



POLITECNICO DI MILANO

Software Engineering II

CodeKataBattle

Requirements Analysis and Specification Document

Version 2.0

Federico Albertini

Aleksandro Aliaj

Leonardo Betti

December, 2023

1. Introduction	4
1.1 Purpose	4
1.2 Goal	4
1.3 Scope	4
1.3.1 World Phenomena	5
1.3.2 Shared phenomena	5
1.4 Definitions, Acronyms, Abbreviations	6
1.4.1 Definitions	6
1.4.2 Acronyms	6
1.4.3 Abbreviations	6
1.5 Revision history	6
1.6 Reference Documents	6
1.7 Document Structure	7
2. Overall Description	8
2.1 Product Perspective	8
2.1.1 Scenarios	8
2.1.2 Domain class diagram	10
2.1.3 Statecharts	11
2.2 Product Functions	14
2.2.1 Sign Up and Login	14
2.2.2 Manage the tournament and the battle	14
2.2.3 Create teams for participating to a battle	14
2.3 User characteristics	15
2.3.1 Admin (Educator)	15
2.3.2 Student	15
2.3.3 Collaborator (Educator)	15
2.4 Assumptions, dependencies and constraints	16
2.4.1 Regulatory policies	16
2.4.2 Domain Assumptions	16
3. Specific Requirements	17
3.1 external interfaces	17
3.1.1 User Interfaces	17
3.1.2 Hardware Interfaces	21
3.1.3 software interfaces	21
3.2 Functional requirement	21
3.2.1 Use case diagrams	24
3.2.2 Use cases	26
3.2.3 Sequence diagrams	33
3.2.4 Requirements mapping	39
3.3 Performance Requirements	42
3.4 Design Constraints	42
3.4.1 Standard Compliance	42
3.4.2 Hardware limitations	42

3.5 Software System Attributes	43
3.5.1 Reliability	43
3.5.3 Security	43
3.5.4 Usability	43
3.5.5 Performance	43
4. Alloy	44
4.1 Signatures	44
4.2 Facts	48
4.3 Show	52
5 Effort	54
6. References	55

1. Introduction

1.1 Purpose

The purpose of this project is to deliver the Requirement Analysis and Specification Document (RASD) for CodeKataBattle. The RASD describes to the fullest extent the mentioned system in terms of both functional and nonfunctional requirements, and furthermore specifies the constraints that it needs to respect as well as its boundaries. On one hand this document is addressed to the developers who will implement the requirements. On the other hand, it is intended as a contractual basis for the users. Therefore it does not contain any specific terminology if not properly defined.

1.2 Goal

- [G1] Educator can manage a tournament
- [G2] Educator can manage a battle inside a tournament
- [G3] Student can participate to battles within tournaments
- [G4] Educator can manage the creation of badges

1.3 Scope

Nowadays, programming skills are fundamental in various contexts. CodeKataBattle (CKB) is a platform designed to make learning and improving coding skills enjoyable and effective. Targeted at students, educators, and coding enthusiasts, CKB offers a collaborative learning environment. Participants can engage in 'Kata Battles,' small coding exercises created by educators to reinforce coding techniques.

CKB is accessible through a web-based platform, fostering collaboration and healthy competition among participants. Educators play a crucial role in crafting challenges and evaluating submissions. The scoring system incorporates both automated assessments, including test-driven development (TDD) practices, and manual evaluations by educators.

Participants have the opportunity to earn badges and achievements based on their performance in tournaments. The platform integrates with GitHub, allowing users to submit their projects and emphasizing real-world application and industry-standard tools.

Overall, CodeKataBattle provides an immersive and fun experience, promoting continuous learning and creative problem-solving in the field of programming.

1.3.1 World Phenomena

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

[W1] Student decide to participate a tournament

[W2] Student enter to the website

[W3] Student work on the task

[W4] Educator decide the task of the battle

1.3.2 Shared phenomena

World controlled:

[SP1] Educator create a Battle

[SP2] Educator create a Tournament

[SP3] Student subscribe to a Tournament

[SP4] Student subscribe to a Battle

[SP5] Educator upload the CK

[SP6] Educator can set minimum and maximum number of students per team

[SP7] Educator can set a registration deadline

[SP8] Educator can set a final submission deadline

[SP9] Educator can set additional configurations for scoring

[SP10] Students use the platform to create teams for that battle

[SP11] Students use the platform to invite other student for a team

[SP12] Student push a new commit into the main branch of their repository

[SP13] Educator can assign a score a personal score

[SP14] Educator can create new badges, new rules and new variables

[SP15] Educators have the option to manually evaluate the work done by students, assigning personal scores.

Machine controlled:

[SP16] Notifications regarding battle and tournament updates, including final rankings, are sent to students and educators.

[SP17] The platform ensures students are informed of upcoming tournaments.

[SP18] The platform dynamically updates battle scores of teams in real-time with each new commit.

[SP19] The platform establishes a tournament rank determined by battle scores.

[SP20] After each battle, the personal tournament score is promptly updated for every student.

[SP21] Tournament ranking of students are consistently updated based on battle scores.

[SP22] Badges are assigned by the platform.

[SP23] Students receive notifications when new battles are published.

1.4 Definitions, Acronyms, Abbreviations

1.4.1 Definitions

- Admin: educator who creates a tournament, and then battles.
- Collaborator: educator who participates in a tournament, to help create new battles.

1.4.2 Acronyms

- RASD: Requirement Analysis and Specification Document.
- UI: User Interface

1.4.3 Abbreviations

- [CK] - code kata(it consists in the brief textual description and a software project with build automation scripts with test cases, but without implementation)
- [CKB] - CodeKataBattle platform
- [TDD] - Test driven development

1.5 Revision history

- Version 1.0
- Version 2.0
 - Grammatical errors have been corrected
 - Functional requirement indexes have been shifted
 - The alloy has been corrected by removing some "one" cardinalities in front of some attributes

1.6 Reference Documents

This document is strictly based on:

- The specification of the RASD assignment of the Software Engineering II course, held by professor Elisabetta Di Nitto A.Y 2023/2024;
- Slides of Software Engineering 2 course on WeBeep;
- official link of <https://www.codewars.com/> to get more information about battle

1.7 Document Structure

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

1. **Introduction**: it's focused on the reasons and the goals that are going to be achieved with project's development;
2. **Overall Description**: it's a high-level description of the system by focusing on the possible phenomena concerning the system 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 and for each component of the group;
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 to a Tournament:

Alfredo, an enthusiastic programmer eager to enhance his coding skills, seeks a platform that can both test and improve his programming knowledge. At the suggestion of his teacher, he discovers CodeKata, a new platform designed to challenge users with a series of interesting programming challenges. Excited to start, Alfredo registers on CodeKata. Upon logging in, he explores the array of open tournaments available on the platform. Aiming to focus on a specific programming topic, he selects a tournament that aligns with his learning goals and knowledge and joins the competition. (As the registration deadline expires.) Once the deadline expires, he receives a notification containing a comprehensive list of battles he can participate in, along with additional essential information.

2. Educator creates a Tournament:

Francesco, an educator using CodeKata as a valuable teaching resource, is excited about introducing a new programming puzzle to challenge his students. To facilitate this, he navigates to the CodeKata portal, where he initiates the creation of a new tournament. In this process, Francesco specifies essential details such as the enrollment deadline and the targeted programming topic, he also creates the badges that at the end of the tournament they will be awarded to the students. Upon completing the tournament setup, CodeKata's notification system promptly informs all enrolled students about the exciting new challenge. As Francesco begins to think about the various battles To streamline the process, he grants permission to other colleagues, empowering them to contribute to the creation of the battles.

3. Educator creates a Battle:

Logan is a university professor who creates tournaments on CodeKataBattle every semester to allow its students to remain trained during all Java topics they face. Logan, after the setup of the tournament, creates a series of battles with specific coding tasks for the students and at the first lesson explains to them that the students who gain points on the platform will also have bonus points during the exam they will have to take in the classroom. Students use the platform to form teams for the upcoming battles, inviting colleagues of their course, ensuring they meet the required minimum and maximum group sizes. Logan carefully crafted each battle to contain a brief textual description and a Gradle project with build automation scripts that contains a set of test cases that the program must pass, but without the program implementation. In the end has configured scoring parameters for the battles, defining aspects like test case pass rates and quality level expectations, but surely after battles will be completed, he will review the student submission, possibly assigning personal scores based on manual evaluation.

4. Students upload battle's solution

Emily, a passionate self-taught coder, discovers the CodeKataBattle platform online and is excited about the opportunity to sharpen her programming skills. Intrigued by the diverse coding challenges presented on the platform, Emily decides to take on the CodeKataBattle challenge independently. Motivated by the prospect of skill enhancement and the thrill of solving real-world coding problems, Emily explores the available tournaments. She finds a particularly interesting one and joins, ready to tackle the series of battles set up by the platform. Choosing to work in a solo team, Emily actively engages with the coding tasks, pushing new commits into her GitHub repository as she progresses. Emily is driven by the motivation of self-improvement and can't wait to see the scores. The CodeKataBattle platform continuously evaluates her work, considering factors such as functional aspects, timeliness, and code quality.

5. Educator reviews battle

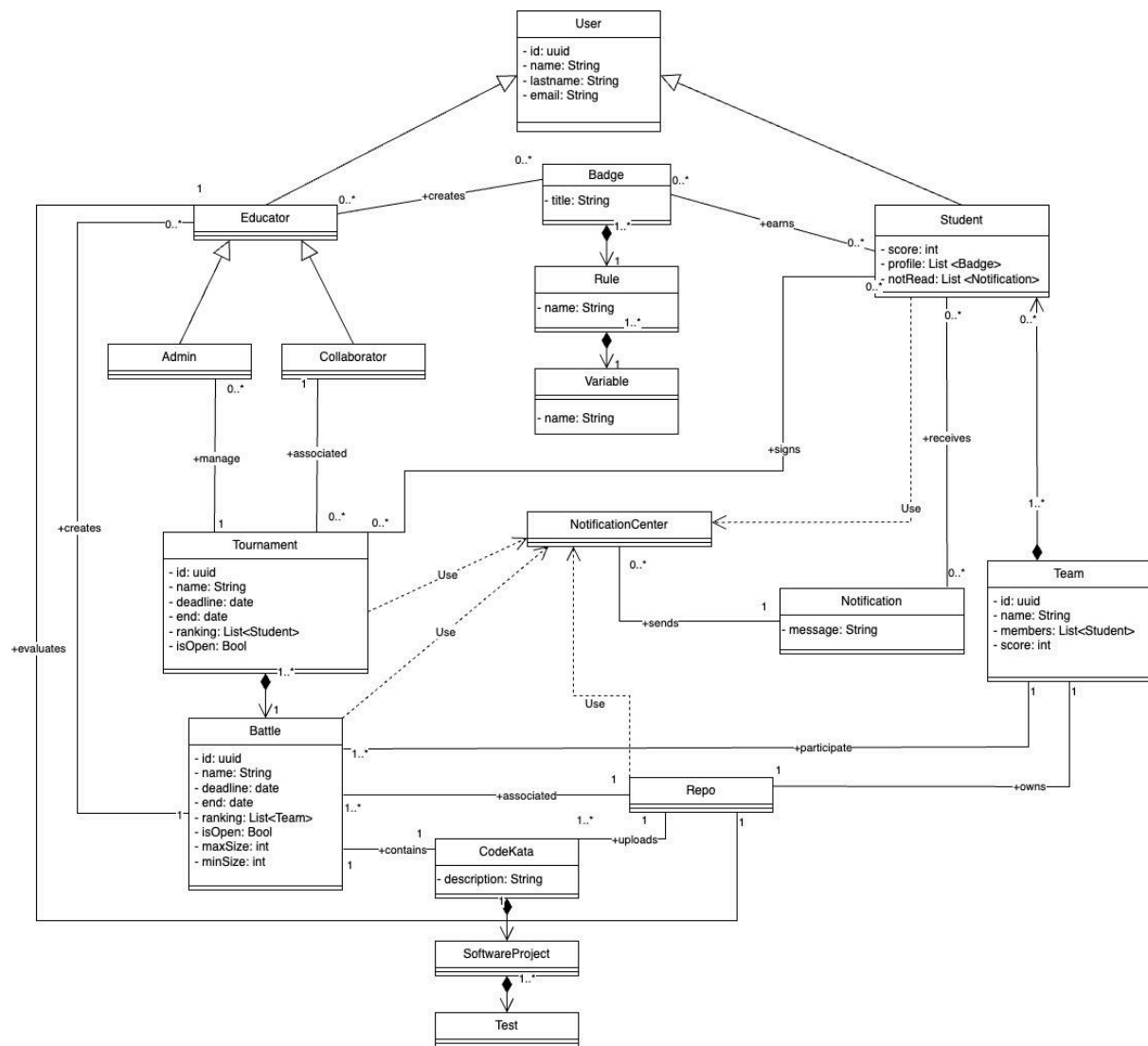
As the battle's deadline draws to a close, James, the dedicated educator, receives a notification promptly informing him that the

battle has concluded. Eager to evaluate the outcomes, James navigates to the battle's dedicated page on the platform. Here, he finds a comprehensive display of all the katas submitted by the participating teams. The platform's automated testing system has run tests on each submission, generating relative scores based on predefined criteria. James reviews the automated test results, gaining insights into the performance of each team. He has the flexibility to adjust the points assigned to each team. After fine-tuning the scores, James updates the battle's standings.

