REQUIREMENT ANALYSIS AND SPECIFICATION DOCUMENT

CODE KATA BATTLE

Armando Fiorini, Samuele Motta, Vajihe Gholami

INFO

Deliverable: RASD
Title: Requirement Analysis and Specification Document
Authors: Armando Fiorini, Samuele Motta, Vajihe Gholami
Version: 1.0.0
Date: 22/12/2023
Download page: https://github.com/ArmaFio/FioriniMottaGholami

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to outline the functionalities and requirements of CodeKataBattle (CKB), a novel platform designed to facilitate the enhancement of students' software development skills through collaborative training on code katas. Educators leverage this platform to challenge and mentor students by creating code kata battles, where teams of students engage in friendly competitions to showcase and enhance their programming skills.

1.2 Scope

Nowadays the world of computer science is more than ever important and present in everyone's life. For this reason, the teaching system must have the best possible means to lead the students to the best possible comprehension and governance of this subject.
CodeKataBattle is a platform supposed to allow Educators to create coding tournaments in which students can compete, in teams or on their own, to improve their skills in any type of programming language.

To ensure that the system allows Educators to manage all the settings and rules of the tournaments (number of battles, allowed Students' team size, score assignment rules, proper programming language . . .), to which students can subscribe.
All the Students logged into the app receive a notification each time a tournament has been created. Students involved in a tournament can create teams (according to the tournament's rules), and then it's time for them to start coding: through GitHub, they will be allowed to receive the problem's text and to submit their solution, within a configured time range.
The solutions will then be evaluated by the system and(if allowed) by the Educator, and will contribute to defining the final rank of the tournament.

Their stats in tournaments will be public and visible in the account of a student, as well as their achieved badges: a badge is a reward that is created by an Educator, who defines the requirement(s) to gain it: the badges can be assigned to one or more students and are relative to a single tournament, at the end of which they are eventually assigned.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Educator: user who signs up to use the system as a mean to improve the programming skills of his students, can create Tournaments and badges.

Student: user who signs up to improve their skills, and participate in tournaments are created by the educators.

User: general user of the system, can be a student or an educator.

Tournament: coding challenge, consisting of a certain number of battles.

Team: group of students, formed to join and compete in a battle.

Battle: every single challenge the tournament is composed of.

(Gamification) Badge: achievement that is created by an educator, and can be obtained from Students who satisfy the established requirements.

1.3.2 Acronyms

- **CKB**: CodeKataBattle
- **RASD**: Requirement Analysis and Specification Document
- **UI**: User Interface
- **UML**: Unified Modelling Language
- **OS**: Operative System

1.3.3 Abbreviations

- $[Gn]$: the n-th goal of the system.
- $[Wn]$: the n-th world phenomena.
- $[SWn]$: the n-th shared phenomena controlled by the world.
- $[SMn]$: the n-th shared phenomena controlled by the machine.
- $[Dn]$: the n-th domain assumption.
- $[FRn]$: the n-th functional requirement.

1.4 Revision History

- Version 1.0 (December 21, 2023)

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 2023/2024
- Slides of Software Engineering II course on WeBeep.

1.6 Document Structure

This document is divided in 6 main chapters:

- **Introduction:** The purpose of this chapter is to provide general information about the project structure and describe its environment. More specifically, it contains brief descriptions regarding the goals that this project aims to achieve.
- **Overall Description:** High-end description, focusing on scenarios and product functions to describe the system behavior in realistic situations.
- **Specific Requirements:** Describes in a very detailed way the requirements that are needed to reach the goals, it also describes the interfaces.
- **Formal Analysis Using Alloy:** Describes the structure and the behavior of the system in a formal way by using alloy.
- **Effort Spent:** Contains information about the time spent to create this document.
- **References:** Contains references to programs and tools used to create this document.

1.7 Overview

1.7.1 Goals

The system is designed to achieve the following goals:

[$G1$] Educators and students can subscribe to the application creating their account.
[$G2$] Educators create Tournaments in which students can compete.
[$G3$] Students subscribe to the tournaments.
[$G4$] Educators create Battles in the context of a tournament, consisting of coding challenges for students.
[$G5$] Students compete, in teams or on their own, in many battles, the results of which will determine the final rank of the tournament.
[$G6$] Educators create badges and define requirements to achieve them.
[$G7$] Students can reach achievements that permit them to gain badges for their account.
[$G8$] Users can visit the account of each other as part of a community, their stats and achieved badges are public.

1.7.2 Phenomena

1.7.2.1 World Phenomena

[$W1$] User downloads the application
[$W2$] An Educator decides in class to use the application to organize a coding tournament.
[$W3$] A Student decides to join a tournament/battle.
[$W4$] A Student contacts some mates to form a team.
[$W5$] A Student pushes a solution on GitHub.
[$W6$] A Student forks the CK repository on GitHub.

1.7.2.2 Shared Phenomena

Controlled by the world and observed by the machine

[$SW1$] A user creates an account and logs into the application.
[$SW2$] An Educator creates a tournament.
[$SW3$] A Student joins a tournament.
[$SW4$] An Educator creates a badge.
[$SW5$] An Educator creates a battle.
[$SW6$] A User decides to visualize his or another one's account page.
[$SW7$] An Educator performs a manual evaluation of the submitted solutions for a battle.

Controlled by the machine and observed by the world

[$SM1$] Student receives a notification regarding the creation of a tournament/battle.
[$SM2$] A Student is notified that the time for submitting their solution is terminating.
[$SM3$] A Student is shown he has achieved a badge.
[$SM4$] A User is shown his account or another one's, with all the badges and stats.
[$SM5$] The system evaluates a solution submitted by a student and gives them the score.

2 Overall Description

2.1 Product Perspective

2.1.1 Scenarios

1. **An Educator creates a tournament**
   John wants to create a coding tournament to make his students train their coding skills: he signs in to the system and clicks on the "My Tournaments" button on his home page, then he clicks on "Create new Tournament": at this point, he can set the name of the tournament, the list of collaborators (Colleagues that can create battles in the context of that tournament), and a subscription deadline for the students who want to join the tournament, after that time the subscription will be closed and the tournament will result as started.
   When he clicks on confirm the tournament has finally been created: now all the students will be notified and can join it.

2. **An Educator creates a battle**
   James wants to create a new battle for the tournament: once he has signed in he can go to the "My Tournaments" section and select a started tournament.
   If the tournament has no ongoing battles the function "Create new battle" is enabled: the Educator can then manage the battle settings. In particular, he has to configure:

   - The programming language(s) students are allowed to use for coding that the evaluating system will recognize
   - The maximum number of students for each team that will compete in the battle
   - The registration deadline for the battle
   - The submission deadline for the solutions
   - The evaluation system options (Totally automated, manual checking)

   He has finally to submit the text of the problem for the battle: it will be automatically uploaded in a repository created by the system at the registration deadline and the students will be automatically given the link to see it and start coding.

3. **An Educator creates a Badge**
   Steve wants to create a gamification badge: badges exist in the context of a tournament and can be created only during the tournament creation phase so he clicks on "My Tournaments" and, after clicking on "Create new Tournament" section, he clicks on "Create a Badge".
   Once reaches this section Steve is shown the list of variables offered by the system which represent properties, such as values of commits number, code length, code speed, joined battles, and so on: using those variables Steve has to write constraints, in form of boolean conditions, to declare the requirements a Student has to satisfy to achieve the badge; he also writes a brief description of the conditions in the dedicated section. Steve decides to create the badge with the following condition: "battle_stud_score $\geq$ 60", which means that each student that takes part in a team that gets more than 60 in a battle of the tournament will gain the badge. Steve also writes a brief description of the condition in the dedicated section. Once the condition is written the Educator uploads an image for the badge and gives it a name, and eventually clicks on "confirm": now the badge is available for all students of that tournament.

4. **A Student joins a tournament**
   Mark has been notified that a new tournament has been created and wants to join it: he can click on the notification, which will redirect him directly to the tournament page, or search for it by himself clicking on "Join a Tournament" in "My Tournaments" section of his homepage. Once he has found the tournament he clicks join and he's in: he can now see the list of the incoming battles to join them on his own or with his team.

5. **A Student joins a battle**
   Jim has seen that a battle is starting in a tournament he has joined and has decided to join it with 2 friends, who have also subscribed to the tournament.
   He selects the tournament from his one's list and then selects the battle: he can then see his team for the battle, to which he can invite people who have subscribed to the tournament by pushing the '+' button and a confirmation button.
   He invites his 2 friends selecting them from a list containing the subscribers and after some time he receives a notification that the invites have been accepted: At that moment he comes back to the battle page and sees his friend's name appeared in his team: now he clicks on confirm and the registration is confirmed, they have to wait for the beginning of the battle.

6. **A Student achieves a badge**
   Paul is competing in a tournament, the last battle of which has just ended: since the Educator who created the tournament defined a badge for the team having committed the biggest number of times, the system verifies that condition and Paul is resulting as the gainer of the badge.
   The system then sends a notification to him: when he sees it he goes to his profile homepage and in the "My Badges" section he sees the new badge he has achieved.
   Eventually, by clicking on it, he can see the information about the badge, including the description of the requirement from which he is now able to know how he has gained the achievement.

2.1.2 Domain Class Diagram

The Domain Class Diagram, depicted in Figure 1, illustrates both the overall structure of the CKB system and the components of the environment in which it functions.
For a better understanding, the main characteristics of the domain are described below.

- There are two types of users: Student and Educator.
  Although both of them have a username, as well as a profile page visible to other users, their roles are completely different; more specifically, students can only join tournaments and battles while educators can only create tournaments and battles.
- The set of rules for each badge is specifically created by the educator therefore each rule belongs to only one badge.

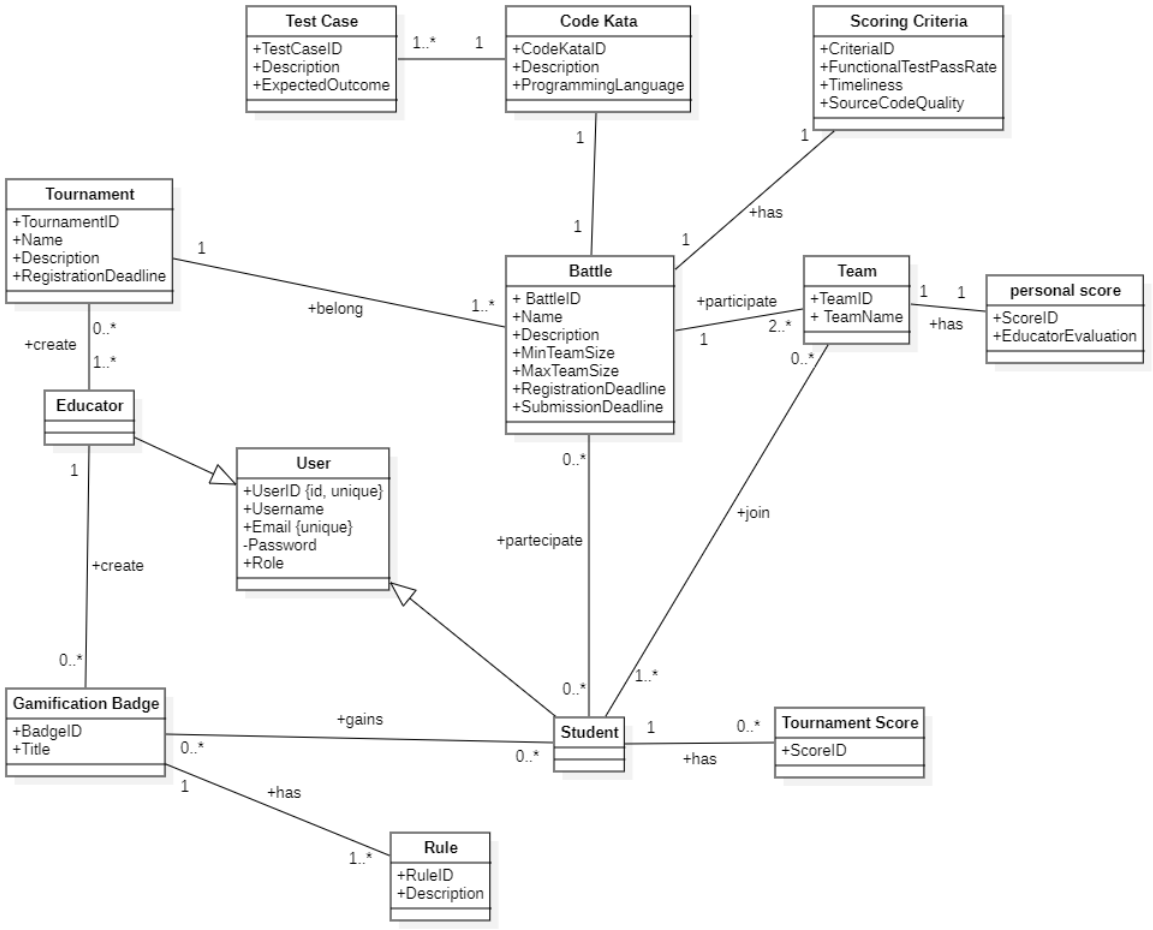
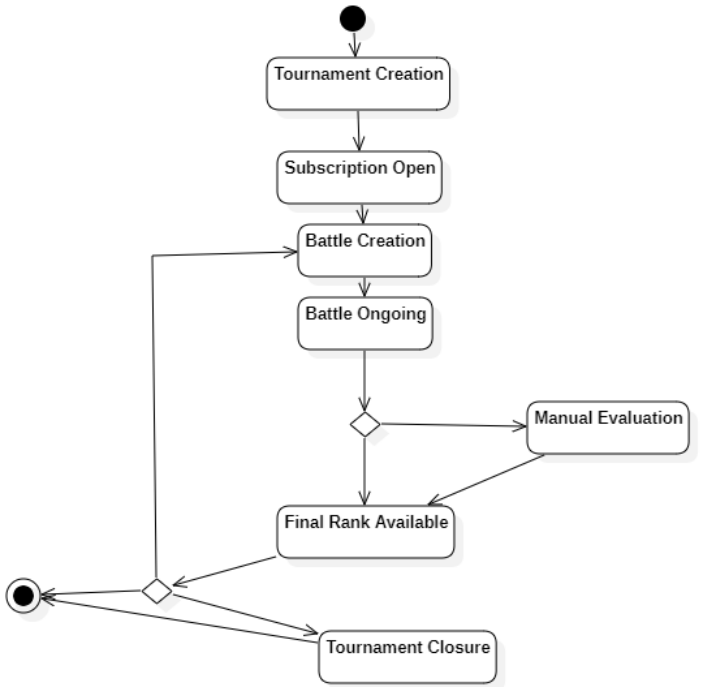


