POLITECNICO
MILANO 1863

CodeKataBattle - Annechini Alessandro, Filippi Nicole, Fiorentini Riccardo

Requirement Analysis and Specification Document

Deliverable:	RASD
Title:	Requirement Analysis and Specification Document
Authors:	Annechini Alessandro, Filippi Nicole, Fiorentini Riccardo
Version:	1.0
Date:	20-December-2023
Download page:	https://github.com/RiccardoFiorentini/AnnechiniFilippiFiorentini

Contents

1 Introduction

1.1 Purpose

The objective of this Requirements Analysis and Specification Document (RASD) is to delineate the foundational framework for the proposed CodeKataBattle platform (CKB). This comprehensive document will thoroughly detail the functional and non-functional requisites of CKB while outlining the system's constraints and boundaries. Primarily, this RASD serves as a directive for the development team tasked with implementing the specified requirements. Simultaneously, it acts as a contractual foundation for stakeholders and end-users, ensuring clarity and alignment between expectations and system capabilities. As such, precision in terminology is important, with explicit definitions provided to enhance mutual understanding.
The CKB platform introduces a revolutionary approach to enhancing students' software development proficiency through collaborative training in code kata exercises. Oriented towards creating a competitive yet supportive learning environment. Educators are enabled to construct engaging challenges for students from the platform. These challenges involve teams of students competing to demonstrate and refine their coding skills.

1.1.1 Goals

Here are the goals to achieve by implementing the CodeKataBattle (CKB) software:

G1	Educator can create tournaments and battles setting all the necessary parameters and badges
G2	Student can complete battles on their own or in groups by providing their code
G3	Educator that creates a tournament can allow other educators to add battles to the tournament
G4	Students and educators can see the list of ongoing and terminated tournaments and the corresponding rank
G5	Students and educators can see the current rank of every battle they are involved in
G6	Educator can go through the sources produced by a team and manually assign a score
G7	Educators and students can visualize all badges and every student's collected badges

Table 1: Goals of the system

1.2 Scope

In the fast-evolving landscape of today's technological era, the demand for adept software developers continues to rise. As the significance of collaborative learning and practical skill development becomes increasingly apparent, educational platforms are crucial in shaping the next generation of software engineers.
CodeKataBattle (CKB) emerges as an innovative solution, providing a dynamic platform to practice their software development skills collaboratively. In a society where real-world coding challenges and industry-relevant practices are fundamental, CKB bridges the gap

between theoretical knowledge and practical application. CKB not only proves instrumental in individual skill enhancement but also promotes a sense of healthy competition. This platform aligns with the contemporary need for hands-on, team-oriented learning experiences.
CKB serves as a vital tool in preparing students for the challenges of today's technology-driven society, where adaptability and collaborative problem-solving are key components of success.

1.2.1 Description of the system

CKB is a project aimed at helping people learn how to code but also to improve their software development skills, offering a dynamic and interactive environment for users to engage in collaborative learning. It supports two types of users:

- Students
- Educators

The platform facilitates the creation and management of code kata battles, where students can compete against each other to solve programming exercises in lots of different languages of choice, either by themselves or by joining a team.
Educators, creating code kata battles, challenge teams of students to compete against each other with diverse coding scenarios. Students can prove and improve not just their coding abilities but also their collaboration abilities by working in teams.
Each battle follows a structured timeline from registration to final submission, with GitHub integration for code versioning and automated assessment. The system also provides a test-first approach and automated evaluation of functional aspects, timeliness, and source code quality.
The platform extends beyond individual battles: educators can also create tournaments where different educators can add challenges. Students can subscribe to tournaments and solve the contained battles. The platform will then provide for each tournament a rank that reflects students' cumulative performance.
The scope includes features for educators to create, configure, and close tournaments, and for students to visualize the tournament's rank.
In addition, the system introduces badges that allow educators to define and award achievements based on specific criteria and students to improve their competitiveness, since gained badges are visible to all students and educators.

1.2.2 World phenomena

WP1	Student wants to learn or practice a programming language
WP2	Educator knows a programming language and wants to teach it
WP3	Student uses GitHub Action to work on a project
WP4	Educator writes battle's code and its tests' code
WP5	Student develops a challenge's solution

Table 2: World phenomena of the system

1.2.3 Shared phenomena

Shared phenomena are divided into:
Controlled by the world and observed by the machine:

SP1	Student creates an account in the system by using a specific form
SP2	Educator creates an account in the system by using a specific form
SP3	Student logs into the system by using a specific form
SP4	Educator logs into the system by using a specific form
SP5	Student subscribes to a tournament
SP6	Educator creates a tournament and then grants other educators to add other battles
SP7	Educator adds a challenge to an existing tournament
SP8	Student uses the platform to form a team for a battle
SP9	Student joins a battle on their own
SP10	GitHub Actions informs the platform when students push a new commit
SP11	Educator creates a challenge by adding textual description, software, and settings
SP12	Group delivers its solution to the platform
SP13	Educator goes through the sources produced by each team to assign their score
SP14	Students and educators involved in the battle visualize the current rank evolving during the battle
SP15	Students and educators visualize the list of ongoing tournaments and their rank
SP16	Educator closes a tournament
SP17	Educator defines badges when creating a tournament specifying title and rules
SP18	Students and educators visualize other students' profiles with their collected badges

Table 3: Shared phenomena of the system (world controlled)

Controlled by the machine and observed by the world:

SP19	Subscribed student receives GitHub repository's link where code kata is contained
SP20	Student is notified when the final battle rank becomes available
SP21	Subscribed student is notified when the final tournament rank becomes available

Table 4: Shared phenomena of the system (machine controlled)

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- Students: all students who are subscribed to the platform to learn and participate in tournaments. They need a unique email to subscribe
- Educators: all educators who are subscribed to the platform to create tournaments and battles. They need a unique email to subscribe
- Users: all students and educators who are subscribed to the platform
- Kata: an exercise in karate where you repeat a form many, many times, making little improvements in each. Code Kata is an attempt to bring this element of practice to software development
- Code kata: exercise created by an educator, composed of a description and software components, including test cases and building automation scripts
- Battle: a code kata battle
- Platform: the CodeKataBattle platform, i.e. a system that provides a set of tools, features, and services

1.3.2 Acronyms

- RASD: Requirement Analysis and Specification Document
- DD: Design Document
- CKB: CodeKataBattle
- GH: GitHub
- GHA: GitHub Action
- DBMS: Database Management System
- UI: User Interface

1.3.3 Abbreviations

- Gn: Goal number n
- Rn: Requirement number n
- Dn: Domain assumption number n
- WPn: World phenomena number n
- SPn: Shared phenomena number n

- DPn: Dependency number n
- UCn: Use case number n

1.4 Revision history

- Version 1.0 - 22/12/2023

1.5 Reference documents

This document is based on:

- The specification of the RASD and DD assignment given by professors Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at Politecnico di Milano, academic year 2023/2024
- Slides of Software Engineering 2 course on WeBeep

1.6 Document structure

The document contains six chapters that describe the whole system and its specifications, to explain it not just to customers and users but also to developers and programmers who will implement the requirements, to systems and requirements analysts, and also to project managers.
The first chapter is an introduction aimed to describe the system and its interaction with the users. The main key of this section is specifying goals, describing the system, and what can be done using this platform.
The second chapter contains a more specific description of the system: to show how CKB can be used, some different scenarios are presented, analyzed, and explained, clarifying the interaction between users and the system.
The third chapter contains more detailed specifications of the requirements. Here, all the interfaces are presented and explained. After this, there are sections dedicated to functional and performance requirements: use case diagrams and sequence activity diagrams are presented to describe functional and non-functional requirements. In the end, this chapter contains sections dedicated to design constraints and the system's attributes such as reliability, availability, security, maintainability, and portability.
The fourth chapter contains a formal analysis with Alloy, here assertions are shown to be meaningful by a series of simulations.
The fifth chapter is a report containing the effort spent by each group member.
The sixth and last chapter contains all the references used in the document.