2.1.2 Domain class diagram

In Figure (2.1) is represented the Domain class diagram related to CodeKataBattle. It contains all the elements of the domain in which the system operates and the interaction between such elements. In the following are described the most relevant parts of the diagram in order to ease the understanding of the domain.

- There are two types of User: Educator and Student. They have in common the fact that they use the system and all of them have an id, name, lastName and email.
- An Educator can be an Admin or a Collaborator of a Tournament and has the authority to create Battles in the context of that Tournament.
- Teams of students participate in Battles, and work on them using a GitHub repository (Repo), directly associated with battles.
- Each Repo belongs to one Team and is associated with a specific Battle.
- Every Battle has a CodeKata, which consists in the brief textual description and a software project with build automation scripts and test cases, but without implementation.
- A Badge is defined by an Educator when he creates a Tournament and is assigned to one or more students depending on the rules checked at the end of the Tournament.

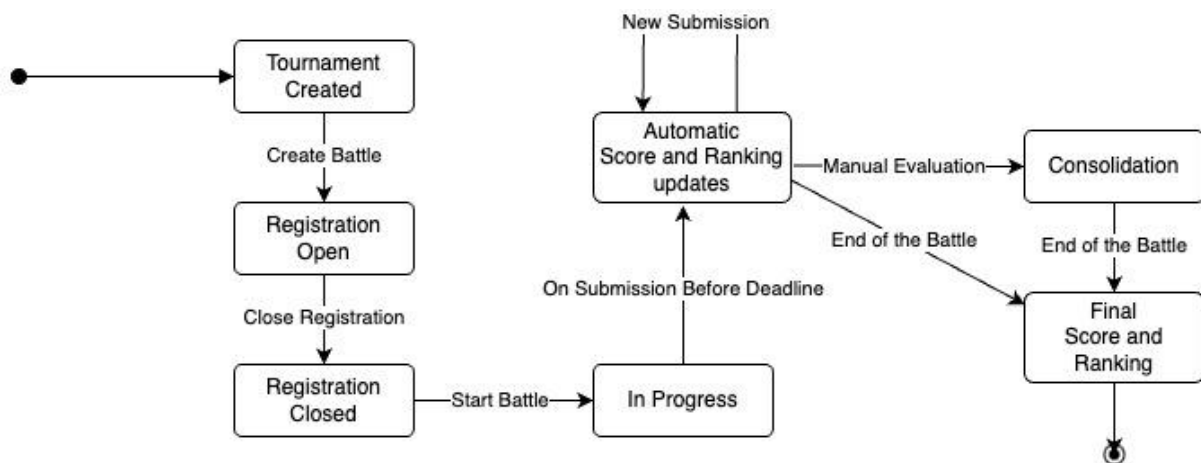


(Figure 2.1) Domain class diagram

2.1.3 Statecharts

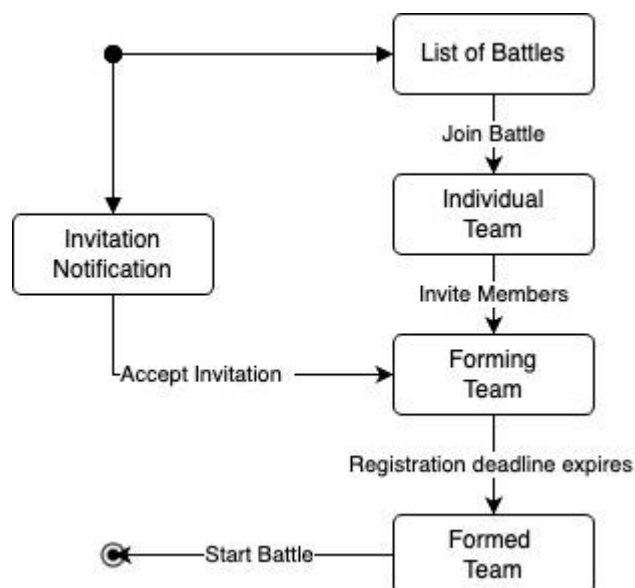
This section delineates key processes within the CKB platform, It captures the dynamics of the Battle process, illustrating the evolution from submission to evaluation and finalization. The Team Formation process illuminates how teams coalesce, either through invitations or individually, fostering collaborative coding environments. Additionally, the Notification Mechanism process showcases the system's communication, ensuring students are seamlessly apprised of updates, maintaining an engaging and informed user experience.

Battle process



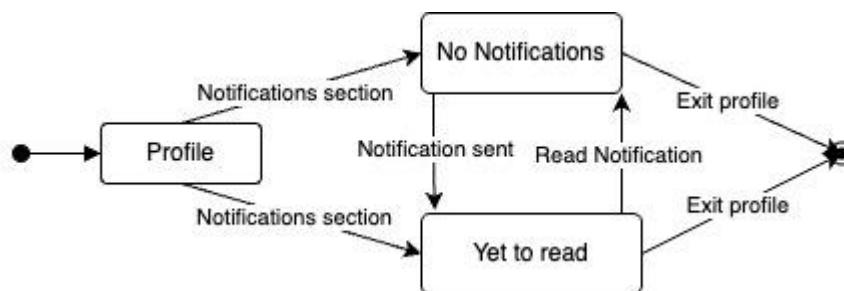
The statechart begins when an educator creates a new tournament, setting the stage for upcoming battles. A registration period is opened, allowing students to enroll in the battles, after a specified period, the registration closes, marking the deadline for students to join the battles. With registrations closed, battles are in progress. Students engage in coding exercises, striving to complete the challenges. As students submit their solutions, the platform automatically evaluates the submissions in real-time, considering factors like test case success and timeliness. If manual evaluation is required, educators enter the scene, carefully assessing the quality of submissions beyond automated checks and their scores are integrated into the overall assessment. The battle scores are published, giving students and educators insights into individual and team performances.

Team Formation



The process of team formation for a battle begins with two distinct paths, reflecting the flexibility of either starting from an invitation or from exploring the list of battles. A student enrolled in a tournament can receive notification to be directly added to a team or can enter the system and explore the list of ongoing battles. If a student finds an interesting battle, moves into the "Individual Team" state, indicating their intention to participate. The system guides the student through the process of forming a team, through the transition to "Forming Team". A student in this state can also decide to not invite other students of the tournament and remain in a team without other members. Finally, users reach the "Formed Team" state after successfully forming a team, whether through invitations or the initial individual team approach. The team is now ready to participate in the selected battle.

Notification



The CKB platform ensures students stay informed and engaged through various notifications. Students can see their notifications in a “notification section” on their profile, they can receive updates on upcoming battles, tournaments, and new tournaments. Real-time notifications reflect battle progress, including dynamic rank changes. After battles, students are promptly notified of their personal tournament scores and any achieved badges. Educators close tournaments, triggering notifications, and the system keeps participants updated about their performance, fostering an interactive and engaging learning environment.

2.2 Product Functions

2.2.1 Sign Up and Login

These functions will be available to all the users. The sign up functionality allows users to register themselves, selecting if they are students or educators, to the system.

In particular, each user will be asked to provide an email (the email will be the username of the user) and a password. Then a verification email is sent to the user.

Once the user has confirmed the email, the user is ready to use the system.

2.2.2 Manage the tournament and the battle

This function will be available to educators.

These functionalities will allow educators to create tournaments and battles

In his dashboard educator can see all the created tournaments and all the battles for each of them. Furthermore educators can see the list of participants enrolled in a battle or in a tournament.

Once battle is created, educators can select the setting of battle (e.g. num of player per team, additional configurations for scoring, deadline submission ecc.) and upload the CK.

2.2.3 Create teams for participating to a battle

This function will be available for the students.

Once the students have been registered they can search for a tournament choosing which language they want to improve.

At this point they have the option to engage in a battle either individually, forming a solo group, or extend invitations to other students to join their team.

2.3 User characteristics

2.3.1 Admin (Educator)

Educators, as primary users, manage the tournament by inviting collaborators. They hold responsibilities in content oversight, ensuring alignment with tournament objectives. Educators may perform administrative tasks, such as setting schedules and defining rules. With authority to review and approve changes, they play a pivotal role in maintaining the quality and coherence of the tournament's overall content.

2.3.2 Student

Students, upon signing into the platform, can explore available tournaments and decide to participate. Once subscribed to a tournament, they have the option to compete in battles individually or form a team with other students within the same group. Students can take the initiative to invite students to join their team or accept invitations from other students to collaborate in battles.

2.3.3 Collaborator (Educator)

Collaborators are users invited by Admin educators to actively contribute to the creation of tournament's battles. Their roles encompass content creation, including development, descriptions, and rule formulation. Some collaborators may specialize in reviewing, ensuring coherence and adherence to guidelines. Additionally, collaborators may serve as valuable feedback providers, suggesting improvements to enhance the overall quality of the battles content. In some cases, a user may act as both an Educator and a Collaborator, but for a different tournament. This dual role allows individuals to participate actively in one tournament while overseeing another.

2.4 Assumptions, dependencies and constraints

2.4.1 Regulatory policies

The regulatory guidelines governing the CodeKataBattle (CKB) platform underscore the paramount importance of ensuring user data privacy, robust security measures, and adherence to intellectual property rights. Clear and transparent evaluation criteria are communicated to students, fostering fair competition. A responsible and secure user experience is maintained through effective communication and thorough testing practices.

The platform's alignment with educational standards and adherence to relevant regulations further solidifies its suitability for educational institutions.

2.4.2 Domain Assumptions

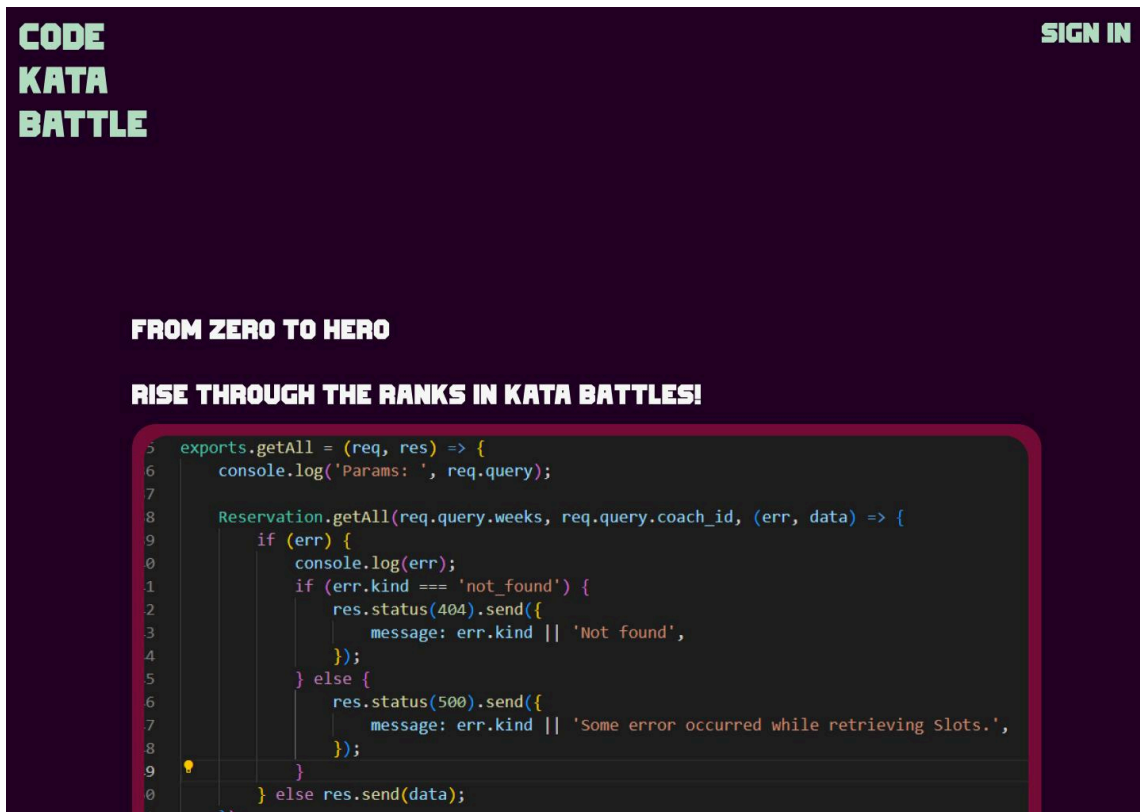
The following are the assumptions made for the CodeKataBattle (CKB) domain. These assumptions are fundamental conditions or properties that the system relies on. Verification of these assumptions ensures the correct functioning of the CKB platform.

