POLITECNICO DI MILANO

SOFTWARE ENGINEERING II

CodeKataBattle

Requirements Analysis and Specification Document

Version 1.0

Authors

Biagio Marra
Raffaele Russo

Contents

1 Introduction

1.1 Purpose

The aim of the 'CodeKataBattle' project is to facilitate the improvement of students' programming skills through training with programming exercises known as CodeKata. In addition, it enables teachers to organise tournaments containing a series of CodeKata battles, offering teams of students the opportunity to participate in such competitions, promoting the development of their programming and problem solving skills.

1.1.1 Goal

[G1] Educator administers the tournaments and decides and manages the battles within them.

[G2] Students participate in tournaments by taking part in related battles, either individually or in groups.

[G3] The tournament implements a ranking derived from each student's individual score. This score is determined by aggregating the scores acquired by the student in each battle in which he took part.

1.2 Scope

In recent years, with the constant progress of information technology, the demand for people with excellent software development skills has steadily increased.
CodeKataBattle is a system that allows through the interface the ability to participate in tournaments where the student can find a series of battles and decide which ones to participate in as an individual or by forming a team with other students.
All students participating in the battle will be sent a link of the main GitHub repository containing the CodeKata project on which they will be working.
The system will show the team's current score with each push the team makes on their repository GitHub, this score is automatically analyzed by the system and a third-party tool, on some predefined aspects and some chosen by the educator.
In addition, the system provides the teams participating in the battle and educators with the ranking of the current battle. Once the battle with its ranking is over, the system informs all participants in the battle.
On the other hand, the system provides the educator with an interface to create and close tournaments and to invite other educators to create new battles in the context of the tournament.
At the creation of each battle, the system gives the opportunity to upload the CodeKata specifying a set of information inherent to the battle.
When a tournament is closed, the system notifies all tournament participants of the final ranking.

1.2.1 World Phenomena

Phenomena events that take place in the real world and that the machine cannot observe.

[W1] Student wants to improve his software development skills
[W2] Student forks GitHub repository
[W3] Student set up an automated workflow through GitHub Actions
[W4] Students who are part of a team discuss implementations to be performed on the project
[W5] Educator creates the codekata project

1.2.2 Shared Phenomena

Phenomena controlled by the world and observed by the machine.

[SP1] Student signs up for a tournament
[SP2] Student unsubscribes from tournament
[SP3] Student signs up for a battle
[SP4] Student invites other students to form battle team
[SP5] Student makes a push on GitHub
[SP6] Student accept or decline the invitation to battle
[SP7] Educator opens a tournament
[SP8] Educator closes a tournament
[SP9] Educator invites other educators to participate in their own tournament

[SP10] Educator creates and configures a battle
[SP11] Educator associates a score to a group's final project
[SP12] Educator accept or decline the invitation to tournament

Phenomena controlled by the machine and observed by the world.

[SP13] System shows the tournaments
[SP14] System shows the battles of the tournament
[SP15] System shows the current score and ranking to each team in the battle
[SP16] System shows and notifies the final ranking of a battle to the students who participate in such a battle
[SP17] System shows the ranking of the tournament to all users of the platform
[SP18] System notifies all registered students when a tournament is opened
[SP19] System notifies all tournament participants when a battle is created
[SP20] System sends GitHub repository link

1.2.3 Machine Phenomena

Phenomena performed by the machine that the world cannot observe.

[M1] System analyses the project and calculates the score
[M2] System creates the GitHub repository

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Fork the GitHub repository**: This action means creating one's own independent copy of a project hosted on GitHub. This operation allows a user to work on a version of the project without directly affecting the original repository.
- **GitHub Actions**: GitHub Actions is an automation service built directly into GitHub that allows developers to automate software workflow. It allows processes to be defined, customized and automated through a series of actions performed in response to specific events within a GitHub repository.
- **Test first-approach**: The Test-First approach has been known as Test-First Development or TDD. It is an integrated quality method used in the Extreme Programming methodology in which developers write unit tests before writing production code.

1.3.2 Acronyms

- **CKB** : CodeKataBattle
- **CK** : CodeKata
- **RASD** : Requirement Analysis and Specification Document.

1.3.3 Abbreviations

- [Gn] - the n-th goal of the system
- [WPn] - the n-th world phenomena
- [SPn] - the n-th shared phenomena
- [UCn] - the n-th use case
- [Rn] - the n-th functional requirement

1.4 Revision history

- Version 1.0

1.5 Reference Documents

This document is based on:

- The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2022/2023;
- Slides of Software Engineering 2 course on WeBeep;

1.6 Document Structure

Mainly the current document is divided in 6 chapters, which are:

1. **Introduction**: it aims to describe the environment and the demands taken into account for this project. In particular it's focused on the reasons and the goals that are going to be achieved with its development;
2. **Overall Description**: it's a high-level description of the system by focusing on the shared phenomena and the domain model (with its assumption);
3. **Specific Requirements**: it describes in very detail the requirements needed to reach the goals. In addition it contains more details useful for developers (i.e information about HW and SW interfaces);
4. **Formal Analysis**: this section contains a formal description of the main aspect of the World phenomena by using Alloy;
5. **Effort Spent**: it shows the time spent to realize this document, divided for each section;
6. **References**: it contains the references to any documents and to the Software used in this document.

2 Overall Description

2.1 Product Perspective

2.1.1 Scenarios

1. **Student signs up for a tournament**
   Luca, eager to increase his skills in software development, learned of the existence of CodeKataBattle (CKB), a system dedicated to programming tournaments based on CodeKata exercises in which students can compete. Taken by interest, Luke registered on the CKB platform and checked out the tournaments that had not yet started. Having chosen the tournament, Luca registers. The CKB system presents Luca with a list of available battles within the tournament, allowing him to select one when the tournament begins. The system notified Luca whenever a new battle was created.

2. **Student signs up for a battle**
   Pasquale, an experienced Java programming student, is looking for opportunities to expand his skills in different languages. Exploring the list of available challenges on CodeKata, he would like to select a challenge based on Python, a language with which he has less experience. However, to tackle this new challenge with more confidence, he is interested in participating in a group with other students. Next, Pasquale examines the still open battles in which it is possible to participate.
   Once enrolled in a specific battle, Pasquale decides to invite his friend Marco to join him in tackling this challenge. Once the battle begins, the CodeKataBattle system sends Pasquale the link to the GitHub repository containing the CodeKata and the instructions necessary to configure the platform to receive notifications regarding each commit and push made by the team.

3. **The team works on the project**
   Team members, consisting of a group of friends, start collaborating on the project. To make sure everything gets done right, they create a fork of the GitHub repository and, using GitHub Actions, set up an automated workflow to inform the CodeKata-Battle (CKB) platform whenever a commit is made to the main branch of their repository.
   Once the first push is made, , the CKB pulls the source uploaded by the team to GitHub, and assigns a score from 0 to 100 based on the number of test cases that pass, the time difference between the battle registration deadline and the last commit made, and uses an external tool to do static analysis and evaluate the source on certain aspects entered at battle creation time by the educated.
   After scoring, the system proceeds to update the current ranking. This procedure allows the team to access information about the current position occupied within the ranking.
   After the final project submission deadline, the CKB platform notifies the team that the final battle ranking is available.

4. **Educator manages a tournament**
   Alessandro, a university lecturer in object-oriented programming, chose to adopt

the CKB platform to organise a tournament. This initiative allows the students on his course to compete against individuals outside the academic context, giving them a broader opportunity for challenge and comparison.
Alexander creates the tournament by setting a tournament entry deadline, and with the goal of enriching the variety of battles, he invited his fellow educators to collaborate in creating the battles. Once created, the platform will inform all participants of the new competition.
Once Alexander considers the number of battles that occurred in the context of the tournament he organized to be satisfactory, and once these have ended, he makes the decision to end the tournament. To this end, the system proceeds to notify all students participating in the tournament, of the available of the final ranking.

5. **Educator organises a battle**
   Gianluca, an expert in C++ programming, decided to organise a battle concerning a specific CodeKata that aroused particular interest. The CKB platform requested him to provide all the necessary specifications to set up the battle, including CodeKata upload, the deadline for registrations, the minimum and maximum limits of students per group and the deadline for the delivery of projects, and he choose maintainability and reliability as evaluation aspects.
   In addition, Gianluca chose to participate in the final evaluation of the projects submitted in the battle.
   During the course of the battle, the system provides Gianluca with the current score and ranking of the teams participating in the battle. At the end of the battle, Gianluca examines the final sources of each team and evaluates them personally.

2.1.2 Domain class diagram

In Fig(2.1) is represented the Domain Class Diagram related to the CKB. It contains all the domain elements in which the system operates and interacts between elements. In the following we will describe the most relevant parts of the diagram:

- There are two types of users: Student and Educator. They have things in common such as first name, last name and email.
- Only one educator is allowed to create a tournament; however, multiple educators may participate in that tournament. Each tournament must include at least one battle.
- In the context of the battle, one or more teams may participate, each consisting of at least one student who is involved in the tournament in which this battle takes place .
- Each team has a GitHub repository in which the last project actually uploaded (pushed) is stored, along with the direct link of the GitHub repository and the score assigned by the system for each push. The mentioned repository hosts the CodeKata related to the battle in which the team participated. Each team may be subject to a score by a specific one, on the final project submitted.

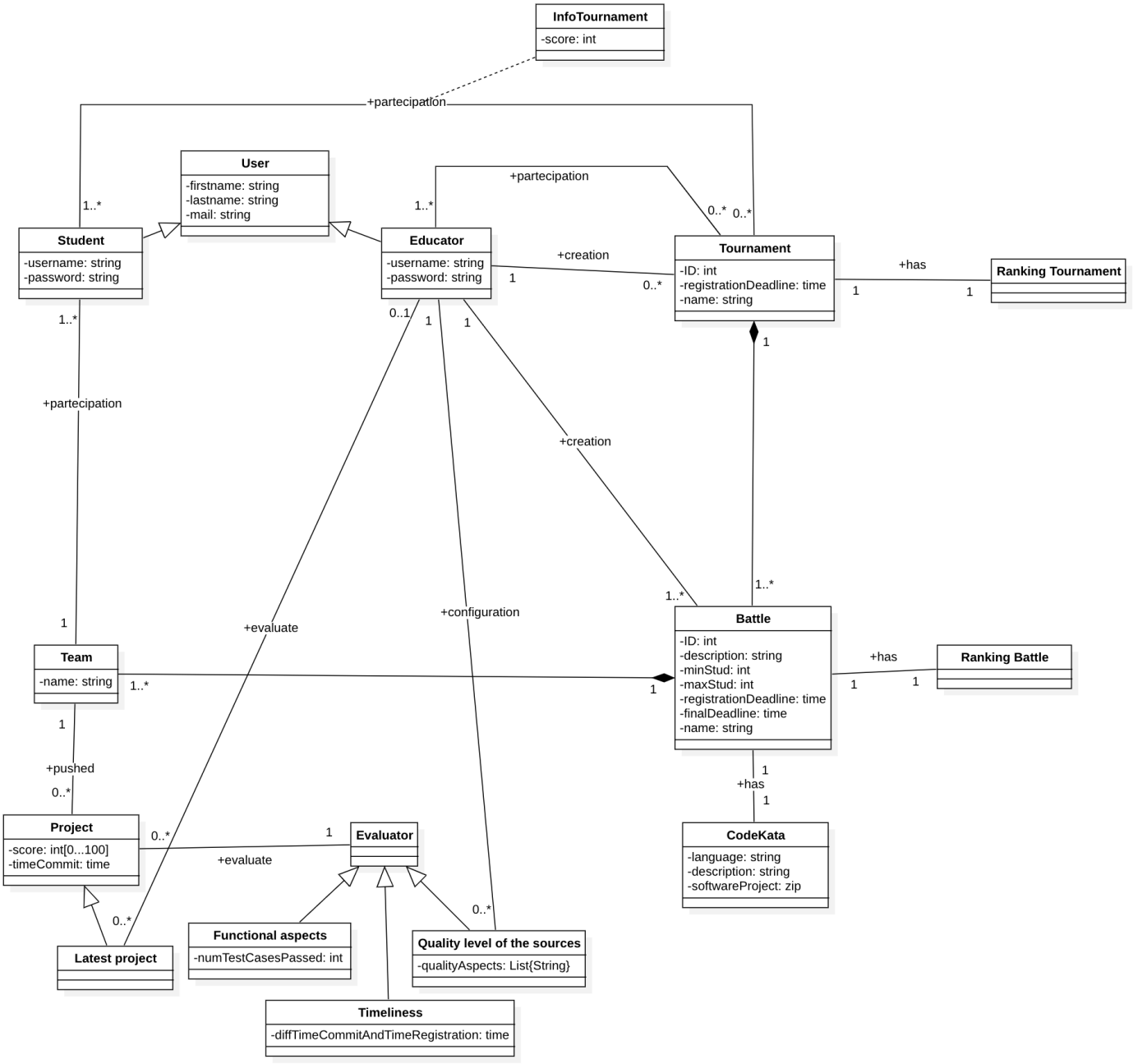


Figure 2.1: Domain Class Diagram

2.1.3 Statecharts

In this section, the state diagrams for Tournament, Team and Battle were examined. These diagrams offer a representation of the life cycle of an object of a specific class, highlighting the states through which the object transits during its life cycle and the conditions that determine these transitions from one state to another.

Status of Tournament

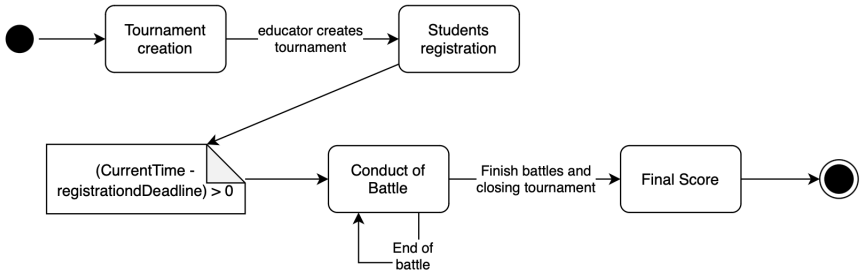


Figure 2.1.2: Tournament status

Figure 2.1.2 presents the state diagram describing the life cycle of an instance of the tournament class.
The process begins in the initial state, called ***Tournament Creation***, the phase in which an educator wants to start a tournament. During this phase, the educator may invite other colleagues to participate in the creation of battles within the tournament.
After the creation of the tournament by the educator, the system moves to ***Student Registration*** status, allowing students to register for the tournament.
Once the deadline to register for the tournament has passed,calculated by the difference between the currentTime and the registrationDeadline, the tournament begins, proceeding in the ***Conduct of Battle***. In this state, the educators manage the battles they have created. At the end of each battle, the participants' scores are updated and notified.
Once all battles within the tournament have been concluded and the educator responsible for the tournament decides to close the event, the process transitions into the ***Final Score*** state. In this state, the ranking of the tournament participants is updated and each participant is notified of the final results of the tournament.