Figure 1: Domain Class Diagram

2.1.3 Statecharts

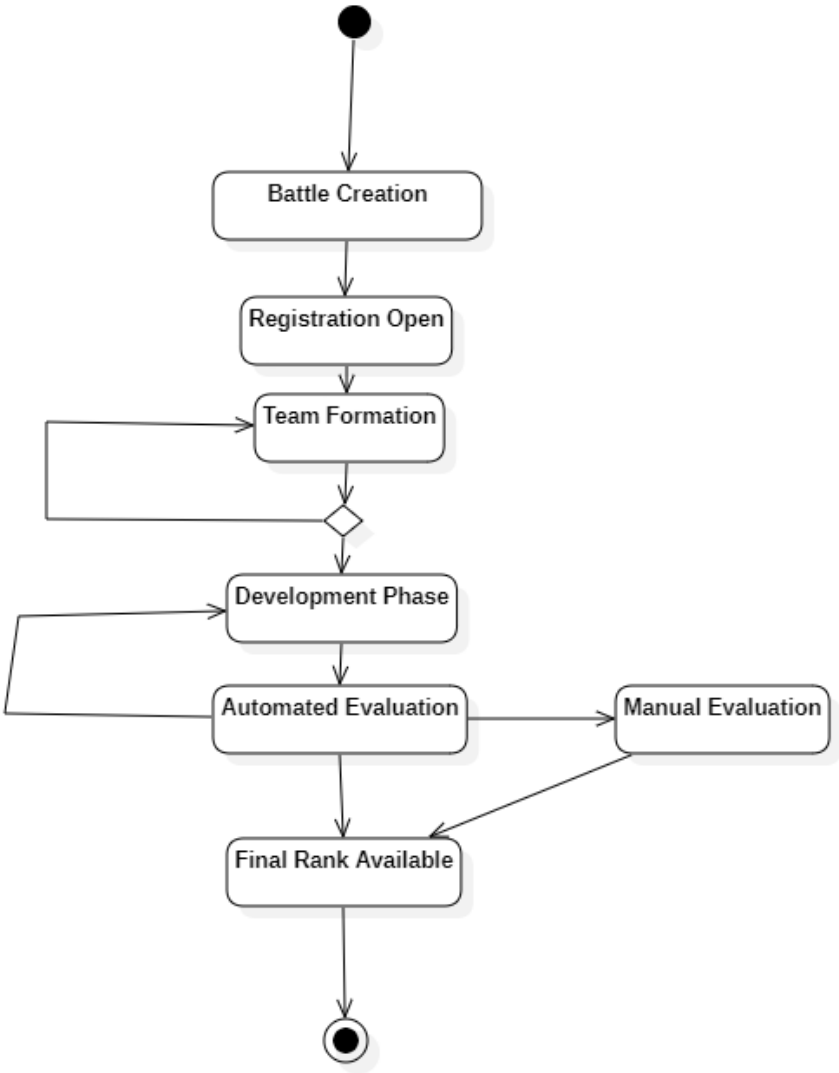
Two state charts are discussed in this section which are fundamental for understanding the life cycle of a tournament and the process to join a battle.

Tournament life cycle



The first state is *Tournament Creation*, in this state the educator creates the tournament and invites collaborators; Once done, the tournament goes in the *Subscription Open* state waiting for students to join the tournament.
After the registration deadline, educators can create battles for the students who joined the tournament in the *Battle Creation* state.
Students who successfully joined the battle can now compete by sending their solution before the deadline in the *Battle Ongoing* state; Their solution will be automatically evaluated every time they are submitted and after the end of the battle, if required, a *Manual Evaluation* is performed by the educator.
After the evaluation step, the *Final Rank Available* state is reached and educators can then decide whether to create new battles(and return to the *Battle Creation* state) or close the tournament reaching the final *Tournament Closure* state.

Battle life cycle



The first state is *Battle Creation*: in this state, the battle is being created by the Educator, who sets the available programming language(s), the minimum and maximum team size, the time slots for subscribing and submitting the solution, and eventually enables the manual evaluation, then proceeds to upload the files about the problem.
Then, when the battle has been successfully created, the *Registration Open* event is reached: after the registrations have opened students can proceed to start the *Team Formation*, inviting others who are participating in the tournament in their team for the battle.
When the deadline for subscription ends the battle starts: every team starts its *Development Phase*,

during which students work to find a solution to the problem of the battle.
Every time a solution is submitted the system performs an *Automated Evaluation* on it to give it a score, which is immediately visible in the provisional rank of the battle.
When the deadline for solution submission expires, a*Manual Evaluation*, if previously enabled while creating the battle, is performed by the Educator on all the submitted solutions.
After that, the battle has ended, and it reaches the *final Rank Available* state: the final results are visible and the tournament general rank is updated.

2.2 Product Functions

2.2.1 Sign Up and Login

These functions will be available to all the users: during the sign-up procedure everyone has to insert an e-mail and a password: these will be also the credentials to log in to the account that is being created.
The inserted e-mail will be verified from the system with a confirmation e-mail sent to the user.
Once this step is done the user will have to insert their personal data: Name, Surname, a username that will be the one who will be seen by the other users, and the type of user (Educator or Student).

2.2.2 Tournament creation

Each Educator can use the system to create tournaments for Students, that will compete singularly in them. When creating a tournament an Educator has to set the list of available programming languages, the number of battles and to give it a name, they can in addition choose to set it as private protecting it with an alphanumerical access key.
The tournaments can also be handled by more than one Educator: collaborators can be added from the creation page, and they can create battles and manually evaluate the solutions provided by the Students if needed, but they cannot create badges. Once an Educator has created a tournament every user in the system is notified and can join it. The creator has to set a subscription deadline: as the deadline expires the tournament status passes from opened to ongoing and no one can join it anymore.

2.2.3 Tournament subscription

Each student can subscribe to a tournament: once they have found a tournament in which they want to compete, exploring the list or searching it there by name or creator, they can click and join it, inserting the access key if it's required.

2.2.4 Battle creation

In the context of an ongoing tournament Educators have to set the options of each battle before making it start: they have to choose the maximum team size in that battle, the language(s) for that specific battle picked from the tournament's ones, and to set the subscription and solution submission deadline. He has also to decide if the solutions will be evaluated totally by the system or if he also wants to manually check the solutions at the end of the battle. After having done that the Educator has to upload the text and test cases for the problem that the Students have to solve

in that battle: when the Educator confirms the creation of a battle a repository containing the uploaded files will be created on GitHub; it will be made available to all Students involved in the battle when the deadline expires. Once all this has been done the Educator confirms the creation of the battle: all the Students that are subscribed to the tournament are notified and the timer for the registration starts.

2.2.5 Battle joining

Each student subscribed to a tournament can join its battles: they receive a notification every time a new battle is ready to start and they have time to join it until the deadline set by the Educator has expired.
As the deadline for subscribing expires each competing student will be allowed to see the link of the repository where they will find the text of the exercise and the test cases that the educator uploaded: each team has to fork the repository and paste the link of his copy in an apposite form in the app so that the System can visit the repository and check the solutions when it's needed.

2.2.6 Code Evaluation

The System, during a battle, will use CodeScene, a tool for Code Static Analysis, to give a score to each competitor: the score is assigned each time a solution is pushed on the GitHub repository so that a live provisional rank is always visible for all the duration of the battle to all the Students involved. When the battle ends, and after the eventual manual evaluation by the Educator, Students will be able to see the final scores of all the competitors in the rank of the battle, and also the updated tournament rank.
The score is a number from 0 to 100, determined based on criteria like correctness but also time and space efficiency and so on. If the manual evaluation is disabled for the battle the score will be entirely assigned by the System, otherwise, it will assign at maximum only a part of the available 100 points, to which it will be summed the other part of the score that will be assigned from the Educator.

2.2.7 Gamification Badges

Educators are allowed, in the context of a tournament, to create Gamification Badges: they can be only added by an educator when he is creating a tournament, neither he nor the eventual collaborator can modify them or add other ones any later .
Badges are achievements that the Students can gain by satisfying a requirement.
When creating a badge, an educator has to give it a name, optionally an image, and to write a brief description of the requirement: these are the things that students will be able to see.
To make the system be able to analyze the requirement and eventually assign the badge when it's necessary, the Educator has also to write it as a boolean condition built on variables given by the System.
The variables represent the most important stats regarding battles and tournaments, and they are related to many aspects:

- **Score**: there is a variable related to the tournament and battle score of a single student

- **Commits**: there are variables related to team commit number and to the maximum number of commits made by a team in a battle.
  There are also the same variables regarding single students in the context of the whole tournament.
- **Code**: there are variables about the code correctness, the execution speed for every single team and two variables representing the fastest working code in the battle and the first team to submit a correct solution to the problem.
- **Rank**: there are variables representing the rank position of each student in a battle or in the total rank of the tournament.

If a parameter value changes the system verifies the conditions that have been created and, if someone has satisfied it, gives him the badge and sends him a notification.

2.2.8 Stats and Rankings

Each user is allowed to explore other accounts: in particular, the tournament ranks (ended and ongoing ones) and the badges of a user are public; everyone can see them clicking on their account.

2.3 User Characteristics

The system is designed to interact with the following two different kinds of users:

2.3.1 Educator

An educator is a user who registers through the CKB app by creating an "educator account", it can create tournaments and battles, invite collaborators to the tournament, evaluate the solutions provided by the students, and create badges for each tournament (optional).

2.3.2 Student

A student is a user who registered through the CKB app by creating a "student account", it can join tournaments and battles, invite other students in their current team, and submit solutions for the battles in which it joined.

2.4 Assumptions, dependencies and constraints

2.4.1 Regulatory Polices

The CKB application will ask for user's 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

The following are the domain assumptions that need to be satisfied for the correct behavior of CKB. Since those situations are out of the control of the system, they are taken for granted.

[$D1$] User must have an internet connection.
[$D2$] User must have a GitHub account.
[$D3$] User must know how to use GitHub properly (push, pull, branch, etc.).
[$D4$] Connection between systems must be reliable.
[$D5$] Test cases provided by educators must be correct.
[$D6$] GitHub works as expected.
[$D7$] The static Analysis tool works reliably.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

This section presents the user interface through which both students and educators can:

- Log in
- Sign Up
- Access personal dashboard

The needs of educators and students are not the same, therefore they are going to get access to different views:
Educators have access to specific views that allow them to handle tournaments and battles while students must be able to join tournaments and battles, as well as invite members in their group.