- [D1]: Educators and students have devices with internet connectivity.
- [D2]: Students adhere to the registration and submission deadlines for battles.
- [D3]: Students adhere to the registration and submission deadlines for battles.
- [D4]: Information provided about battles and tournaments is accurate.
- [D5]: Reliable connections exist between the CKB platform and GitHub.
- [D6]: Students have the required equipment for coding exercises.
- [D7]: Notifications are delivered promptly within 10 seconds.
- [D8]: Time left and scoring details are consistent with the platform display.
- [D9]: Educators responsibly input all required information about tournaments and battles.

3. Specific Requirements

3.1 external interfaces

3.1.1 User Interfaces



(Figure 3.1.1) Home page

EMAIL

PASSWORD

SIGN IN

[FORGOT PASSWORD?](#) [SIGN UP](#)

(Figure 3.1.2) Sign in

A sign-up form with a dark purple background. The form is a light purple rounded rectangle containing five input fields and a button. The fields are labeled NAME, LASTNAME, EMAIL, PASSWORD, and REPEAT PASSWORD. The button is labeled SIGN UP.

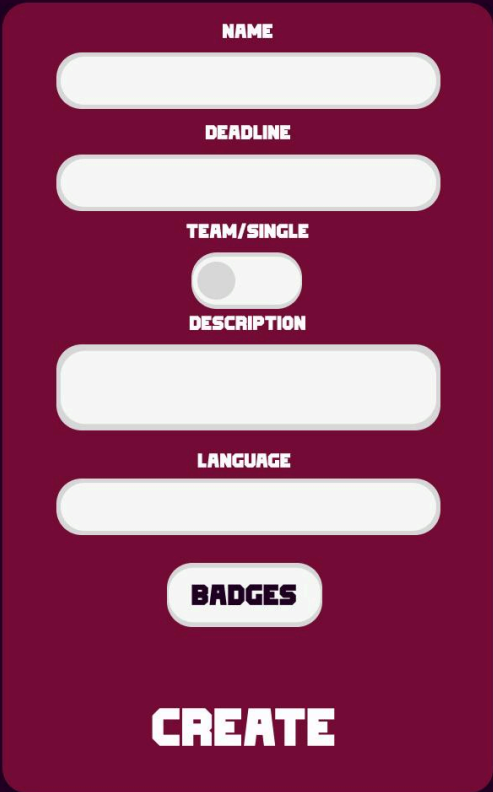
(Figure 3.1.3) Sign up

CODE
KATA
BATTLE

TOURNAMENTS

NAME	DEADLINE	TYPE	LANGUAGE	DESCRIPTION	
Code Clash Carnival	10/12/2023	Single	C++	Join us under the virtual big top for the Code Clash Carnival! Engage in a coding spectacle where programmers showcase their skills in a variety of challenges. From algorithm acrobatics to code circus stunts, this carnival is a celebration of coding excellence.	JOIN
Algorithm Adventure Extravaganza	10/1/2024	Teams	Java	The Algorithm Adventure Extravaganza invites you to navigate through the realms of complex algorithms and unravel coding mysteries. Prepare for a journey that will test your problem-solving prowess and lead you to undiscovered coding treasures.	JOIN
Byte Brawl Bonanza	15/12/2023	Single	Javascript	Get ready for an epic showdown in the Byte Brawl Bonanza! Programmers from around the world will engage in a byte-sized battle, competing in fast-paced coding skirmishes. Only the strongest algorithms will emerge victorious in this byte-packed coding extravaganza.	JOIN
Hackathon Fiesta Frenzy	20/3/2024	Single	Assembly	It's time to party with code at the Hackathon Fiesta Frenzy! Join the coding celebration where participants dance through challenges, salsa with syntax, and tango with technology. Bring your A-game, and let the coding fiesta begin!	JOIN
Debugging Derby Delight	19/2/2024	Teams	C++	Saddle up for the Debugging Derby Delight! Enter the coding arena where programmers will race against time to debug their way to victory. It's a thrilling competition of speed, precision, and debugging dexterity.	JOIN
Bit Battle Royale	10/12/2023	Single	C++	Witness the clash of the coding titans in the Bit Battle Royale! Programmers will face off in an epic coding showdown where bits and bytes collide. Only the most resilient and strategic coders will emerge as the champions of this digital battlefield.	JOIN

(Figure 3.1.4) Tournaments



NAME

DEADLINE

TEAM/SINGLE

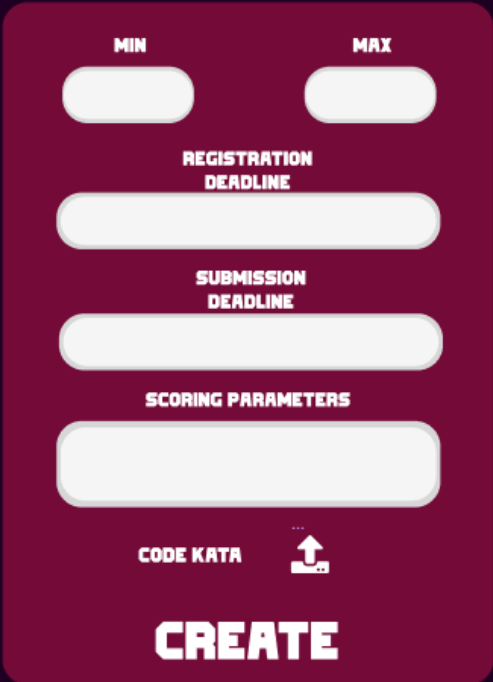
DESCRIPTION

LANGUAGE

BADGES

CREATE

(Figure 3.1.5) Create Tournament



MIN

MAX

REGISTRATION DEADLINE

SUBMISSION DEADLINE

SCORING PARAMETERS

CODE KATA

CREATE

(Figure 3.1.6) Create Battle

3.1.2 Hardware Interfaces

- Users, including Students, Educators, and Collaborators, are required to access the platform through a standard web browser. This ensures compatibility with various devices, and users only need a machine with internet connectivity to engage with the platform effectively.
- The platform itself is hosted on a robust remote server, which not only hosts the web application but also executes all automatic code validation processes when students upload their submissions. This centralized hosting enhances the scalability, reliability, and efficiency of the platform's operations.

3.1.3 software interfaces

The system requires some software interfaces in order to provide its services. Below are reported the most significant :

- GitHub API: Students, upon forking the repo, set up workflows triggering API calls to CKB
- Email provider: provider used to send verification emails to users

3.2 Functional requirement

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

- [R1] The platform allows students to sign up
- [R2] The platform allows educators to sign up
- [R3] The platform allows already registered students to login
- [R4] The platform allows already registered educator to login
- [R5] The platform allows educators to create a tournament
- [R6] The platform allows educators to set a tournament topic
- [R7] The platform allows educators to set a tournament deadline
- [R8] The platform allows educators to invite a collaborator to a tournament
- [R9] The platform allows educators to create a new battle
- [R10] The platform allows educators to upload a CK for a battle

[R11] The platform allows educators to set minimum number of students per team

[R12] The platform allows educators to set maximum number of students per team

[R13] The platform allows educators to set the language about a tournament

[R14] The platform allows educators to set the details about a battle

[R15] The platform allows educators to set registration deadline to a battle

[R16] The platform allows educator to set a final submission deadline to a battle

[R17] The platform allows educator to set additional configuration for scoring

[R18] The platform allows the student to enroll to a tournament

[R19] The platform allows student to invite others students to a their team for a battle

[R20] The platform allows student to accept a team invitation for a battle respecting the constraints about the size of the team

[R21] The platform allows student to enroll to a battle

[R22] The platform creates a github repository containing the CK when registration deadline expires

[R23] The platform send the repo's link to all student who are members of a subscribed team when registration deadline expires

[R24] The platform must receive information through API calls from github action

[R25] The platform is triggered when a push is committed before deadline

[R26] When a push is committed the platform must pull the latest sources

[R27] When the platform pulls latest sources from the repo of a team, must analyze them and run the tests on the corresponding executable

[R28] When the tests of a team's code are executed, the platform must calculate and update the battle score of the team

[R29] The final score of a team must be a natural number between 0 and 100

[R30] The platform must evaluate the score in a fully automated way

[R31] The platform allows educators to manually adjust the score of the battle

[R32] The platform automatically updates the battle scores of a time as soon as new push actions on github are performed

[R33] Both student and educator involved in a battle must see the current rank evolving during the battle

[R34] Once the consolidation stage finishes all students participating in the battle must be notified when the final rank becomes available.

[R35] The platform, for each tournament, updates a rank that measure how a student's performance compares to other students in the context of that tournament

[R36] The student ranking in a tournament is available for all students and educators subscribed to the platform

[R37] The list of ongoing tournaments is available for all students and educators subscribed to the platform

[R38] When an educator closes a tournament, as soon as the final tournament rank becomes available the platform notify all students involved in the tournament

[R39] An educator can define badges when they create a tournament

[R40] Each badge has a title and one or more rules that must be fulfilled to achieve the badge

[R41] At the end of a tournament the badge rules must be checked

[R42] An educator can define new rules for a badges

[R43] An educator can define new variables for a badges

[R44] Badges can be visualized by all users

[R45] Both students and educators can see collected badges when they visualize the profile of a student.

[R46] An user can view the profile of any Student

[R47] The platform allows the user to choose during registration whether to be a Student or an Educator

[R48] The platform supports file upload functionality.

[R49] Educators only see battles they manage.

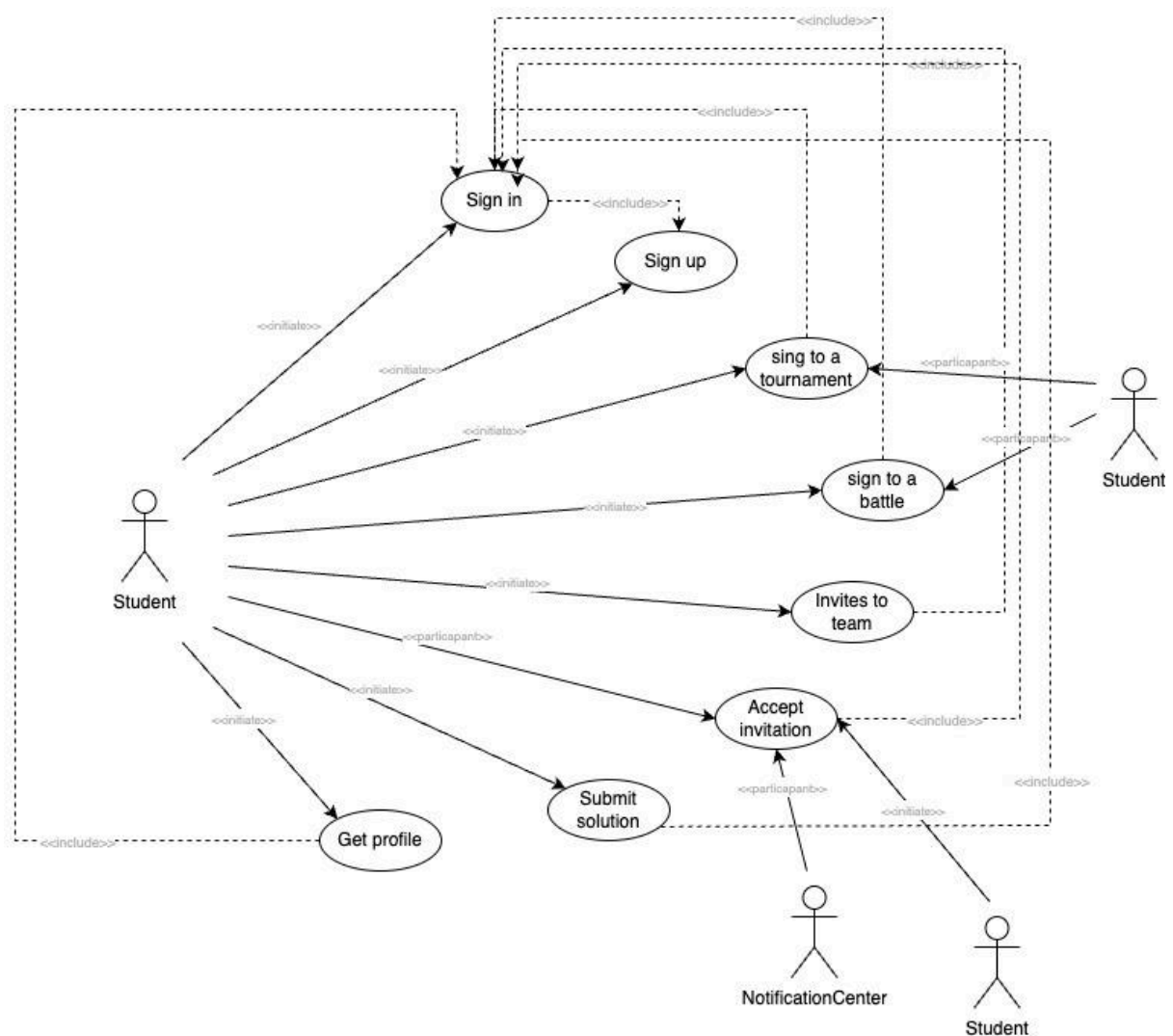
[R50] Educators have visibility of the list of teams enrolled.