Status of a Team

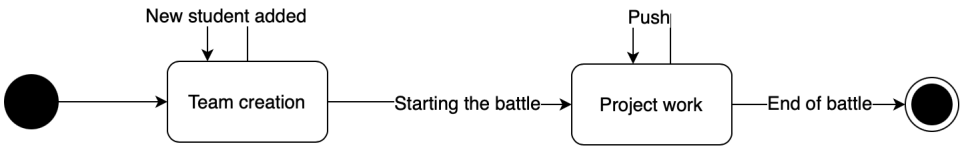


Figure 2.1.3: Team Status

The figure 2.1.3 presents the statechart describing the status of a team within a battle context.

The process begins in its initial state, called ***Team Creation***, in which a student participating in a battle forms a team with other students by inviting them, respecting the specific constraints of the battle they are joining.
After the start of the battle, the team enters the ***Project Work*** state. In this phase, team members work on the Code Kata associated with the battle, making regular pushes to GitHub and receiving feedback.
The process finally reaches its terminal state at the end of the battle.

Status of Battle

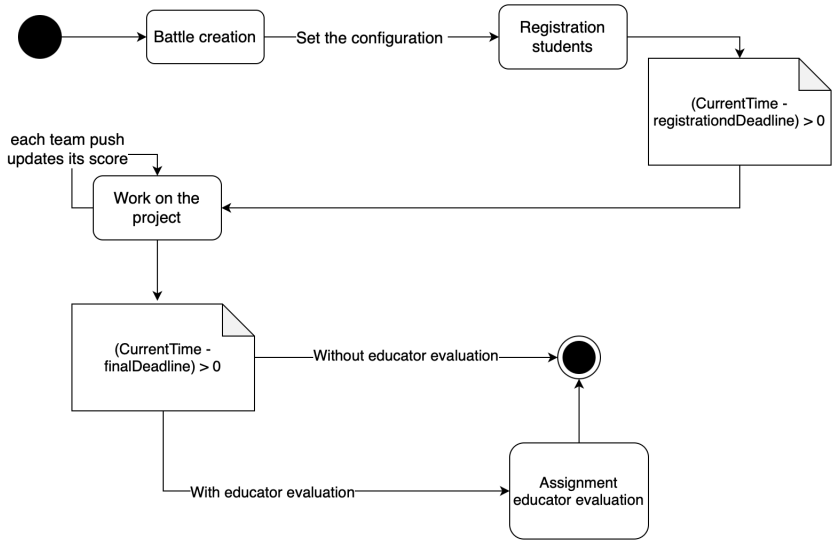


Figure 2.1.4: Battle Creation

The figure 2.1.4 illustrates the statechart relating to the procedure of creating a battle by an educator.
The process begins in the initial state, represented by ***Battle Creation***, in which the battle parameters are configured.
It then transitions into the ***Registration students*** state, where interested students register and invite other students to form teams to participate in the battle. When the registration deadline is reached, calculated by the difference between the currentTime and the registrationDeadline, the battle starts the operational phase by entering the ***Work on the Project*** state, during this phase the teams receive the link to the GitHub repository and the instructions to be followed to enable correct operation, after which they start working on the project by pushing on the repository, in this way every push before the deadline, the platform calculates and updates the team score.
At the expiry of the ***Final Deadline***, calculated through the difference between the currentTime and the finalDeadline, if the educator has not made a decision regarding scoring, the process transits to the final state.
Conversely, if the educator decides to evaluate the final project by assigning a score, the process moves to the ***Assignment Educator Evaluation*** state. At this stage, the educator proceeds with the scoring of the team, finally leading the process to the final state.

Status of Student

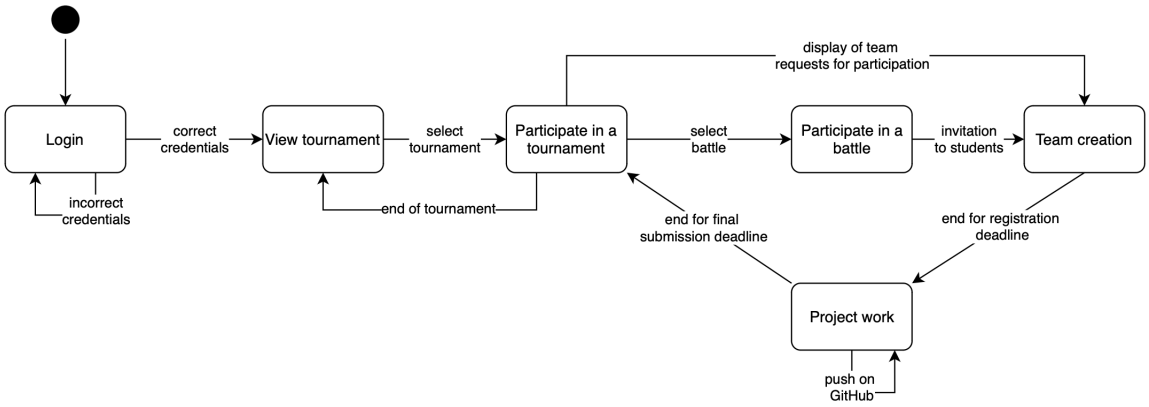


Figure 2.1.5: Student status

The figure 2.1.5 illustrates the statechart representing the status of a student within the system.
The process starts from the initial state ***Login***, where the student enters their credentials. If the credentials entered are incorrect, the student remains in that state; otherwise, he moves on to the next state, ***View tournament***, where the available tournaments are displayed.
Once the tournaments are displayed, the student selects the one in which he wishes to participate, passing into the ***Partecipate in a tournament*** state. In this status, the available battles within the tournament are listed. In this state, two scenarios can occur:

- The student selects a specific battle in which he wishes to participate, passing through the ***Participate in a battle*** status. At this point, the student enrolled in the battle must create a team, inviting other student participants. This transition takes place in the ***Team creation*** state, where the student enters the team name and invites other participants registered for the tournament.

- The student receives and accepts a request to participate in a team, moving directly into ***Team creation*** status.

Once the battle registration period has expired, the process moves into the ***Project work*** state. In this phase, the team works on the received codekata, pushing on GitHub. When the final submission phase expires, the student returns to the ***Participate in a tournament*** state. If the tournament is still running, the student goes through the same status as previously described. However, if the tournament has been closed by the responsible educator, we return to the ***View tournament*** status, where the available tournaments are shown.

2.2 Product Functions

2.2.1 Sign Up and Login

This function is accessible to all users. The registration process for students and educators allows them to register on the platform by entering the required data, including first name, last name, e-mail address, username and password.
The login function allows users to access the platform using username and password.

2.2.2 Manage tournaments

The tournament management function offers educators the possibility of creating tournaments by defining their name and setting a deadline for registrations. They can also invite other educators to participate. In addition, it allows students to register for tournaments and revoke their registration if necessary.
Each time a tournament is created, the system notifies each platform member. At the end of each battle, the participants' individual scores are updated, representing the sum of the points obtained in the battles completed within the tournament. The ranking of the tournament accordingly is also updated.
Once all battles have been completed and wishing to close the tournament, the creating educator can proceed to close it. In addition, each user of the CKB platform can view the list of all ongoing tournaments and their respective rankings.

2.2.3 Participation in tournament

This functions allows students to sign up for a tournament, and once they sign up and the tournament has started they can see the list of available battles and the tournament ranking with their current score. When the tournament closes, they will be notified that the final ranking is available.

2.2.4 Manage battle

The battle management function allows each educator involved to create multiple battles in a tournament.
When each battle is created, a notification is sent to all students registered for that tournament. At each battle, it is required to upload the CodeKata on which the teams are to focus and to specify the minimum and maximum number of students for each team. In addition, the educator enters a brief description, sets the deadline for registration, the deadline for delivery of the final project and indicates the quality aspects to be sent by the system to a third-party tool for evaluation.
At the battle deadline, if declared during battle creation, the educator can view the source of the teams and can assign a personal score. Otherwise, the system will close the battle and directly present the final ranking of the battle along with the corresponding score for each team. The system will also inform each student participating in the battle of the availability of the final ranking.

2.2.5 Participation in battle

The function allows students to sign up for only one battle at a time, the delivery of invitations for the formation of groups that conform to the limits set by the educator. Once the groups are formed and the battle begins, the system will send each member of each team the link to the main GitHub folder and the instructions to be executed to allow the platform to stay updated on commits and pushes performed by teams. For each push made by the team on its main repository, the system will use 3 parameters: functional aspects, measured in terms of number of test cases that pass out of all test cases, timeliness, measured in terms of time passed between the registration deadline and the last commit, quality level of the sources, where the system uses SonarQube to perform static analysis of the source and evaluate it on aspects entered at the time of battle creation by the educator. By assigning a score between 0 and 100. In this way, the team can view the current score and relative position in the ranking with each push.

2.3 User characteristics

There are two types of users who interact with the system: Student and Educator.

2.3.1 Student

A Student is a person who possesses basic skills in software programming and demonstrates a high level of competence in the use of computing devices. He wishes to increase his skills in software development and uses the CKB system for this purpose.

2.3.2 Educator

An Educator is a person with advanced skills in software development. He is responsible for managing the administrative functions of the system, including the creation of tournaments and battles.

2.4 Assumptions, dependencies and constraints

2.4.1 Regulatory policies

The CKB system will ask for user personal information like name, surname and email address. Email addresses won't be used for commercial purposes. Personal information will be processed in compliance with the GDPR.

2.4.2 Domain Assumptions

Below are the assumptions made for the domain. These assumptions are properties and/or conditions that the system takes for granted, mainly because they are outside the control of the system itself, and therefore must be verified to ensure the correct behavior of CodeKataBattle.

[D1] Student must have an internet connection
[D2] Educator must have an internet connection
[D3] Educator be a very knowledgeable person in the area of software development
[D4] Student must have a GitHub account
[D5] Students who are part of the team when they receive the main GitHub repository must fork it
[D6] Students who are part of the team must set up an automated workflow through GitHub Actions to inform the platform
[D7] Students must have all the tools(e.g., IDE) to work on the project
[D8] Educator must make sure that the CodeKata he intends to upload, we have no errors inside
[D9] Students participating in a battle are expected to follow a test-first approach

3 Specific requirements

3.1 External Interfaces

3.1.1 User Interface

This section of the document illustrates the platform's user interfaces, graphical tools that facilitate users' interaction with the software.
Within the platform, there are two different types of views based on user roles: the Student and the Educator. Each role offers specific and distinct functionalities.
The Educator, can create and manage tournaments and battles. The Student, on the other hand, is able to actively participate in tournaments and battles.
This distinction of roles results in a differentiated user interface, specially designed to offer specific functionalities according to the profile and actions allowed to each type of user.

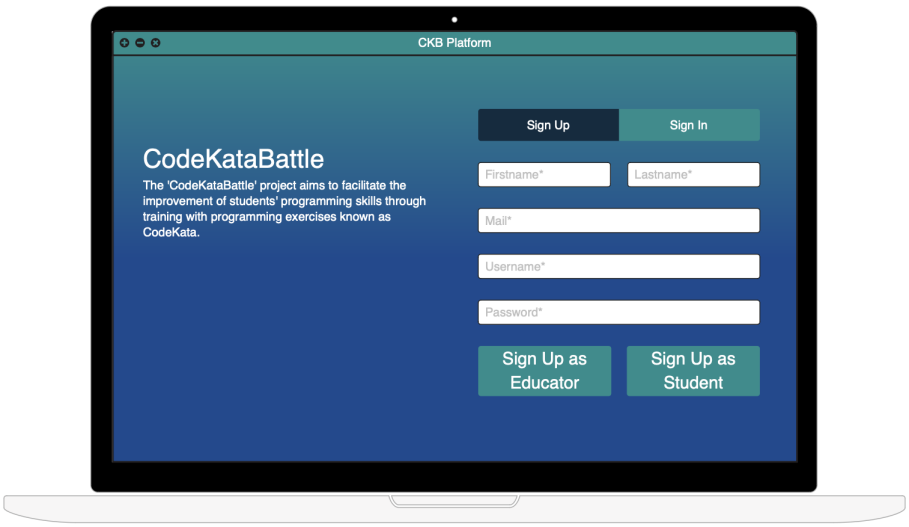


Figure 3.1.1: User Interface to register on the platform

The figure 3.1.1 shows the user interface relating to the platform registration procedure. This interface allows users to enter their personal data, such as firstname, lastname and e-mail address, as well as provide a username and password to access the system. It also offers the possibility to select the desired type of registration, allowing users to register as a student or as an educator.

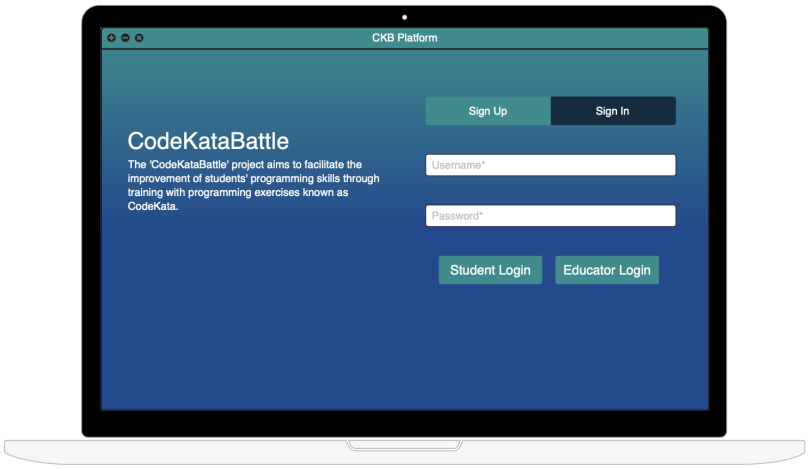


Figure 3.1.2: User Interface to login on the platform

Figure 3.1.2 illustrates the interface dedicated to accessing the platform. This screen allows users to enter their login credentials, including username and password. It also offers the choice of logging in as Educator or Student.