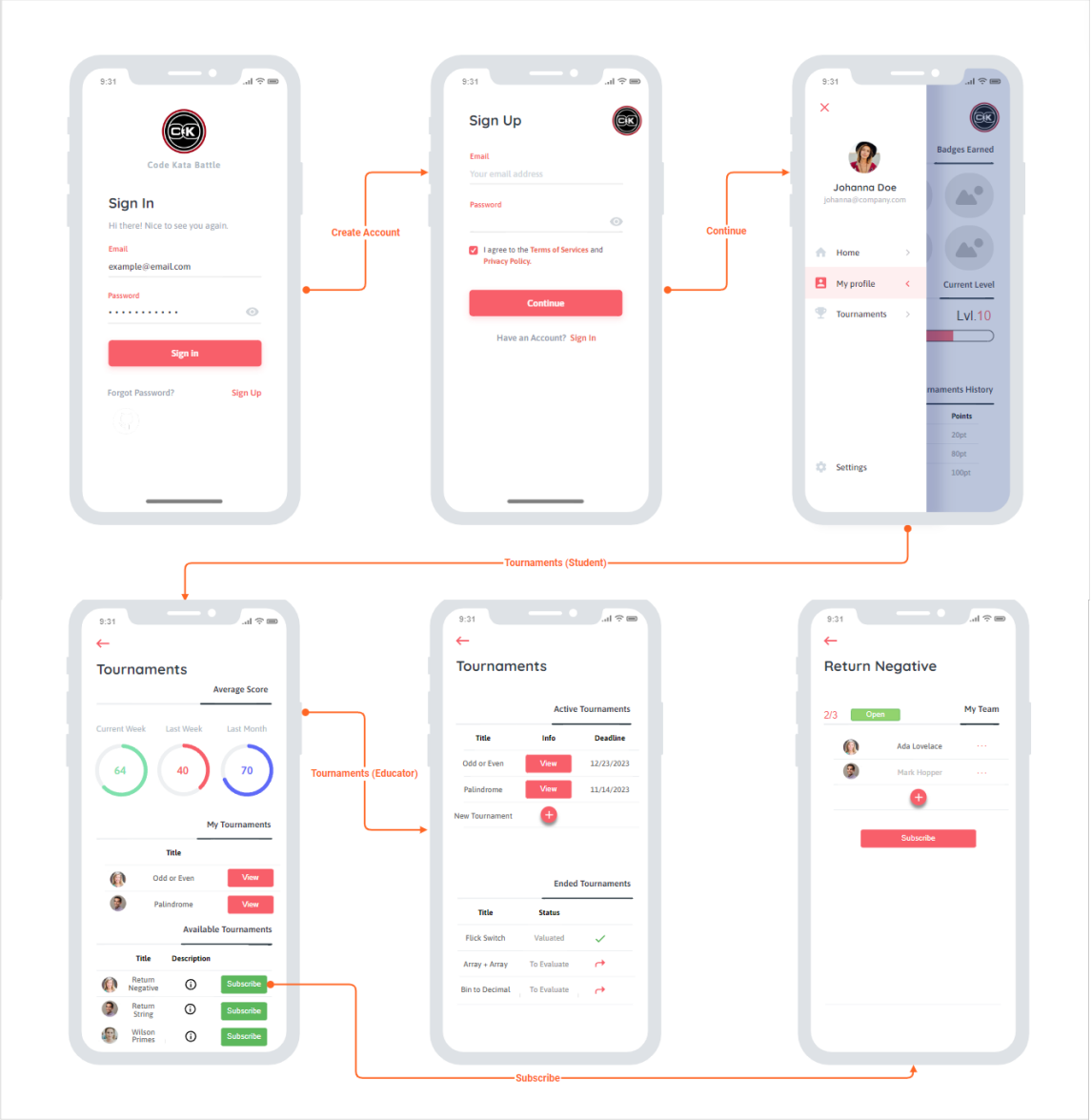


Figure 2: Mobile User Interface

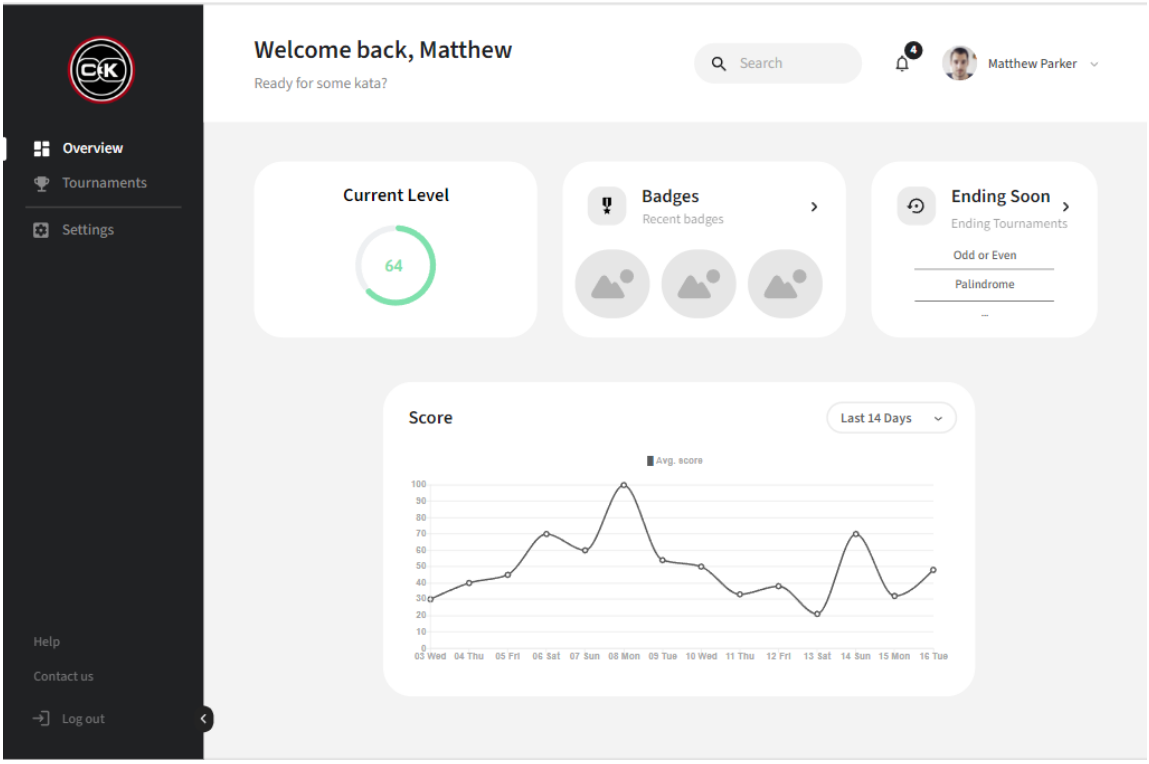


Figure 3: Web User Interface

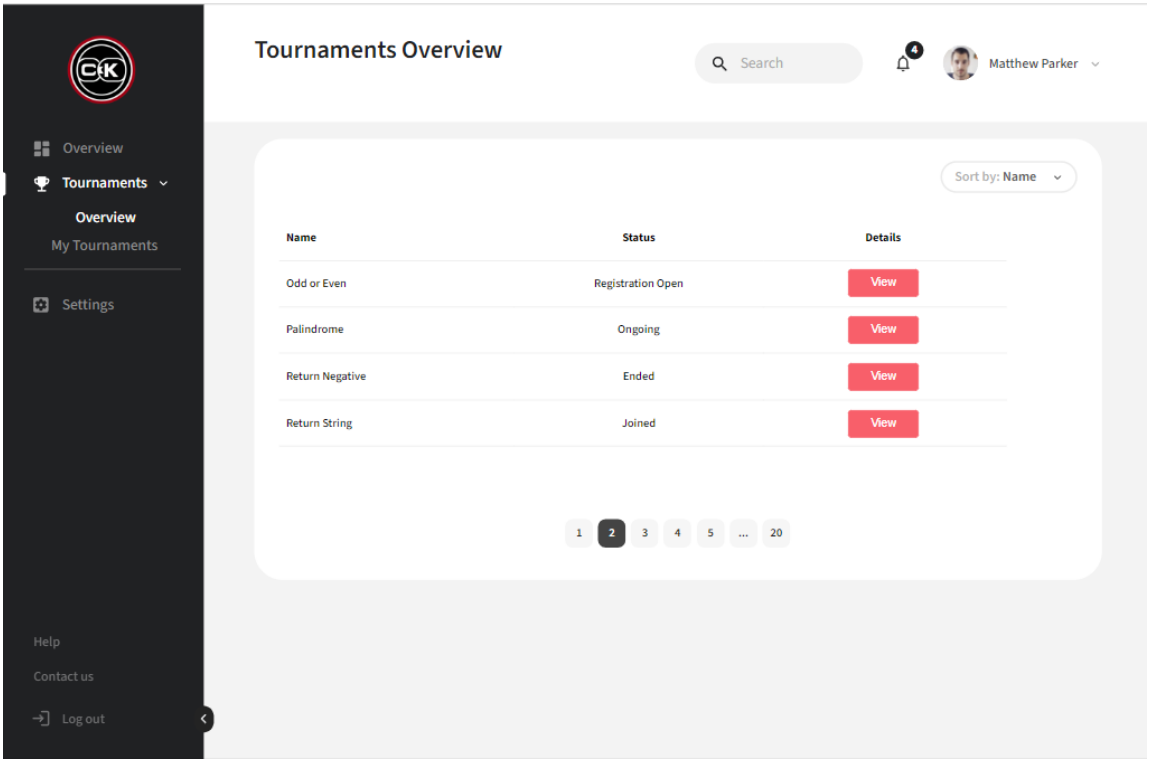


Figure 4: Web Interface Student

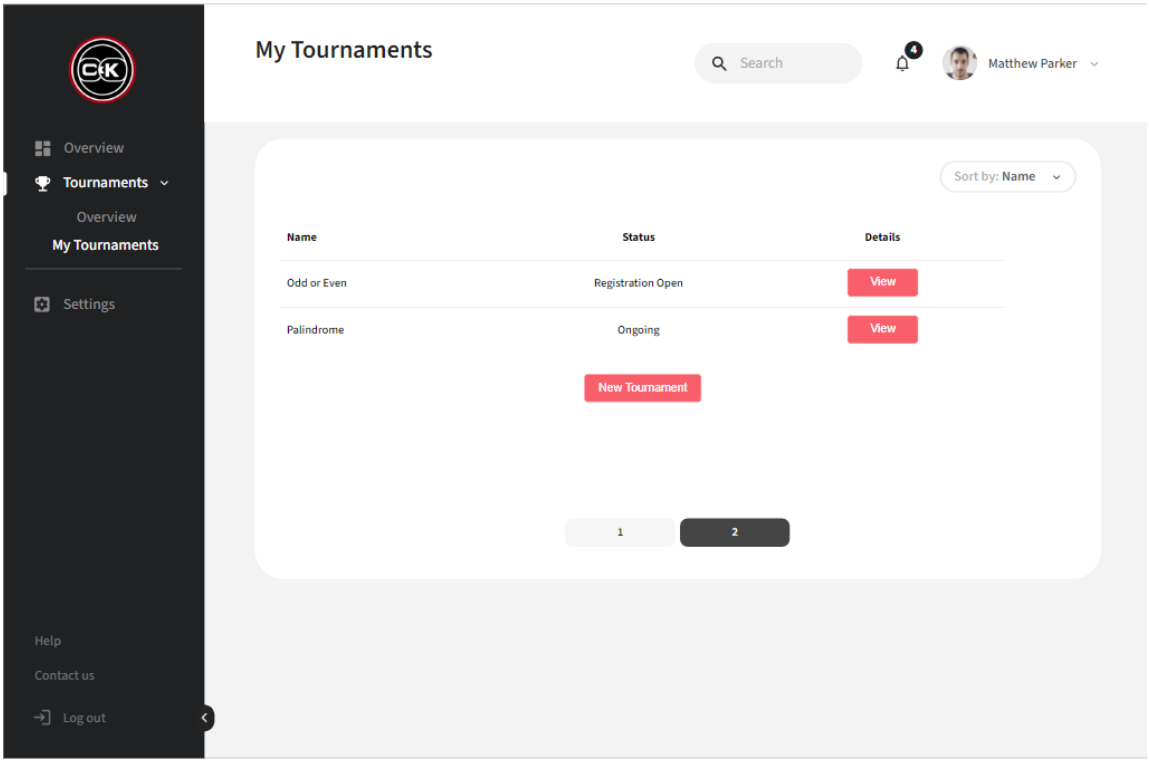


Figure 5: Web Interface Educator

3.1.2 Hardware Interfaces

The system offers to the users different services (see paragraph **2.2**) that can be accessed by any device with an internet connection whether it's through the app or web app.

3.1.3 Software Interfaces

The system relies on different software interfaces, more specifically:

- **GitHub API:** The system uses this API to create the repository of the battle, moreover, can receive notifications from GitHub every time a new solution to the problem has been uploaded by a student.
- **CodeScene API:** The system uses this API to evaluate all the solutions uploaded by students: every time a new solution is uploaded, the system sends it to CodeScene for evaluation and then, converts the result into an integer number.

3.1.4 Communication Interfaces

The System uses an Internet connection to communicate with GitHub and all the users: it should have a uniform interface so that the communication is always reliable.

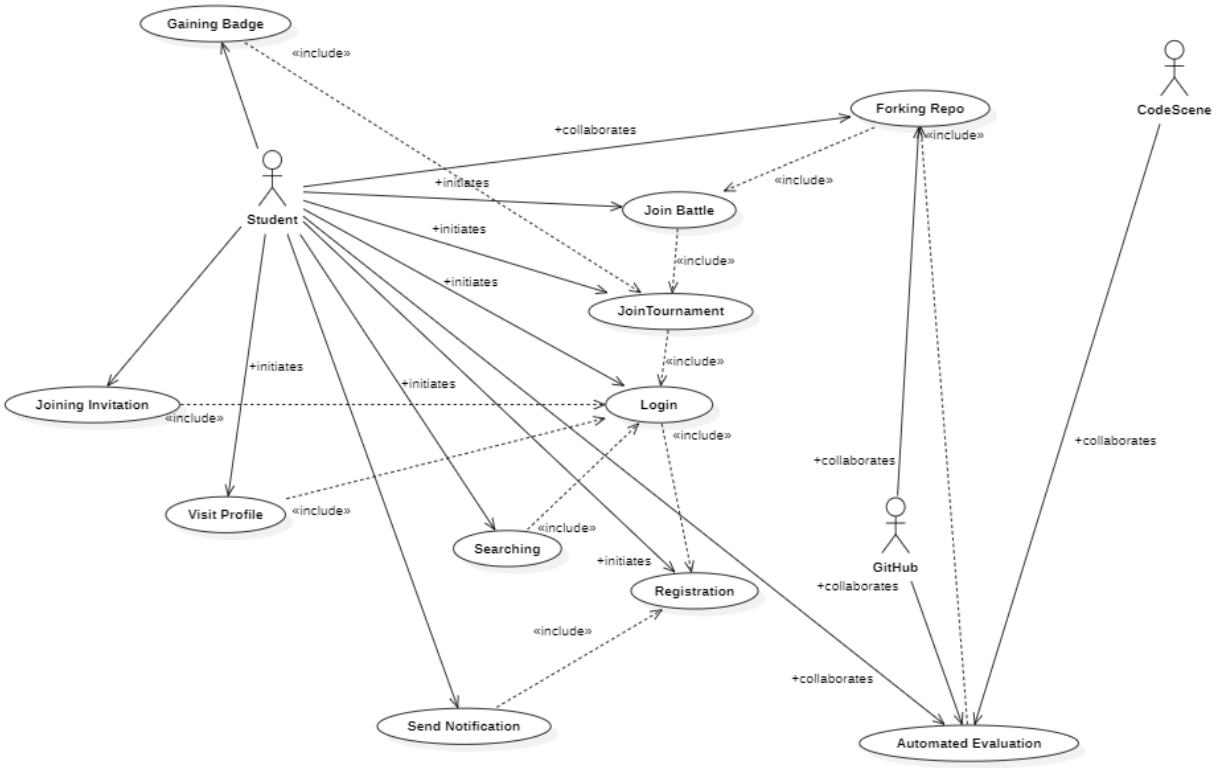
3.2 Functional Requirements

[$FR1$] The system allows Users to sign up.
[$FR2$] The system allows Users to log in.
[$FR3$] The system allows Educators to create tournaments.
[$FR4$] The system allows Educators to set a deadline for subscribing to the tournament.
[$FR5$] The System allows Educators to set a list of programming languages that will be used in the tournament.
[$FR6$] The system allows Educators to invite other Educators as collaborators for a tournament
[$FR7$] The system allows Educators to set the number of battles in a tournament.
[$FR8$] The system allows Educators to create battles.
[$FR9$] The system allows Educators to set a deadline for the subscription to the battle.
[$FR10$ The system allows Educators to set a deadline for the submission of the solutions to the battle.
[$FR11$] The System allows educators to choose which of the languages of the tournament will be used in each battle.
[$FR12$] The System allows Educators to upload the files with the text of the problem and test cases for the battle
[$FR13$] The system allows Educators to set the maximum number of students per team in a battle.
[$FR14$] The system allows Educators to choose whether and when the scores will be assigned in a totally automatic way or also through a manual verification by themselves
[$FR15$] The system allows Educators to create badges for a specific tournament.
[$FR16$] The system allows Educators to build the requirements for achieving a badge.
[$FR17$] The system allows Students to see what are the active badges in a tournament and what they have to do to achieve them.
[$FR18$] The system allows Students to join a tournament.