2 Overall Description

2.1 Product perspective

2.1.1 Scenarios

1. **Creating a Tournament:**
   John, a university professor, wants to give his students the opportunity to improve their skills before the upcoming exam session. To achieve this, he logs into the CKB platform using his educator account. Once logged in, John initiates the process by creating a new tournament. During the tournament creation, John writes a comprehensive description of the game. Following the tournament's creation, the CKB platform notifies all subscribed students about the newly initiated competition.

2. **Creating a Code Kata Battle:**
   Sophia, to contribute to her colleague Liam's programming tournament, takes the initiative to create a new Code Kata Battle within the existing tournament. Liam has granted her the authorization to add Code Kata Battles to this specific tournament. Sophia begins by logging into the CKB platform (with her educator account) and navigating to the battles' section. Here, she uploads the challenge details, composing a descriptive introduction to the Code Kata and including the necessary software project files along with building automation scripts to facilitate a seamless development and evaluation process. Moving forward, Sophia configures the battle parameters, specifying the minimum and maximum number of students permitted per group. She sets a registration deadline, indicating when students should form or join teams for the battle, and a final submission deadline. Additionally, she customizes scoring configurations to tailor the evaluation process for this specific challenge. Upon successfully creating the Code Kata Battle, the CKB platform notifies all subscribed students within Liam's tournament. Students receive notifications about the newly introduced challenge, including essential details such as registration and submission deadlines.

3. **Student Participates to a Battle:**
   Olivia, willing to play a C++ coding challenge with her coding club friends, initiates her participation by logging into the CKB platform. Within the platform, she navigates to the desired tournament and selects the specific battle she wishes to join. Once Olivia joins the battle, she enters the subscription phase. During the subscription phase, Olivia invites her friends to form a team, ensuring to respect the specified minimum and maximum number of members for that particular battle. Her friends, upon receiving the invitation, log into the CKB platform and promptly accept her invitation to become members of the team. To submit their code, Olivia and her teammates fork the designated GitHub repository, and they set up an automated workflow using GitHub Actions to streamline the continuous integration process. The team can submit their code by committing changes to the main branch. The CKB platform is triggered on each push, pulling the latest sources and executing

the necessary tests to calculate and promptly update the team's score. This real-time scoring mechanism ensures that students and educators have instantaneous access to the latest results, fostering an environment of transparency and healthy competition.

4. **Educator Manually Evaluates Students' Solutions:**
   As the submission deadline for Jacob's battle approaches, the consolidation phase begins. Once the deadline expires, the CKB platform automatically evaluates essential components such as functional aspects, timeliness, and source code quality. This automated evaluation ensures a comprehensive and standardized assessment of each participating group's performance. In addition to the automated evaluation, Jacob, as the organizer, takes an extra step to guarantee the fairness of the competition. He personally reviews and evaluates the submissions from all participating groups, applying his expertise to acknowledge exceptional efforts and contributions. The final scores are finally assigned, marking the conclusion of the consolidation phase. At this point, all scores become visible to the users.

5. **Tournament Closure and Notifications:**
   Emily decides to conclude her tournament on the CKB platform. Following this decision, the CKB platform promptly took action, notifying all students who actively participated in the tournament about its closure. Simultaneously, the platform updates the personal tournament scores for each student involved. This ensures that participants are promptly informed of their final standings and achievements in the context of the concluded tournament.

6. **Gamification Badges:** Robert has recently created a tournament and wants to improve the gamification aspect of the competition by introducing badges. To integrate these badges into the CKB platform, Robert utilizes the platform's commands, specifying distinct titles and rules for each badge. Students earn badges based on their performance in battles. After each battle, the CKB platform automatically assesses and assigns badges to students who have fulfilled the specified requirements for each badge. The badges, complete with their titles and associated rules, are displayed for all participants.

2.1.2 Domain class diagram

In Figure 1, the Domain Class Diagram of the system is represented to illustrate the most relevant entities of the system along with the relations between them. There are two types of Users in the system: Students and Educators. Students can join Tournaments and Battles, either cooperating with other students by forming a Group or playing on their own (situation represented by a Group with one element). Groups can perform multiple submissions in the context of a Battle, all of which get instantaneously. Educators can create Tournaments and Battles, choosing whether to manually evaluate the final submissions or not. When creating a Battle, Educators must describe the Exercise that the students will try to solve, along with all the required Software Components for automated

building and testing. Finally, in the context of a Tournament, Badges can be created and assigned to deserving students once the Tournament has closed. Note that this illustration does not allow for a dynamic view of the system, while in the rest of the document, time-dependent aspects will be presented, starting from the state evolution of the main components of the system.