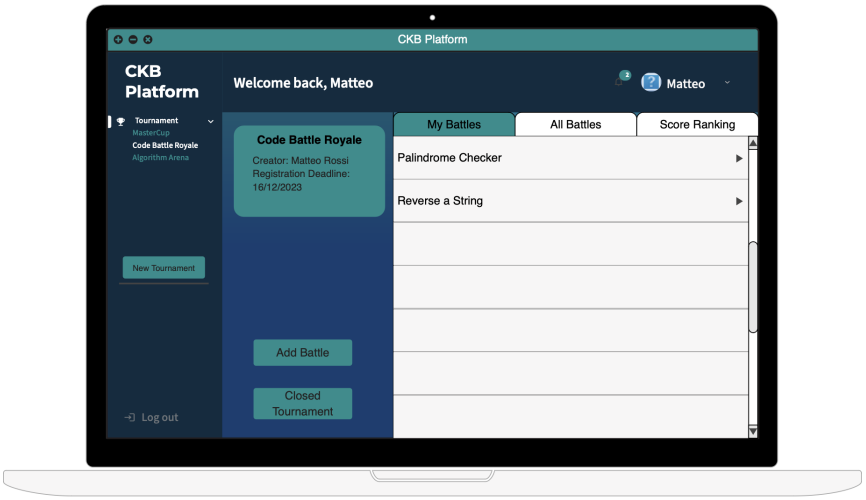


Figure 3.1.3: Educator Interface for Tournament Management

Figure 3.1.3 represents the educator's interface for managing tournaments and related battles.
In the left section of the interface, a list of tournaments the educator is a creator of or participates in is displayed, as well as a 'New Tournament' button to create a new tournament.
Each tournament is accompanied by a descriptive section containing information such as the name of the tournament, the creator and the registration deadline . In addition there are two buttons:

- "Add Battle": allows the addition of new battles within the tournament.
- "Closed Tournament": allows to close the tournament only to the educator who created the tournament and if all battles have been completed.

The section to the right of the interface presents a series of selectable tabs that allow various information to be displayed:

- "My Battles": shows all battles created for that specific tournament by the educator.
- "All Battles": shows the complete list of all battles in the tournament.
- "Score Ranking: displays the ranking based on the scores of all those participating in the tournament.

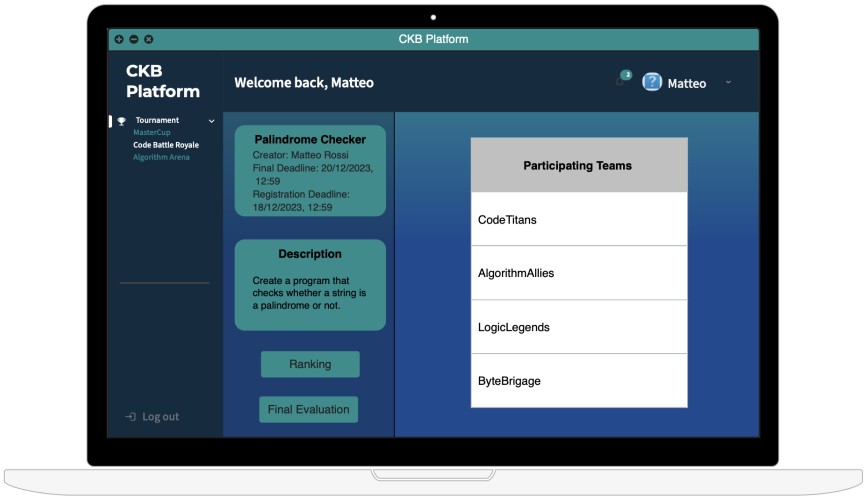


Figure 3.1.4: Educator Interface for Battle Management

Once the battle has been selected, Figure 3.1.4 presents the interface dedicated to its management by the educator who is its creator.
In the left section of the interface, there are two containers. The first contains key information regarding the battle, including details such as the name, creator educator, final deadline and registration deadline. The second container contains a detailed description of the battle.
In addition, two buttons are available:

- "Ranking": displays the ranking for the battle.
- "Final Evaluation": only accessible if the battle is over, defined by the Final Deadline. It provides access to a dedicated screen to view the last source uploaded by each team and evaluate it..

In addition, the list of teams actively participating in the battle is presented in the right section of the interface.

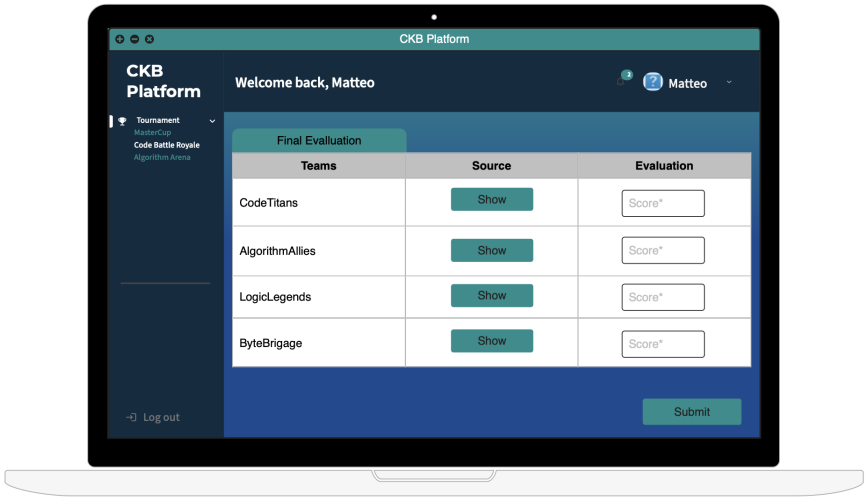


Figure 3.1.5: Educator Interface for Final Evaluation

At the end of the battle, defined by the final deadline date, when the educator presses the 'Final Evaluation' button, he is shown the page in the Figure 3.1.5.
This section lists the participating teams with a corresponding button to access the source file. In addition, there is also a dedicated scoring section, which allows you to manually enter the value to be assigned to each team.
Once the scoring is completed, the operation is confirmed by pressing the 'Submit' button.

Figure 3.1.6: Student Interface to register for a battle

Figure 3.1.6 shows the interface that offers students the opportunity to register for a specific battle. This screen allows students to form a team, entering the team name and inviting other students to the tournament, always respecting the limits set for the tournament itself, which are detailed in the dedicated section on the left of the interface.

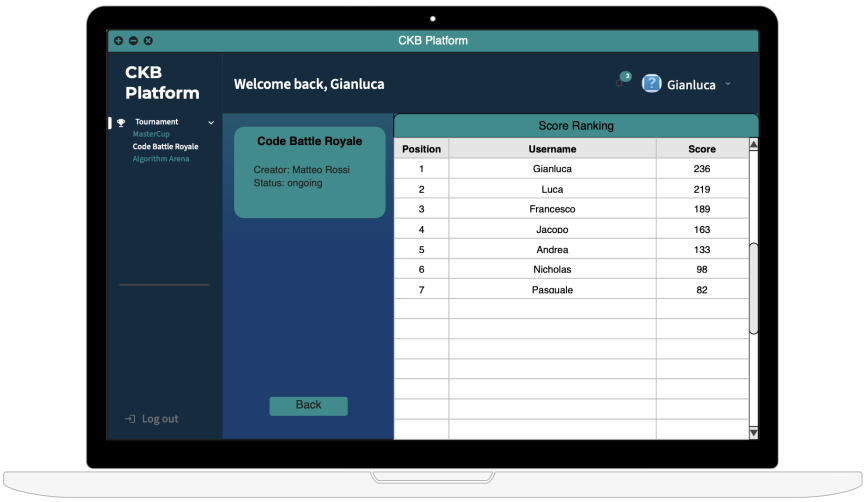


Figure 3.1.7: Interface for the Tournament Ranking

The interface illustrated in Figure 3.1.7 shows the detailed ranking of the tournament entrants, together with their respective scores, accessible to all registered members of the platform. In addition, there is a button in the left section that allows you to return to the battle list

3.1.2 Hardware interfaces

The CKB system has no external hardware equipment to interact with.

3.1.3 Software interfaces

The system requires certain software interfaces in order to provide specific services. These interfaces are detailed below:

- **GitHub REST API v3**: the system uses these APIs to enable communication with the GitHub platform and the execution of specific actions. Such actions include, for example, creating folders and performing pull operations related to specific directories.
- **SonarQube Web API**: these APIs make it possible to automate various operations, including the ability to upload projects and perform static analysis in accordance with specified parameters.

3.1.4 Communication interfaces

The system uses the Internet connection to communicate with all connected devices. The backend of the system will expose a unified RESTful API to communicate with all clients using HTTPS and TCP/IP.

3.2 Functional requirements

Following are all the functional requirements divided by functionality.

Registration and login requirements

Rn	Description
R1	The system allows the student to register by entering firstname, lastname, email, username and password.
R2	The system allows registered students to log in by entering username and password.
R3	The system allows the educator to register by entering firstname, lastname, email, password and username.
R4	The system allows registered educator to log in by entering username and password.

Tournament requirements

Rn	Description
R5	The system allows the educator the ability to create of a tournament, allowing him or her to set a deadline for entries.
R6	The system allows the educator to invite other educators to join the tournament.
R7	The system allows the educator who created the tournament, to close the tournament if all battles are completed.
R8	The system allows students to register for the tournament by the deadline.
R9	The system allows students to unsubscribe from the tournament.
R10	The system allows students enrolled in the tournament to see the list of all battles(unstarted, ongoing, and completed).
R11	The system allows educators to see the list of all battles(unstarted, ongoing, and completed) in that tournament.
R12	The system notifies all students registered on the platform to inform them of the creation of a new tournament.
R13	The system, at the end of each battle, updates each student's score by summing the scores obtained in each completed battle in which he participated, reflecting these changes in the overall tournament ranking.
R14	The system when the educator closes the tournament, notifies all students in that tournament of the availability of the final ranking.
R15	The system allows all members of the platform to see the list of ongoing tournaments.
R16	The system allows all members of the platform to see the ranking of a tournaments, with the relative score associated with each student participating in the tournament.
R17	The system sends a notification to educators who have been invited by the educator who created the tournament to participate in the creation of battles within the tournament.
R18	System allows educators to accept or decline invitations to participate in the tournament.

Battle requirements

Rn	Description
R19	The system allows all educators participating in the tournament to create battles by uploading the CodeKata and specifying the description, entry deadline, final project delivery deadline, minimum and maximum number of students in each team, if desired manual evaluation, and qualitative aspects of the source to be evaluated.
R20	The system when the battle is over allows the educator who created the battle to see each team's final project.
R21	The system allows the educator to evaluate each team's final projects.
R22	The system allows students registered for the tournament to see all the specifics of a battle.
R23	The system allows students participating in the tournament to register for the battle by the deadline.
R24	The system allows students participating in the battle to create teams by inviting other students participating in the same tournament.
R25	The system when the battle enrollment expires must create the GitHub repository with the CodeKata in it.
R26	The system when the battle enrollment expires sends the GitHub repository link to all members of each team and the instructions to fork the repository and set up an automated workflow.
R27	The system must retrieve the source from the team repository after each push by the team.
R28	The system assigns a integer score from 0 to 100 to the project retrieved from the GitHub repository, according to the following aspects: functional aspects (number of test cases passed), timeliness (difference between the last team commit and the battle registration deadline), and source quality level, the latter evaluated by a third-party tool Sonarqube.
R29	The system allows students participating in the battle to see each team's current score.
R30	The system allows students participating in the battle to see the current ranking of the battle.
R31	The system allows the battle educator to see the current ranking.
R32	The system allows the battle educator to see the score of each team.
R33	The system, at the end of the battle, allows students participating in tournament to see the final ranking.
R34	The system, at the end of the battle, allows educators participating in tournament to see the final ranking.
R35	The system, at the end of the battle, notifies all students participating in the battle of the availability of the final ranking.
R36	The system sends a notification to all students participating in a tournament when a new battle is created.
R37	The system sends a notification to students who have been invited by another student to join and form a team.
R38	The system offers the possibility for students to accept or reject invitations received from other students to form a team.

3.2.1 Use cases diagrams

Following are use case diagrams, which help us understand how actors interact with the system.

Unregistered user Use Case Diagram

Figure 3.2.1: Unregistered user Use Case Diagram

Use Case Diagram

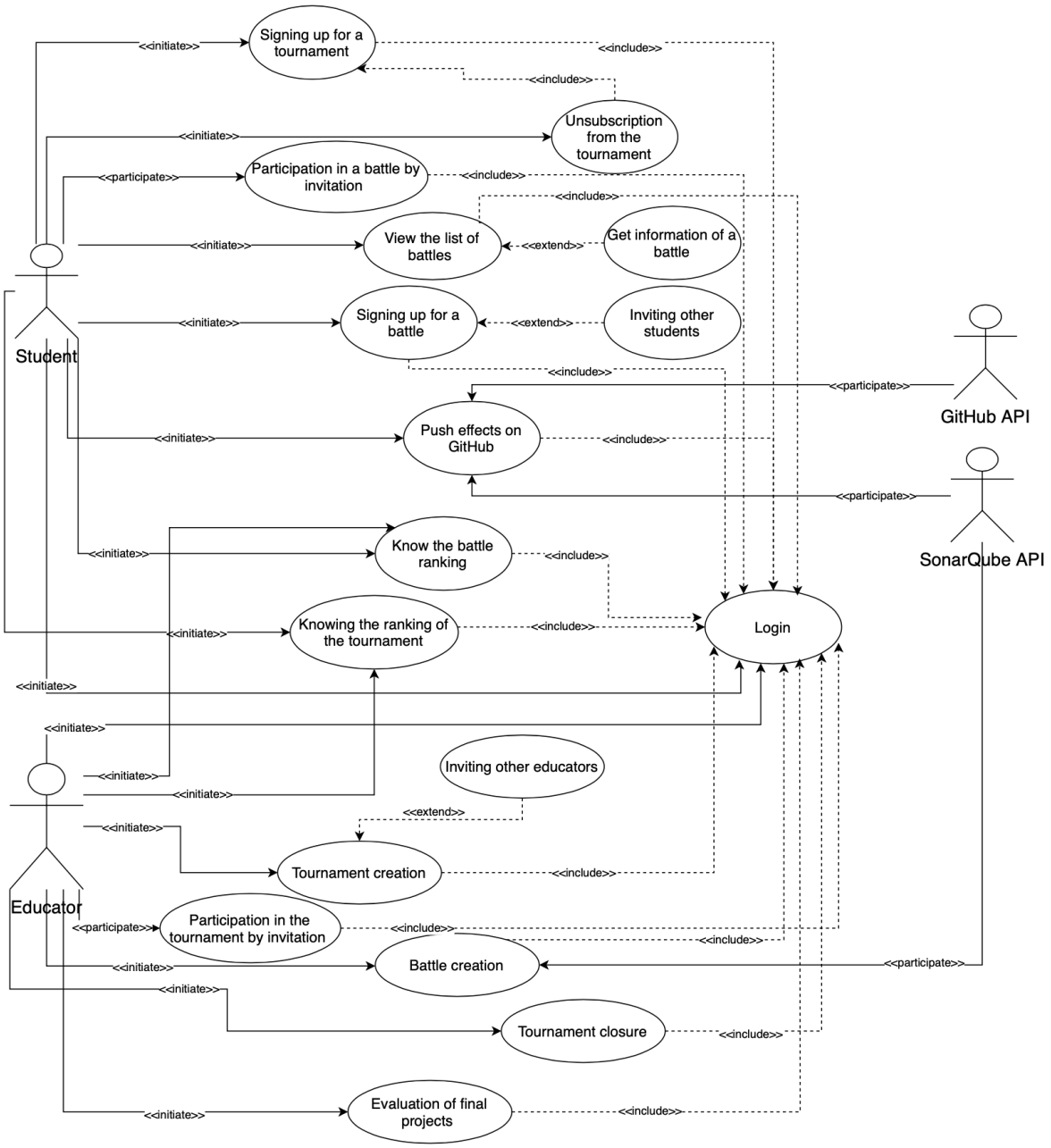


Figure 3.2.2: Use Case Diagram

3.2.2 Use Cases and Sequence Diagrams

[UC1] - Registration

Name	Registration
Actors	Unregistered User
Entry Condition	Unregistered User accesses the platform.
Event Flow	**1.** Unregistered User presses on the "Sign Up" button. **2.** The system shows a screen with the relevant fields: Firstname, Lastname, Email, Username, Password. And two buttons " Sign up as Student" and "Sign up as Educator". **3.** Unregistered User fills in the fields. **4.** Unregistered User clicks on the "Sign Up as Student" button. **5.** The system displays the initial dashboard for the Student or Educator.
Exit Condition	Unregistered User is correctly registered as a Student or Educator.
Alternative	**4a.** Unregistered User clicks on the "Sign Up as Educator" button.