[R51] The platform supports different programming languages for coding exercises.

3.2.1 Use case diagrams

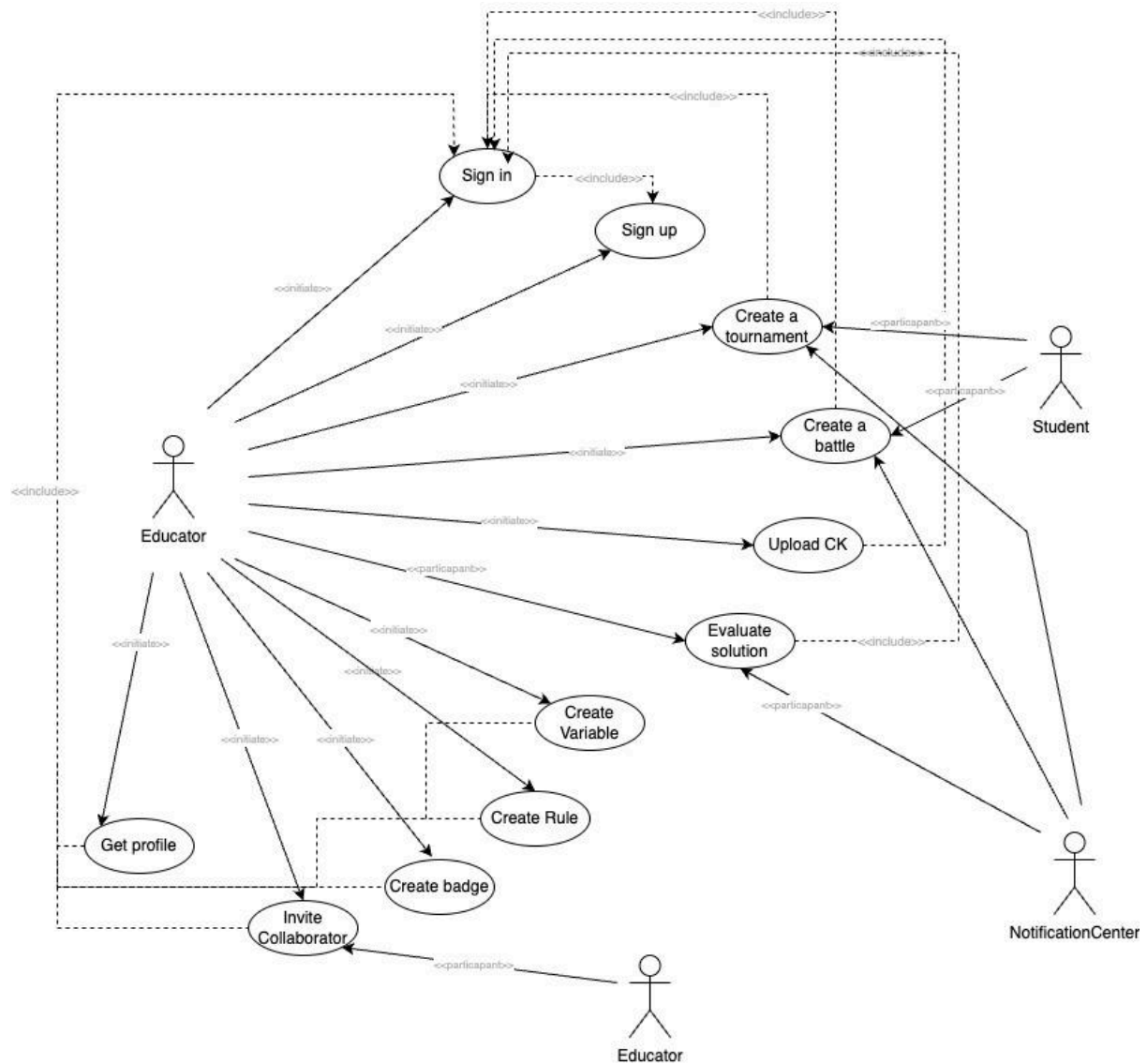
In the following are provided the use case diagrams deduced from the scenarios reported in paragraph 2.1.1. They help identify the actors interacting with the system and their role within each use case.

Student Use Case Diagram



(Figure 3.2) Student Use Case Diagram

Educator Use Case Diagram



(Figure 3.3) Educator Use Case Diagram

3.2.2 Use cases

[UC1] - Login User

Name	Login User
Actors	User
Entry condition	The user goes to codekatabattle.com
Event flow	<ol style="list-style-type: none">1. The user inserts his username and password in the form2. The user clicks on the “Login” button3. The system validate the credentials4. The application shows the proper dashboard
Exit condition	The user has been authorized to access the services provided by CKB
Exception	<ol style="list-style-type: none">3. The data inserted are not valid. The system returns to the entry condition.

[UC2] - User signup

Name	User registration
Actors	User
Entry condition	The User is not registered do CKB
Event flow	<ol style="list-style-type: none">1. The user goes to CKB2. The user clicks on the “Sign up” button3. The platform opens the signup UI4. The user insert name, lastname, email and password5. The platform send a verification email to the user6. The platform creates the user7. The platform makes the user choose whether they want to be a student or an educator
Exit condition	The user has been authorized to access the services provided by CKB

[UC3] - Educator creates a tournament

Name	Tournament creation
Actors	Educator, Students
Entry condition	The educator goes to the tournament creation page
Event flow	<ol style="list-style-type: none"> 1. Educator clicks on “create new tournament” 2. The site show the creation view 3. Educator inserts the: <ul style="list-style-type: none"> • Name • Deadline • Team/Single • Description • Language • Badges 4. Educator clicks on “create” 5. The platform creates the tournament 6. The platform sends a notification to all students
Exit condition	The platform has created the new tournament
Exception	6. The platform fails to create the tournament. The system returns to the entry condition.
Alternative	3.2 Educator invites the Collaborators

[UC4] - Student submit to a tournament

Name	Student submission to a tournament
Actors	Student
Entry condition	The student wants to submit to a tournament
Event flow	<ol style="list-style-type: none"> 1. Student clicks on “tournaments” 2. The site show the list of tournaments 3. Student clicks on “submit” 4. The platform signs the student to the tournament
Exit condition	The student is registered to the tournament
Exception	<ol style="list-style-type: none"> 4. The student’s submission fails. The view returns to tournament’s list

[UC5] - Educator creates a battle

Name	Battle Creation
Actors	Educator, Collaborator, Students
Entry condition	The educator is logged into the CKB platform and wants to create a new battle within an already created tournament.
Event flow	<ol style="list-style-type: none"> 1. Educator navigates to the selected tournament's management interface. 2. The platform shows the battle creation view. 3. Educator sets: <ul style="list-style-type: none"> • Minimum and maximum number of students per group • Registration deadline • Submission deadline • Additional scoring parameters 4. Educator confirms and submits the battle configuration. 5. The educator uploads the CK. 6. The platform generates a GitHub repository for the battle, containing the CK. 7. The platform notifies all subscribed students.
Exit condition	The educator successfully creates a new battle within the selected tournament, and the platform notifies students about the upcoming coding challenge.
Exception	5. The platform fails to create the battle. The view display an error message

[UC6] - Student joins a battle

Name	Join battle
Actors	Student, System
Entry condition	The student has joined a specific tournament with team battles.
Event flow	<ol style="list-style-type: none"> 1. A Student navigates to the list of available battles within the selected tournament. 2. The platform displays a list of open battles 3. Student selects a specific battle 4. The platform signs the student to the battle
Exit condition	The student successfully forms a team and is ready to participate in the selected battle within the tournament.
Alternative	<ol style="list-style-type: none"> 4.a The student invites other students to team 5.a The platform signs the team to the battle

[UC7] - Student receives an invitation to form a team

Name	Student receive an invitation
Actors	Student
Entry condition	The student is logged into the CKB platform
Event flow	<ol style="list-style-type: none"> 1. A student receives a team invitation notification. 2. The student accepts the invitation. 3. The platform updates the team composition, adding the student to the team for the specified battle.
Exit condition	The student has joined the team for the battle
Alternative	<ol style="list-style-type: none"> 2a. The student declines the invitation. 3a. The platform notify the team owner about the refusal of the invitation

[UC8] - Student submits solution to a battle

Name	Battle Submission
Actors	Student, GitHub
Entry condition	The students push the battle's solution to the main branch of the GitHub repo
Event flow	<ol style="list-style-type: none"> 1. The students push the battle's solution to the main branch of the GitHub repo 2. GitHub actions informs the platform that a commit has been pushed 3. The platform get the code from the repo 4. The platform runs the tests. 5. The platform generates the score of the team. 6. The platform updates the ranking of the battle. 7. The platform updates the tournament's ranking
Exit condition	The student successfully submits the solution to the battle, and the platform updates the battle score for the team.
Exception	The student pushes the battle's solution after the submission deadline.

[UC9] - Educator manually evaluates student submissions

Name	Manual Evaluation
Actors	Educator
Entry condition	The platform has runned tests on a team's battle solution.
Event flow	<ol style="list-style-type: none"> 1. Educator selects the specific team's solution that requires manual evaluation. 2. Educator inserts the manual score 3. The platform updates the team's score 4. The platform updates the battle's ranking 5. The platform updates the tournament's ranking
Exit condition	The platform shows the new scores

[UC10] - Educator creates badges

Name	Badge creation
Actors	Educator
Entry condition	An educator is creating a new Tournament and wants to create new badges
Event flow	<ol style="list-style-type: none"> 1. The Educator opens the badge creations page 2. The Educator inserts the badge name 3. The Educator chooses the rules of the badge 4. The Educator submit the data 5. The system saves all the badges 6. The system shows all the badges of the tournament
Exit condition	The educator has successfully created new badges and the tournament

[UC11] - Educator creates rules

Name	Rules creation
Actors	Educator
Entry condition	An educator wants to create new rules for the badges
Event flow	<ol style="list-style-type: none"> 1. The Educator opens the rules creations page 2. The Educator inserts the rule name 3. The Educator creates the variables that need to be used 4. The Educator submit the data 5. The system saves all the rules
Exit condition	The educator has successfully created new rules for the badges creation

[UC12] - Educator creates variables

Name	Variables creation
Actors	Educator
Entry condition	An educator wants to create new variables
Event flow	<ol style="list-style-type: none"> 1. The Educator opens the variables creations page 2. The Educator inserts the variable name 3. The Educator defines the variable using the custom language 4. The Educator submit the data 5. The system checks the variable definition 6. The system saves the variable
Exit condition	The educator has successfully created new badges and the tournament
Exception	The variable definition is not valid, the system print an error message

