her mistakes and eventually submit a new solution. Unfortunately, we cannot interfere with each student's own transmission speed, so we'll only need to focus on how our system will behave once that new delivery becomes readable, while keeping also in mind that faster equipment will also be more expensive, so we have to balance those two contrasting needs.
Therefore, we should be able to check whether a new solution has been delivered at least once every second, and have a fast computational speed to be able to evaluate a solution of average length, in a situation of average workload in at most 3 seconds.
Finally, estimating the number of concurrent active user that we should expect is a hard question, since we have very few data regarding this topic, but we can see that a website with a similar structure called `www.codewars.com` has on average 2.4M access each month[4], so we can assume for it to have an average of 833 concurrent user active at the same time[5], therefore given the minor popularity that our system will have initially but also to deal with peak hours the system is required to be able to handle up to 1300 concurrent active users

[5]to calculate this approximate value we have assumed the user session on our website to last on average 15 minutes and the access to be uniformly spread over the course of the whole day and month, therefore we have assumed that every access made in the same quarter of hour will be considered concurrent: $\text{concurrent\_acceses} = \frac{\text{monthly\_acces}}{30*24*4}$

3.4 Design Constraints

3.4.1 Standards compliance

CodeKataBattle's user data must be compliant with the European Union's General Data Protection Regulation (GDPR), a European Union regulation on information privacy, therefore the system must be able to store and process user data securely, and must be able to provide the user with a way to access, modify or delete his/her own data at any time.

3.4.2 Hardware limitations

Given the nature of the software, not much computational power is needed on the user-side, so any fairly powered machine would probably be enough for our users to effectively use the software. On the other hand an internet connection is required, in order to communicate and exchange data with the platform, and this internet connection is also strongly suggested to be fast and stable in order to grant the user a fair competition with his/her opponents (for example to avoid delays while reading the code kata, sending the delivery or receiving his/her updated battle score)

3.5 Software System Attributes

3.5.1 Reliability

The system must have a backup infrastructure to ensure that the data is always available and that the system can be restored in case of a failure. The components of this infrastructure must be characterized by a high MTTF (Mean Time To Failure) and a low MTTR (Mean Time To Repair), in order to both reduce the amount of failures to a minimum and to be able to quickly recover when those failures actually happen. Moreover, all the needed components must be redundant and parallelized as much as possible, to ensure a high reliability of the overall system as well as fault-resilience in the case of a single component failure

3.5.2 Availability

CodeKataBattle is not an emergency service or anything related to critical situations, so it is considered enough for the system to provide an availability of 99.9%, this means an average downtime of 8.77 hours per year, or 10.08 minutes a week, which is perfectly fine for our uses.
Also, since CKB is expected to be used also by teacher during exercise classes and by student/workers in days when they are not going to school or work, the peak hours are expected to be from 9 to 12 during working days and from 15 to 19 during weekends, so the system should be able to handle a higher than average workload during these time-slots. For this reason all the maintenance activities should be done at night from Monday to Thursday when the system is less used.

3.5.3 Security

The system store some sensitive personal information about its users, so the security aspect cannot be under-estimated, the data stored by the system must be protected and its communication with other components must be encrypted, to avoid any unauthorized interception.
Therefore, all the user data must be stored securely, hashed and/or encrypted using approved algorithms that have not been cryptographically broken, like SHA-256 for hashing or AES-256 for encryption. Specifically, passwords should be stored only in their hashed version and subsequently the whole data stored regarding a user (including the hashed password) should be encrypted.
Furthermore, the communications between the server and any other component or external entity should be encrypted using TLS 1.3 and powered by HTTPS, to grant confidentiality, integrity, and authenticity to such communications.
Finally, all stored data must be compliant with the GDPR.