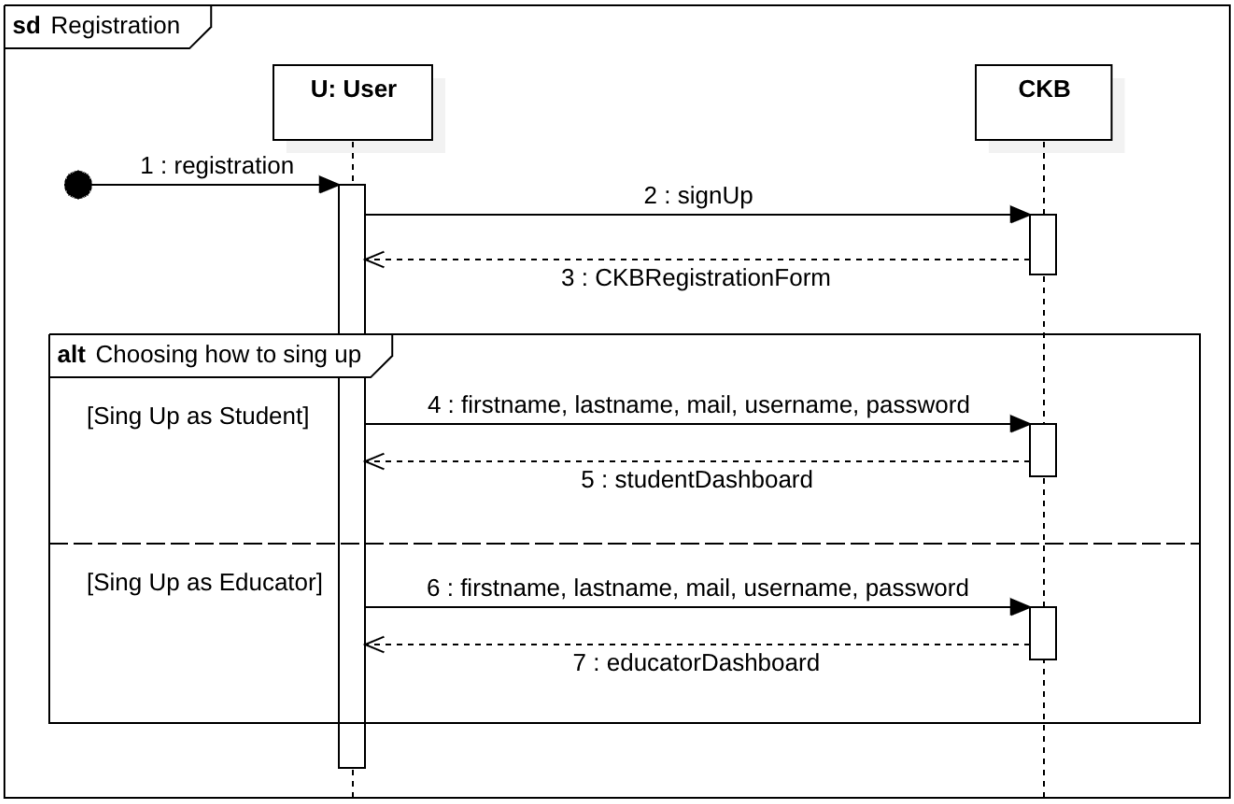


Figure 3.2.3: Registration Sequence Diagram

[UC2] - Login

Name	Login
Actors	User
Entry Condition	User accesses the platform.
Event Flow	**1.** User presses the "Sign In" button. **2.** The system shows a screen with the relevant fields: Username, Password. And two buttons " Student Login" and "Educator Login". **3.** User fills in the fields. **4a.** User clicks on the "Student Login" button. **5.** The system displays the initial dashboard for the Student or Educator.
Exit Condition	Student or Educator are logged in correctly to the platform.
Alternative	**4b.** User clicks on the "Educator Login" button.

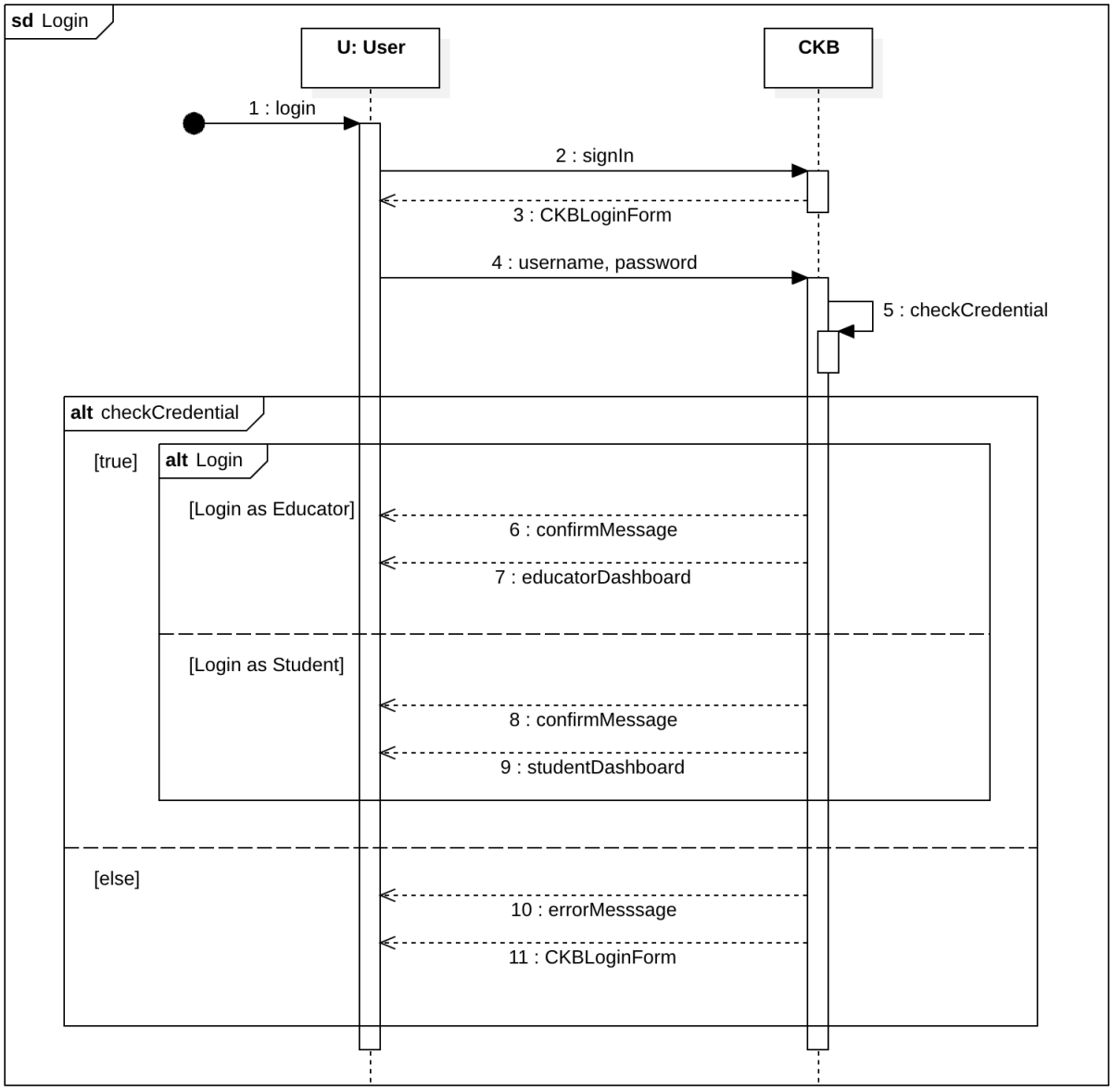


Figure 3.2.4: Sign In Sequence Diagram

[UC3] - Tournament Creation

Name	Tournament Creation
Actors	Educator, Student
Entry Condition	Educator logged into the system.
Event Flow	**1.** Educator presses the "New Tournament" button. **2.** The system shows him a screen with fields to fill in: - Tournament name. - Entry Deadline. It also shows a list of educators with a checkbox next to it. **3.** The educator fills in these fields and selects educators from the list. **4.** Educator presses the "Confirm" button. **5.** System notifies all students registered on the platform of the new tournament. **6.** System sends a confirmation message
Exit Condition	The tournament was created correctly and notifications are sent correctly.
Exception	In the event of a duplication of the tournament name, the system will ask you to enter the required fields again.

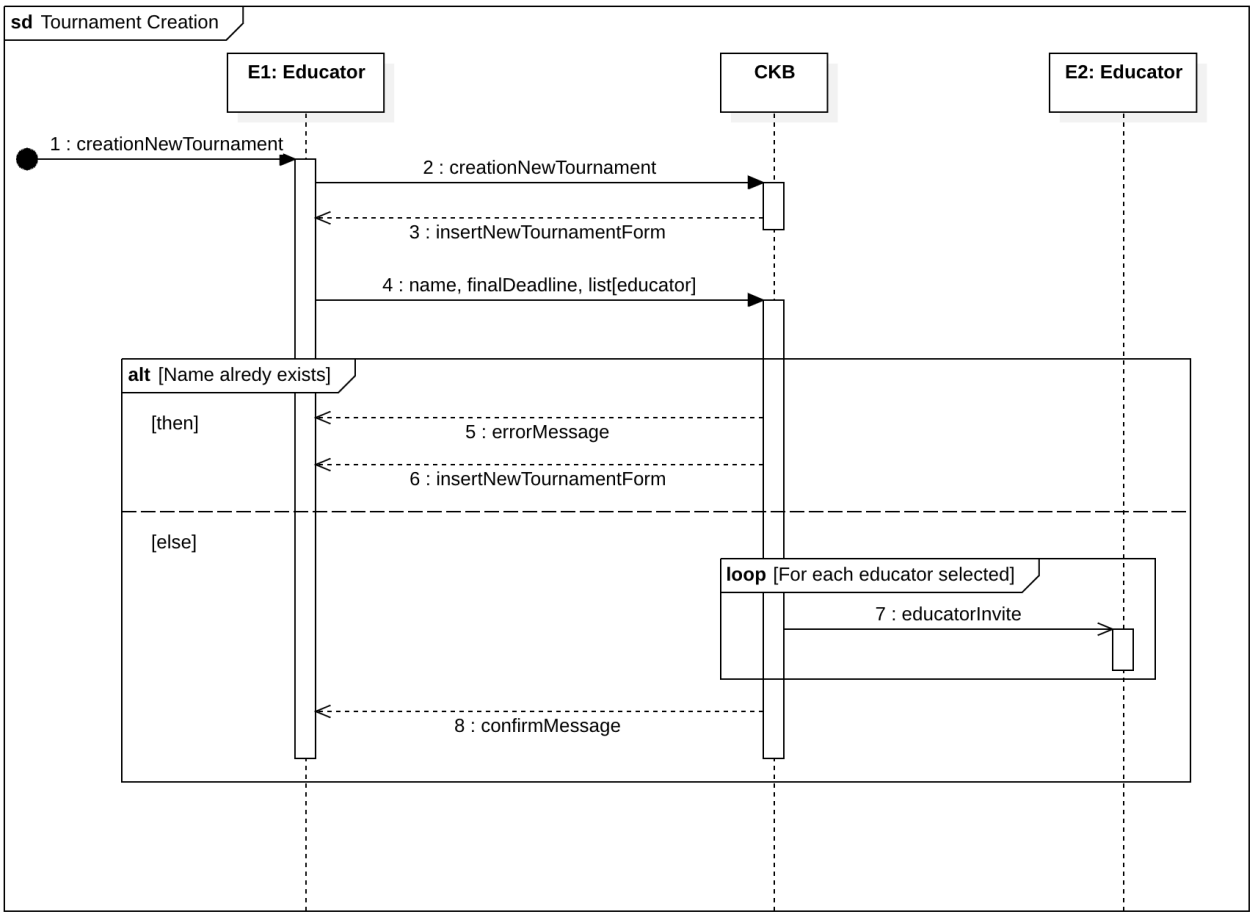


Figure 3.2.5: Tournament Creation Sequence Diagram

[UC4] - Battle Creation

Name	Battle Creation
Actors	Educator, SonarQube, Student
Entry Condition	Educator logged into the system and must participate in a tournament.
Event Flow	**1.** Educator presses the "Add Battle" button. **2.** The system shows him a screen with fields to fill in: - Battle name. - Registration Deadline. - Minimum and maximum number of students in each group. - Final project upload deadline. - A field to upload the CodeKata. - Aspects that must be evaluated in the source. - Manual evaluation. **3.** Educator fills in these fields. **4.** Educator presses the "Confirm" button. **5.** System sends SonaQube the aspects on which it must evaluate all the sources of each team in this battle **6.** System notifies all students participating in the tournament of the new battle. **7.** System sends a confirmation message
Exit Condition	The battle was created correctly, notifications are sent correctly and correct configuration of SonarQube.
Exception	In the event of a duplication of the battle name, the system will ask you to enter the required fields again.