[UC13] - Tournament ends

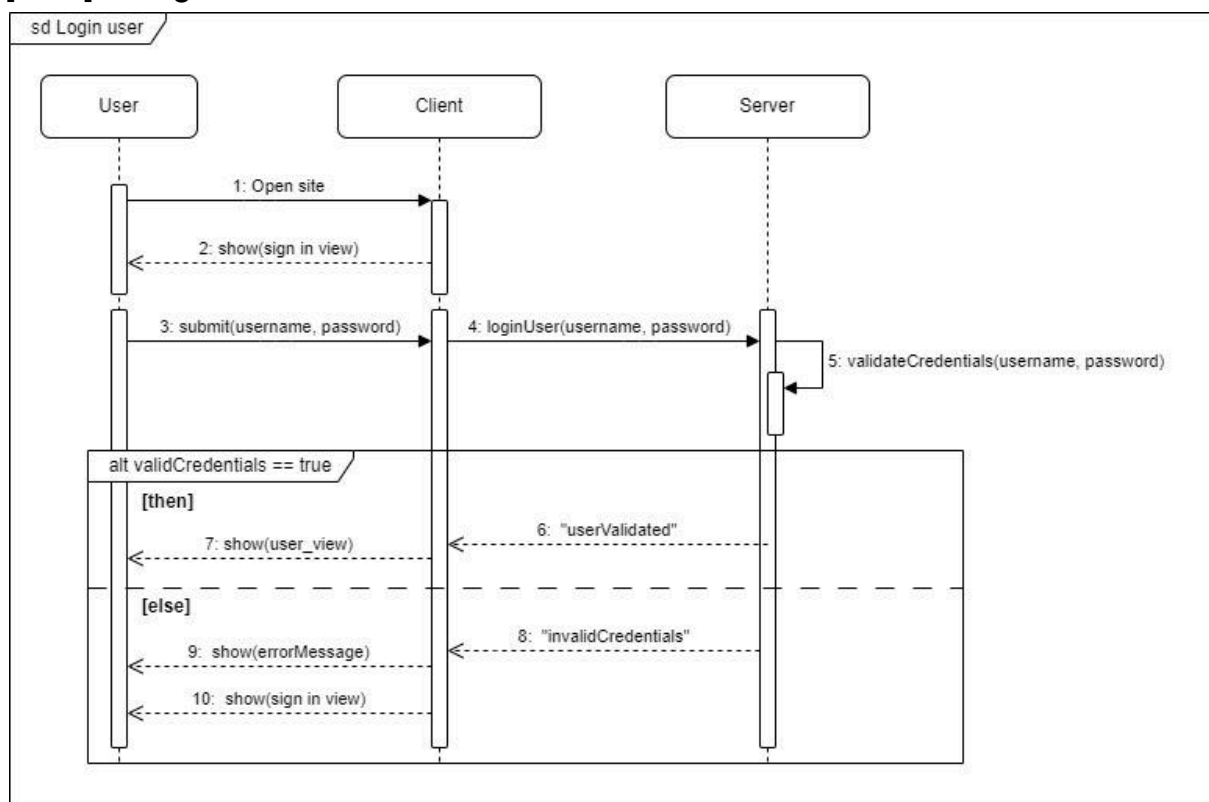
Name	Tournament ending
Actors	Students, system
Entry condition	A tournament has concluded, and personal tournament scores are finalized.
Event flow	<ol style="list-style-type: none"> 1. The deadline of a Tournament expires 2. The system closes the tournaments and all battles 3. The system notifies the students the Tournament has ended 4. The system calculate the final ranking 5. The system assigns the badges to the students 6. The system show the final ranking
Exit condition	The student completes the viewing of the tournament rank, gaining insights into their performance and that of other participants.

[UC14] - Get students profile

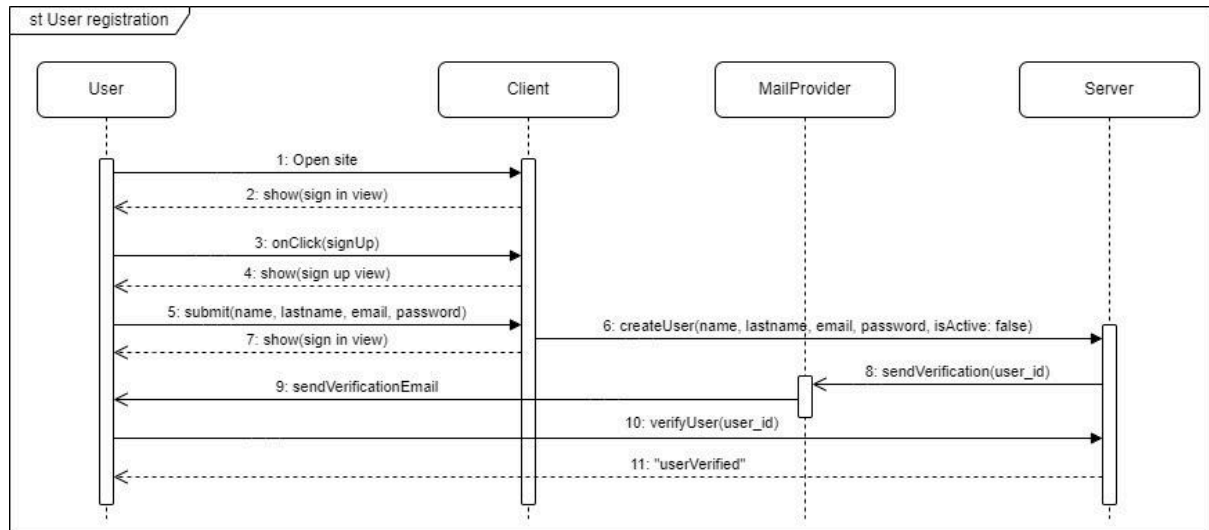
Name	Get Student profile
Actors	User, system
Entry condition	An user wants to see the student profile
Event flow	<ol style="list-style-type: none"> 1. The user clicks on a student profile 2. The system gets all the student's data 3. The system shows the profile view of the student
Exit condition	The system show the profile view of the student with all the data: Name, badges and analytics