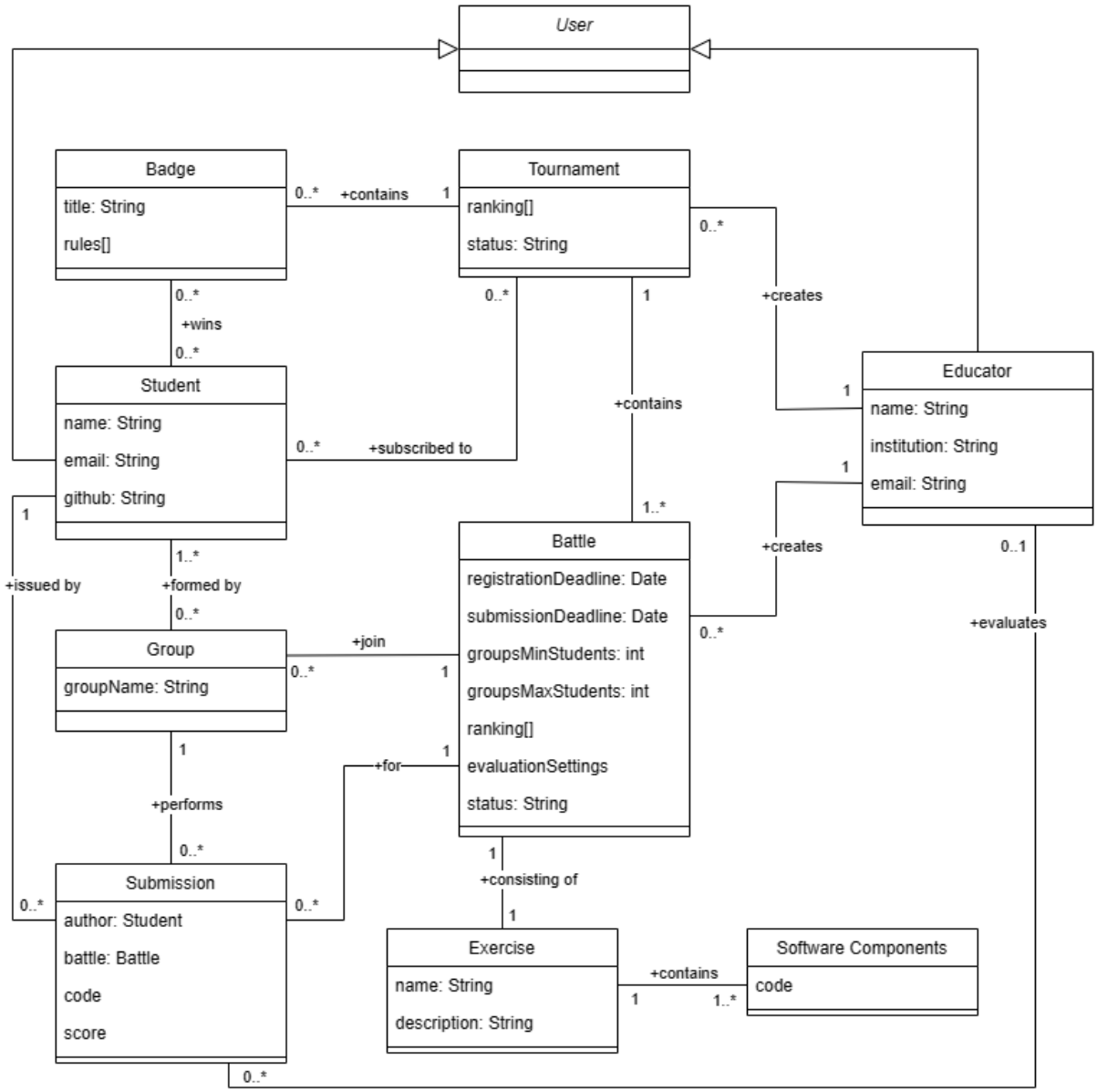

Figure 1: Domain class diagram

2.1.3 Statecharts

The CKB platform involves several key states and transitions to facilitate the seamless execution of code kata battles and tournaments. The primary states and transitions are described below:

1. **Tournament Lifecycle:**
   Upon the creation of a tournament, the Registration state commences, enabling students to submit their applications. Upon the conclusion of the registration phase, the tournament seamlessly transitions to the Started state, officially allowing users to subscribe to the battles. Upon reaching its conclusion, the educator who initiated the tournament holds the authority to close it, guiding it into the Closed state. This state transition mechanism ensures a structured progression throughout the tournament lifecycle.

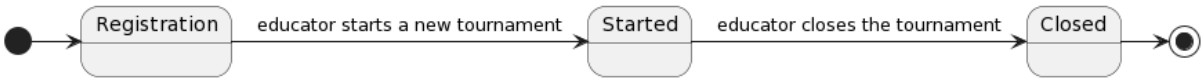

Figure 2: Tournament Lifecycle

2. **Code Kata Battle Lifecycle:**
   Upon the creation of a code kata battle, it assumes the Registration state, enabling students to participate by entering the battle and forming teams. Following the expiration of the registration deadline, the state seamlessly transitions to Started, marking the commencement of the game.

   After reaching its conclusion, the battle undergoes a state transition. If necessary, and at the educator's discretion, it can move to the Consolidation state for a manual review of the code. Alternatively, if manual review is not required, the battle transitions directly to the Closed state, revealing all the scores and outcomes.

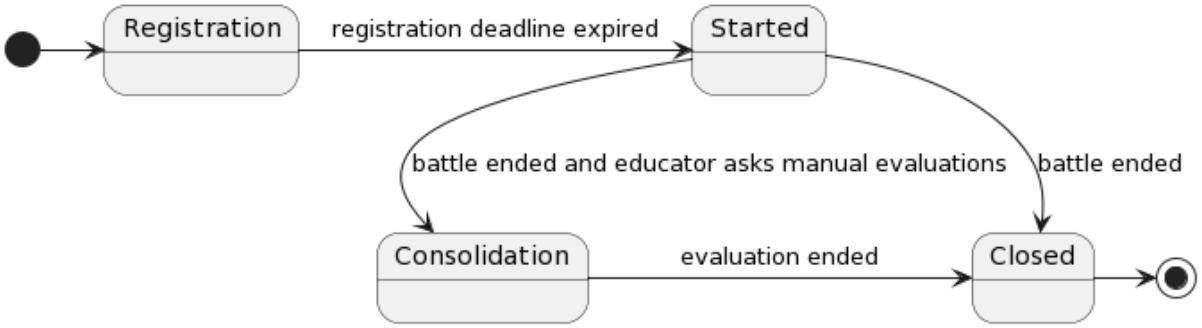

Figure 3: Code kata battle lifecycle

3. **Student Team Formation:**
   Upon entering a battle, a student can create a team by extending invitations to other students via email. The group initially resides in the Pending state until the

commencement of the battle. At this point, if the team aligns with the specified requirements of the battle (the required number of students), the team's state transitions to Accepted. Contrarily, if the team falls short of meeting the stipulated requirements, the state shifts to Rejected.

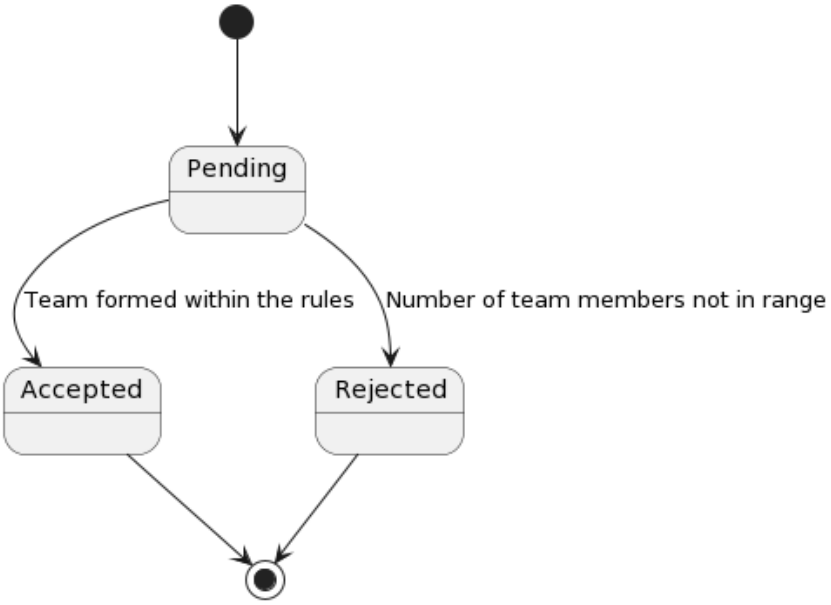

Figure 4: Students team formation

4. **GitHub Integration:**
   Following the creation of the repository by the educator, denoting the RepositoryCreated state, students are required to fork the repository. Subsequently, they must set up the automated workflow using GitHub Actions, marking the transition to the WorkflowEstablished state. As the battle concludes, the state progresses to BattleClosed, meaning that no further pushes to the repository are permitted.