3.5.4 Maintainability

The system must be easy to maintain and update, and must be able to be extended with new features in the future with ease. For this reason the system must be designed using a modular architecture, and must be developed using modern design patterns and best practices guaranteeing high reusability of its code, as well as future extensions of its features. A documentation for such code must also be produced which should be clear, complete and easy to understand. Finally, the system should also be tested using unit tests and integration tests covering all its modules before its release, to assert that no errors or conflicts are present

3.5.5 Portability

The system should be designed to be easily ported into many platforms, providing equal functionalities in each of these versions, for example the system could be designed to work with most modern web browsers (like Google Chrome, Microsoft Edge, Mozilla Firefox, Opera or Safari, as well as possibly other Chromium based web browsers) and/or most of the modern desktop and mobile operating systems (like Windows, macOS, Linux, Android or iOS).

4 Formal Analysis using Alloy

The analysis using Alloy tool aims to formally prove the correctness of the model previously described. To achieve so, a test to assert the satisfaction of the goals has been made. In particular the following goals have been tested:

- **(G1) Have his/her own profile on the application**: for this goal an assertion called *DifferentEmailOrNicknameImpliesDifferentUsers* checks that no user have same e-mail or nickname
- **(G2) Participate in a coding tournament**: to satisfy this goal an assertion called *StudentsSubscribedToBattleAlsoSubscribedToTournament* verifies that only a user that is subscribed to the tournament where the battle is being hosted can enroll to that battle. The predicate *StudentParticipatesInTournament* shows that some students can subscribe to a tournament where a battle is being hosted.
- **(G3) Cooperate with colleagues**: to fulfill this goal some assertions have been made:
  - *StudentInGroupAlsoInBattle* checks that a student must be enrolled in a battle to be part of a group
  - *NoLoneSolution* checks that a solution must be submitted by a group that is participating to such a battle
  - *GroupSizeIsCorrect* checks that every group is in the size constrain limit of a battle

  Furthermore, a predicate called *StudentCollaborateWithColleagues* shows a student collaborating with a group and the relative solution
- **(G4) Review their own performance score**: using the predicate *showStudentSolutions* the model shows all the solutions published by a student in the context of a tournament. Given a set of solution is then possible to calculate, using the desired performance metrics, the score of a student(in the context of such a tournament)
- **(G5) Organize coding tournaments**: for this goal an assertion called *NoEducatorWithoutTournament* checks that all educators are linked to a tournament. The predicate *showEducatorsInTournament* instead shows some educators administering a tournament
- **(G6) Run coding battles in their tournaments**: to satisfy this goal an assertion called *BattleOwnerIsEducatorOfTournament* verifies that all the battles inside a tournament have been created by an educator of such a tournament

- **(G7) Close an existing coding battle in their tournaments**: using the predicate *showClosedBattle* the model shows an instance were a battle has been closed
- **(G8 & G9) Obtain an automated evaluation over a student solution & Manually review students submissions**: to fulfill these goals an assertion called *CorrectSolutionScore* checks that all the solutions have an automated score and that the manual score is set if and only if it was needed(manual review score was enabled in the settings of a battle)