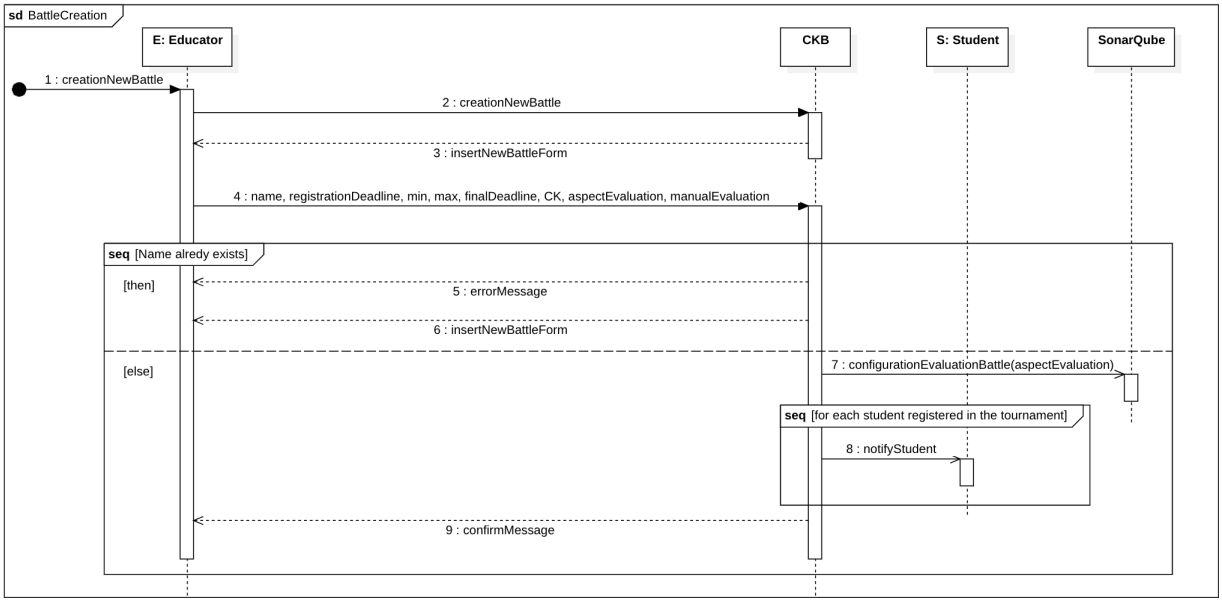


Figure 3.2.6: Battle Creation Sequence Diagram

[UC5] - Tournament Closing

Name	Tournament Closing
Actors	Educator, Student
Entry Condition	Educator logged into the system, the educator who created the tournament.
Event Flow	**1.** Educator pushes the button on his tournament. **2.** System shows him the screen of his tournament. **3.** Educator presses the "Close Tournament" button. **4.** System notifies all students participating in the tournament of the final ranking. **5.** System sends a confirmation message
Exit Condition	The tournament is successfully closed and notifications are sent correctly.
Exception	The battles are not all finished. The system returns an error message.

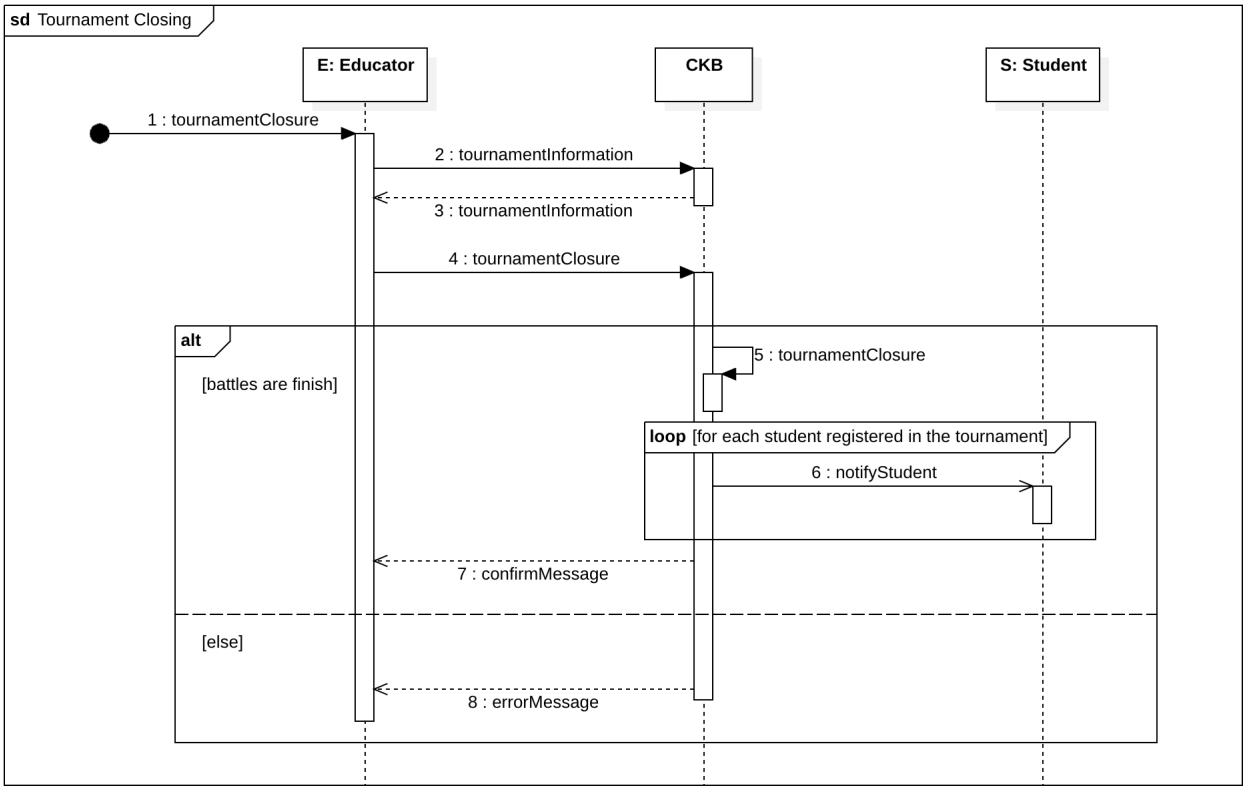


Figure 3.2.7: Tournament Closing Sequence Diagram

[UC6] - Evaluation of final project

Name	Evalutation of final project
Actors	Educator
Entry Condition	Educator connected to the system, the educator who created the battle, the battles is finished.
Event Flow	**1.** Educator presses the "Final Evaluation" button. **2.** System shows the screen with the list of teams inside, with a button next to it to see the latest source and a field to enter the rating. **3.** The educator presses the button to examine the source of a team. **5.** The system shows it the source. **6.** The educator presses the button to go back. **7.** System shows the previous screen. **8.** Educator enters assessment to relevant team.
Exit Condition	Educator assigned the assessment to each team.

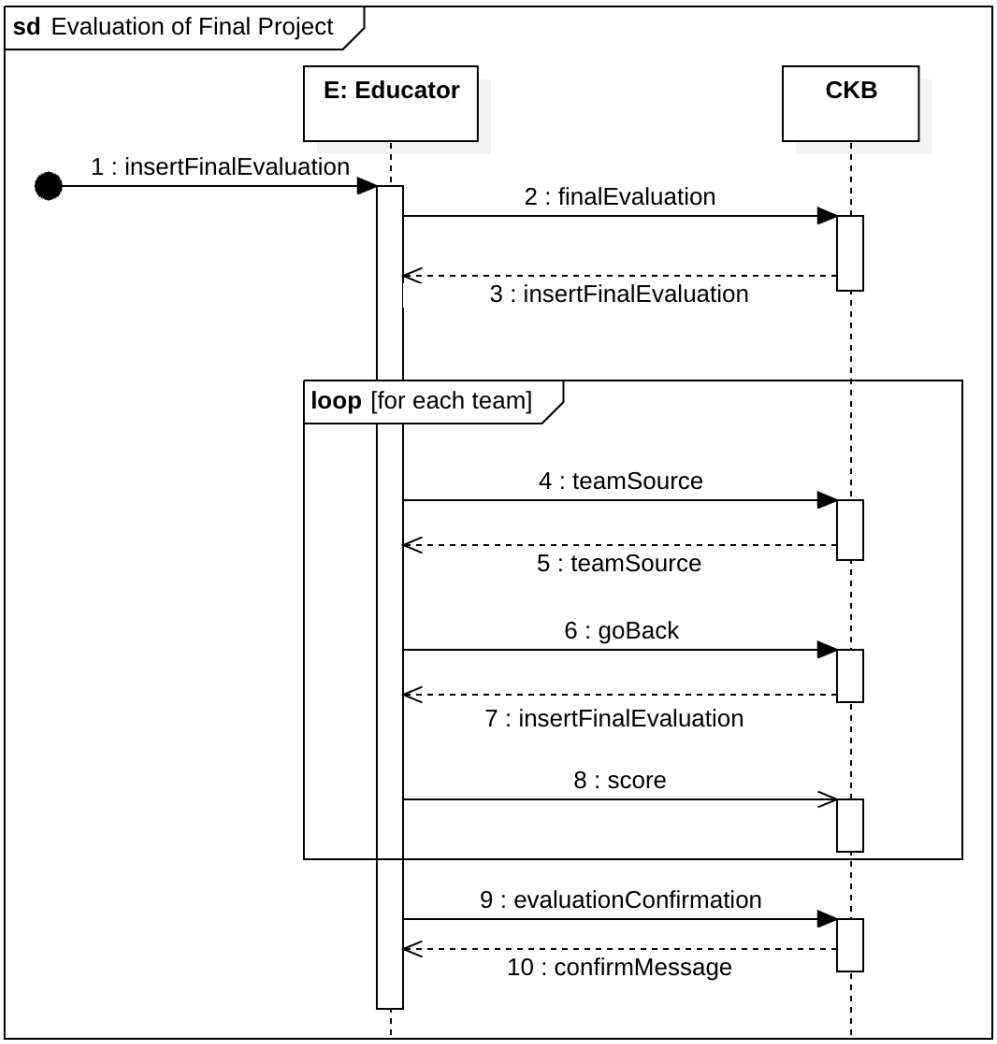


Figure 3.2.8: Evaluation of Final Project Sequence Diagram

[UC7] - Know the battle ranking

Name	Know the battle ranking
Actors	Educator and Student
Entry Condition	Actors logged into the system. Educator created the battle. Student is enrolled in battle.
Event Flow	**1.** Actors pushes the button on his battle. **2.** The system shows the screen of his battle. **3.** Actors presses the "Ranking" button. **4.** System shows screen with battle ranking.
Exit Condition	Actors sees ranking of teams.

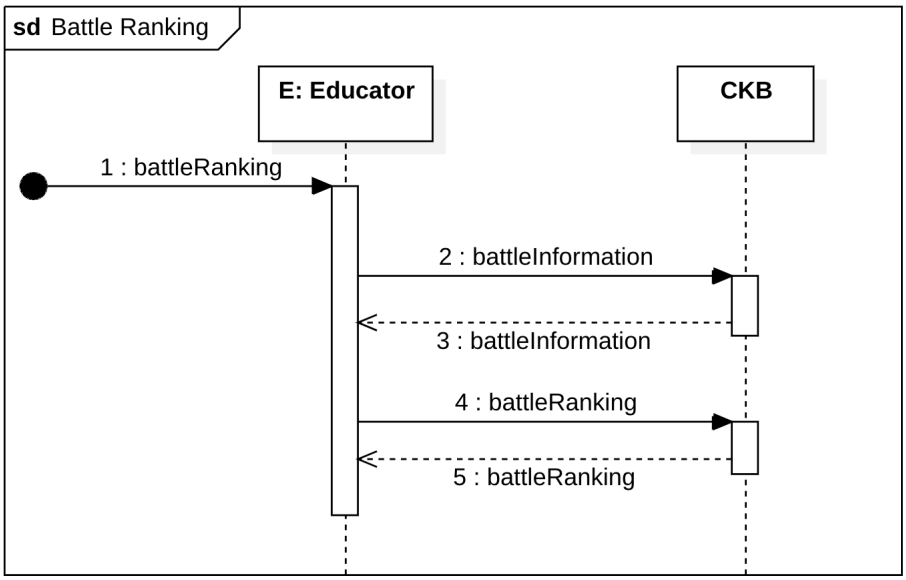


Figure 3.2.9: Battle Ranking Sequence Diagram, same for student

[UC8] - Know the tournament ranking

Name	Know the tournament ranking
Actors	Registered Users
Entry Condition	User logged into the system.
Event Flow	**1.** User select tournament to view. **2.** System shows them the tournament page. **3.** User press on the "Score Ranking" button. **4.** The system displays a page with the ranking and score of each student entered in the tournament.
Exit Condition	User take a look at the ranking.