Figure 5: GitHub Integration

5. **Battle Score Update:**
   At the onset of a battle and throughout its active phase (denoted by the Started state for the code kata battle), the battle score update operates in the RealTimeUpdate state. This signifies that with every push made by each student, the scores undergo real-time updates. Upon the expiration of the submission deadline, the state transitions to the ConsolidationStage. In this phase, the educator has the option to perform a manual evaluation of the students if desired. Following the manual evaluation or, if no manual evaluation is conducted, the state moves to the FinalScore, revealing the conclusive rank of the battle. This systematic flow ensures a seamless and structured progression throughout the different stages of the battle.

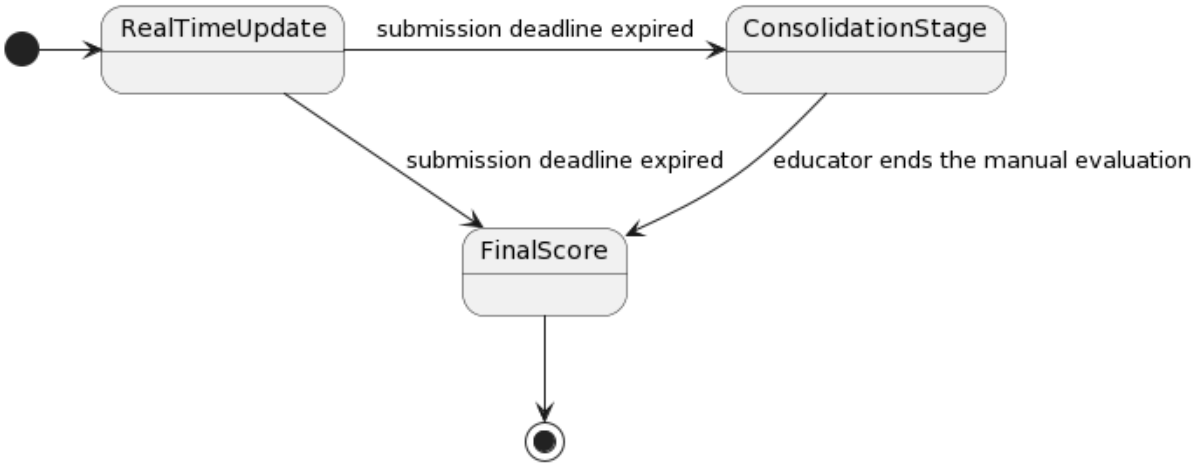

Figure 6: Battle score update

2.2 Product functions

1. **Tournament Management**
   Only educators possess the authority to initiate the creation of a tournament on the CKB platform. Once a tournament concludes, the creator has the option to officially close it (only if all the battles are concluded). After the closure, the CKB platform signals the end of the competition notifying all participants. Throughout the tournament's progression, the educator can either personally add battles or delegate this task to other educators.

2. **Code Kata Battle Management**
   The privilege of creating a Code Kata Battle within a tournament is reserved for educators. During this process, educators are asked to furnish essential details such as the kata description, the software project, build scripts, and scoring configurations, thereby shaping a well-defined coding challenge. In addition, educators have to establish timelines, including setting registration and submission deadlines for each battle. Furthermore, educators have the flexibility to determine the minimum and maximum number of individuals permitted within a team.

3. **Student Interaction**
   Students have the option to register either individually for specific battles or collaboratively form teams, adhering to the predetermined team size limits. It's important to note that the team formation process occurs prior to the registration deadline.

4. **GitHub Integration**
   Following the registration deadline, the platform generates a dedicated GitHub repository for each participating team, a repository that students are required to fork. Additionally, students must configure automated workflows utilizing GitHub Actions to facilitate continuous integration and evaluation. This setup ensures that with every commit made to the main branch, the CKB platform is triggered, up-

dating scores in real time. This workflow guarantees efficient collaboration and real-time feedback for participating teams.

5. **Scoring evaluation**
   The platform conducts automated evaluations containing functional aspects, timeliness, and source code quality. These evaluations result in real-time updates to scores, providing instant visibility for students and educators on the platform. Moreover, educators can perform manual evaluations and assign personal scores if necessary, contributing to a comprehensive and dynamic assessment process.

6. **Gamification Badges**
   Educators can craft gamification badges tailored to specific tournaments, with each badge automatically assigned to students based on predefined rules and their performance. The earned badges become prominently displayed in a user's profile, offering a visual representation of their achievements. Both students and educators can access and view these badges, fostering a sense of recognition and accomplishment within the CKB platform.

7. **Notification System**
   The CKB platform ensures communication by automatically notifying students and educators of new tournaments, battles, and tournament closures. These notifications are comprehensive, offering essential information such as important deadlines and updates. This proactive approach improves user engagement and keeps participants well-informed throughout their CodeKataBattle experience.

2.3 User characteristics

1. **Educators**
   An educator is a user with the power of managing the CKB platform. They have the authority to create, manage, and close tournaments. In addition, they can define code kata battles, and badges, and evaluate students' submissions.

2. **Students**
   Students are the primary users who participate in code kata battles and tournaments. They register for battles, form teams, and actively engage in coding exercises.

2.4 Assumptions, dependencies and constraints

2.4.1 Assumptions

The following assumptions are made to clarify the domain in which the software runs. Such assumptions are properties of CKB or conditions that the system takes for granted because they are out of its control. For that reason, they need to be verified to ensure the correct behavior of CKB.

D1	Users (educators and students) have basic proficiency in using web-based platforms and are familiar with version control systems, such as Git
D2	Educators have the necessary knowledge to create meaningful code kata battles, including defining test cases and scoring criteria
D3	Educators and students can connect to the Internet with their devices
D4	Notification must arrive to connected users in one minute
D5	Students are required to have a GitHub account for participation, and the platform assumes that students have the necessary permissions to fork repositories
D6	Users dispone of an e-mail address
D7	Users need a reliable internet connection to interact with the CKB platform

Table 5: Assumptions

2.4.2 Dependencies

In the previous sections, several dependencies from third-party software regarding core features of the system have been identified:

DP1	The CKB platform relies on GitHub for repository hosting and continuous integration processes
DP2	To allow static code analysis external API could be required

Table 6: Dependencies

2.4.3 Constraints

Since users will use the platform to compete and improve their coding skills the CKB platform must support a variety of programming languages. In order to give a good user experience the CKB platform is optimized for modern web browsers (Chrome, Firefox, Safari) to allow each person to play with his favorite browser. The platform must be designed to handle a scalable number of concurrent users during peak periods, such as registration deadlines. In addition, the platform must adhere to security best practices to protect user data and prevent unauthorized access, to prevent any malicious scripts from breaking into the system it exploits a sandboxing system to.