3.2.3 Sequence diagrams

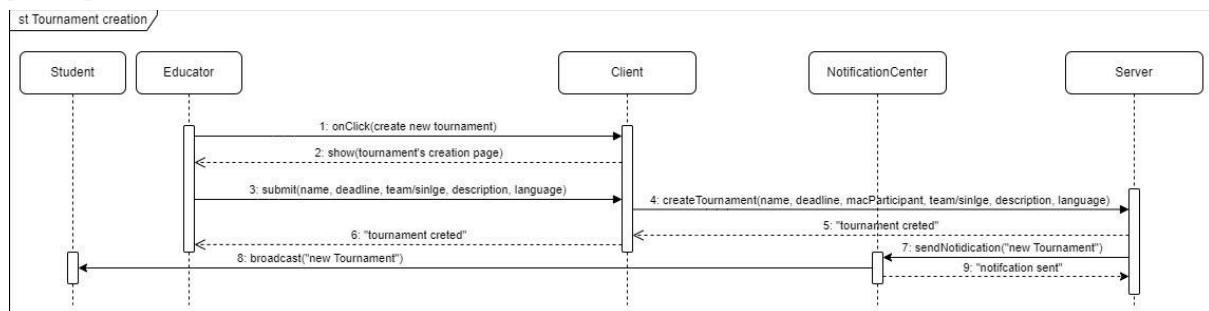
[UC1] - Login User



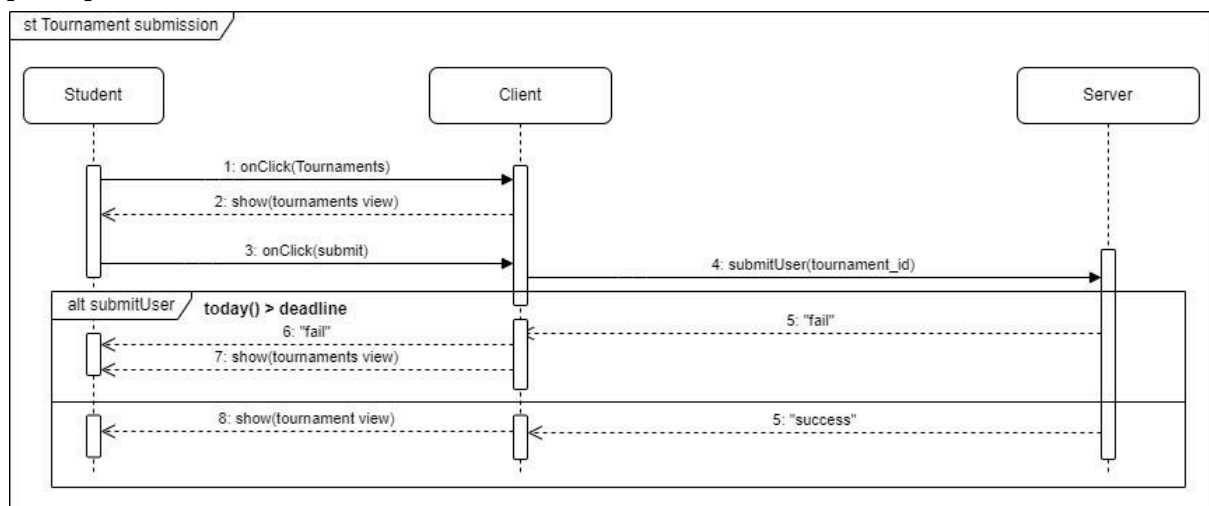
[UC2] - User signup



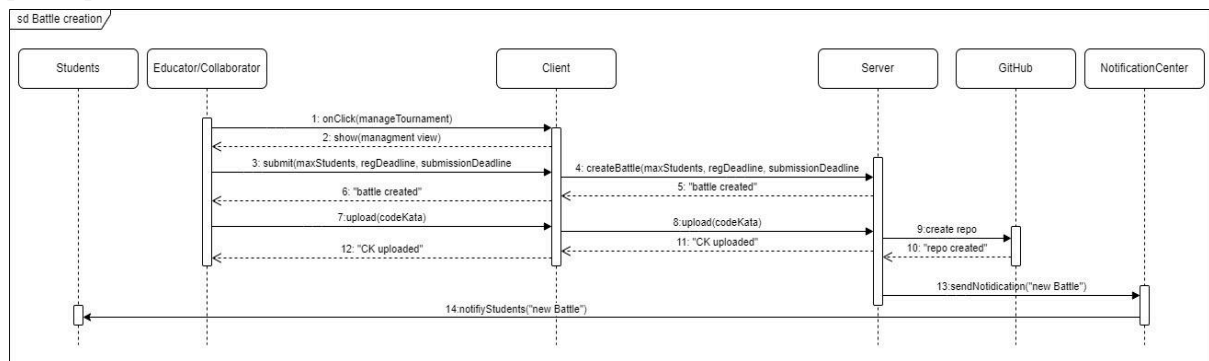
[UC3] - Educator creates a tournament



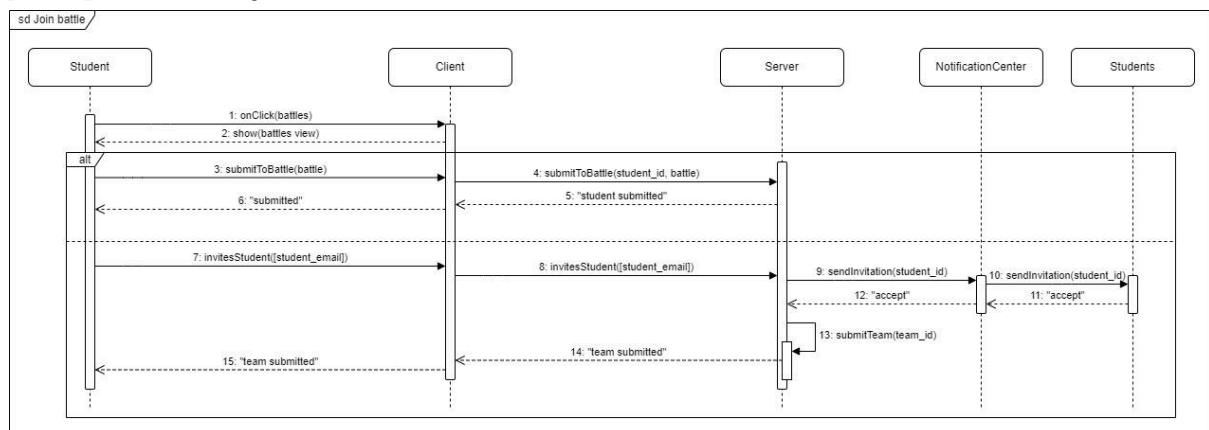
[UC4] - Student submit to a tournament



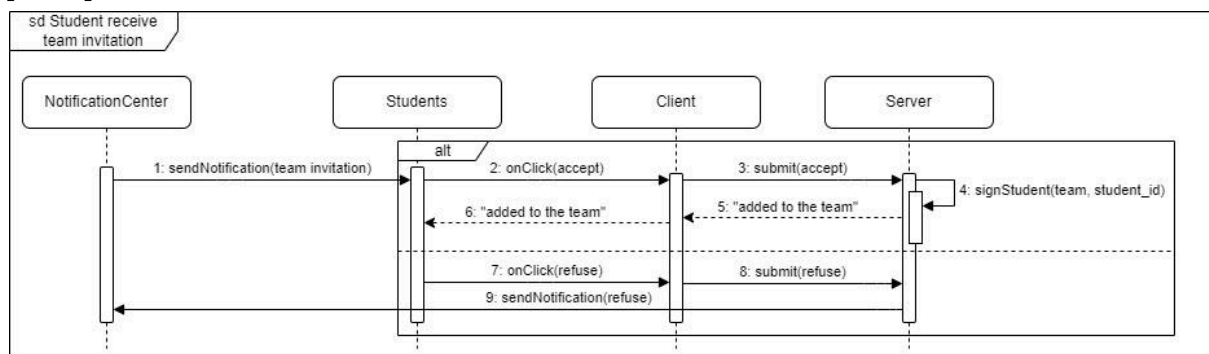
[UC5] - Educator creates a battle



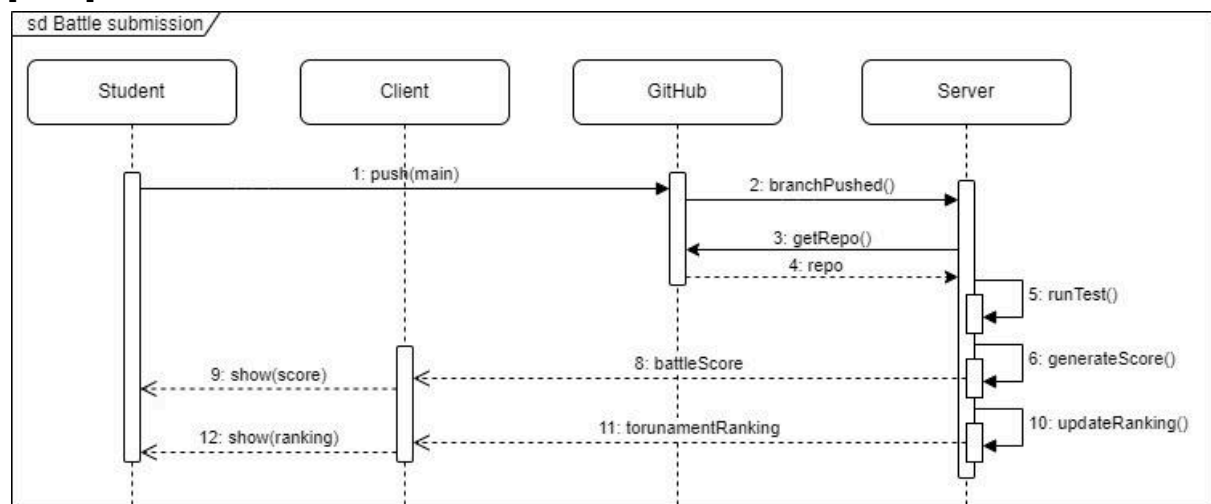
[UC6] - Student joins a battle



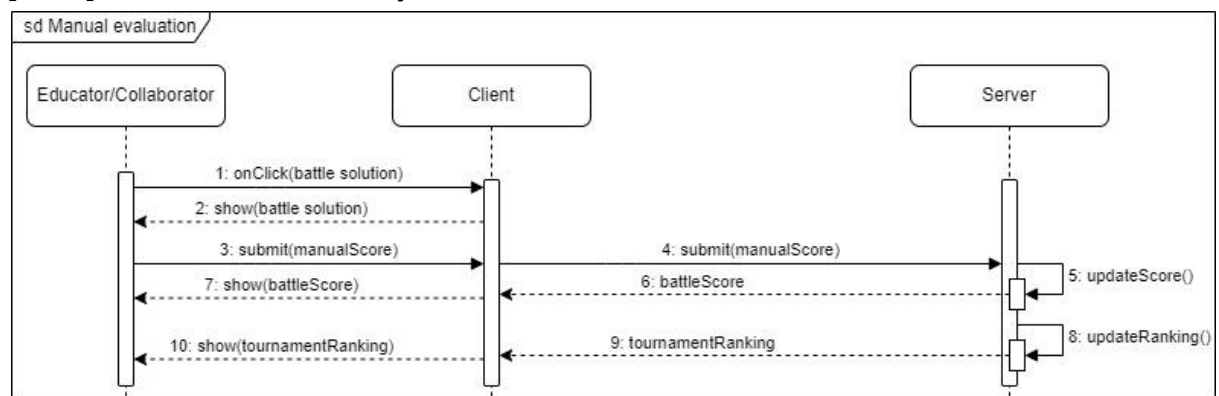
[UC7] - Student receives an invitation to form a team



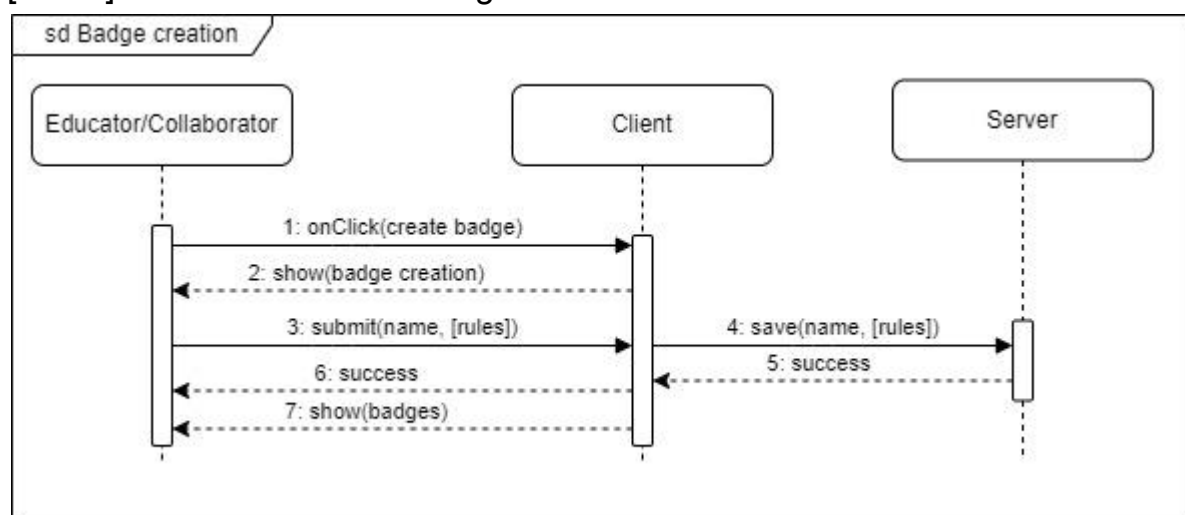
[UC8] - Student submits solution to a battle



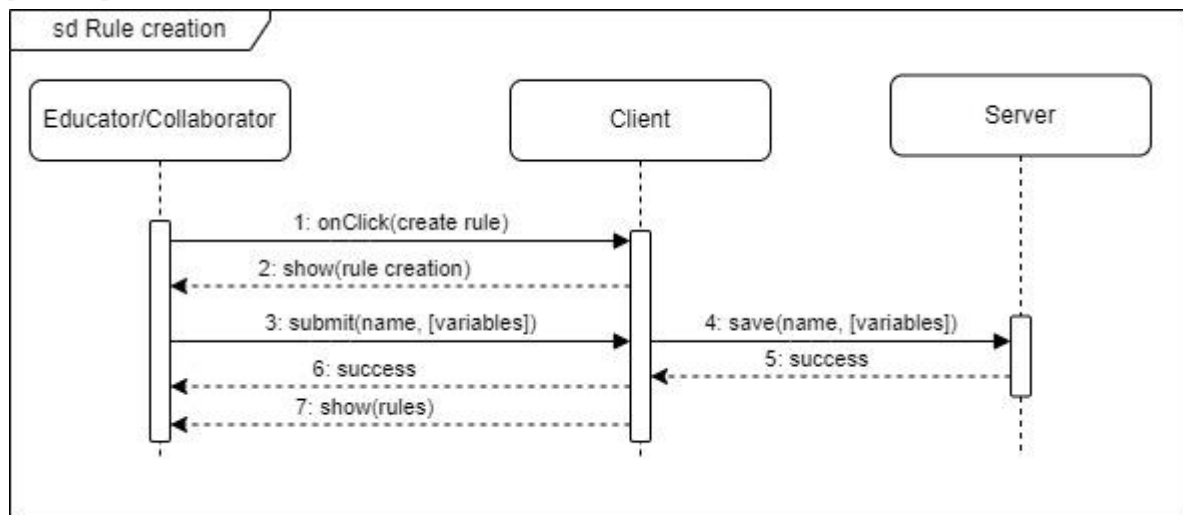
[UC9] - Educator manually evaluates student submissions