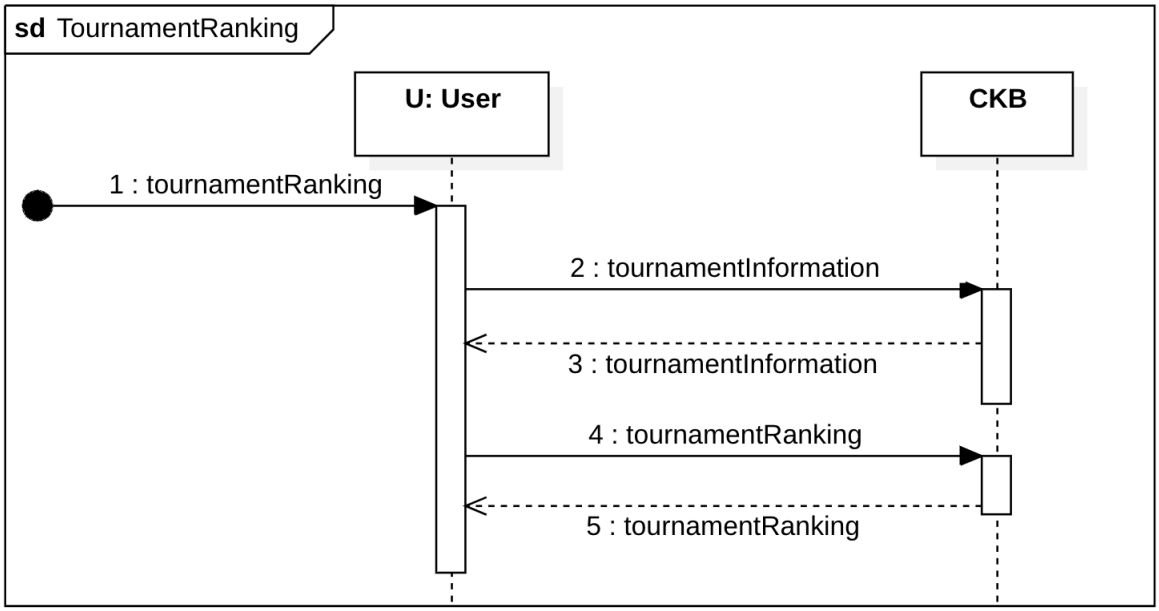


Figure 3.2.10: Tournament Ranking Sequence Diagram

[UC9] - Signing up for a tournament

Name	Signing up for a tournament
Actors	Student
Entry Condition	Student logged into the system.
Event Flow	**1.** Student accesses the system. **2.** System shows screen with available tournaments. **3.** Student presses on a tournament. **4.** System shows the tournament screen, with related information such as creator and registration deadline. **5.** Student presses the "Enrollment" button. **6.** System sends a confirmation message.
Exit Condition	Student successfully enrolled in tournament.

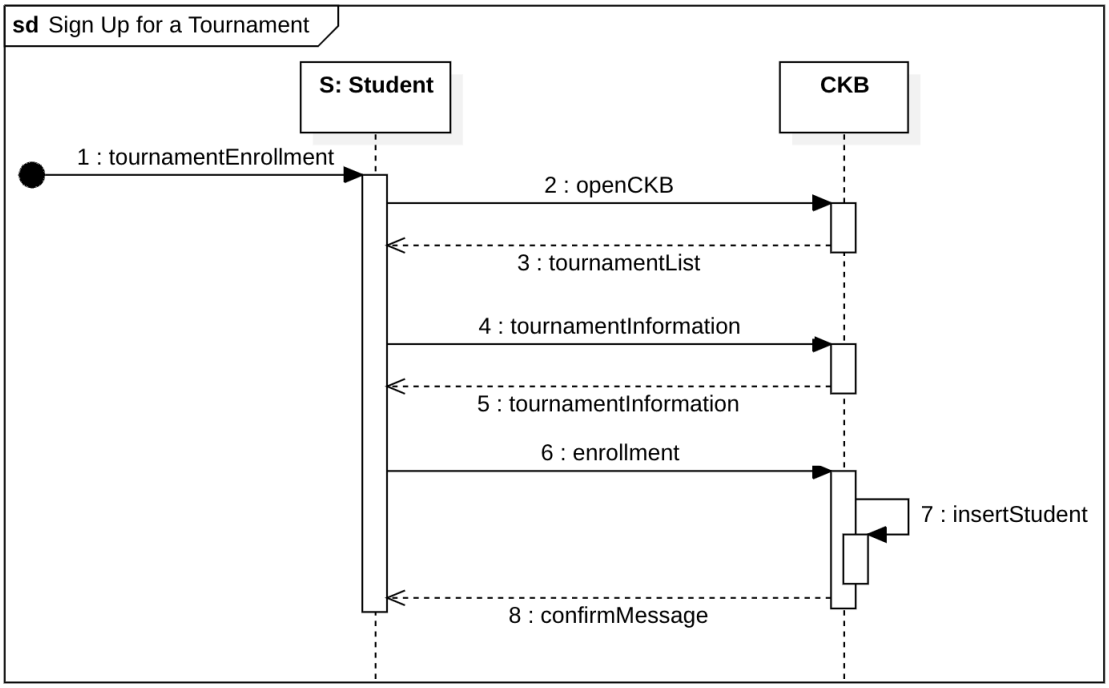


Figure 3.2.11: Sign Up for a Tournament Sequence Diagram

[UC10] - Unsubscription from the tournament

Name	Unsubscription from the tournament
Actors	Student
Entry Condition	Student logged into the system and student is enrolled in a tournament.
Event Flow	**1.** Student presses the tournament button where he participates. **2.** The system shows the tournament screen. **3.** Student presses on "unsubscribe" button. **4.** System removes student from tournament participants and ranking and sends a confirmation message.
Exit Condition	Student unsubscribe correctly.

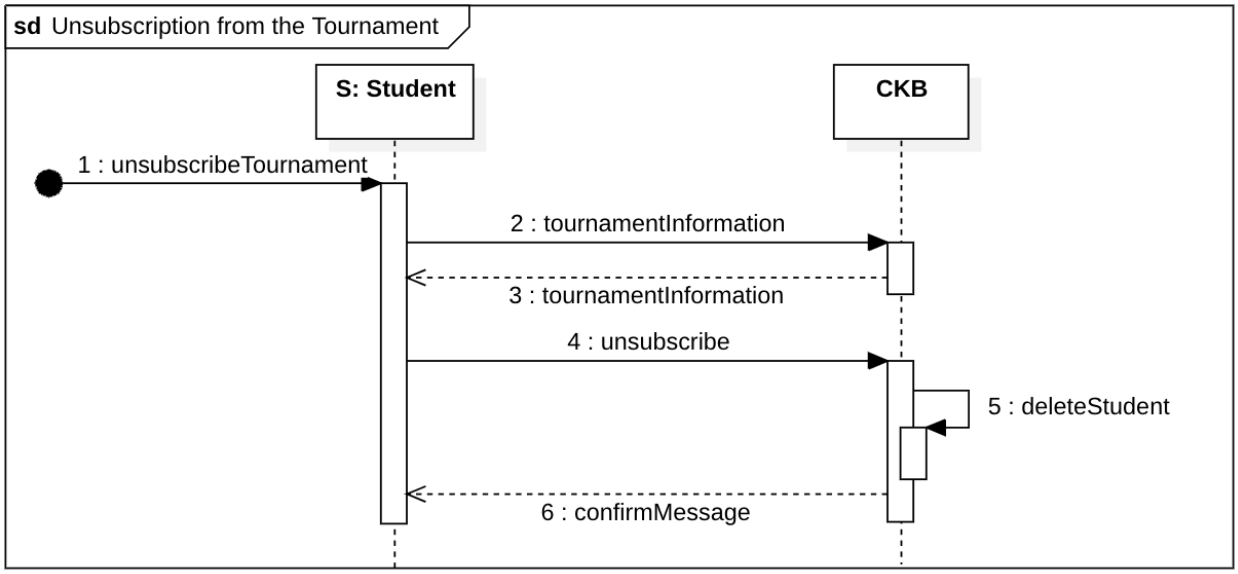


Figure 3.2.12: Unsubscription from the Tournament Sequence Diagram

[UC11] - View the list of battles

Name	View the list of battles
Actors	Student
Entry Condition	Student logged into the system and registered for a tournament.
Event Flow	**1.** Student presses on his tournament button. **2.** System shows the tournament screen, with all the battles. **3.** Student presses on a battle. **4.** The system shows the screen with all the information related to that battle.
Exit Condition	Student reviews the battles.

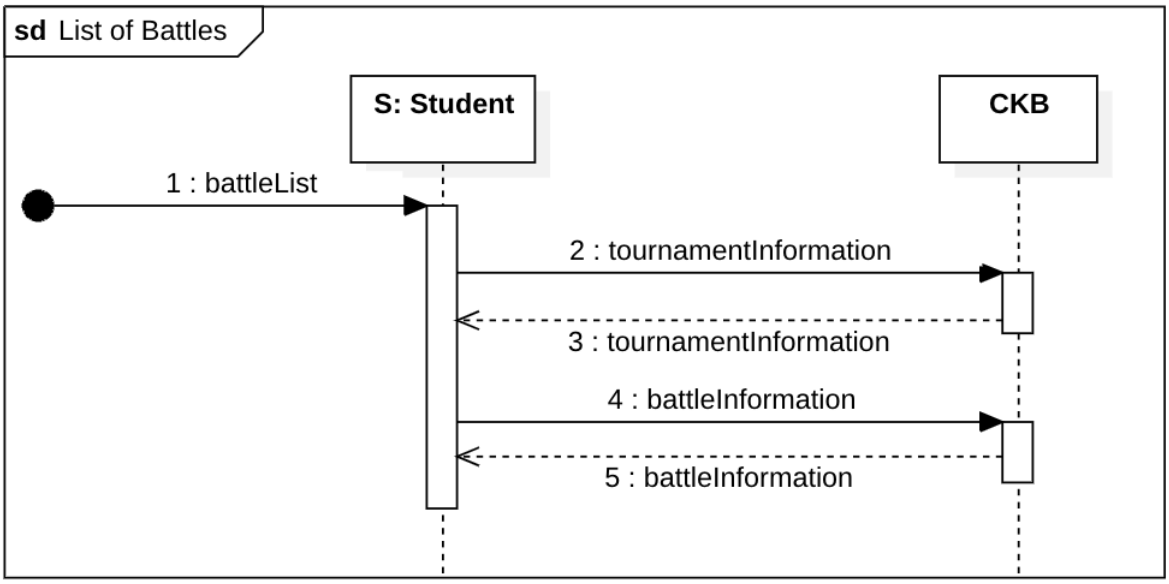


Figure 3.2.13: View the list of battles Sequence Diagram

[UC12] - Signing up for a battle

Name	Signing up for a battle
Actors	Student
Entry Condition	Student logged into the system and registered for a tournament.
Event Flow	**1.** Student selects the battle that has not yet begun. **2.** The system displays the battle page with inside: the relevant information, a field to enter the team name, a list of students with a check box next to it, and the "Join" button. **3.** The student fills in the text field and selects the students they wish to invite using a checkbox. **4.** Student presses the "Join" button. **5.** System notifies all students selected by the student to participate in his team. **6.** The system displays a confirmation message.
Exit Condition	The student is ready to work on the project.

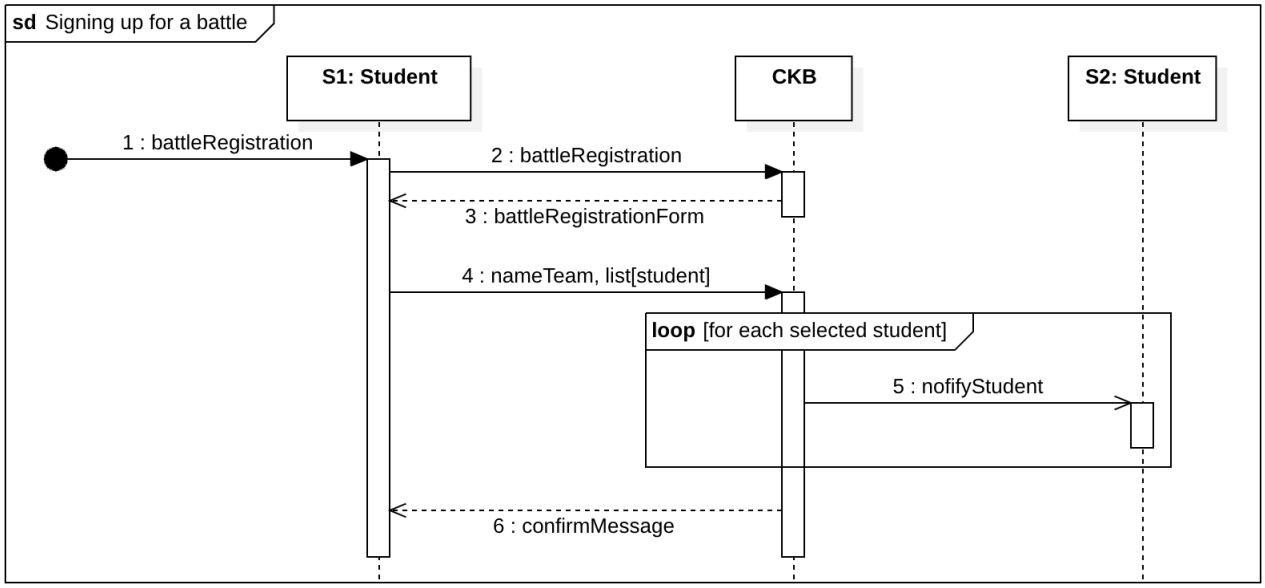


Figure 3.2.14: Signing up for a battle Sequence Diagram

[UC13] - Push effects on GitHub

Name	Push effects on GitHub
Actors	Student, GitHub and SonarQube
Entry Condition	Student logged into the system and registered to a battle.
Event Flow	**1.** Student makes a push on his GitHub repository of the new source. **2.** System is informed by GitHub that the team in which the student participates has made a new push to GitHub. **3.** System pulls on the student team's GitHub repository. **4.** System sends the retrieved source code to SonarQube to perform static analysis of the code and be evaluated on the aspects entered by the educator at the battle creation stage. **5.** System once received the data from SonarQube completes the evaluation, also calculating timeliness and functional aspects, scoring the project from 0 to 100. **6.** System updates the team's current score with the new score and updates the ranking.
Exit Condition	Student's team score and battle ranking is updated correctly.