4.1 Alloy Code

```
module projectSE2
open util/boolean

sig Email {}
sig TestCase {}
sig BuildAutomationScript {}
sig string {}

sig User {
    email: one Email,
    password: one string,
    name: one string,
    surname: one string,
    nickname: lone string
}

sig Student extends User {
    var tournaments: some Tournament
}

sig Educator extends User {
    var tournamentsOwned: set Tournament,
    var tournamentsModerator: set Tournament
}{
    #tournamentsOwned + #tournamentsModerator > 0
}

sig Battle {
    name: one string,
    description: lone string,
    owner: one Educator,
    var enrolledStudentList: set Student,
    codeKata: one CodeKata,
    teamMinimumSize: one Int,
    teamMaximumSize: one Int,
    var groupList: set Group,
    battleEvent: one Event,
    var solutionList: set Solution,
    manualReview: one Bool
}{
    teamMinimumSize ≤ teamMaximumSize
    teamMinimumSize ≥ 1
    #enrolledStudentList ≤ #enrolledStudentList'
    #groupList ≤ #groupList'
    #solutionList ≤ #solutionList'
}

sig Tournament {
    name: one string,
    owner: one Educator,
    var administratorList: set Educator,
    var subscribedStudentList: set Student,
    var battleList: set Battle,
    tournamentEvent: one Event
}{
    owner in administratorList
    #administratorList > 0
    #subscribedStudentList ≤ #subscribedStudentList'

}
```

```
sig Solution {
    groupID: one Int,
    automatedScore: one Int,
    manualReviewScore: lone Int
}{
    automatedScore ≥ 0 and automatedScore ≤ 7 // should be 100, limited by alloy
}

sig CodeKata {
    description: one string,
    testCases: some TestCase,
    buildAutomationScript: some BuildAutomationScript
}

sig Group {
    name: one string,
    groupID: one Int,
    var studentList: some Student,
    var size: one Int
}{
    size = #studentList
}

sig Event {
    startDate: one Int,
    endDate: one Int,
}{
    startDate < endDate
}


// ------------------------ MODEL CONSTRAINTS ------------------------

// all user have a unique email address and a unique nickname if set
fact {
    always (all disj u1, u2: User | u1.email ≠ u2.email and
    (u1.nickname ≠ none implies u1.nickname ≠ u2.nickname))
}

// every Email is associated to one User
fact {
    always (all e: Email | one u: User | u.email = e)
}

// students and tournaments are associated to each other
fact {
    always (all s: Student, t: Tournament | s in t.subscribedStudentList iff t in s.
        ↪ tournaments)
}

// educators and tournaments are associated to each other
fact {
    always (all t: Tournament, e: Educator | t.owner = e iff t in e.tournamentsOwned
        ↪ )
    always (all t: Tournament, e: Educator | e in t.administratorList iff (t in e.
        ↪ tournamentsModerator or t.owner = e))
}

// all battles are associated to a tournament
fact {
    always (all b: Battle | one t: Tournament | b in t.battleList)
}
```

```
// all battles in a tournament must have a unique name
fact {
    always (all t: Tournament | all disj b1, b2: t.battleList |  b1.name ≠ b2.name)
}

// all battles' owner must be an educator of the tournament
fact {
    always (all t: Tournament | all b: t.battleList | b.owner in t.administratorList
        ↪ )
}

// all groups in the battle must have the cardinality between teamMinimumSize and
    ↪ teamMaximumSize
fact {
    always (all b: Battle | all g: b.groupList | g.size ≥ b.teamMinimumSize ∧ g.size
        ↪  ≤ b.teamMaximumSize)
}