3 Specific Requirements

3.1 External interfaces

3.1.1 User interfaces

In this section, we report a mockup of the application's User Interface. For the two types of users, Students and Educators, two different views are needed to guarantee all and only the desired functionalities for the two categories. Since coding on a smartphone's keyboard is infeasible, computers are the main target of the application and the following User Interfaces are represented on a computer screen.
The Student UI allows the students to see the status of all the tournaments they are registered to, along with all the details regarding the battles within each tournament. Student must be able to join new battles by forming a team and see their ranking in the battle both before and after the closing deadline.
The Educator UI allows educators to create new tournaments and battles, invite other educators to post new battles in their tournaments and review the submissions of each time after the battle's closing deadline, if required.
Note that this is only a representation of the user interface of the system and does not represent a final product.

Student UI

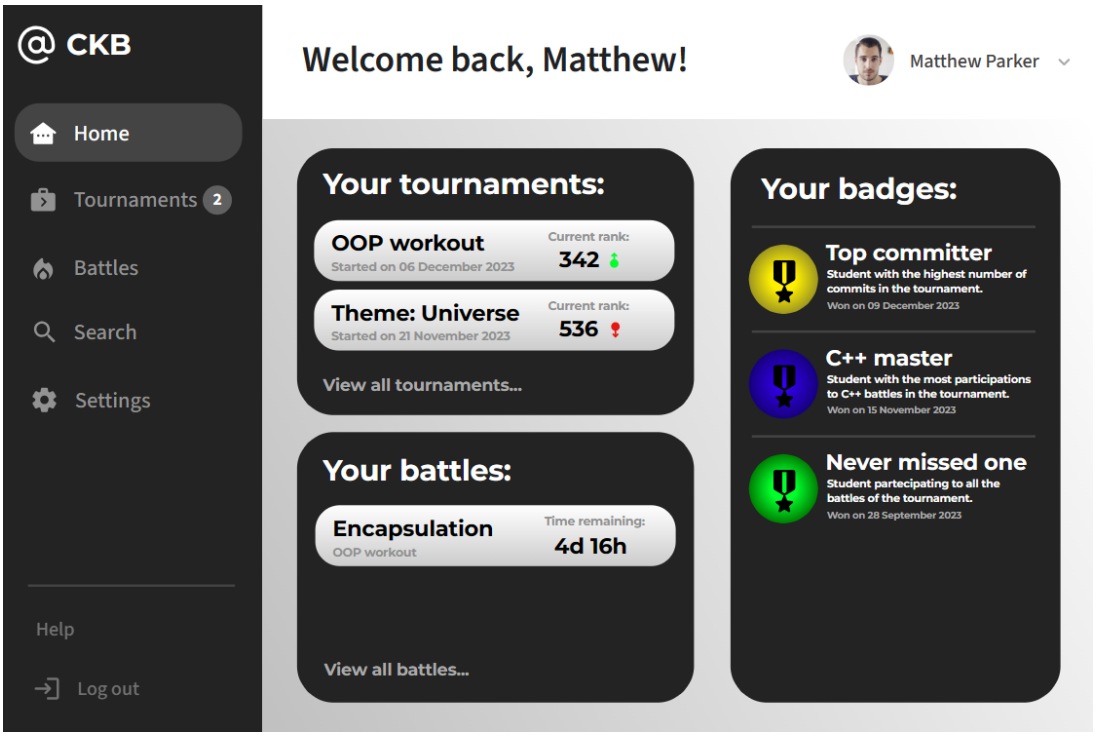

Figure 7: Student Home screen

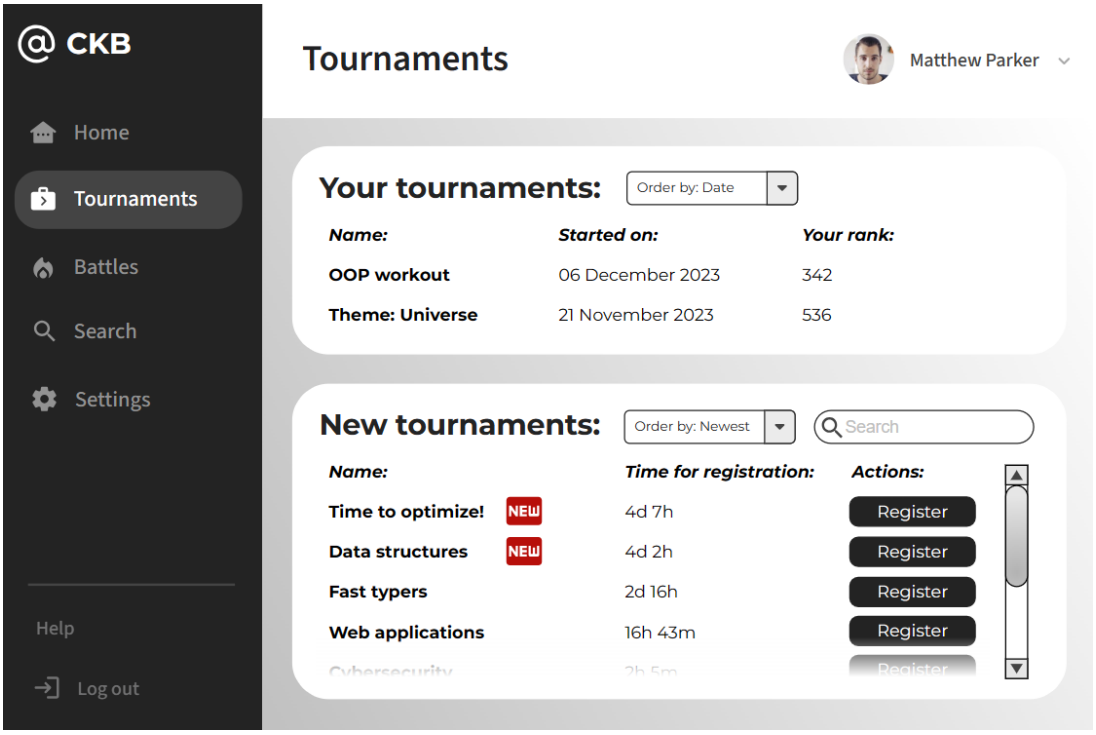

Figure 8: Student Tournaments screen

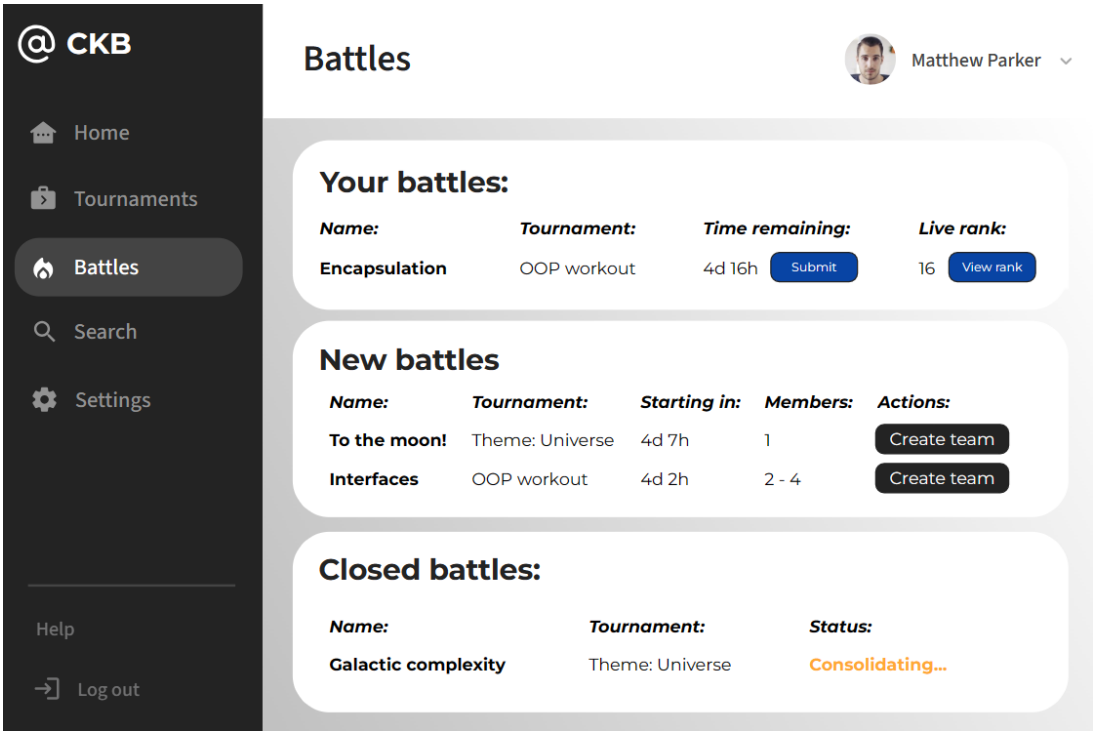

Figure 9: Student Battles screen

Educator UI

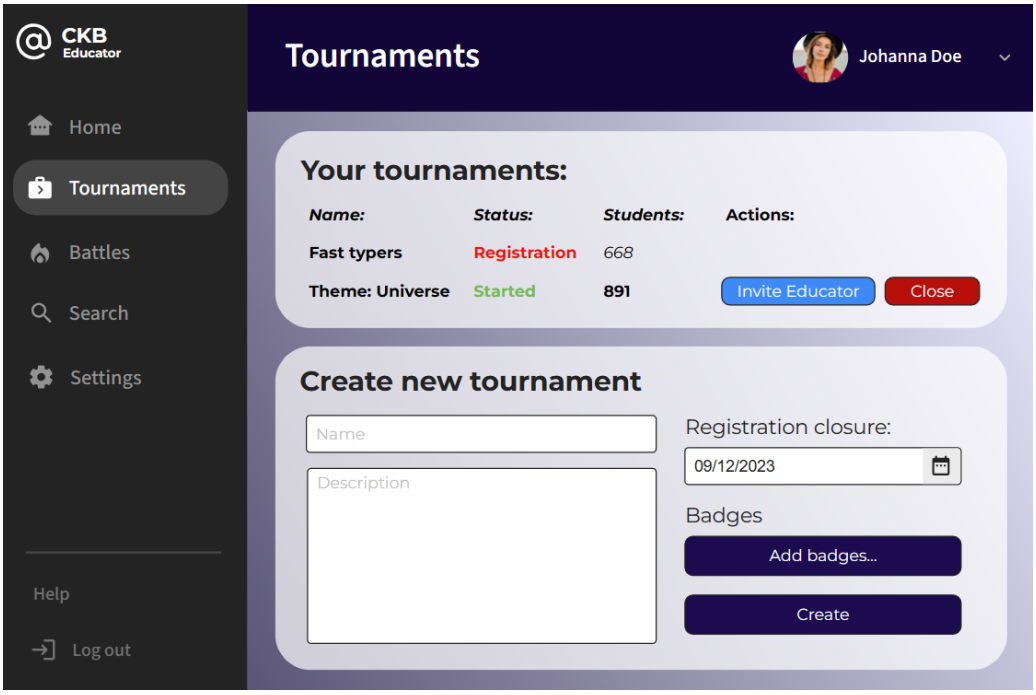

Figure 10: Educator Tournaments screen

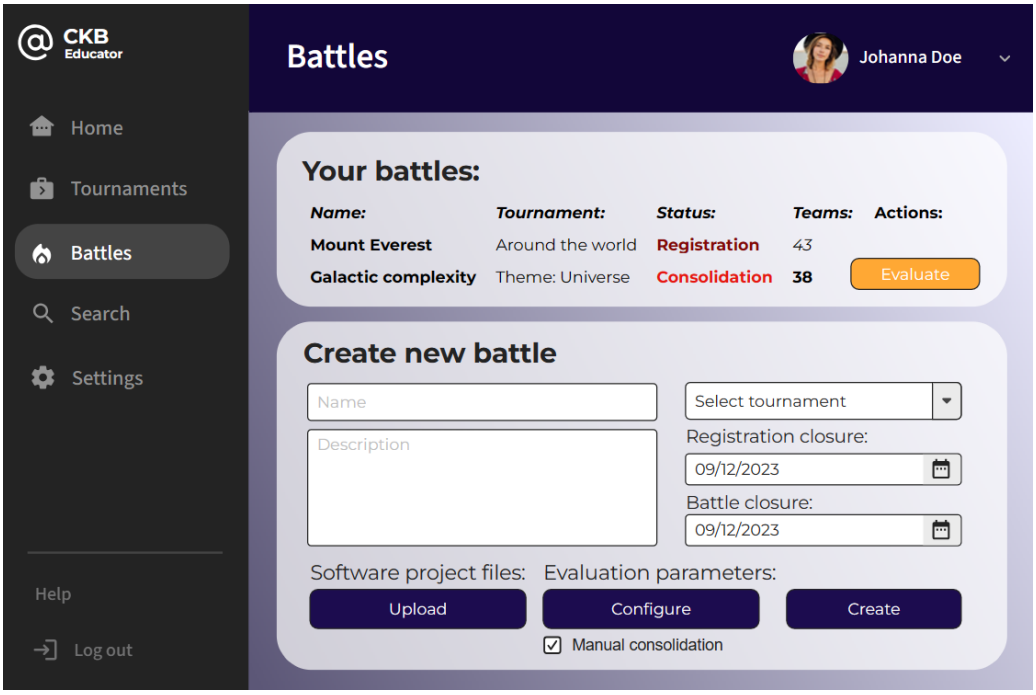

Figure 11: Educator Battle screen

Badges Integration

Another important interface of the system is the one regarding the specification of the badges, along with the respective rules that must be met to earn the badge, happening during the creation of a tournament. The possible variables that can be exploited to decide whether to assign a badge or not are plenty, and it is not possible to define a-priori all of them. To solve this problem, the system offers a fixed set of variables, along with the possibility of defining new variables and new rules.