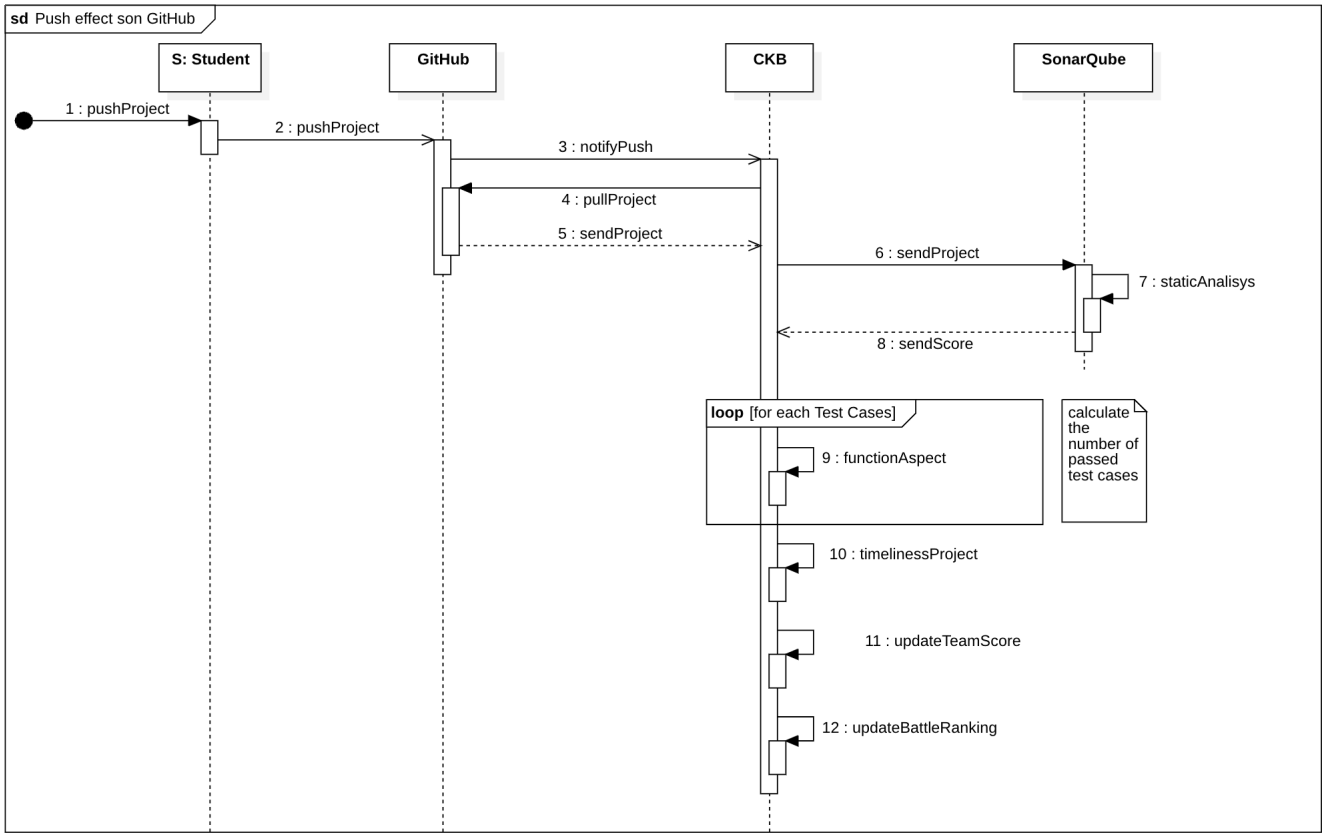


Figure 3.2.15: Push effects on GitHub Sequence Diagram

[UC14] - Participation in a battle by invitation

Name	Participation in a battle by invitation
Actors	Student
Entry Condition	Student logged into the system and registered for a tournament
Event Flow	**1.** The system sends a notification to the Student requesting to join a team. **2.** The Student requests notifications. **3.** The System displays the notifications. **4a.** The Student accepts the invitation. **5.** System sends a confirmation message.
Exit Condition	The Student is a member of the team.
Alternative	**4b.** The Student declines the invitation.
Exception	The battle has already begun. The system sends an error message.

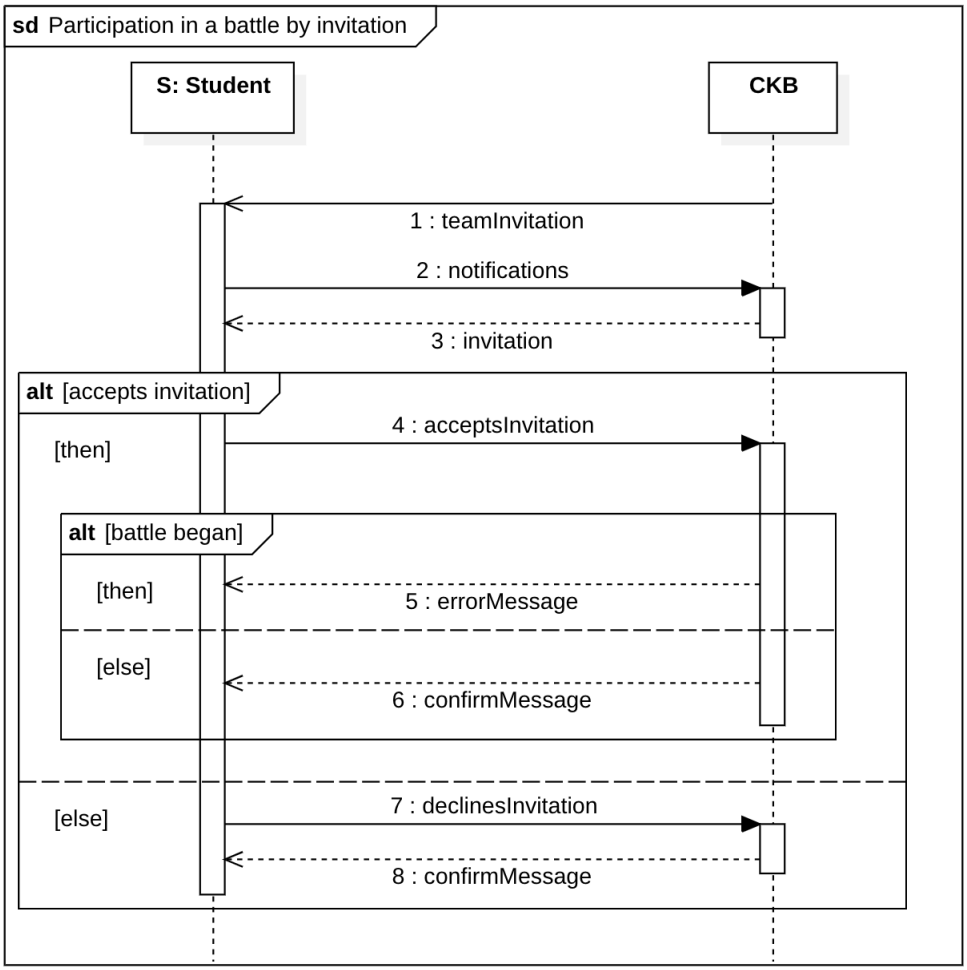


Figure 3.2.16: Participation in a Battle by invitation Sequence Diagram

[UC15] - Participation in a tournament by invitation

Name	Participation in a tournament by invitation
Actors	Educator
Entry Condition	Educator logged into the system.
Event Flow	**1.** The system sends a notification to the Educator asking to join a tournament. **2.** The Educator requests notifications. **3.** The System displays the notifications. **4a.** The Educator accepts the invitation. **5.** System sends a confirmation message.
Exit Condition	The Educator is a member of the tournament.
Alternative	**4b.** The Educator declines the invitation.
Exception	The tournament has already begun. The system sends an error message.

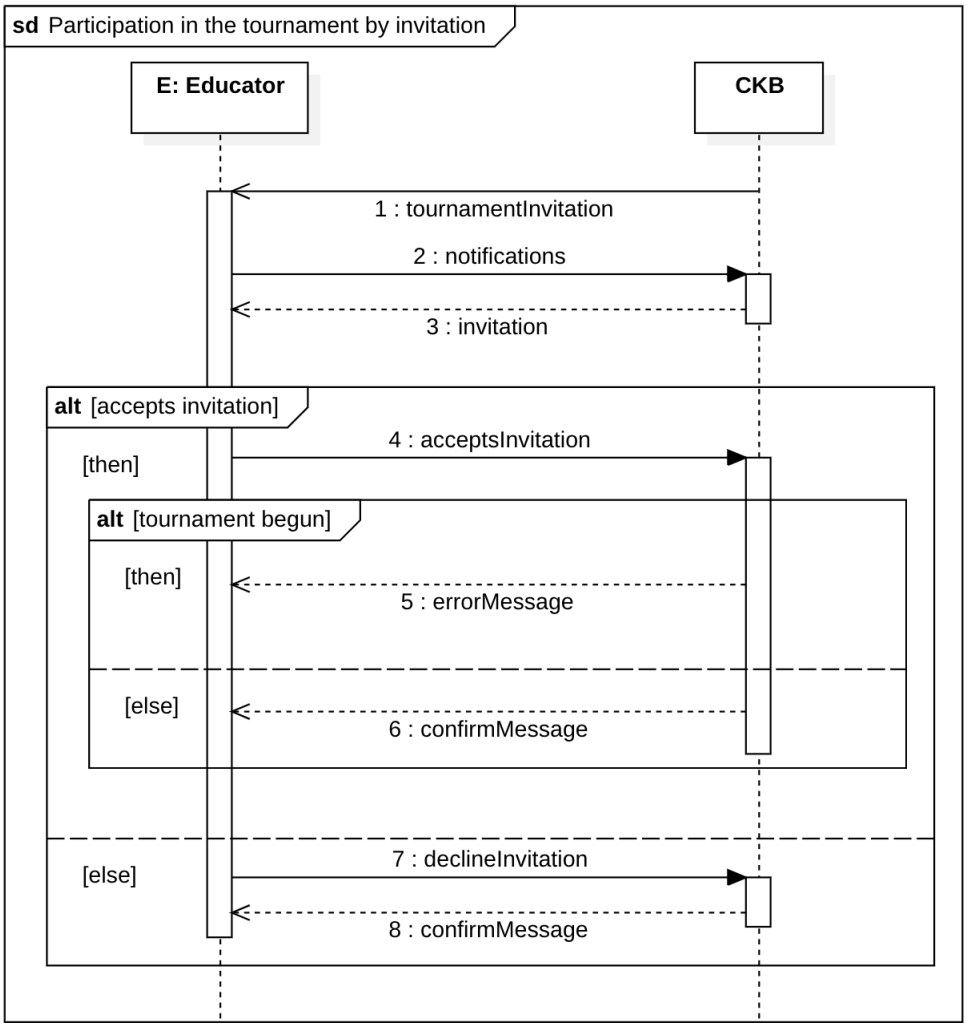


Figure 3.2.17: Participation in a Tournament by invitation Sequence Diagram

3.2.3 Requirements mapping

[G1] Educator administers the tournaments and decides and manages the battles within them.	
[R3] The system allows the educator to register by entering firstname, lastname, email, password and username.	[D2] Educator must have an internet connection.
[R4] The system allows registered educator to log in by entering username and password.	[D3] Educator be a very knowledgeable person in the area of software development.
[R5]The system allows the educator the ability to create of a tournament, allowing him or her to set a deadline for entries.	[D8] Educator must make sure that the CodeKata he intends to upload, we have no errors inside.
[R6] The system allows the educator to invite other educators to join the tournament.	
[R7] The system allows the educator who created the tournament, to close the tournament if all battles are completed.	
[R11] The system allows educators to see the list of all battles(unstarted, ongoing, and completed) in that tournament.	
[R15] The system allows all members of the platform to see the list of ongoing tournaments.	
[R16] The system allows all members of the platform to see the ranking of a tournaments, with the relative score associated with each student participating in the tournament.	
[R17] The system sends a notification to educators who have been invited by the educator who created the tournament to participate in the creation of battles within the tournament.	
[R18] System allows educators to accept or decline invitations to participate in the tournament.	

[R19] The system allows all educators participating in the tournament to create battles by uploading the CodeKata and specifying the description, entry deadline, final project delivery deadline, minimum and maximum number of students in each team, if desired manual evaluation, and qualitative aspects of the source to be evaluated. [R20] The system when the battle is over allows the educator who created the battle to see each team's final project. [R21] The system allows the educator to evaluate each team's final projects. [R31] The system allows the battle educator to see the current ranking. [R32] The system allows the battle educator to see the score of each team. [R34] The system, at the end of the battle, allows educators participating in tournament to see the final ranking.	

[G2] Students participate in tournaments by taking part in related battles, either individually or in groups.	
[R1] The system allows the student to register by entering firstname, lastname, email, password and username.	[D1] Student must have an internet connection.
[R2] The system allows registered students to log in by entering username and password.	[D4] Student must have a GitHub account.
[R8] The system allows students to register for the tournament by the deadline.	[D5] Students who are part of the team when they receive the main GitHub repository must fork it.
[R9] The system allows student to unsubscribe from the tournament.	[D6] Students who are part of the team must set up an automated workflow through GitHub Actions to inform the platform.
[R10] The system allows students enrolled in the tournament to see the list of all battles(unstarted, ongoing, and completed).	[D7] Students must have all the tools(e.g., IDE) to work on the project.
[R12] The system notifies all students registered on the platform to inform them of the creation of a new tournament.	[D9] Students participating in a battle are expected to follow a test-first approach.
[R14] The system when the educator closes the tournament, notifies all students in that tournament of the availability of the final ranking.	
[R15] The system allows all members of the platform to see the list of ongoing tournaments.	
[R22] The system allows students registered for the tournament to see all the specifics of a battle.	
[R23] The system allows students participating in the tournament to register for the battle by the deadline.	

[R24] The system allows students participating in the battle to create teams by inviting other students participating in the same tournament. [R25] The system when the battle enrollment expires must create the GitHub repository with the CodeKata in it. [R26] The system when the battle enrollment expires sends the GitHub repository link to all members of each team and the instructions to fork the repository and set up an automated workflow. [R29] The system allows students participating in the battle to see each team's current score. [R30] The system allows students participating in the battle to see the current ranking of the battle. [R33] The system, at the end of the battle, allows students participating in tournament to see the final ranking. [R35] The system, at the end of the battle, notifies all students participating in the battle of the availability of the final ranking. [R36] The system sends a notification to all students participating in a tournament when a new battle is created. [R37] The system sends a notification to students who have been invited by another student to join and form a team. [R38] The system offers the possibility for students to accept or reject invitations received from other students to form a team.	

[G3] The tournament implements a ranking derived from each student's individual score. This score is determined by aggregating the scores acquired by the student in each battle in which he took part.	
[R13] The system, at the end of each battle, updates each student's score by summing the scores obtained in each completed battle in which he participated, reflecting these changes in the overall tournament ranking.	[D1] Student must have an internet connection. [D4] Student must have a GitHub account. [D5] Students who are part of the team when they receive the main GitHub repository must fork it.
[R25] The system when the battle enrollment expires must create the GitHub repository with the CodeKata in it.	[D6] Students who are part of the team must set up an automated workflow through GitHub Actions to inform the platform.
[R27] The system must retrieve the source from the team repository after each push by the team.	[D7] Students must have all the tools(e.g., IDE) to work on the project.
[R28] The system assigns a integer score from 0 to 100 to the project retrieved from the GitHub repository, according to the following aspects: functional aspects (number of test cases passed), timeliness (difference between the last team commit and the battle registration deadline), and source quality level, the latter evaluated by a third-party tool Sonarqube.	[D9] Students participating in a battle are expected to follow a test-first approach.

3.3 Performance Requirements

3.3.1 Number of Users

In accordance with data published by other such applications, the CKB system expects to support about 20,000 users. So, we can consider that the system should be able to handle simultaneously the 50% of them.