// all enrolled students in the battle must be also subscribed to the tournament
fact {
    always (all t: Tournament | all b: t.battleList | all s: b.enrolledStudentList |
        ↪  s in t.subscribedStudentList)
}

// all battles associtated to a tournament must start and end during the tournament
fact {
    always (all t: Tournament | all b: t.battleList | b.battleEvent.startDate ≥ t.
        ↪ tournamentEvent.startDate ∧ b.battleEvent.endDate ≤ t.tournamentEvent.
        ↪ endDate)
}

// all battles in a tournament must not overlap in time
fact {
    always (all t: Tournament | all disj b1, b2: t.battleList | b1.battleEvent.
        ↪ startDate > b2.battleEvent.endDate or b1.battleEvent.endDate < b2.
        ↪ battleEvent.startDate)
}

// all groups in a battle must have a unique name
fact {
    always (all b: Battle | all disj g1, g2: b.groupList | g1.name ≠ g2.name)
}

// all groups in a battle must be composed by students that are enrolled in the
    ↪ battle
fact {
    always (all b: Battle | all g: b.groupList | all s: g.studentList | s in b.
        ↪ enrolledStudentList)
}

// a student can be in only subscribe to a group per battle
fact {
    always (all b: Battle | all g: b.groupList | all disj s1, s2: g.studentList | s1
        ↪  ≠ s2)
    always (all b: Battle | all disj g1, g2: b.groupList | all s: Student | s in g1.
        ↪ studentList implies (not s in g2.studentList))
}

// all tournaments must have a unique name
fact {
    always (all disj t1, t2: Tournament | t1.name ≠ t2.name)
}
```

```
// all solutions are associated to a battle
fact {
    always (all s: Solution | one b: Battle | s in b.solutionList)
}

// a solution is present in a battle if and only if the group is present in the
    ↪ battle
fact {
    always (all b: Battle, s: Solution | (s in b.solutionList) iff (some g: b.
        ↪ groupList | s.groupID = g.groupID))
}

// all solutions must have an existing groupID
fact {
    always (all s: Solution | one g: Group | s.groupID = g.groupID)
}

// all solutions have a manualReviewScore if and only if the manualReview is true in
    ↪  the battle
fact {
    always (all b: Battle, s: b.solutionList | (s.manualReviewScore ≠ none iff b.
        ↪ manualReview = True))
}

// all groups have a uniqie groupID
fact {
    always (all disj g1, g2: Group | g1.groupID ≠ g2.groupID)
}

// all groups are associated to a battle
fact {
    always (all g: Group | one b: Battle | g in b.groupList)
}

// all event is associated to a tournament or a battle
fact {
    always (all e: Event | one t: Tournament | e in t.tournamentEvent ∨ one b:
        ↪ Battle | e in b.battleEvent)
}

// all code katas are associated to a battle
fact {
    always (all ck: CodeKata | one b: Battle | ck in b.codeKata)
}

// all test cases are associated to a code kata
fact {
    always (all tc: TestCase | one ck: CodeKata | tc in ck.testCases)
}

// all build automation scripts are associated to a code kata
fact {
    always (all bas: BuildAutomationScript | one ck: CodeKata | bas in ck.
        ↪ buildAutomationScript)
}

// ------------------------------ GOALS -------------------------------

// G1: Have his/her own profile on the application
assert DifferentEmailOrNicknameImpliesDifferentUsers {
    all u1, u2: User | u1.email ≠ u2.email implies u1 ≠ u2
    all u1, u2: User | u1.nickname ≠ u2.nickname implies u1 ≠ u2
```

```
}


// G2: Participate in a coding tournament
assert StudentsSubscribedToBattleAlsoSubscribedToTournament {