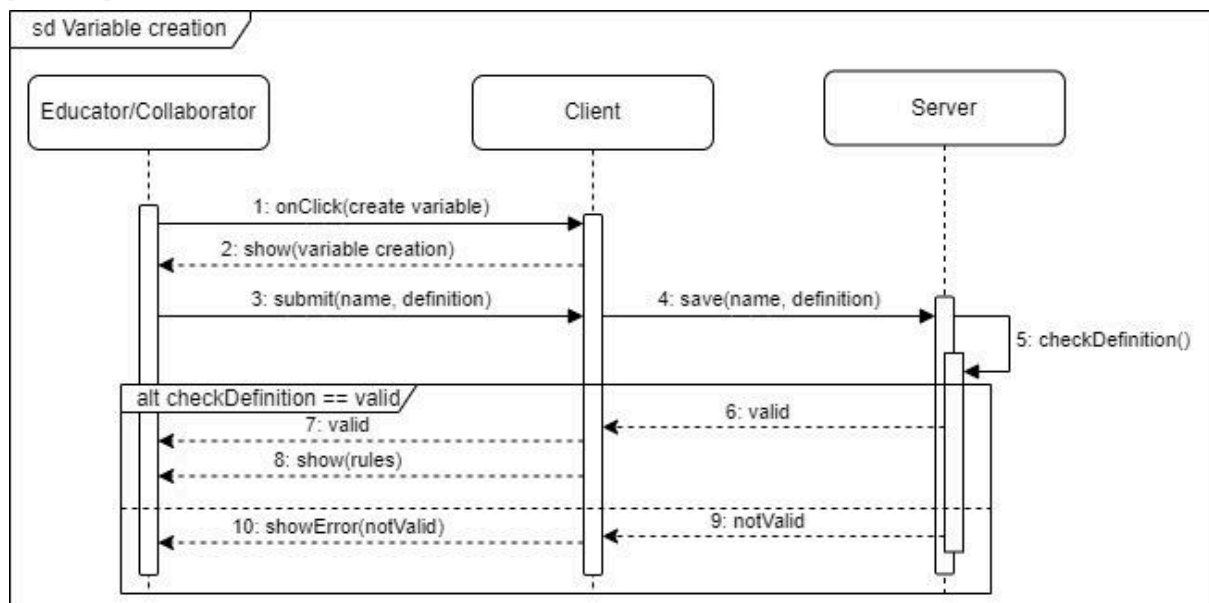
[UC10] - Educator creates badges



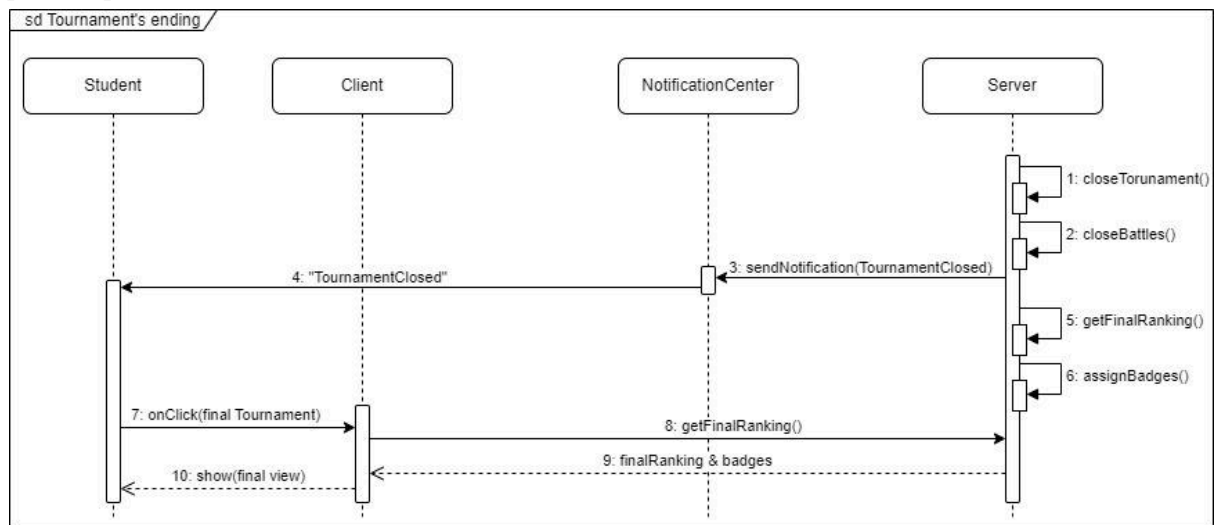
[UC11] - Educator creates rules



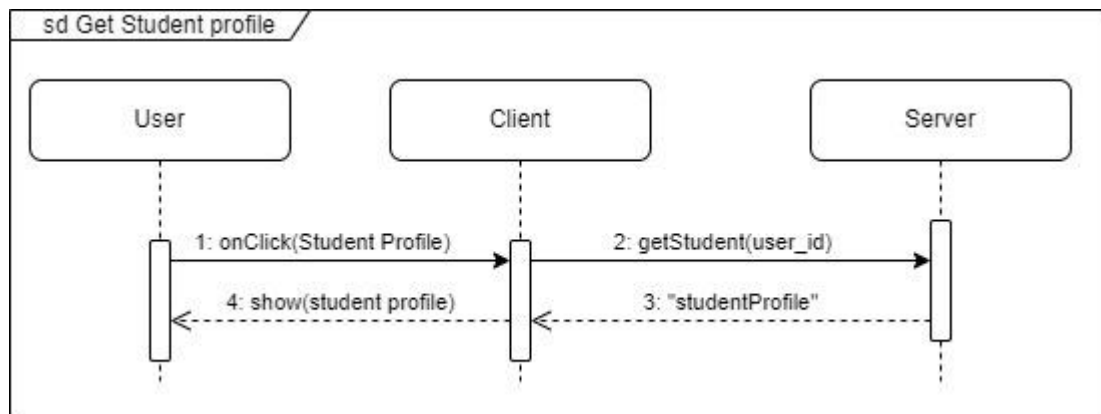
[UC12] - Educator creates variables



[UC13] - Tournament ends



[UC14] - Get students profile



3.2.4 Requirements mapping

[G1] Educator can manage tournament	
<p>[R4] The platform allows already registered educator to login</p> <p>[R5] The platform allows educators to create a tournament</p> <p>[R6] The platform allows educators to set a tournament topic</p> <p>[R7] The platform allows educators to set a tournament deadline</p> <p>[R8] The platform allows educators to invite a collaborator to a tournament</p> <p>[R13] The platform allows educators to set the language about a tournament</p> <p>[R38] When an educator closes a tournament, as soon as the final tournament rank becomes available the platform notify all students involved in the tournament</p> <p>[R39] An educator can define badges when they create a tournament</p>	<p>[D1]: Educators and students have devices with internet connectivity.</p> <p>[D4]: Information provided about battles and tournaments is accurate.</p> <p>[D8]: Educators responsibly input all required information about tournaments and battles.</p>
[G2] Educator can manage a battle inside a tournament	
<p>[R9] The platform allows educators to create a new battle</p> <p>[R10] The platform allows educators to upload a CK to a battle</p> <p>[R11] The platform allows educators to set minimum number of students per team</p> <p>[R12] The platform allows educators to set maximum number of students per team</p> <p>[R14] The platform allows educators to set the details about a battle</p> <p>[R15] The platform allows educators to set registration deadline to a battle</p>	<p>[D1]: Educators and students have devices with internet connectivity.</p> <p>[D4]: Information provided about battles and tournaments is accurate.</p> <p>[D7]: Notifications are delivered promptly within 10 seconds.</p> <p>[D9]: Educators responsibly input all required information about tournaments and battles.</p>

<p>[R16] The platform allows educator to set a final submission deadline to a battle</p> <p>[R17] The platform allows educator to set additional configuration for scoring</p> <p>[R31] The platform allows educators to manually adjust the score of the battle</p> <p>[R33] Both student and educator involved in a battle must see the current rank evolving during the battle</p> <p>[R49]: The platform supports file upload functionality.</p> <p>[R50]: Educators only see battles they manage.</p> <p>[R51]: Educators have visibility of the list of teams enrolled.</p>	
---	--

[G3] Student can participate to battles within tournaments	
<p>[R18] The platform allows the student to enroll to a tournament</p> <p>[R19] The platform allows student to invite others students to their team for a battle</p> <p>[R20] The platform allows student to accept a team invitation for a battle respecting the constraints about the size of the team</p> <p>[R21] The platform allows student to enroll to a battle</p> <p>[R23] The platform sends the repo's link to all student who are members of a subscribed team when registration deadline expires</p> <p>[R29] The final score of a team must be a natural number between 0 and 100</p> <p>[R33] Both student and educator involved in a battle must see the current rank evolving during the battle</p>	<p>[D1]: Educators and students have devices with internet connectivity.</p> <p>[D2]: Students adhere to the registration and submission deadlines for battles.</p> <p>[D4]: Information provided about battles and tournaments is accurate.</p> <p>[D6]: Students have the required equipment for coding exercises.</p> <p>[D7]: Notifications are delivered promptly within 10 seconds.</p> <p>[D8]: Time left and scoring details are consistent with the platform display.</p>

<p>[R34] Once the consolidation stage finishes all students participating in the battle must be notified when the final rank becomes available.</p> <p>[R36] The student ranking in a tournament is available for all students and educators subscribed to the platform</p> <p>[R37] The list of ongoing tournaments is available for all students and educators subscribed to the platform</p> <p>[R38] When an educator closes a tournament, as soon as the final tournament rank becomes available the platform notify all students involved in the tournament</p>	
---	--

[G4] Educator can manage the creation badges	
<p>[R39] An educator can define badges when they create a tournament</p> <p>[R40] Each badge has a title and one or more rules that must be fulfilled to achieve the badge</p> <p>[R41] At the end of a tournament the badge rules must be checked</p> <p>[R42] An educator can define new rules for a badges</p> <p>[R43] An educator can define new variables for a badges</p> <p>[R44] Badges can be visualized by all users</p> <p>[R45] Both students and educators can see collected badges when they visualize the profile of a student.</p> <p>[R49]: The platform supports file upload functionality.</p>	<p>[D1]: Educators and students have devices with internet connectivity.</p>

3.3 Performance Requirements

The platform itself is hosted on a highly capable remote server, designed with a dynamic architecture that minimizes downtime and efficiently handles significant user traffic. This dynamic architecture allows for the deployment of multiple instances of the system to seamlessly manage high volumes of user interactions, ensuring a responsive and reliable user experience even during peak usage periods. In addition, the server is equipped with substantial computational power, enabling it to perform automated tests of battles in the shortest possible time. This computational power is crucial in validating code submissions and delivering prompt feedback to students, contributing to a productive and responsive learning environment.

3.4 Design Constraints

3.4.1 Standard Compliance

In adherence to data privacy regulations, the CDK platform strictly complies with the General Data Protection Regulation (GDPR), which governs data protection and privacy for individuals within the European Union (EU) and the European Economic Area (EEA). This ensures that all data handling and processing within the system align with the highest privacy standards.

Furthermore, the system also adheres to international standards regarding the use and representation of date and time. This commitment to international conventions ensures consistency and compatibility in date and time-related functionalities across the platform, facilitating a seamless user experience for individuals from various regions around the world.

3.4.2 Hardware limitations

- Students: Internet connection (WiFi), modern web browser, medium-end computer for solve the battle
- Educator/Collaborator: Internet connection (WiFi), modern web browser

3.5 Software System Attributes

3.5.1 Reliability

Reliability is a fundamental attribute of our platform, ensuring that it consistently performs its intended functions without unexpected downtime or failures. To achieve this, the system employs robust error handling, fault tolerance mechanisms, and automated backup systems.

3.5.2 Scalability

The platform is designed to scale efficiently in response to varying user loads. As user traffic increases, additional resources and server instances can be dynamically allocated to ensure optimal performance. Scalability is a key consideration to accommodate both small-scale usage scenarios and large-scale concurrent activities during peak periods.

3.5.3 Security

The system incorporates industry-standard encryption protocols for data transmission and storage. Access controls, authentication mechanisms, and role-based permissions are implemented to safeguard sensitive information.

3.5.4 Usability

Usability is a critical attribute to enhance the user experience. The platform features an intuitive and user-friendly interface, ensuring that users can easily navigate and utilize its functionalities. Accessibility standards are followed to cater to users with diverse needs.

3.5.5 Performance

The system is designed to deliver fast response times and efficient processing of tasks. Caching mechanisms, load balancing, and efficient database queries are employed to minimize latency and ensure that actions such as uploading content or participating in battles occur swiftly.

4. Alloy

4.1 Signatures

The formal modeling activity using Alloy is driven by the need to achieve a thorough understanding, validation, and verification of the CKB platform's structure and behavior. It serves as a valuable tool for ensuring the correctness, consistency, and reliability of the system, aligning with the specified requirements and objectives outlined in the previous chapters. In the following are stated all the signatures defined for a complete alloy modelization. Notice that some signatures have intrinsic constraints which we thought were important to include here, for example Battle, Tournament, Score and Date.

// Signatures

// Useful for defining all the attributes of a user

```
abstract sig User {  
    id: Int,  
    name: String,  
    lastname: String,  
    email: String  
}
```

//Profile of a Student

```
sig Profile{  
    badgesCollected: set Badge  
}
```

//Definition of a Student

```
sig Student extends User {  
    profile: one Profile,  
    notificationsNotYetRead: set Notification  
}
```

//Definition of an Educator, concretized in Admin and Collaborator

```
abstract sig Educator extends User {  
}
```

//Definition of an Admin, who is the creator of a Tournament

```

sig Admin extends Educator{
    tournamentsCreated: set Tournament
}

//Definition of a Collaborator, who can collaborate in tournaments
sig Collaborator extends Educator{
    ParticipatingTournaments: set Tournament
}

// Definition of a Variable, useful for defining new rules
sig Variable {
    name: String
}

//Definition of a Rule, useful for defining new badges
sig Rule {
    name: String,
    variables: some Variable
}

//Definition of a Badge, which are created by educators and can be assigned
//to students
sig Badge {
    title: String,
    rules: some Rule
}

//Boolean useful to modelize the status of tournaments and battles (open or
//closed)
abstract sig Bool {}
one sig TRUE extends Bool {}
one sig FALSE extends Bool {}

// Simplified model of a date, managed with integers
sig Date{
    date: Int
}{} date >= 0 }

//maxScore can be different from battle to battle, to allow Educators
//decide for an optional manual evaluation

```

```

sig Score {
  value: Int
} {
  value >= 0 and value <= Battle.maxScore
}

```

//Definition of the programming Language

```

sig ProgrammingLanguage {}

```

//Definition of a Tournament

```

sig Tournament {
  id: Int,
  name: String,
  registrationDeadline: Date,
  endOfTournament: Date,
  rankingOfStudents: set Student,
  battles: set Battle,
  programmingLanguage: ProgrammingLanguage,
  isOpen: Bool,
  badgesCreated: set Badge
} {
  //Integrity constraint on dates
  registrationDeadline.date > 0
  registrationDeadline.date < endOfTournament.date

  all b: Battle | b in battles implies registrationDeadline.date <
  b.registrationDeadline.date
  and b.endOfBattle.date <= endOfTournament.date
}

```

//Definition of a Battle

```

sig Battle {
  id: Int,
  name: String,
  registrationDeadline: Date,
  endOfBattle: Date,
  rankingOfTeams: set Team,
  teams: set Team,
  isOpen: Bool,

```

```

    maxScore: Int,
    maxTeamMembers: Int,
    minTeamMembers: Int,
    codekata: one CodeKata
  }

  // Constraints on some attributes
  minTeamMembers <= maxTeamMembers and minTeamMembers > 0
  registrationDeadline.date < endOfBattle.date
  maxScore > 0

  // Ensure that rankingOfTeams is the same as the set of teams in the battle
  rankingOfTeams = teams

  // Ensure that codekata in a battle is the same as codekata in a repo,
  // if the repo belongs to a team in that battle
  all t: teams | codekata = t.repo.codekata
}

//Definition of a Team
sig Team {
  id: Int,
  name: String,
  members: some Student,
  score: Score,
  repo: one Repo
}

//Definition of a Repo, to manage the association with the GitHub repository
sig Repo{
  codekata: one CodeKata
}

//Definition of a CodeKata
sig CodeKata{
  programmingLanguage: ProgrammingLanguage,
  description: String,
  softwareProject: one SoftwareProject
}

```

```

//Definition of Software Project, which includes test cases and build
//automation scripts
sig SoftwareProject{
    tests: some Test
}

//Definition of a test
sig Test{}

//The Notification Center is responsible to manage the notification mechanism
one sig NotificationCenter{}

//Definition of a Notification
sig Notification {
    message: String
}

```

4.2 Facts

In the following are stated in Alloy the facts that must hold for the domain modeled in order to maintain consistency with the real world.

```

// Facts and Constraints

// Constraint to ensure that if isOpen of a Battle in a Tournament is TRUE,
// then isOpen of the corresponding Tournament can't be FALSE
fact consistentBattleTournamentStatus{
    all t: Tournament, b: t.battles |
        b.isOpen = TRUE implies t.isOpen = TRUE
}

// Constraint to ensure that rankingOfTeams has only teams that belong to the
//set teams of the battle
fact validRankingOfTeams {
    all b: Battle | brankingOfTeams in b.teams
}

// Constraint to ensure that the profile of a Student is always different
fact differentProfilesOfStudents{
    all s1, s2: Student | s1 != s2 implies s1.profile != s2.profile
}