The default variables are divided into two categories: student variables and global variables. The default rules consist of a comparison between the student's specific variable and the global variable. For example, assigning a badge to the student with the highest number of commits can be achieved in the following way:

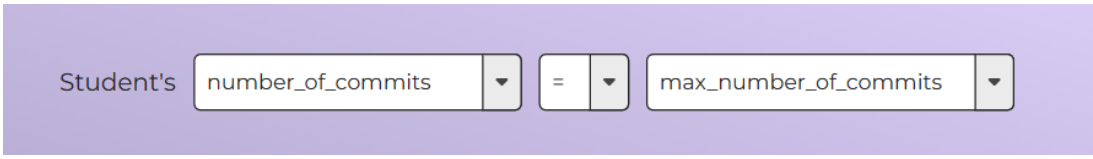

Figure 12: Student with maximum number of commits

Another example, assigning a badge to a student participating in all the battles in the tournament and reaching the first ten positions in the final tournament. Ranking can be assigned in the following way:

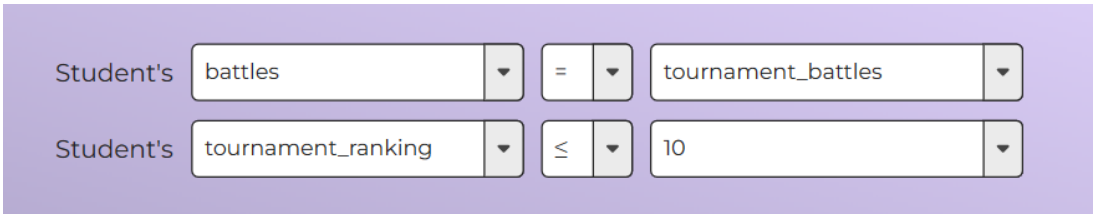

Figure 13: Student participating in all the battles and reaching the first ten positions in the tournament

The set of student variables, which will have a dedicated section explaining their meaning in the final application, is the following:

```
battles, //the number of battles in which the student participated
battles_won, //the number of battles in which the student
                //reached the first position in the final rank
max_battle_ranking, //the maximum rank reached at the end of a battle
max_commits, //maximum number of commits issued in a battle
max_tournament_ranking, //the maximum rank reached during the tournament
number_of_commits, //total number of commits issued by the student
```

```
number_of_teammates, //total number of distinct teammates that
                     //participated with the student to the battles
tournament_ranking, //the final rank of the student in the tournament
```

The set of global variables, which will have a dedicated section explaining their meaning in the final application, is the following:

```
tournament_battles, //the number of battles in the tournament
max_battles_won, //the maximum number of battles won
                 //by any student
max_number_of_commits, //maximum number of commits issued by any student
max_number_of_max_commits, //maximum number of commits issued
                           //by any student in a battle
```

In addition to this set of variables, the system offers a way to define new variables by writing a JavaScript function that returns a number. The variables offered to the educator are the following:

```
tournament
    .startTime //timestamp of the beginning of the tournament
    //timestamps are expressed as second elapsed since 00:00:00 of 1/1/1970 UTC
    .finishTime //timestamp of the beginning of the tournament
    .students[] //array of students
        .email //email of the student
        .finalRank //final rank of the student
        .ranks[] //rank of the student at the end of each battle
                 //regardless if the student participated or not
    .battles[] //array of battles
        .startTime //timestamp of the beginning of the battle
        .finishTime //timestamp of the ending of the battle
        .languages[] //programming languages of the battle
        .registeredStudents[] //array of student emails
        .teams[] //array of teams
            .members[] //emails of members of the team
            .rank //final position in the rank of the battle
            .commits[] //array of commits:
                       //the last entry is the final submission
                .timestamp //timestamp of the submission
                .language //programming language used
                .author //student email
                .text[] //lines of code
```

All the educator has to do is define a function that uses the previous data to return a quantity: if the function describes a student's variable, then the "student" object will be passed as a parameter.

As an example, here is the process to describe the number of participations of each student in battles solvable with the programming language C++:

```
function cpp_partecipations(student){
    let count = 0;
    for(let b of tournament.battles){
        if(b.registeredStudents.includes(student.email) &&
        b.languages.includes("C++")){
            count++;
        }
    }
    return count;
}
```

After this, we can define a new global variable as the maximum number of participations in a C++ battle by any student:

```
function max_cpp_partecipations(){
    let max = 0;
    for(let s of tournament.students){
        let curr = cpp_partecipations(s);
        if(max < curr)
            max = curr;
    }
    return max;
}
```

Once the two variables have been defined, the rule indicating the student with the most participations to C++ battles can be expressed as above:

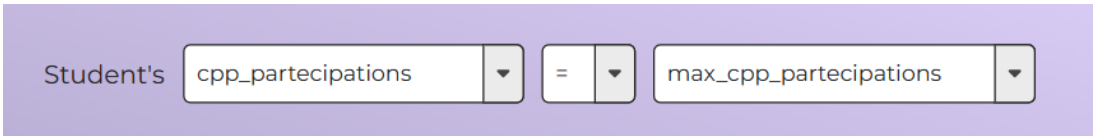

Figure 14: Student with the most participations to C++ battles

We note that the scripts proposed by the educators will be executed in the same controlled environment as the student submissions, and badges with incorrect descriptions will be discarded.

To add yet another layer of expressiveness, the system allows educators to define a rule function that directly returns if a student is eligible for a certain badge or not. To make a final example, here is the rule for selecting the students that issued two commits in less than a minute:

```
function two_commits_one_minute(student){
   for(let b of tournament.battles){
      for(let t of b.teams){
         if(t.members.includes(student.email)){
            for(let i=0; i<t.commits.length-1; i++){
               for(let j=i+1; j<t.commits.length &&
               t.commits[j].timestamp - t.commits[i].timestamp < 60; j++){
                  if(t.commits[i].author == student.email &&
                  t.commits[j].author == student.email){
                     return true;
                  }
               }
            }
         }
      }
   }
   return false;
}
```

These are all the options that an educator has to describe the rules for assigning badges to students. The system offers both a simple, intuitive way to assign badges to students based on simple parameters and a more complex, expressive way of defining detailed rules. The scripts used for defining rules can be saved to be reused and shared among educators and many of them are offered as templates to make the draft of complex rules more intuitive, along with removing the strict need for educators to eventually learn JavaScript in depth just for this purpose.

3.1.2 Hardware interfaces

As the system functions as an online service accessible through standard web browsers, specific hardware interfaces are unnecessary. Users can interact with the system using any computer with good internet connectivity, eliminating the need for specialized hardware requirements.

3.1.3 Software interfaces

In order to provide all the desired services and functionalities, the system requires different software interfaces. The most important ones are the following:

- GitHub Action: the system relies on GitHub Actions to automate the forwarding of the student commits to the system. GitHub Actions provides the required framework for continuous integration.
- Sandboxing: the system requires a robust sandboxing framework to execute external code in a secure and controlled environment. The primary purpose of this

framework is to mitigate potential security risks associated with running untrusted code, ensuring the integrity and safety of the system. This framework is also needed to measure memory usage and execution time of the submitted scripts, in order to assess the quality of the submissions in case of an automated evaluation.

- Email Provider: the system requires an e-mail service provider to send confirmation e-mails to the users.