    all b: Battle, s: b.enrolledStudentList | one t: Tournament | b in t.battleList
        ↪ and s in t.subscribedStudentList
}

pred StudentParticipatesInTournament[s: Student, t: Tournament] {
    one b: Battle | b in t.battleList and s in b.enrolledStudentList
    #t.subscribedStudentList < #t.subscribedStudentList'
}


// G3: Cooperate with colleagues
assert StudentInGroupAlsoInBattle {
    all g: Group, s: g.studentList, b: Battle | g in b.groupList implies s in b.
        ↪ enrolledStudentList
}

assert NoLoneSolution {
    all b: Battle, s: b.solutionList | one g: Group | s.groupID = g.groupID
    all b: Battle, g: Group, s: Solution | (not g in b.groupList) implies (not (s in
        ↪  b.solutionList and s.groupID = g.groupID))
}

assert GroupSizeIsCorrect {
    all g: Group | g.size = #g.studentList
    all b: Battle, g: b.groupList | g.size ≥ b.teamMinimumSize and g.size ≤ b.
        ↪ teamMaximumSize
}

pred StudentCollaborateWithColleagues[stud: Student, g: Group, sol: Solution] {
    stud in g.studentList
    sol.groupID = g.groupID
    g.size > 1
}


// G4 : Review their own performance score
fun solutionsOfStudentInTournament(s: Student, t: Tournament): set Solution {
    {sol: Solution | some b: t.battleList, g: b.groupList | s in g.studentList and
        ↪ sol in b.solutionList and sol.groupID = g.groupID}
}

pred showStudentSolutions[stud: Student, t: Tournament, sols: set Solution] {
    sols = solutionsOfStudentInTournament[stud, t]
    #sols > 1
}


// G5: Organize coding tournaments
assert NoEducatorWithoutTournament {
    all e: Educator | some t: Tournament | e in t.administratorList
}

assert NoOverlappingBattles {
    all t: Tournament | all disj b1, b2: t.battleList | (b1.battleEvent.startDate <
        ↪ b2.battleEvent.startDate) implies (b1.battleEvent.endDate < b2.
        ↪ battleEvent.startDate)
}
```

```
pred showEducatorsInTournament[t: Tournament, edus: set Educator] {
    edus = t.administratorList
    #edus > 1
}


// G6: Run coding battles in their tournaments
assert BattleOwnerIsEducatorOfTournament {
    all t: Tournament, b: t.battleList | b.owner in t.administratorList
}


// G7: Close an existing coding battle in their tournaments
pred showClosedBattle[t: Tournament] {
    #t.battleList > #t.battleList'
}


// G8 & G9: Obtain an automated evaluation over a student solution & Manually review
    ↪  students submissions
assert CorrectSolutionScore{
    all s: Solution | s.automatedScore ≠ none
    all b: Battle, s: b.solutionList | s.manualReviewScore ≠ none iff b.manualReview
        ↪  = True
}

// ------------------------------ RUNS --------------------------------
check DifferentEmailOrNicknameImpliesDifferentUsers for 15 but 3 steps
check StudentsSubscribedToBattleAlsoSubscribedToTournament for 15 but 3 steps
check StudentInGroupAlsoInBattle for 15 but 3 steps
check NoLoneSolution for 10 but 3 steps
check GroupSizeIsCorrect for 15 but 3 steps
check NoEducatorWithoutTournament for 15 but 3 steps
check NoOverlappingBattles for 5 but 3 steps
check BattleOwnerIsEducatorOfTournament for 15 but 3 steps
check CorrectSolutionScore for 15 but 3 steps

run StudentParticipatesInTournament for 20 but exactly 3 steps
run StudentCollaborateWithColleagues for 20 but exactly 3 steps
run showStudentSolutions for 20 but exactly 3 steps
run showEducatorsInTournament for 20 but exactly 3 steps
run showClosedBattle for 20 but exactly 3 steps
```