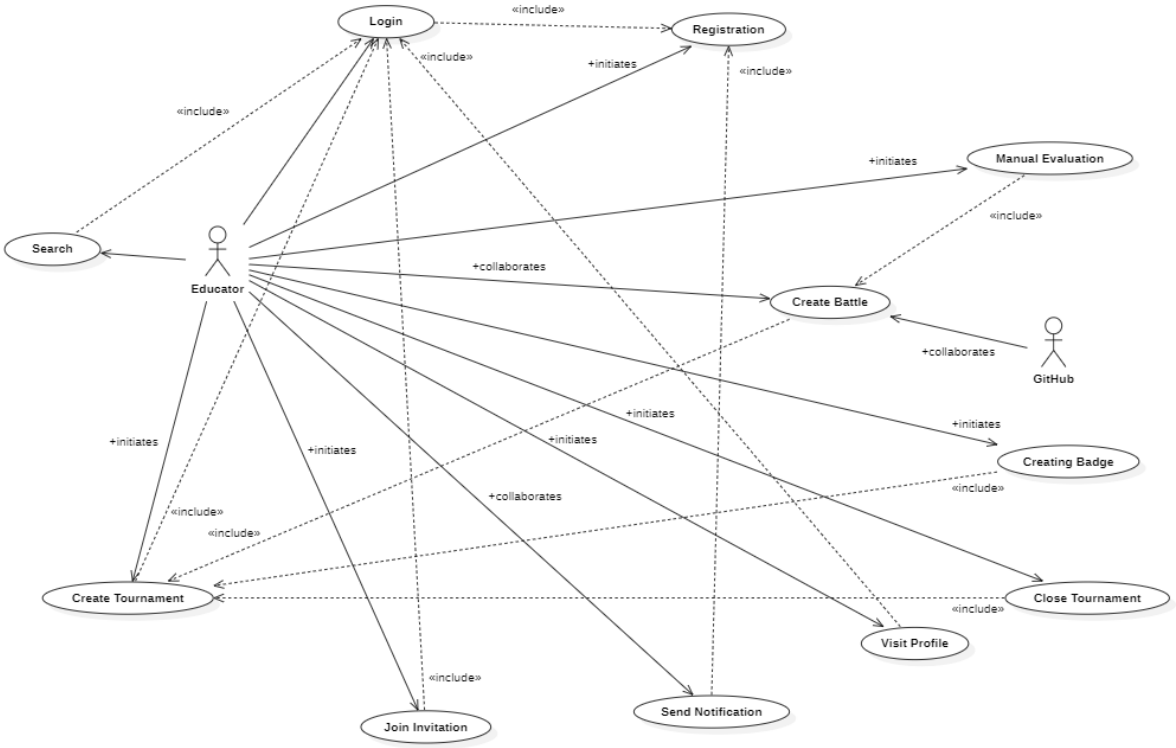
[$FR19$] The system allows Students to invite other students to form teams for a battle
[$FR20$] The system allows Students to join a battle.
[$FR21$] The system allows Students to know about the provisional scores of all students, updated for every solution submitted, for all the duration of a battle
[$FR22$] The system allows Educators to see the list of the active tournaments they have created.
[$FR23$] The system allows Students to see the list of the active tournaments they have joined.
[$FR24$] The system allows Users to see the stats of any other account registered in the system
[$FR25$] The system allows Users to search for other accounts or tournaments on the search bar

3.2.1 Use Case Diagrams

Student Use Case Diagram



Educator Use Case Diagram



3.2.2 Use Cases

The following are the use cases that can happen within the system:

[UC1] Registration

Name	Registration
Actor	User
Entry Condition	The user has opened the application
Event Flow	1. The user opens the application 2. The user clicks on the button" Sign Up" 3. The system shows the registration form 4. The user inserts his email and a password 5. The system sends a confirmation email to the user through an email provider 6. The user confirms his email by clicking on the link received by email 7. The system shows a form asking for personal information, including the type of account the user wants to create (Student or Educator) 8. The user inserts the requested data
Exit Condition	Registration has been successful. The user's data is stored in the system's database. The user then will be able to log into the system by using their credentials.
Exception	The user is already registered to the system. The system will send a notification message telling the user that there is an account with that email.

[UC2] User Login

Name	User Login
Actor	User
Entry Condition	The user has opened the application
Event Flow	1. The user inserts his username and password in the form 2. The user clicks on the "Login" button 3. The system checks the credentials 4. The application shows the dashboard
Exit Condition	The user has access to the services
Exception	The data inserted is not valid. The system returns to the entry condition.

[UC3] Create Tournament

Name	Create Tournament
Actor	Educator
Entry Condition	The educator is logged into the platform
Event Flow	1. The educator clicks on the button to create a tournament 2. The educator puts information about the tournament 3. The educator creates badges **[UC7]** 4. The educator invites other educators to join the tournament as collaborators 5. The educator clicks on the "Confirm Creation" button 6. The application shows the tournament environment
Exit Condition	The educator successfully created the tournament

[UC4] Create Battle

Name	Battle
Actor	Educator, GitHub
Entry Condition	The educators logged into the platform and selected the tournament in which they wanted to create a battle
Event Flow	1. The educator clicks on "create battle" button 2. The educator specifies information about the battle (name, text information, subscription deadline, submission deadline, etc. . .) 3. The educator uploads the track of the problem and the test cases 4. The educator clicks on the "confirm creation" button 5. The system creates the repo with the uploaded files 6. The system sends a notification to users about the creation of the battle **[UC9]**
Exit Condition	The educator successfully created the battle

[UC5] Join Battle

Name	Join Battle
Actor	Student, GitHub
Entry Condition	User logged into the application and is on the battle page; The battle is open for registration
Event Flow	1. The student clicks on the "join battle" button 2. The student sees all the information about the battle 3. The student sends the invitations to other students and waits for acceptances/rejections to form their team 4. Students click on "Confirm"
Exit Condition	Student/Team successfully joined the battle

[UC6] Student Joins Tournament

Name	Student Joins Tournament
Actor	Student
Entry Condition	User logged into the application and is on the tournament page; The tournament is open for registration
Event Flow	1. The student clicks on the "Join Tournament" button 2. The student sees all the information about the tournament 3. The student clicks on the "Confirm Joining" button
Exit Condition	The student joined the tournament successfully

[UC7] Create Badge

Name	Badge Creation
Actor	Educator
Entry Condition	The educators logged into the platform and started the creation of the tournament
Event Flow	1. The educator clicks on the "Create Badge" button 2. The educator chooses a title for the badge 3. The educator uploads an image for the badge 4. The educator writes the description of requirements for the badge 5. The educator builds the conditions with variables for the badge 6. The educator clicks on the "confirm badge" button
Exit Condition	The educator successfully created a badge
Exception	The educator didn't build the conditions in step **5** correctly

[UC8] Assigning Badge

Name	Assigning Badge
Actor	Student
Entry Condition	The educators have defined badges with associated rules and criteria; A battle or the tournament has ended
Event Flow	1. The system verifies the fulfillment of conditions for achieving a badge 2. The system eventually awards the badge to the Student
Exit Condition	The student receives the badge

[UC9] Notify Participants

Name	Notification System
Actor	User
Entry Condition	A significant event occurs within the CKB platform
Event Flow	1. The system identifies a significant event 2. The system chooses the nature of the event 3. The system chooses the Users to whom the notification will be sent 4. The system generates notifications containing relevant information 5. The system sends out notifications
Exit Condition	User receive the notification successfully

[UC10] Automated Evaluation

Name	Automated Evaluation
Actor	Static Analysis Tool
Entry Condition	The student/team pushes a new commit to the main branch of their forked repository containing their code solution
Event Flow	1. The CKB platform retrieves the latest source code from the Student/Team's repository related to the specific battle 2. The system sends code and test cases to the Static Analysis Tool 3. The Static Analysis Tool analyzes the code 4. The Static Analysis Tool sends back the results to the System 5. The system calculates a score 6. The system updates the battle score of the respective Team 7. The system notifies participants [**UC9**]
Exit Condition	The battle score is updated

[UC11] Manual Evaluation

Name	Manual Evaluation
Actor	Educator
Entry Condition	The battle has ended and the solutions have been submitted, manual evaluation enabled
Event Flow	1. The educator accesses the battle page 2. The educator selects the link of a solution he wants to evaluate 3. The educator evaluates the solution 4. The educator inserts the score in the system (the sum of the maximum score assignable by the system and the one assignable by the Educator is 100) 5. The educator confirms the inserted scores
Exit Condition	Final marks are calculated and published by the system
Exception	• Invalid score assigned by the educator • Confirmation while one or more scores are missing

[UC12] Creating Repository Fork

Name	Creating Fork
Actor	Student, GitHub
Entry Condition	Subscription deadline for the battle has expired
Event Flow	1. The student accesses the battle page 2. The system shows the User the link to the GitHub main repository for the battle 3. The student creates a branch for the repository 4. The student sends the link of the branch to the system 5. The student clicks on "confirm" 6. The system checks that the link is valid
Exit Condition	Student/team successfully registered their branch
Exception	Invalid link

[UC13] Close Tournament

Name	Tournament Closure
Actor	Educator
Entry Condition	All battles have ended
Event Flow	1. The educator clicks on "Close Tournament" 2. Students are notified that the tournament has ended [**UC9**] 3. The system shows the final results on the tournament homepage
Exit Condition	The tournament is closed
Exception	Ongoing battle

[UC14] Visit Profile

Name	Visit Profile
Actor	User
Entry Condition	User has logged into the system
Event Flow	1. The user clicks on an account or searches it **[UC16]** 2. The system shows the account information to the user
Exit Condition	The user can see the requested profile

[UC15] Joining Invitation

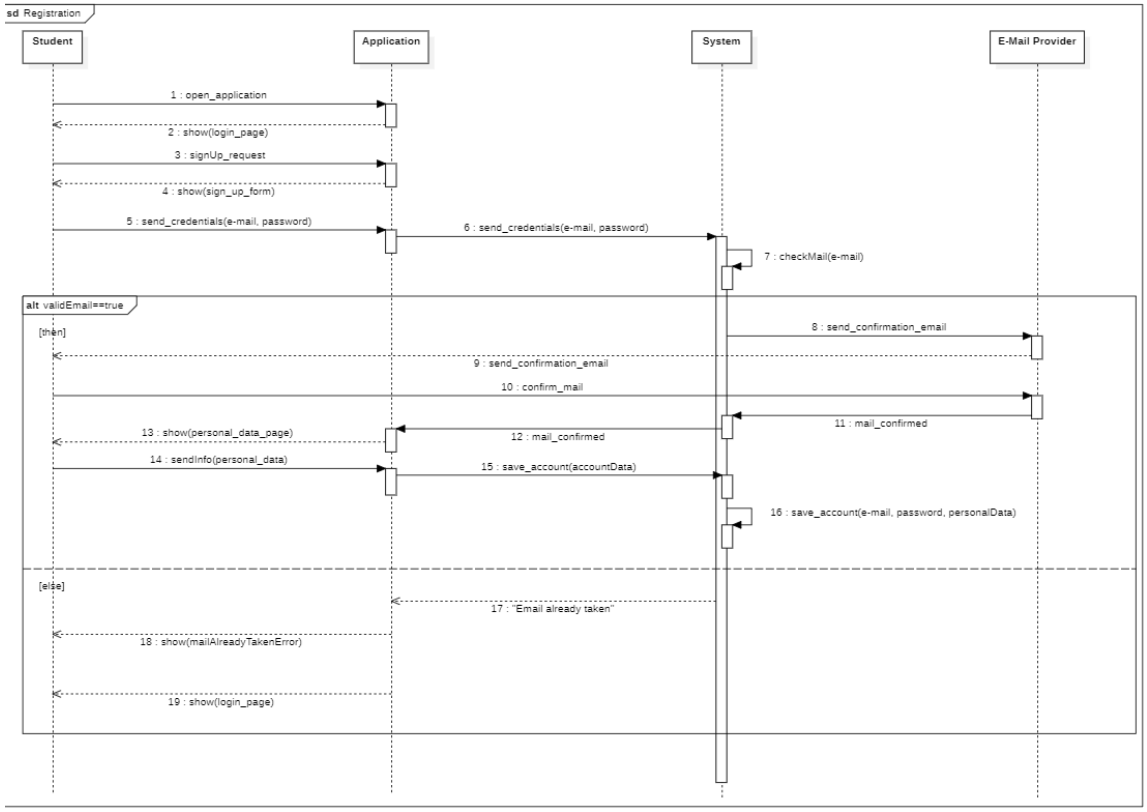
Name	Joining Invitation
Actor	User
Entry Condition	A user has received an invitation notification and has logged into the system
Event Flow	1. The user accesses the notifications section 2. The user clicks on the notification 3. The system shows the information about the invitation 4. The user accepts or declines the invitation
Exit Condition	The invitation has been accepted or declined
Exception	The invitation has been retired