3.3.2 Data Storage

From the data storage point of view, the CKB system should consider several sources of data:

- **Student's data**: about 19,000 students are expected
- **Educator's data**: about 1,000 educator are expected
- **Tournaments data**: 1,500 tournaments a year are expected
- **Battles data**: 12,000 battles a year are expected

3.3.3 Time response

CKB should ensure that the delay between an interaction with the system and its response should be no more than a couple of seconds, provided that the Internet connection is working correctly.

3.4 Design Constraints

3.4.1 Standards compliance

The system should respect all the laws regarding privacy and data treatment and exchange with third parties, to work in Europe, the system should respect the EU GDPR. In particular, a general description of the main principles that data should have in order to guarantee their privacy is given in Art. 5 of the GDPR document.

3.4.2 Hardware limitations

Following are all the hardware requirements:

- An Internet connection that can be Wi-Fi, 2G/3G/4G/5G or Ethernet.
- And any electronic device that can connect to the internet and open a web browser.

3.4.3 Any other constraint

There are not other constraints.

3.5 Software System Attributes

3.5.1 Reliability

With regard to reliability, since the system has no critical operations, the reliability rate can be about 1%, which is in line with common standards. However, it is crucial to consider that the system also depends on third-party components over which no direct control is exercised. Despite this, it is critical to emphasize that any failure of such external components should not result in total system failure and loss of data.

3.5.2 Availability

Regarding the availability aspect, the platform can accept a level of 99%. This implies that the system could be offline for a total period of 3.65 days per year. This choice is motivated by the fact that the platform does not provide critical or emergency services. This decision reflects consideration of the non-critical nature of platform operations, allowing the system a limited period of downtime without significant impact on daily operations.

3.5.3 Security

The system handles sensitive personal data of users, emphasizing the crucial importance of security. The store must be subject to strict security measures to prevent any external or internal threats. Passwords inside are subjected to cryptographic processes. For communication through the Internet, the application uses an encryption protocol in order to avoid the possibility of eavesdropping and traffic manipulation (sniffing and spoofing). These precautions aim to ensure protection from fraudulent attacks, preserving user privacy and ensuring data integrity and consistency.

3.5.4 Maintainability

The system should be structured into distinct modules, each dedicated to specific functionality, with the goal of conforming to defined standards. This approach is intended to facilitate maintenance, replacement and, possibly, extension of modules, ensuring efficient management of the system over time. Each feature implemented must be thoroughly documented, providing clear and understandable details to facilitate understanding and management of the code. The combination of a modular structure, standards compliance, comprehensive documentation and a rigorous testing routine is critical to ensure effective management, maintainability and sustainable growth of the system over time.

3.5.5 Portability

The system is designed to operate on any web browser (e.g., Google Chrome, Safari, Firefox) in order to maximize its portability. This approach is intended to offer maximum versatility, allowing users to exploit the system across a wide range of web browsers.

4 Alloy

This section is dedicated to the Alloy model of the CodeKataBattle software, included all the functions of the application and their most important constraints.
In this model, we want to prove that:

4.1 Signatures

```
//Definition of the Student
sig Student {
}

//Definition of the Educator
sig Educator {
}

//Definition of the Tournament
sig Tournament {
      creator: one Educator,
      invitedEducators: set Educator,
      participantEducators: set Educator,
      Battles: disj some Battle,
      participantStudents: set Student
}

//Definition of the Battle
sig Battle {
    creator: one Educator,
    participantTeams: set Team,
    evaluator: disj one Evaluator,
    has: disj one FinalRankingBattle
}

//Definition of the Team
sig Team {
    creator: one Student,
    members: set Student,
    push: disj set Project,
    lastPush: disj lone LastProject
}

//Definition of the Evaluator
sig Evaluator{
   configurator: one Educator
}

//Abstract definition of a project
abstract sig Proje {
```

```
    rated: one Evaluator,
    has: disj one Score
}

//Definition of a Project
sig Project extends Proje{

}

//Definition of a LastProject
sig LastProject extends Proje{
    manualEvaluation: lone Educator
}

//Definition of a Score
sig Score {
}

//Definition of a FinalRankingBattle
sig FinalRankingBattle{
    basedOn: set Score
}
```

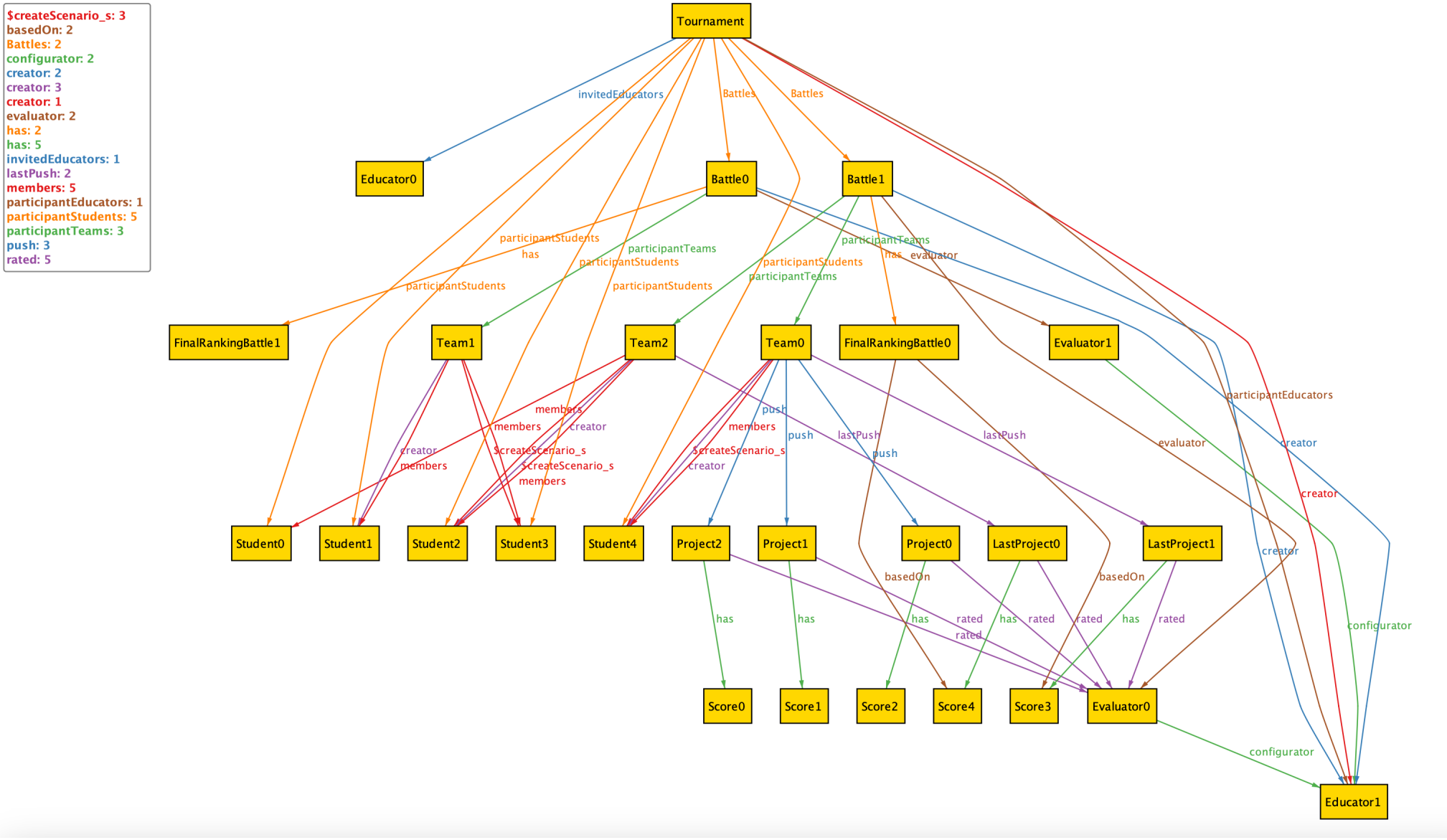
4.2 Facts

```
//All final battle rankings have a battle in which the following participate
fact EachFinalRankingBelongsToBattle{
all rb: FinalRankingBattle | one b: Battle | rb in b.has
}


// The final battle ranking is based on the scores of the last projects pushed
fact FinalRankingBasedOnScores{
all rb: FinalRankingBattle, b: Battle | rb in b.has implies rb.basedOn =
    b.participantTeams.lastPush.has
}


//Evaluator can only be configured by the creator of the battle where the team
    participates
fact EvaluatorConf{
   all ev: Evaluator, b: Battle | ev in b.evaluator implies ev.configurator =
       b.creator
}

//A team's pushed project is evaluated by the evaluator who has the battle
    where the team is enrolled
fact ProjectRated{
   all p: Proje, tm: Team, b: Battle | tm in b.participantTeams and (p in
       tm.push or p in tm.lastPush) implies p.rated = b.evaluator
}

//The educator who created the battle can decide whether or not to do manual
    evaluation of the teams' latest projects
fact SameManualEvaluationInBattle {
  all b: Battle |
      all lp1, lp2: LastProject |
        (lp1 in b.participantTeams.lastPush and lp2 in
            b.participantTeams.lastPush) implies
          lp1.manualEvaluation = lp2.manualEvaluation or no lp1.manualEvaluation
}

//This facts allows us to say that if one team has the last project delivered
    the others will have it too because the battle will be over
fact LastProjectConsistency {
 all b: Battle | #b.participantTeams.lastPush > 0 implies #b.participantTeams
     = #b.participantTeams.lastPush
}


//All the latest projects have a Team where the following are included.
```

```
fact LastProjectInTeam{
all p: LastProject | one tm: Team| p in tm.lastPush
}

//All the projects have a Team where the following are included.
fact ProjectInTeam{
all p: Project | one tm: Team| p in tm.push
}

//All the scores have a project where the following are included.
fact ScoreInProje{
all sc: Score | one pj: Proje | sc in pj.has
}

//All the Evaluator have a Battle where the following are included.
fact EvaluatorInBattle{
all e: Evaluator | one b: Battle| e in b.evaluator
}

//All the Battle have a Tournament where the following are included.
fact BattleInTournament{
all b: Battle | one t: Tournament| b in t.Battles
}

//All the Team have a Battle where the following are included.
fact TeamInBattle{
all tm: Team | one b: Battle| tm in b.participantTeams
}

//A battle in the tournament implies that the creator of the battle must be
    part of the tournament educators and the participants of the battle must
    be part of the tournament
fact EachBattleBelongsToOneTournament {
  all t: Tournament, b: Battle |
    (b in t.Battles) implies (b.creator in t.participantEducators and
        b.participantTeams.members in t.participantStudents)
}

//A team can only participate in one battle at a time
fact TeamBattaglia {
all disj b1, b2: Battle, tm: Team | tm in b1.participantTeams implies tm not
    in b2.participantTeams
}

// Each team has some students within
fact StudentCanParticipateInOneTeam {
  all tm: Team | some s: Student | s in tm.members
}
```

```
//A student may participate in one team at a time
fact NoDuplicateStudentAcrossTeam {
  all disj t1, t2: Team, s: Student | s in t1.members implies s not in
      t2.members
}


//The creator educator is part of the tournament participants
fact CreatorPartecipate {
  all t: Tournament, e: Educator | e in t.creator implies e in
      t.participantEducators and e not in t.invitedEducators
}

//Only invited eductors can be participants
fact OnlyInvitedEducatorsCanParticipate {
  all t: Tournament, e: Educator | e in t.participantEducators and e not in
      t.creator implies e in t.invitedEducators
}


//All educators participating in the tournament create at least one battle
fact EducatorParticipatesInBattleCreateAtLeastOneBattle {
  all e: Educator, t: Tournament |
    e in t.participantEducators implies some b: Battle | b in t.Battles and
       b.creator = e
}


//The creator of a team is part of the members of the same team
fact CreatorIsMemberOfTeam {
  all tm: Team | tm.creator in tm.members


//A general scenerio of the system
pred createScenario {
   #Educator = 2

   #Student = 5

   #Tournament = 1

   #Battle = 2

   #Team = 3

   #LastProject > 1

   #Project > 1
}
```

```
run createScenario for 5


}
```


$createScenario_s: 3
basedOn: 2
Battles: 2
configurator: 2
creator: 2
creator: 3
creator: 1
evaluator: 2
has: 2
has: 5
invitedEducators: 1
lastPush: 2
members: 5
participantEducators: 1
participantStudents: 5
participantTeams: 3
push: 3
rated: 5
Tournament
Educator0
Battle0
Battle1
FinalRankingBattle1
Team1
Team2
Team0
FinalRankingBattle0
Evaluator1
Student0
Student1
Student2
Student3
Student4
Project2
Project1
Project0
LastProject0
LastProject1
Score0
Score1
Score2
Score4
Score3
Evaluator0
Educator1
invitedEducators
Battles
Battles
participantStudents
has
participantTeams
participantStudents
participantStudents
participantStudents
participantTeams
participantTeams
has
evaluator
participantStudents
participantTeams
members
members
creator
creator
members
$createScenario_s
$createScenario_s
members
members
creator
$createScenario_s
push
push
lastPush
push
lastPush
evaluator
participantEducators
creator
creator
creator
basedOn
basedOn
has
has
has
rated
rated
has
rated
rated
has
rated
configurator
configurator

In this schematic representation of the system, we can observe a generic tournament instance with connections to several relationships. From above we notice that there is a tournament instance connected to several relationships. The first relationship highlighted is invitedEducator, which indicates that educator0 has been invited to participate in the tournament. However, since there is no participantEducators relationship, we can infer that the educator declined the invitation to participate.

Next, we notice two relationships, participantEducators and creator, that link the tournament to the educator who created it. Also, we see that the tournament contains two battles and has 5 students enrolled. The two instances of the battles are different from each other. For example, battle0 has only team1 as a participant, which includes students Student1 and Student3. This team has not yet done any project push, as indicated by the lack of project-related reports.

Next, we observe battle1, which contains team2 and team0. Both teams have pushed their projects, including the last delivered projects. Each project is evaluated by the evaluator configured by the educator who created the battle. A manual evaluation by the battle creator of the last project is also possible, but in this case it did not happen. Finally, we note that the final battle ranking1 is based on the scores obtained from the last projects delivered by the two teams.

5 Time Spent

Raffaele Russo

Chapter	Effort (in hours)
1	8
2	13
3	25
4	12

Biagio Marra

Chapter	Effort (in hours)
1	8
2	13
3	21
4	16

6 References

- Statecharts made with: draw.io
- Diagrams made with: StarUML
- Mockups made with: moqups.com
- Creating and verifying Alloy models with: Alloy Tools.