Here below some predicates generated by the model are shown.

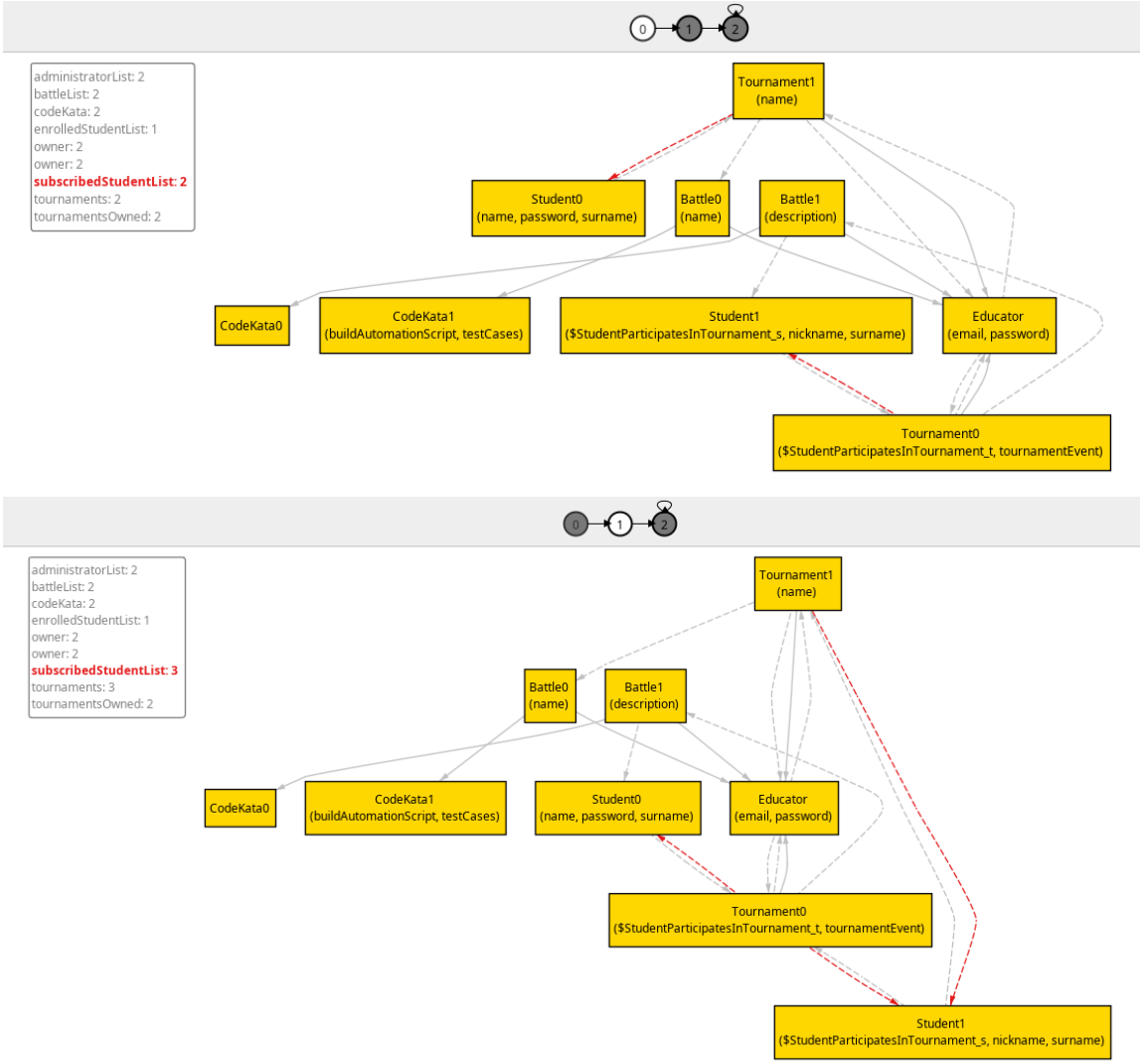


Figure 9: Predicate *StudentParticipatesInTournament*.

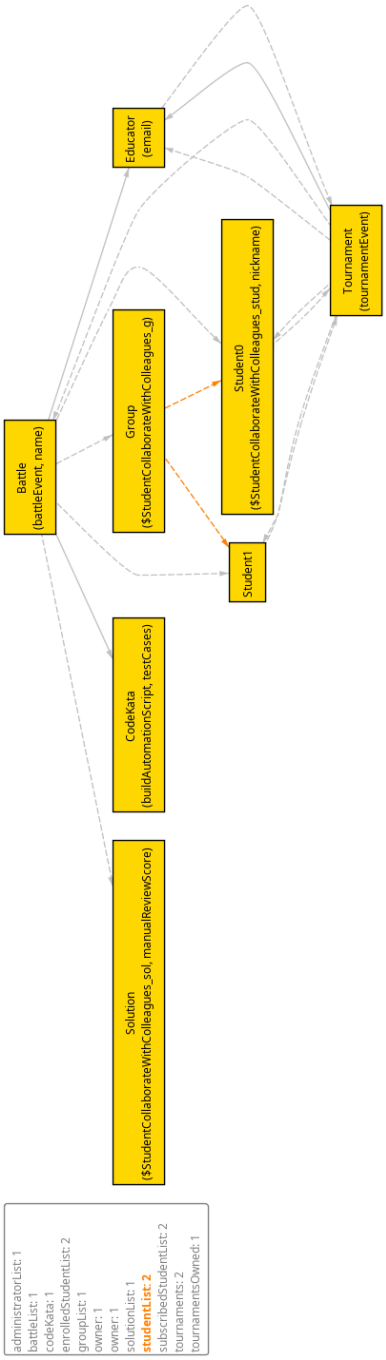


Figure 10: Predicate *StudentCollaborateWithColleagues*.