[UC16] Searching

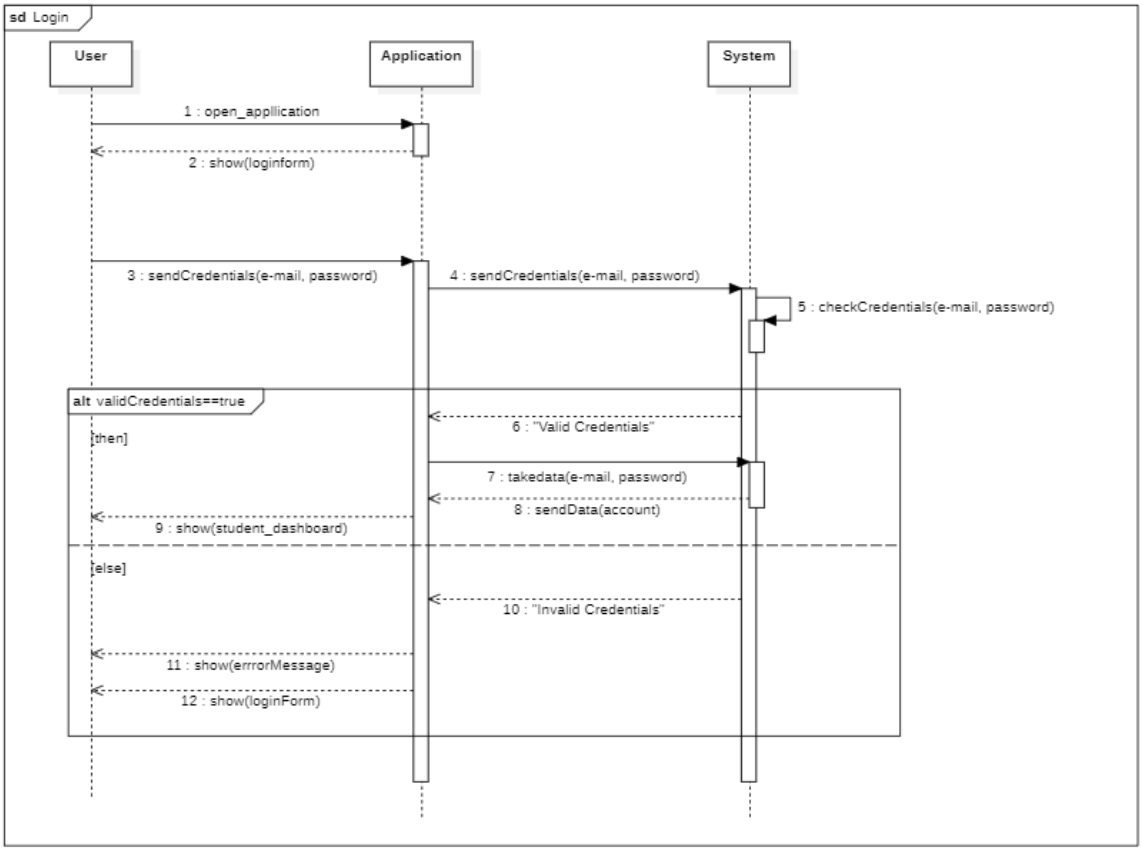
Name	Searching
Actor	User
Entry Condition	User has logged into the system
Event Flow	1. The user clicks on the search bar 2. The user digits a keyword 3. The system shows the results of the research
Exit Condition	The user can see the result of his research

3.2.3 Sequence Diagrams

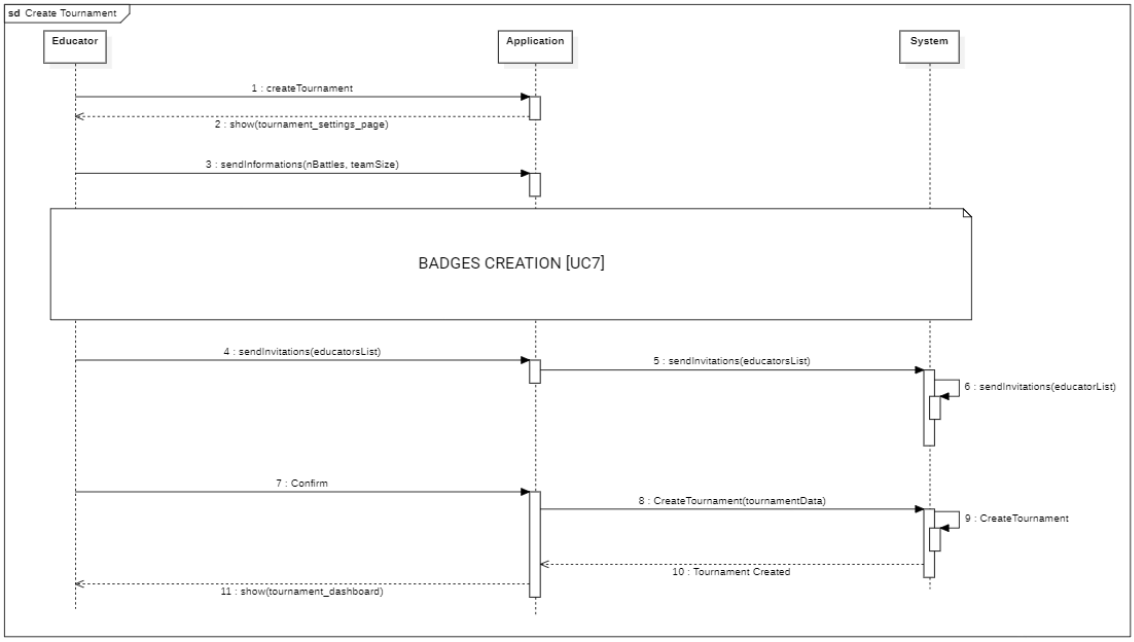
[UC1] Registration



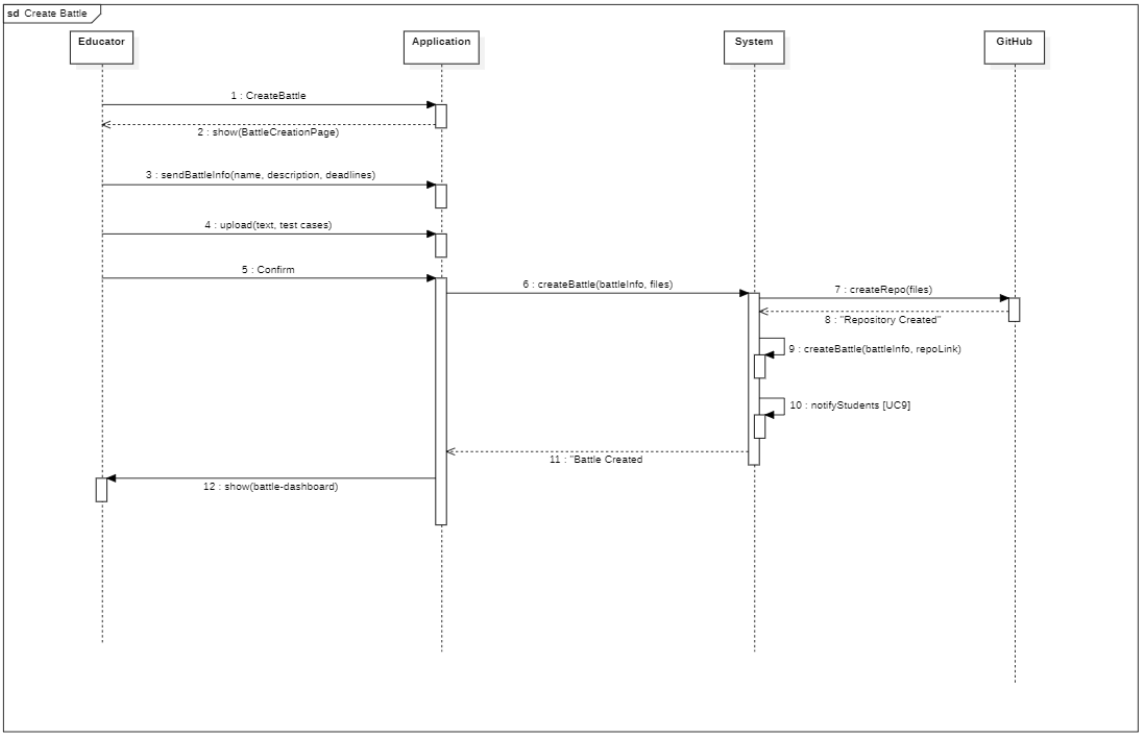
[UC2] Login


sd Login
User
Application
System
1 : open_appllication
2 : show(loginform)
3 : sendCredentials(e-mail, password)
4 : sendCredentials(e-mail, password)
5 : checkCredentials(e-mail, password)
alt validCredentials==true
6 : "Valid Credentials"
[then]
7 : takedata(e-mail, password)
8 : sendData(account)
9 : show(student_dashboard)
[else]
10 : "Invalid Credentials"
11 : show(errrorMessage)
12 : show(loginForm)