```



```

}

// Fact to ensure uniqueness of IDs
fact uniqueIDs {
    all disj u1, u2: User | u1.id != u2.id
    all disj t1, t2: Tournament | t1.id != t2.id
    all disj b1, b2: Battle | b1.id != b2.id
    all disj tm1, tm2: Team | tm1.id != tm2.id
}

// Fact to ensure positive IDs
fact positiveID{
    all u: User | u.id >= 0
    all t: Tournament | t.id >= 0
    all b: Battle | b.id >= 0
    all t: Team | t.id >= 0
}

// Fact to ensure unique Team names in a Battle and other name constraints
fact uniqueNames {
    all b: Battle | all disj tm1, tm2: b.teams | tm1.name != tm2.name
    all t: Tournament | all disj b1, b2: t.battles | b1.name != b2.name
    all disj t1, t2: Tournament | t1.name != t2.name
}

// To guarantee different teams in the same Battle
fact DifferentTeamMembersInABattle {
    all b: Battle | all t1, t2: b.teams | t1 != t2 implies no t1.members &
t2.members
}

// To guarantee different battles in Tournaments
fact DifferentBattlesInTournaments {
    all disj t1, t2: Tournament | no t1.battles & t2.battles
}

// Teams have different Repo
fact DifferentRepoForTeams{
    all disj tm1,tm2: Team, r1: tm1.repo, r2: tm2.repo | r1 != r2
}

```

```

//Battle have different teams
fact DifferentTeamInDifferentBattle{
    all disj b1, b2: Battle, tm1: b1.teams, tm2: b2.teams | tm1!=tm2
}

// Constraint to ensure that each Battle belongs to a Tournament
fact battleBelongsToTournament {
    all b: Battle | one t: Tournament | b in t.battles
}

// Constraint to ensure that each Team belongs to a Battle
fact teamBelongsToBattle {
    all t: Team | one b: Battle | t in b.teams
}

// Constraint to ensure that profiles are always different from student to
//student
fact differentProfilesOfStudents {
    all s1, s2: Student | s1 != s2 implies s1.profile != s2.profile
}

// Ensures that every Profile belongs to a Student
fact everyProfileBelongsToStudent {
    all p: Profile | one s: Student | s.profile = p
}

//Ensures that every CodeKata has his own Software Project
fact differentSoftwareProjectsOfCodeKata{
    all c1, c2: CodeKata | c1 != c2 implies c1.softwareProject !=
c2.softwareProject
}

// Constraint to ensure that each sig Test belongs to a CodeKata
fact testBelongsToSoftwareProject {
    all t: Test | one sw: SoftwareProject | t in sw.tests
}

// Constraint to ensure that for all Tournaments there exists exactly one Admin

```

```

fact adminAndCollaboratorsForTournaments {
  all t: Tournament |
    one a: Admin | t in a.tournamentsCreated
}

```

```

// To manage strings in alloy, simplified model of a String
fact stringPool {
  none != "a" + "b" + "c" + "d" + "e"
}

```

```

// Constraint to ensure that programmingLanguage in CodeKata must be the
//same of programmingLanguage in a Tournament (if the CodeKata belongs to
//a repo of a Team inside the Tournament)
fact consistentProgrammingLanguage {
  all t: Tournament, b: t.battles, tm: b.teams, r: tm.repo, ck: r.codekata |
    ck.programmingLanguage = t.programmingLanguage
}

```

```

// Constraint to ensure that for every Tournament there is at least one Badge
//created
fact atLeastOneBadgePerTournament {
  all t: Tournament |
    some b: Badge | b in t.badgesCreated
}

```

```

// Constraint to ensure that if a Tournament is closed, a badge can be
//assigned to some students
// (In a profile of a Student can also exist other badges he may have collected)
fact badgeAssignedAtTheEndOfTournament {
  all t: Tournament, b: t.battles, tm: b.teams, s: tm.members, p: s.profile, bd:
p.badgesCollected |
    t.isOpen = FALSE iff bd in t.badgesCreated
}

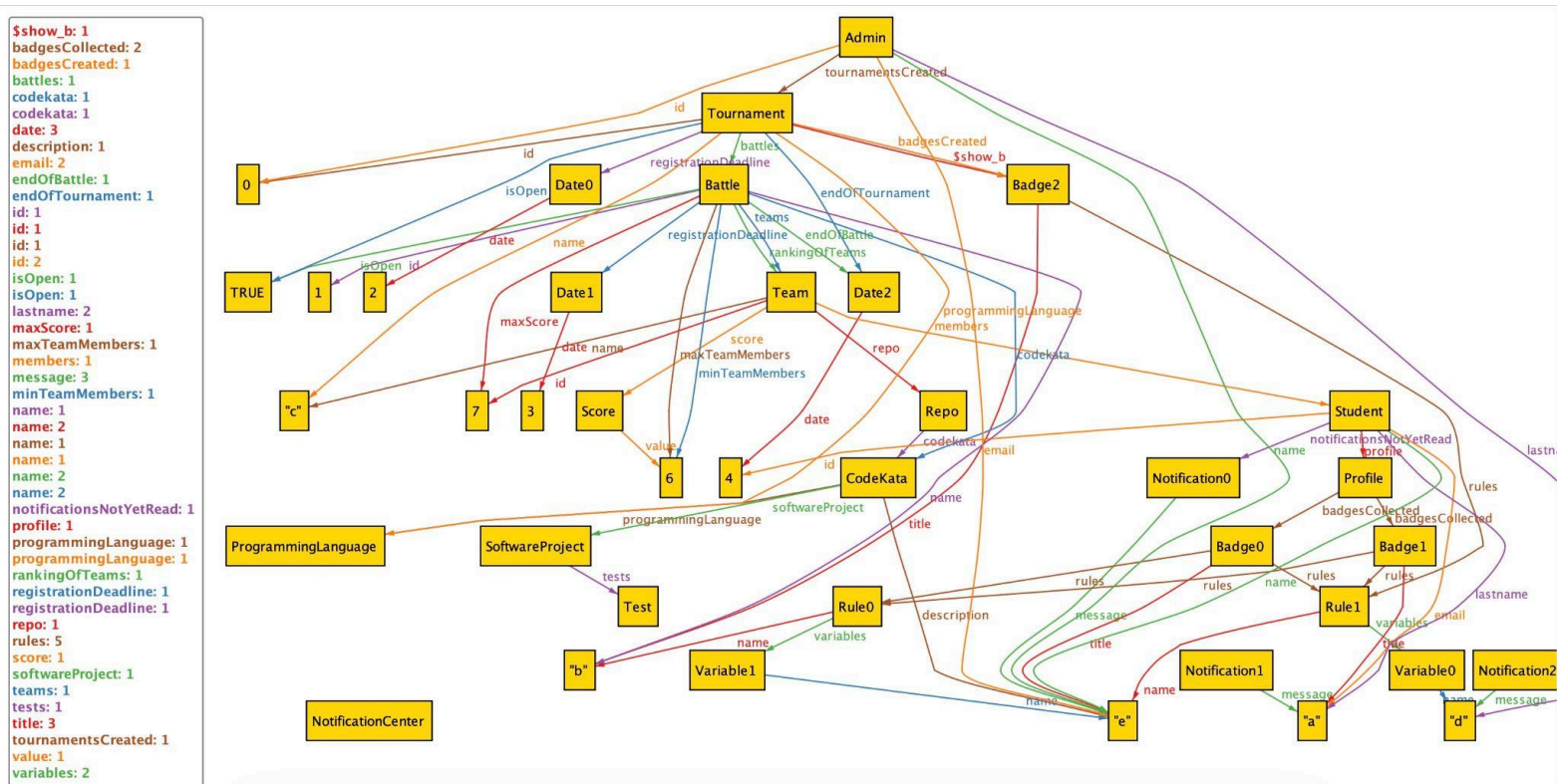
```

4.3 Show

It is possible to see some worlds obtained by running the model, starting with running a very simple model:

```
pred show []{
  #Tournament = 1
  #Battle = 1
  #Educator = 1
  #Team = 1
}
```

run show for 3



Metamodel showing the situation where an Educator has created only one tournament and a student joins the only existing battle in the tournament. It is possible to analyze increasingly complex models, but in doing so the number of signatures present in the model will also increase, for example we can run the following:

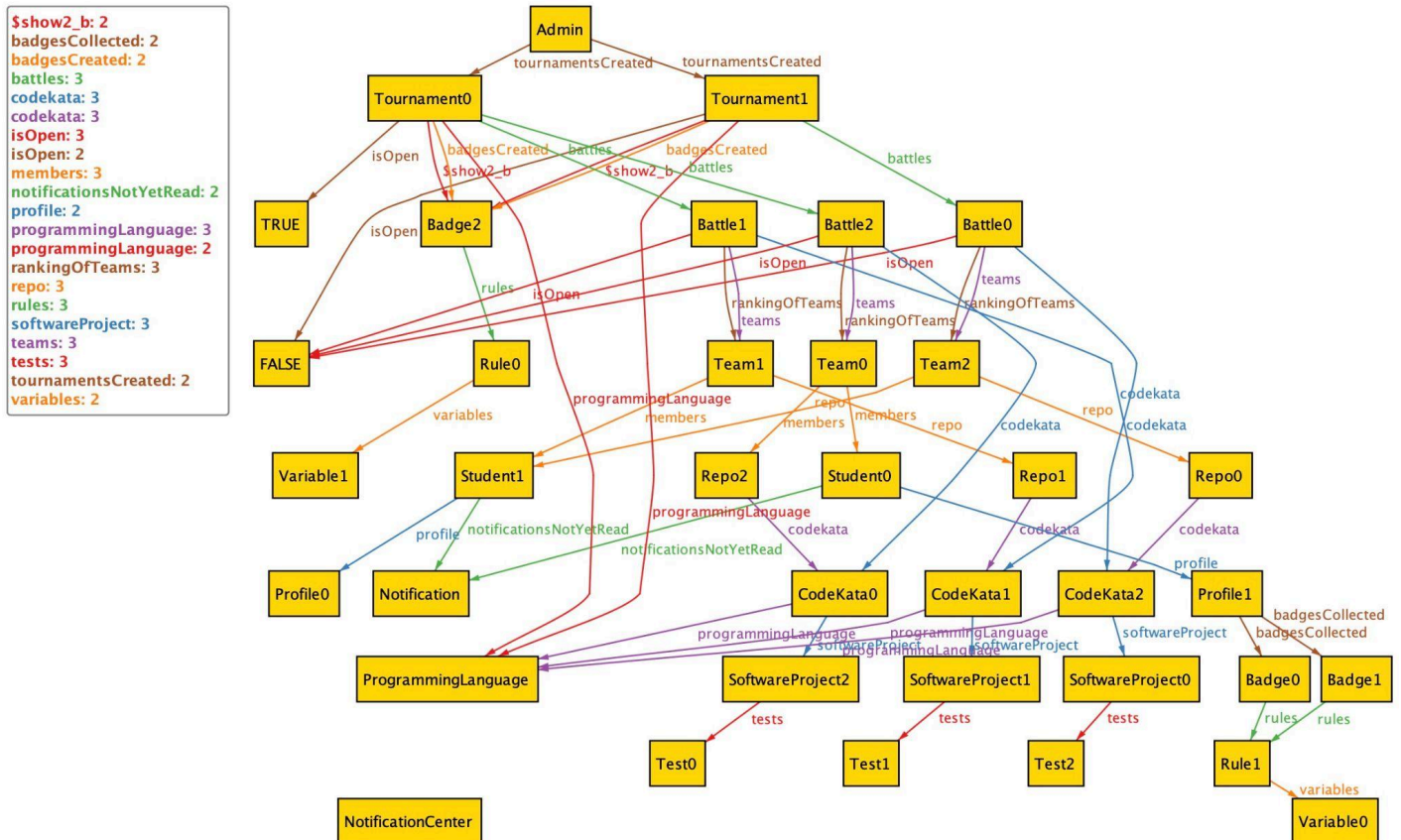
```
pred show2 []{
  #Tournament = 2
  #Battle = 3
}
```

```

#Team = 3
#CodeKata = 3
#Badge = 3
}

```

run show2



For readability reasons of the image we have removed from the model all the int and string attributes of the signatures (for example the various IDs and names).

5 Effort

In the following table is reported the effort of each component of the group for each chapter

Aleksandro Aliaj

chapter	hour spent
1	7
2	12
3	14
4	22

Federico Albertini

chapter	hour spent
1	7
2	15
3	20
4	13

Leonardo Betti

chapter	hour spent
1	7
2	14
3	21

6. References

- Diagrams made with: draw.io
- Mockups made with: figma
- Alloy models made, runned and checked with: alloytools