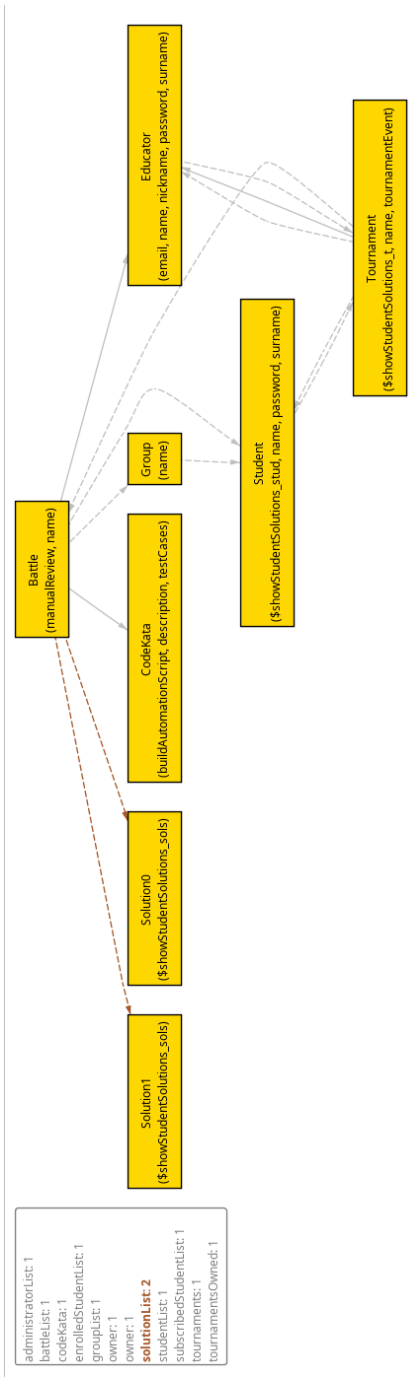


Figure 11: Predicate *showStudentSolutions*.

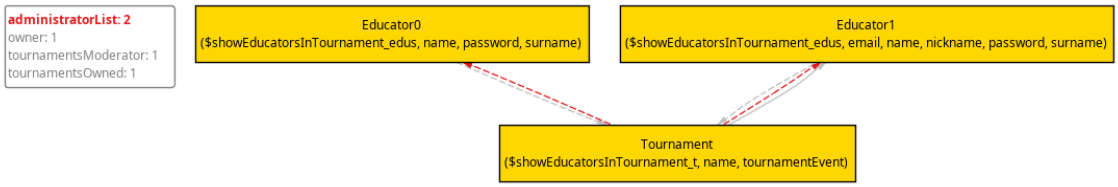


Figure 12: Predicate *showEducatorsInTournament*.