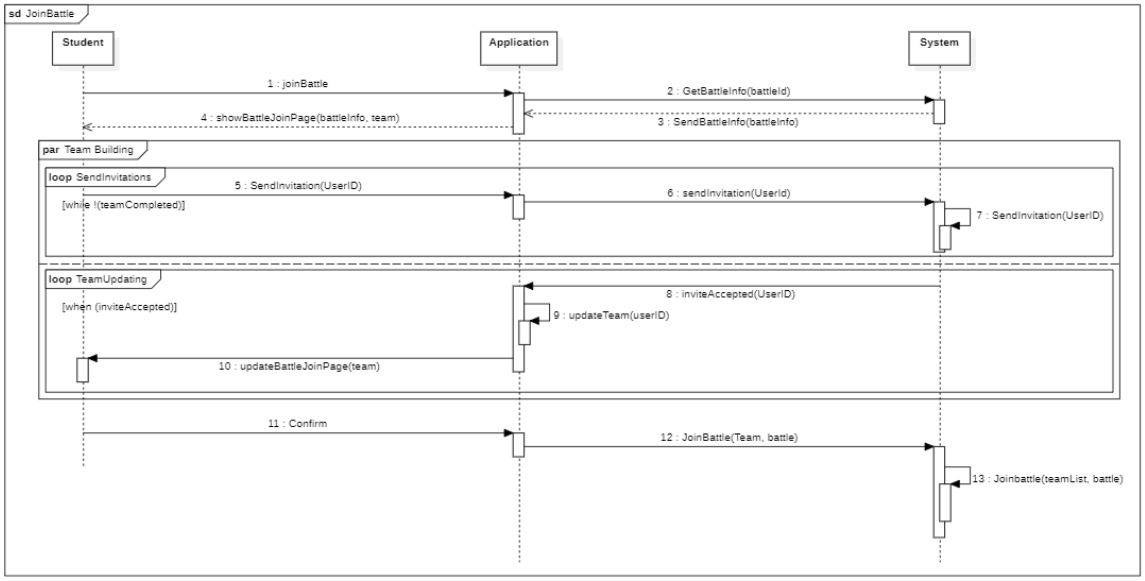
[UC3] Create Tournament



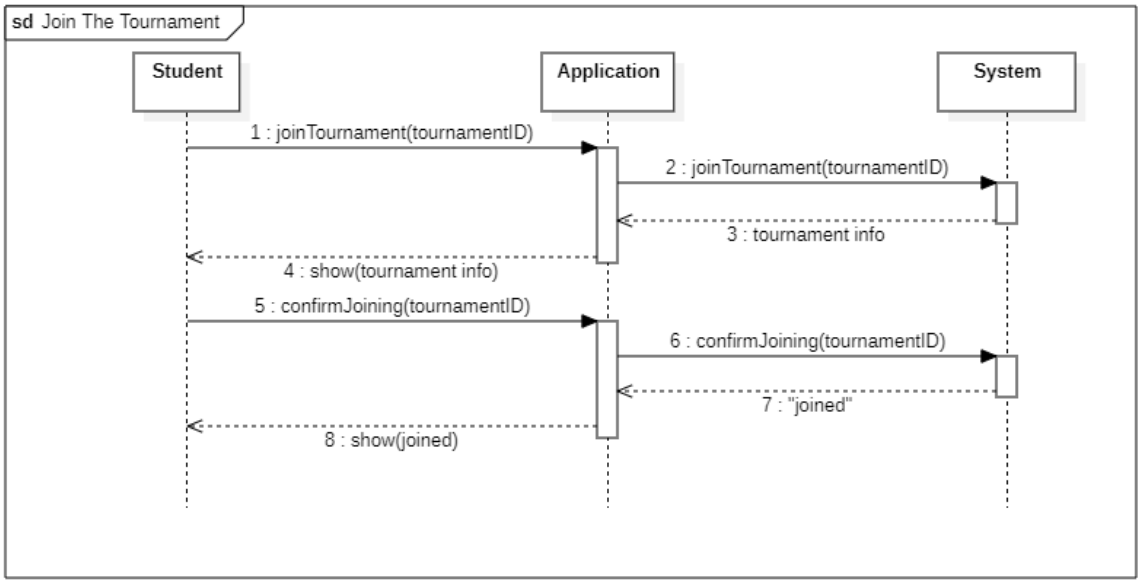
[UC4] Create Battle



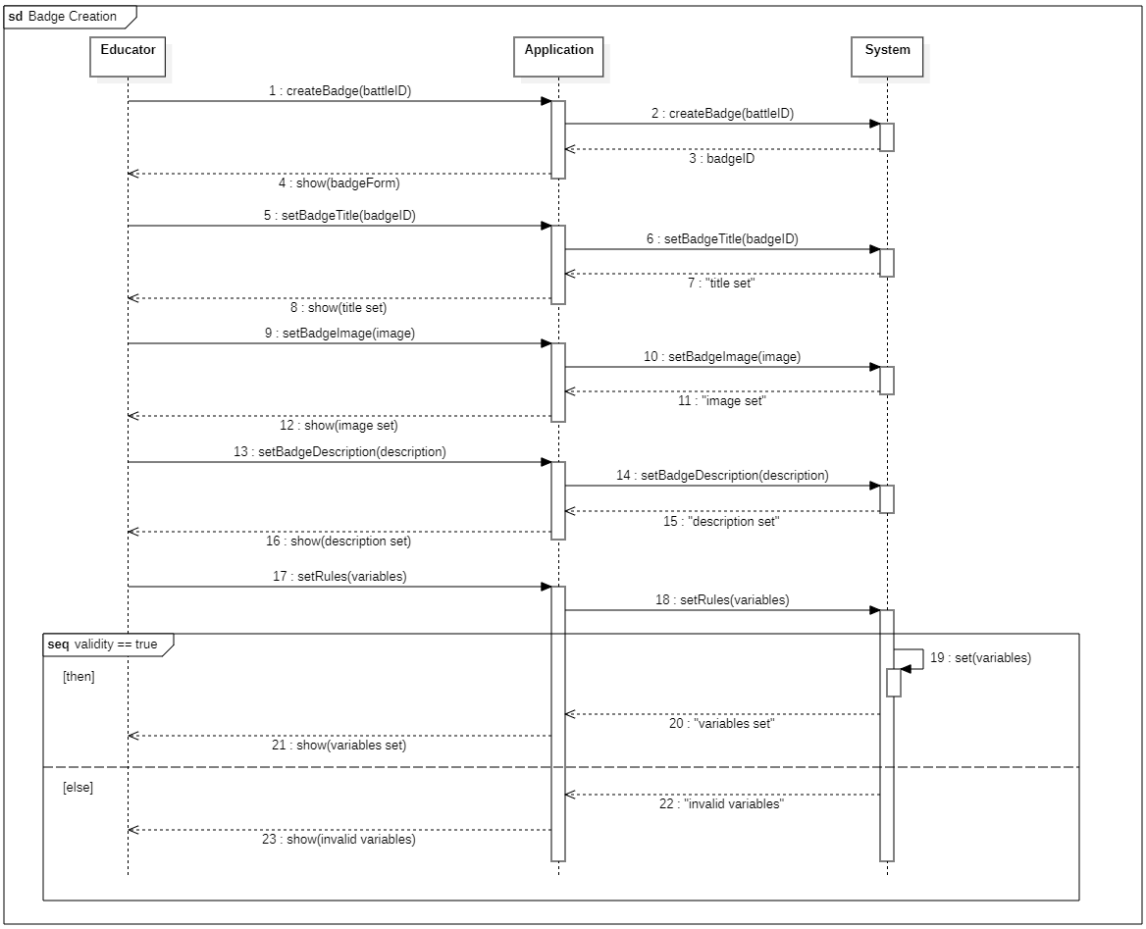
[UC5] Join Battle



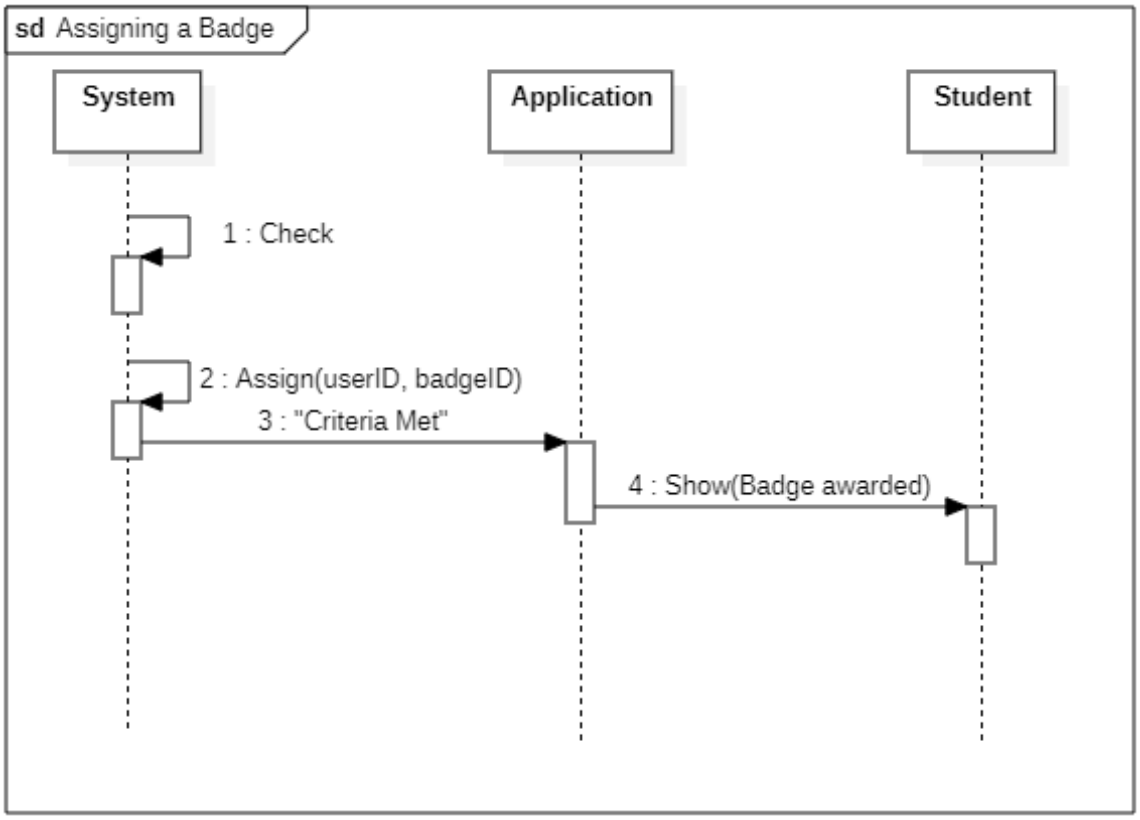
[UC6] Join The Tournament



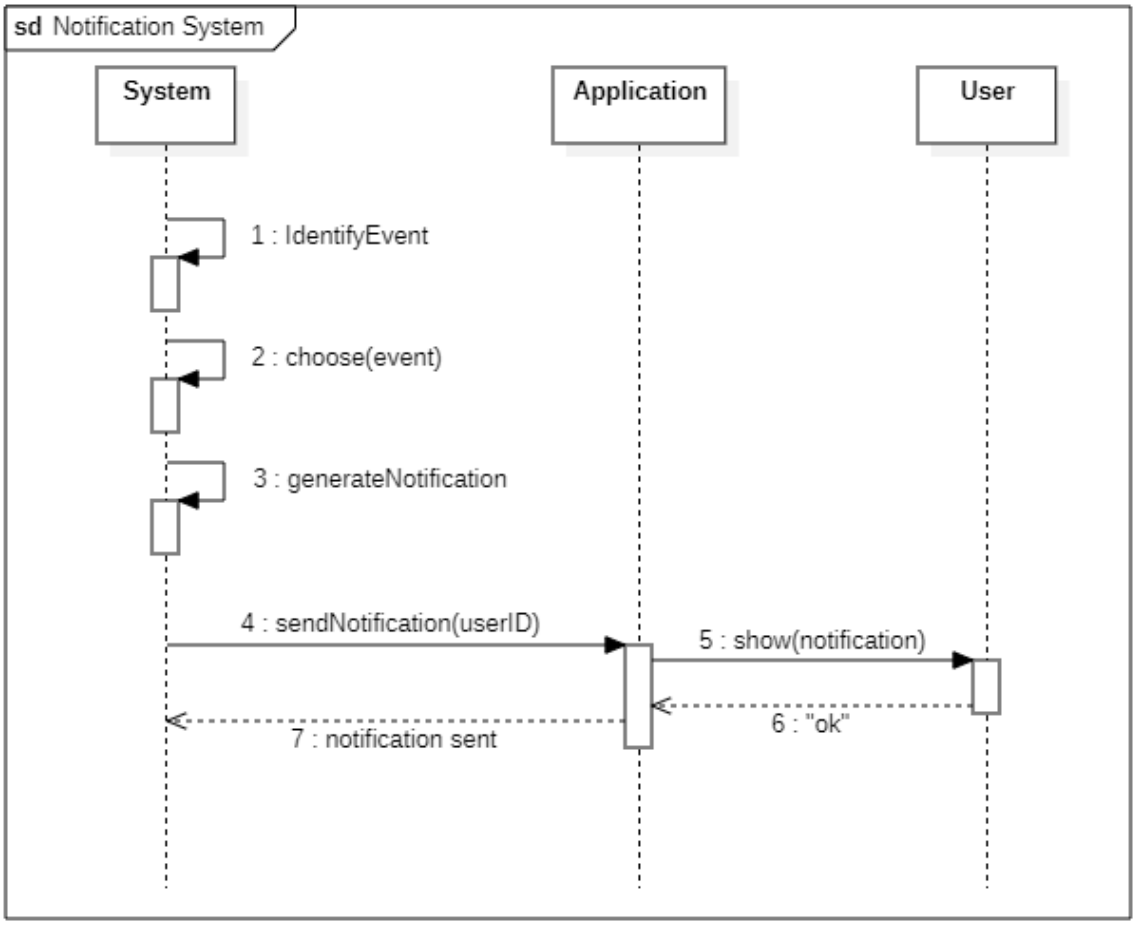
[UC7] Badge Creation



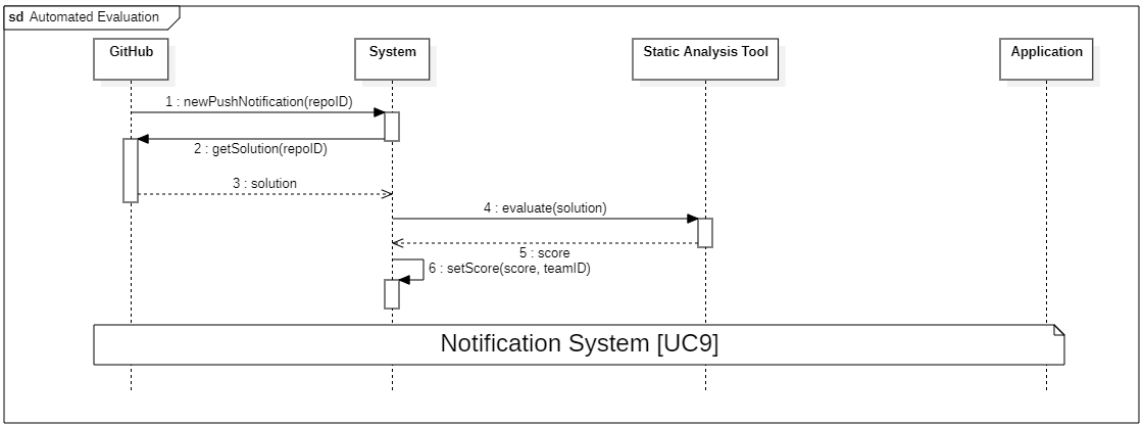
[UC8] Assigning A Badge



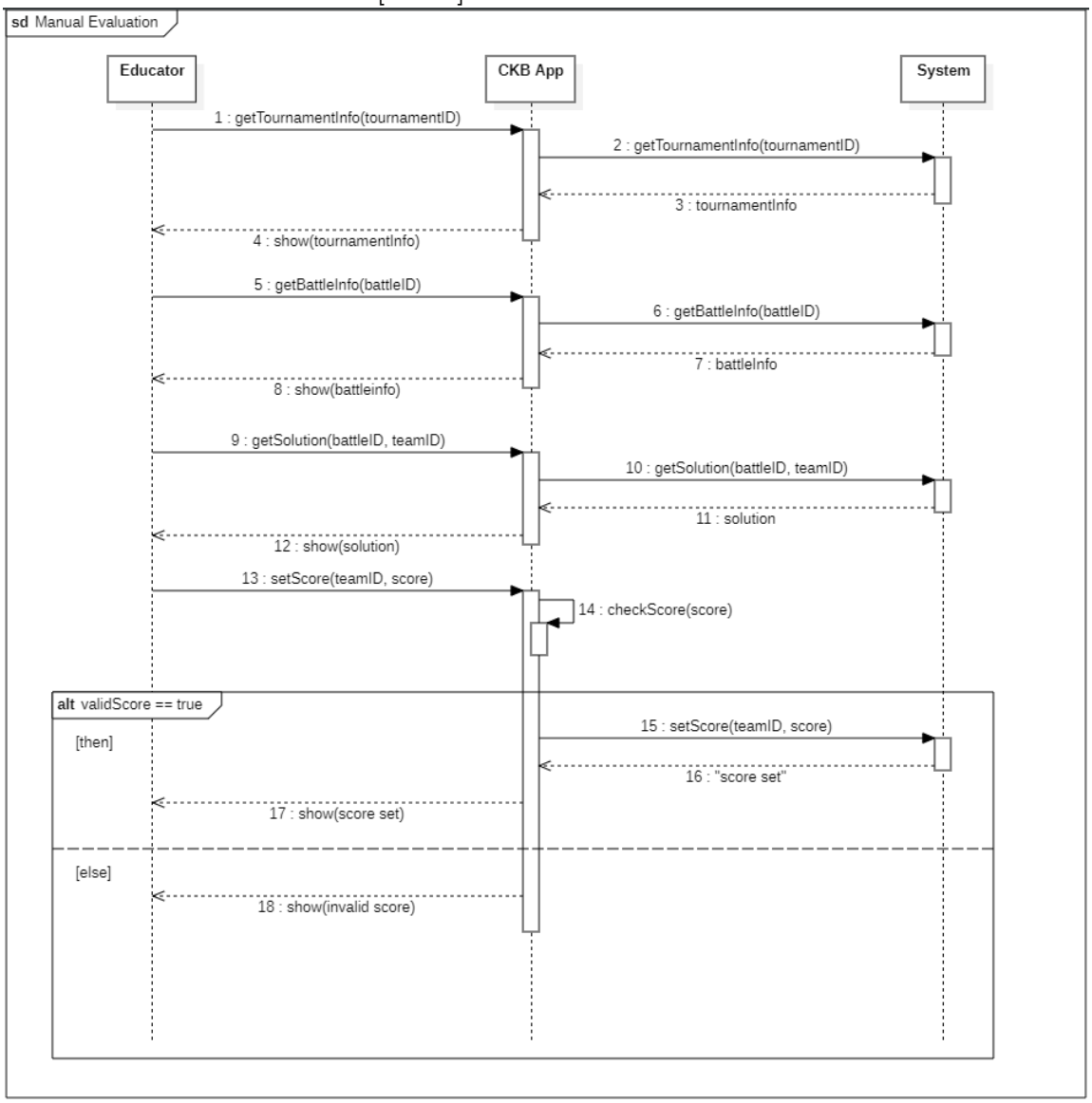
[UC9] Notification System



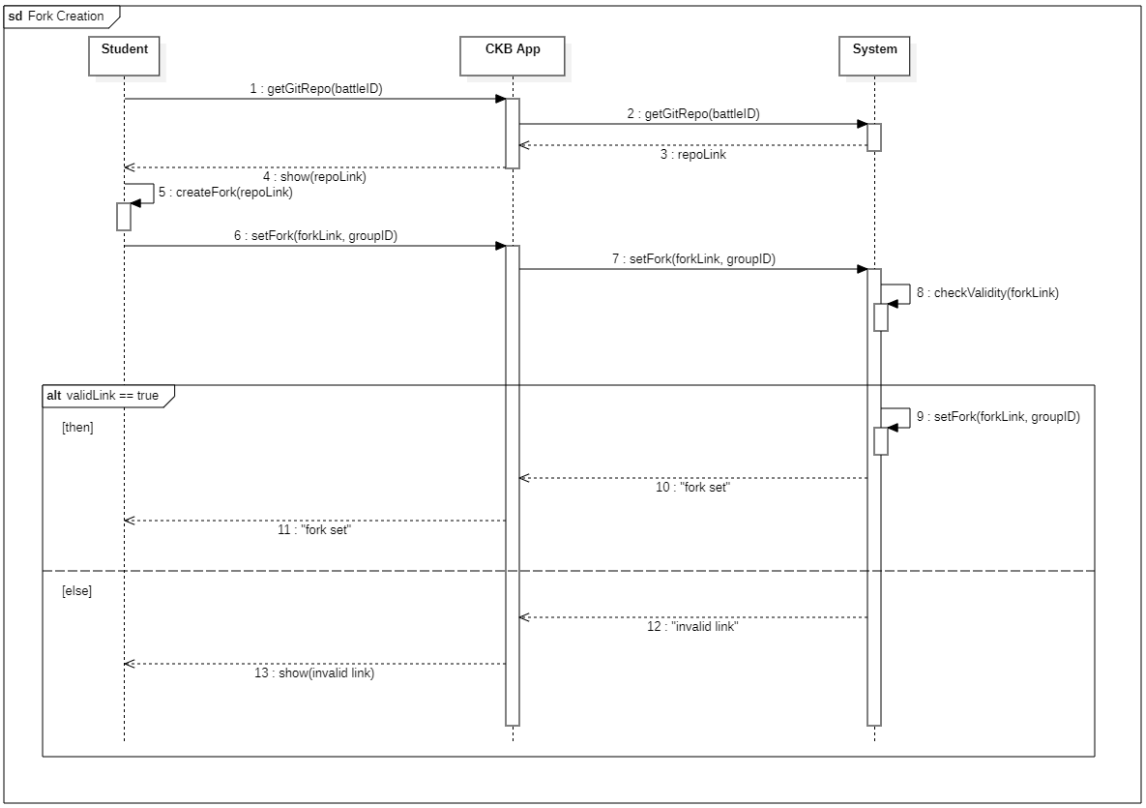
[UC10] Automated Evaluation



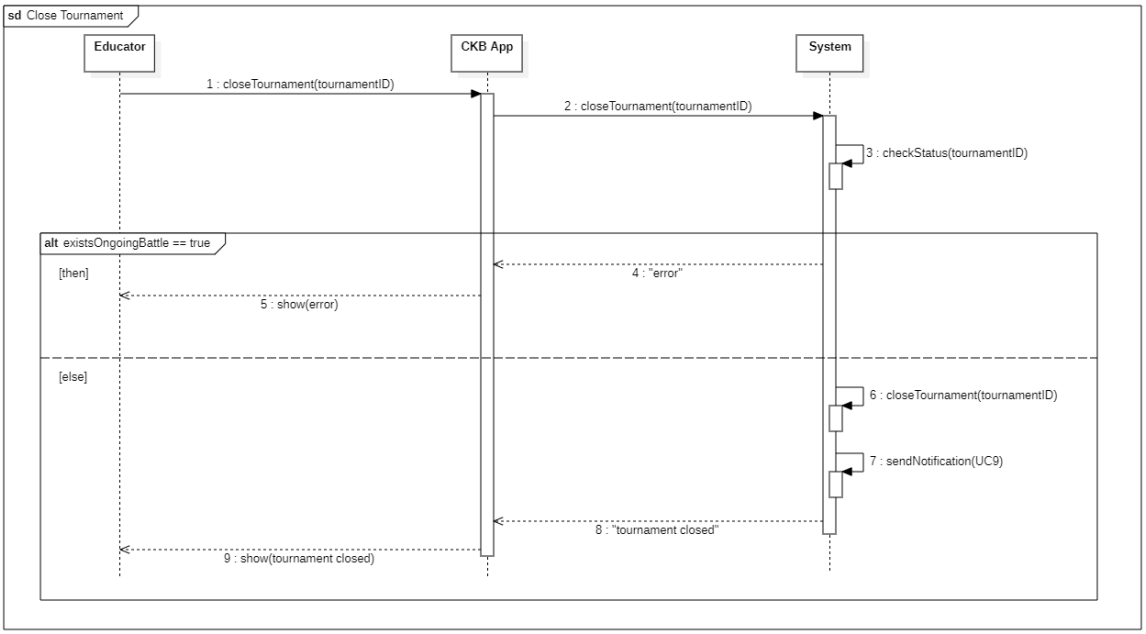
[UC11] Manual Evaluation



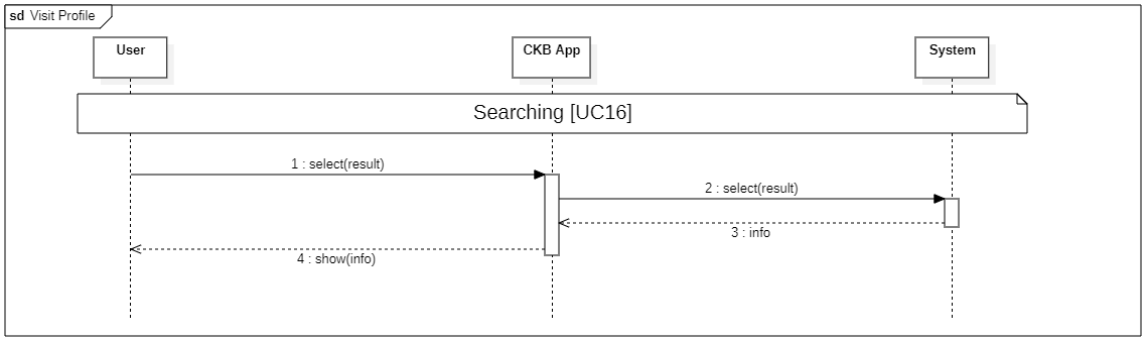
[UC12] Fork Creation



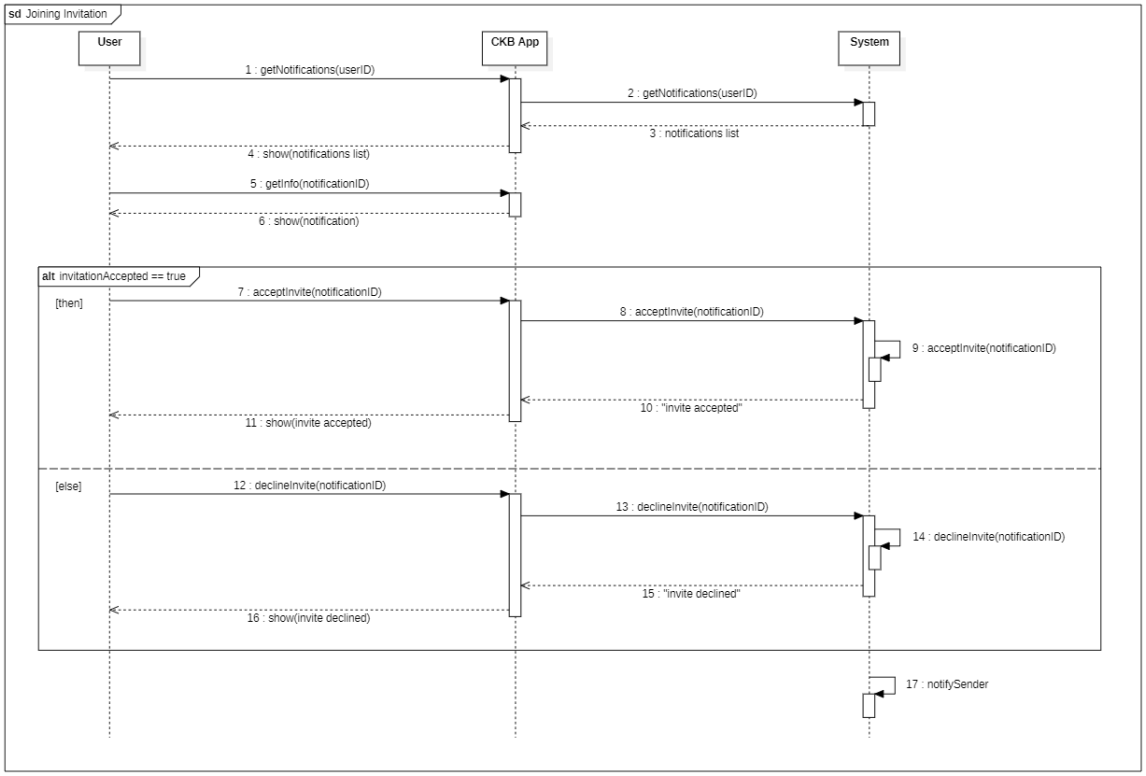
[UC13] Close Tournament



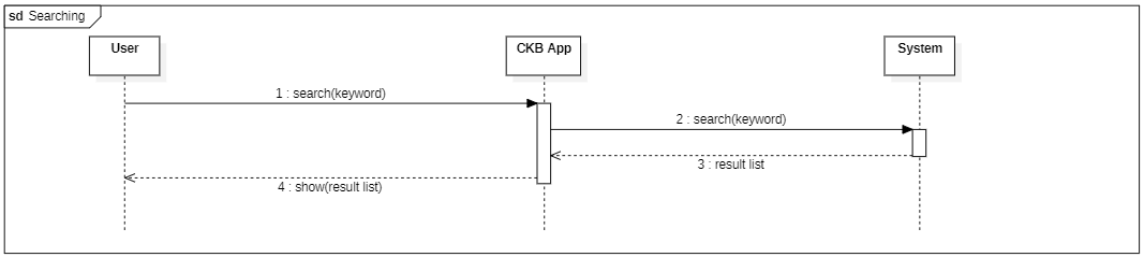
[UC14] Visit Profile



[UC15] Joining Invitation



[UC16] Searching



3.2.4 Requirements Mapping

[G1] EDUCATORS AND STUDENTS CAN SUBSCRIBE TO THE APPLICATION CREATING THEIR PERSONAL ACCOUNT	
[FR1] The system allows Users (Students and Educators) to sign up. **[FR2]** The system allows Users (Students and Educators) to login.	**[D1]** User must have an internet connection. **[D4]** Connection between systems must be reliable.

[G2] EDUCATORS CREATE TOURNAMENTS IN WHICH STUDENTS CAN COMPETE	
[FR3] The system allows Educators to create tournaments. **[FR4]** The system allows Educators to set a deadline for subscribing to the tournament. **[FR5]** The System allows Educators to set a list of programming languages that will be used in the tournament. **[FR6]** The system allows Educators to invite other Educators as collaborators for a tournament **[FR7]** The system allows Educators to set the number of battles in a tournament. **[FR22]** The system allows Educators to see the list of the active tournaments they have created.	**[D1]** User must have an internet connection. **[D4]** Connection between systems must be reliable.

[G3] STUDENTS SUBSCRIBE TO THE TOURNAMENTS	
[FR18] The system allows Students to join a tournament. **[FR23]** The system allows Students to see the list of the active tournaments they have joined.	**[D1]** User must have an internet connection. **[D4]** Connection between systems must be reliable.

[G4] EDUCATORS CREATE BATTLES IN THE CONTEXT OF A TOURNAMENT, CONSISTING IN CODING CHALLENGES FOR STUDENTS	
[FR8] The system allows Educators to create battles. **[FR9]** The system allows Educators to set a deadline for the subscription to the battle. **[FR10]** The system allows Educators to set a deadline for the submission of the solutions to the battle. **[FR11]** The System allows educators to choose which of the languages of the tournament will be used in each battle. **[FR12]** The System allows Educators to upload the files with the text of the problem and test cases for the battle **[FR13]** The system allows Educators to set the maximum number of students per team in a battle. **[FR14]** The system allows Educators to choose whether and when the scores will be assigned in a totally automatic way or also through manual verification by themselves	**[D1]** User must have an internet connection. **[D4]** Connection between systems must be reliable. **[D5]** Test cases provided by educators must be correct. **[D6]** GitHub works as expected.