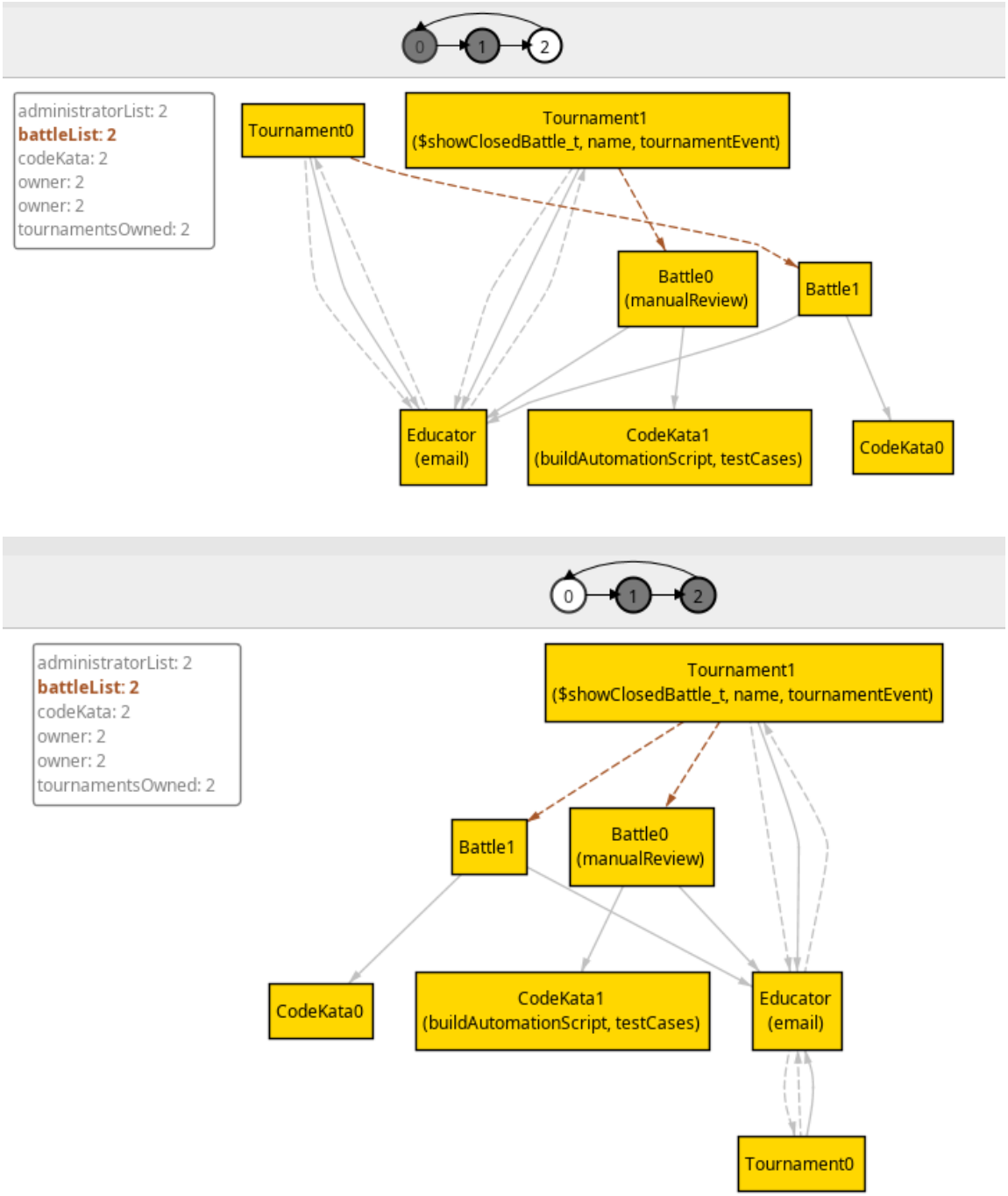


Figure 13: Predicate *showClosedBattle*.

5 Effort Spent

Student	Time for S.1	S.2	S.3	S.4
Davide Grazzani	3h	17h	2h	27h
Alessandro Masini	6h	12h	40h	2h

6 References

References

[1] https://www.researchgate.net/publication/369608775_APPLICATION_OF_TEAM_GAME_TOURNAMENT_TGT_AS_A_COOPERATIVE_LEARNING_MODEL_TO_INCREASE_STUDENTS'_LEARNING_MOTIVATION_OF_CLASS_X_OTKP_IN_BUSINESS_ECONOMICS_LEARNING_SUBJECTS_AT_SMK_TELKOM_PEKANBARU

[2] https://www.atlantis-press.com/article/125947409.pdf

[3] https://en.wikipedia.org/wiki/Test-driven_development

[4] https://www.similarweb.com/website/codewars.com/#overview