[G5] STUDENTS COMPETE, IN TEAMS OR ON THEIR OWN, IN MANY BATTLES, THE RESULTS OF WHICH WILL DETERMINE THE FINAL RANK OF THE TOURNAMENT	
[FR19] The system allows Students to invite other students to form teams for a battle **[FR20]** The system allows Students to join a battle. **[FR21]** The system allows Students to know about the provisional scores of all students, updated for every solution submitted, for the duration of a battle.	**[D1]** User must have an internet connection. **[D2]** User must have a GitHub account. **[D3]** User must know how to use GitHub in the proper way (push, pull, branch, etc.). **[D4]** Connection between systems must be reliable. **[D5]** Test cases provided by educators must be correct. **[D6]** GitHub works as expected. **[D7]** The static Analysis tool works reliably.

[G6] EDUCATORS CREATE BADGES AND DEFINE REQUIREMENTS TO ACHIEVE THEM	
[FR15] The system allows Educators to create badges for a specific tournament. **[FR16]** The system allows Educators to build the requirements for achieving a badge.	**[D1]** User must have an internet connection. **[D4]** Connection between systems must be reliable.

[G7] STUDENTS CAN REACH ACHIEVEMENTS THAT PERMIT THEM TO GAIN BADGES FOR THEIR ACCOUNT	
[FR17] The system allows Students to see what are the active badges in a tournament and what they have to do to achieve them.	**[D1]** User must have an internet connection. **[D4]** Connection between systems must be reliable.

[G8] USERS CAN VISIT THE ACCOUNT OF EACH OTHER AS PART OF A COMMUNITY, THEIR STATS AND ACHIEVED BADGES ARE PUBLIC.	
[FR24] The system allows Users to see the stats of any other account registered in the system. **[FR25]** The system allows Users to search for other accounts or tournaments on the search bar.	**[D1]** User must have an internet connection. **[D4]** Connection between systems must be reliable.

3.3 Performance Requirements

The CKB app is designed to ensure optimal performance, capable of serving a large number of users. Users can expect a swift system response, typically within a couple of seconds, assuming a stable internet connection. In instances of slower internet connections, response times may vary due to external factors beyond the system's control. The app is equipped to detect and handle such conditions to provide the best possible user experience.

3.4 Design Constraints

3.4.1 Standards compliance

Regarding data privacy, the KCB project is committed to compliance with the General Data Protection Regulation (GDPR). The GDPR is a comprehensive regulation within EU law designed to protect and ensure the privacy of individuals within the European Union (EU) and the European Economic Area (EEA) concerning the processing of personal data.
In addition to data privacy regulations, the system also adheres to international standards concerning the use and representation of date and time. This ensures consistency and compatibility in handling temporal information, aligning the system with global norms.

3.4.2 Hardware limitations

The following system requirements must be met for CKB to function properly:

- For the mobile app, an internet connection(mobile data or Wi-Fi) and an updated OS(iOS or Android) are required.
- For the web app, an internet connection(mobile data or Wi-Fi) and an updated web browser is required.

3.4.3 Any other constraint

Since the system is designed to allow users to participate in coding battles, the system should not be too simple nor too detailed, allowing users to easily navigate through the various sections of the app.

3.5 Software System Attributes

3.5.1 Reliability

The system should achieve an uptime of at least 99.9%, implement mechanisms for graceful degradation during system failures, and ensure consistent and reliable storage of user data and kata progress.

3.5.2 Availability

The system should maintain a 24/7 availability, with advanced communication for scheduled maintenance. Since this is not a critical system, a downtime period(for maintenance or disservice) is acceptable.

3.5.3 Security

Since it stores some personal information, the system should use secure channels for all user interactions, employing industry-standard encryption protocols and implementing robust user authentication and authorization mechanisms, furthermore, there should be conducted regular security audits and vulnerability assessments.

3.5.4 Maintainability

The system should adhere to industry coding standards and best practices for a maintainable codebase, maintain comprehensive documentation, including API references and system architecture, and support easy integration of new code katas with deployable updates and minimal downtime.

3.5.5 Portability

The application is required to be developed for two distinct platforms: iOS and Android devices. Additionally, the web application must be compatible with any operating system—such as Windows, Mac OS, Linux, and others—that supports a standard web browser.

4 Formal Analysis Using Alloy

All the signatures defined for the Alloy modelization are listed below.

4.1 Signatures

```
abstract sig User {}

//Only Students can gain badges
sig Student extends User {
    gained: set Badge
}

//Educators can create or collaborate in a tournament
sig Educator extends User {
    creates: set Tournament,
    collaboratesIn: set Tournament
}

//A Tournament has one creator who can optionally
//invite other Educators to collaborate, a list of available programming
//languages, and is structured in a set of battles
//The Tournament can start only when the number of students has excedeed the
//minParticipants value
//Each tournament has a list of associated badges which students can gain
//(rts= registration time slot)
sig Tournament {
        creator : one Educator,
        collaborators : set Educator,
    students: set Student,
    state: one State,
        languages: some Language,
        battles: set Battle,
    badges: set Badge,
    rts: one TimeSlot,
    minParticipants: one Int
}{minParticipants > 0}

//Each battle consist in a coding challenge in one or more possible programming
//languages, it's competed in teams, that have to find a solution to the exercise
//The Exercise is available for the students in the GitHub repository at repoLink
//and manual evaluation
//can be enabled to be performed at the end of the battle.
//maxTeamSize and minTeamSize are the constraints that Students have to respect
//when forming a team about its size.
//(rts= registration time slot, sts = (solution) submission time slot)
sig Battle {

        creator: one Educator,
    state: one State,
        languages: some Language,
        exercise: one Exercise,
    teams: some Team,
    repoLink: one RepoLink,
    rts: one TimeSlot,
    sts: one TimeSlot,
       manualEvaluation: one Int,
    maxTeamSize: one Int,
    minTeamSize: one Int
}{(manualEvaluation = 1 || manualEvaluation = 0) && minTeamSize >=1 && minTeamSize <= maxTeamSize && maxTeamSize >= 1}
```

```
//Each team is formed by one or more students and has a GitHub Fork when there
//is their solution which evaluated with a Score
sig Team {
    members: some Student,
        repo: lone Fork,
    score: lone Score
}

//The union of the files regarding the codekata: the text of the exercise and the
//TestCases
sig Exercise {
    text: one Text,
    testCases: some TestCase
}

//The fork that a Team does from the codekata repo, identified by his link
//and containing team's solution
sig Fork{
    link: one Link,
    solution: lone Solution
}

//Gamification badge
sig Badge{}

//Team's solution code
sig Solution{}

//Team's solution evaluation
//A part of the score can be given manually by the educator
sig Score{
    automatic: one Automatic,
    manual: lone Manual,
}{}

//A TimeSlot: used for representing deadlines
sig TimeSlot{
    startTime: one DateTime,
    endTime: one DateTime
} {startTime != endTime}

//Part of the score automatically assigned by Codescene
sig Automatic{}

//Part of the score manually assigned by the educator
sig Manual{}

sig DateTime{}
//Programming language
sig Language{}
```

```
//Link of the code kata repo
sig RepoLink{}

//Test Cases for the codekata
sig TestCase{}

//a link
sig Link{}

//Text of the codakata problem
sig Text{}

//Possible states of a tournament or battle
//Ongoing battle or tournament has began but not ended yet
//Evaluating can only be reached by a battle if manual evaluation is enabled
//Open represents the state when the subscription are opened, the tournament or
//battle hasn't began yet
//Closed represents the ended battle or tournament
abstract sig State{}
sig Opened extends State{}
sig Ongoing extends State{}
sig Evaluating extends State{}
sig Closed extends State{}
```

4.2 Facts

The following facts are stated in Alloy, representing the requirements that must be held for the domain to remain consistent with the real world.

```
//Useful for the constraint to avoid inconsistencies in timeslot, two timeslots
//are different if completely disjointed
fact TimeSlotEqualityDefinition{
    all t1, t2: TimeSlot|
    t1!=t2 implies (t1.startTime!=t2.startTime and t1.endTime!=t2.endTime
    and t1.startTime!=t2.endTime and t1.endTime!=t2.startTime)
}

//If an educator creates a tournament then the tournament is set as created by
//that educator
fact TournamentsCreatedBySameEducatorAreInEducatorList {
    all t: Tournament, e:Educator |
        (t.creator=e implies t in e.creates) and
    (e in t.collaborators implies t in e.collaboratesIn) and
    (t in e.collaboratesIn implies e in t.collaborators) and
    (t in e.creates implies t.creator=e)
}

//No inconsistencies in timeslots
fact DifferentTimeSlotsForTournamentAndBattles {
    all t: Tournament, b: Battle |
        b in t.battles implies (b.rts != t.rts and b.sts != t.rts and b.rts!=b.sts)
}


//tournament cannot have started if it hasn't reached the requested minimum number
//of participants
fact MinimumParticipantsConstraints {
    all t: Tournament |
        t.state = Ongoing implies #t.students >= t.minParticipants
}

//An Educator can't be a creator and a collaborator for the tournament at the same time
fact CreatorCantBeACollaborator{
    (all t: Tournament |
            no e: Educator | e in t.collaborators and e = t.creator) &&
    (all e: Educator |
        no t: Tournament | t in e.creates and t in e.collaboratesIn)
}
```

```
//One student can be part of only one team in a battle
fact NoDuplicateTeamMembersInBattle {
    all b: Battle, m: Student |
        (m in b.teams.members) implies (all t1, t2: Team | (t1 in b.teams and t2 in b.teams and t1 != t2) implies
}

//A student has to have subscribed to the tournament to take part to a team for a battle
fact AllTeamMembersMustBeInTheTournament {
    all t: Team, to: Tournament |
        all m: t.members |
            m in to.students
        and
        t in to.battles.teams
}

//each team solution is evaluated separately
fact DifferentScoresForDifferentTeams{
    all disj t1,t2:Team| t1.score!=t2.score
}


//teams of a battle have to respect the size limits for the battle
fact TeamSizeLimit {
    all t: Team, b: Battle|
        t in b.teams implies #t.members >= b.minTeamSize and #t.members <= b.maxTeamSize
}



//A Battle cannot request the solution in a language which is not enabled for the tournament
fact LanguageAllowedInBattlesMustBeAllowedInTheTournament {
    all t: Tournament, b: Battle |
        b in t.battles implies b.languages in t.languages
}

//a battle can only be created in the context of a tournament
fact EachBattleAssociatedWithOneTournament {
        all b: Battle |
            one t: Tournament | b in t.battles

    all t: Tournament |
        t.state = Ongoing implies #t.battles >= 1
}
```

```
//Educator can and has only to manually evaluate all the solution if the manual evaluation
//is enabled
fact ManualEvaluationRules {
    all b: Battle |
        (b.manualEvaluation = 0 implies
            all t: b.teams | t.score.manual = none
        ) and
        (b.manualEvaluation = 1 implies
            all t: b.teams |
                t.score != none implies t.score.manual != none
        )
}

//no team unlinked solution
fact EachSolutionBelongsToATeam {
    all f: Fork |
        one t: Team | f in t.repo
}

//two teams can't have the same repo
fact UniqueRepoLinks {
    all f1, f2: Fork |
        f1 != f2 implies f1.link != f2.link
}


//No unlinked badges
fact AllBadgesInsideTournaments{
    all b: Badge |
        one t: Tournament | b in t.badges
}

//No unlinked exercises
fact AllExercisesInsideBattle{
    all e: Exercise |
        one b: Battle | e in b.exercise
}
```

```
//a tournament can't host battles when it's opened (it has to be ongoing) and can't
//be closed without having hosted at least one battle
fact EachTournamentHasAtLeastOneBattle{
    all t: Tournament |
            t.state in Closed implies #t.battles >= 1 &&
            t.state in Opened implies #t.battles = 0
}

//a tournament cannot assume the "Evaluating" state
fact NoTournamentsInEvaluating{
    all t:Tournament| not t.state in Evaluating
}

//no unlinked state
fact StateExistence {
    all s: State | s in Battle.state or s in Tournament.state
}

//no unlinked scores
fact ScoreExistence {
    all s: Score | s in Team.score
}

//no unlinked repolinks
fact RepoLinkExistence{
    all r: RepoLink| r in Battle.repoLink
}

//no unlinked manual scores
fact ManualExistence{
    all m:Manual| m in Score.manual
}

//no unlinked automatic scores
fact ManualExistence{
    all m:Automatic| m in Score.automatic
}

//no unlinked DateTimes
fact DateTimeExistence{
    all d:DateTime| d in TimeSlot.startTime or d in TimeSlot.endTime
}

//no unlinked Languages
fact LanguageExistence{
    all l: Language | l in Tournament.languages or l in Battle.languages
}

//no unlinked Testcases
fact TestCaseExistance {
    all t: TestCase | t in Exercise.testCases
}
```

```
//no unlinked teams
fact EachTeamIsInABattle{
    all t: Team |
        one b: Battle | t in b.teams
}

//no unlinked solutions
fact EachSolutionInAFork{
    all s: Solution |
        one f: Fork | s in f.solution
}

//no unlinked links
fact EachLinkInAFork{
    all l: Link |
        one f: Fork | l in f.link
}

//if a tournament is closed all his battles have to be closed
fact BattleStateConstraint{
    all b: Battle, t: Tournament |
        b in t.battles && t.state = Closed implies b.state = Closed
}

//if a battle is opened teams cannot have forked the repo at repoLink
//if manualevaluation is not enabled the battle never reaches evaluating state
fact BattleStateConstraint {
    all b: Battle, t: b.teams |
        (b.state in Opened implies t.score = none and t.repo = none) and
        (b.state in Evaluating implies b.manualEvaluation = 1)
}

//if a team has submitted a solution it has to be evaluated
fact SolutionsAreEvaluated{
    all t:Team |
    t.repo.solution!=none implies t.score!=none
}

//If a student owns a badge of a tournament, the student must have joined the tournament
fact StudentOwnsBadgeInTournament {
    all s: Student, b: Badge |
        b in s.gained implies
            one t: Tournament | s in t.students and b in t.badges
}

//useful for avoiding score inconsistencies, scores are different if they are completely disjointed
fact ScoreInequalityDefinition{
    all disj s1,s2:Score | s1!=s2 implies (s1.manual!=s2.manual or s1.manual=none or s2.manual=none) and s1.automatic!=s2.automatic
}
```

```
    //no unlinked Texts
    fact Textexistence{
        all t:Text | t in Exercise.text
    }

//Predicate
pred validTournamentInstance[t: Tournament] {
    #{t:Tournament| #t.collaborators=1}=1
    #{b:Battle | b.manualEvaluation=1&&b.maxTeamSize=2&&b.minTeamSize=2&&b.state in Ongoing}=1
    #Team=2
    #Fork=2
    #Solution=2
    #Battle=1
    #Student=4
}


// Example usage:
run validTournamentInstance for 10
```

```
Executing "Run validTournamentInstance for 6"
   Solver=sat4j Bitwidth=4 MaxSeq=6 SkolemDepth=4 Symmetry=20 Mode=batch
   28836 vars. 1608 primary vars. 56928 clauses. 895ms.
   Instance found. Predicate is consistent. 447ms.
```

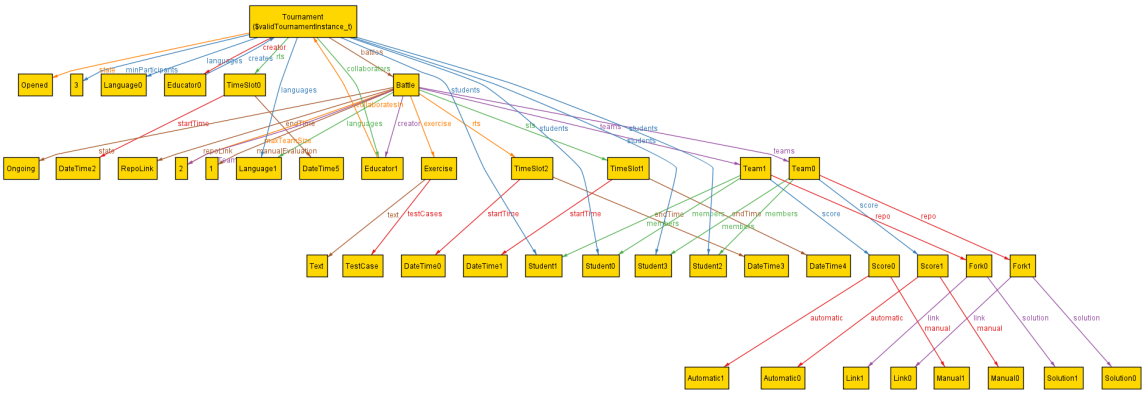


Figure 6: Metamodel showing the status of a tournament

5 Effort Spent

Armando Fiorini

Chapter	Hours Spent
1	3
2	10
3	21
4	18

Samuele Motta

Chapter	Hours Spent
1	4
2	6
3	20
4	14

Vajihe Gholami

Chapter	Hours Spent
1	4
2	11
3	22
4	10

6 References

- Diagrams made with StarUML
- Mockups made with: moqups.com
- Alloy models checked and ran with Alloy analyzer