3.2 Functional Requirements

R1	The system must allow users to register on the platform
R2	The system must allow users to authenticate themselves and login securely
R3	The system must allow educators to create new tournaments
R4	Tournament creation must include specifying a description of the tournament
R5	The system must allow authorized educators to create code kata battles within a tournament
R6	Creation of CKB must include kata description, software project, build scripts, and scoring configurations
R7	The system must allow students to register for individual battles or form teams based on specified team size limits
R8	The system automatically creates GitHub repositories for battles
R9	Students must be able to fork the repository and set up automated workflows using GitHub Actions
R10	During the battle, the platform updates scores with every push made by students in real time
R11	The system must allow educators to evaluate manually submissions
R12	The system must allow authorized educators to add battles to tournaments
R13	The closure triggers updates to personal tournament scores for each student
R14	The system must allow educators to create badges associated with specific tournaments
R15	Badges are automatically assigned based on predefined rules and student performance
R16	The system automatically notifies students and educators about new tournaments, battles, and tournament closures
R17	Notifications must include deadlines and updates
R18	The system must allow the educator owning the tournament to authorize other educators to create CKB

Table 7: Functional requirements

3.2.1 Use cases diagrams

In this section, an exploration of diverse use cases is presented to identify, elucidate, and organize the system's requirements. Each use case represents a series of interactions be-

tween users and the system, for them to achieve specific objectives. Actors involved in these scenarios, their actions, and reactions are here detailed, with potential exceptions leading to unsuccessful outcomes. To complement these descriptions, use case diagrams are employed, offering a visual representation of user-system interactions. These diagrams illustrate the user's actions, system functions executed by the application to fulfill user requests, and the given responses. Function names here provided are examples but maintain consistency across use cases to show the roles of the system's components. In addition, logic units have been defined, these communicate with the Database Management System (DBMS) to execute the required actions.

1. **Student Registration:** After choosing the creation of a student account, a simple interface allows the student to register by inserting their email address, which needs to be unique within other students' accounts. The format will be verified by the system and then an email containing a confirmation link that needs to be opened by the student to complete the registration is sent. They will be asked to also insert a unique username and a safe-enough password. After the registration process, a confirmation email is sent. The logical unit "UserManager" is a component of the system that communicates with DBMS to manage users' registrations and logins.

Name	Registration of a student
Actors	Student, Email provider
Entry Condition	Student has access to the Internet and has not registered as a student in the system yet
Event Flow	• Student opens the web application • Student clicks on the button "Sign Up as Student" • System shows the registration form • Student inserts username, email, and chosen password • System checks inserted data • System saves data and sends a confirmation email • Student clicks on the received link • System sends an email that confirms the subscription
Exit Conditions	Registration has been successful: student's data are correctly inserted and saved into the system's database
Exceptions	Registration has not been successful if: • Student inserts an already existing username or email • Student inserts an unsafe password

Table 8: UC1: Student Registration

2. **Educator Registration:** After choosing the creation of an educator account, a simple interface allows the educator to register by inserting their email address, which needs to be unique within other educators' accounts. The format will be verified by the system and then an email containing a confirmation link that needs to be opened by the educator to complete the registration is sent. They will be asked to also insert a unique username and a safe-enough password. After the registration process, a confirmation email is sent.

Name	Registration of an educator
Actors	Educator, Email provider
Entry Condition	Educator has access to the Internet and has not registered as an educator in the system yet
Event Flow	• Educator opens the web application • Educator clicks on the button "Sign Up as Educator" • System shows the registration form • Educator inserts username, email, and chosen password • System checks inserted data • System saves data and sends a confirmation email • Educator clicks on the received link • System sends an email that confirms the subscription
Exit Conditions	Registration has been successful: educator's data are correctly inserted and saved into the system's database
Exceptions	Registration has not been successful if: • Educator inserts an already existing username or email • Educator inserts an unsafe password

Table 9: UC2: Educator Registration

3. **Student Login:** Here we consider students' log in: they need to insert their credentials (email and password) and then, if they are valid, they are logged into the platform. Credentials' validity is checked by the logical unit "UserManager".

Name	Login of a student
Actors	Student
Entry Condition	Student has access to the Internet and has already registered as a student in the system
Event Flow	• Student opens the web application • Student clicks on the button "Sign In as Student" • System shows the login form • Student inserts username and password • System checks credentials • System redirects to correct Home screen
Exit Conditions	Student logs in correctly: inserts correct credentials and sees correct web page
Exceptions	Student doesn't log in correctly if: • Student inserts a nonexisting username • Student inserts the wrong password

Table 10: UC3: Student Login

4. **Educator Login:** Here we consider the login to an educator account. The educator who wants to log in must be registered in the system. They need to insert their credentials (email and password) and then, if they are valid, they are logged into the platform.

Name	Login of an educator
Actors	Educator
Entry Condition	Educator has access to the Internet and has already registered as an educator in the system
Event Flow	• Educator opens the web application • Educator clicks on the button "Sign In as Educator" • System shows the login form • Educator inserts username and password • System checks credentials • System redirects to correct Home screen
Exit Conditions	Educator logs in correctly: inserts correct credentials and sees correct web page
Exceptions	Educator doesn't log in correctly if: • Educator inserts a nonexisting username • Educator inserts a wrong password

Table 11: UC4: Educator Login

5. **Tournament Creation:** This use case considers the educator's creation of a new tournament on the platform. Of course, to do this the educator needs to be logged into the platform. The educator inserts the name, description, and registration deadline. The "TournamentManager" logic unit is responsible for inserting the new tournament and its information into the database and communicating with the DBMS. All subscribed students are then notified about the newly inserted tournament.

Name	Creation of a tournament
Actors	Educator, Students
Entry Condition	Educator has already logged in
Event Flow	• Educator opens Tournament screen • Educator writes name and description of the tournament • Educator selects registration deadline • Educator can add badges • Educator clicks on the "Create" button • System inserts new tournament's data into DB • System shows the new tournament in the students' Tournaments screen • System notifies subscribed students
Exit Conditions	Tournament has been correctly created and inserted into DB, it's also visible to all students subscribed to the platform
Exceptions	Tournament has not been created if: • Tournament with an inserted name already exists • Registration deadline is not valid

Table 12: UC5: Tournament Creation

6. **Add Battle Permission Granting:** After creating a tournament, the educator can grant other educators the possibility to add their battle to that tournament. The educator needs to be still logged into the platform and the other educators must exist. To do this, the educator uses a specific form where they can search for the educator they want to give permission to, inserting their email or their username. The "TournamentManager" will then insert this information into the database, communicating with the DBMS.

Name	Granting the permission to add a battle to a tournament
Actors	Educators
Entry Condition	Educator has already logged in and has created at least one tournament
Event Flow	• Educator opens Tournament screen • Educator clicks on the "Invite Educator" button for the correct tournament • System shows a specific form • Educator searches for the other educator(s) they want to invite inserting their email or username • System inserts tournament's new data into DB
Exit Conditions	Other educator(s) have now the possibility to add their battle to the tournament
Exceptions	Permission is not given if: • Tournament is already closed or doesn't exist • Educator is not the tournament's creator • Inserted names or emails don't exist • Selected educator has been already invited

Table 13: UC6: Permission Granting

7. **Battle Creation:** An educator can create a battle and add it to a tournament on the platform. To do this the educator must be not just logged into the platform, but they must have permission to insert a battle in that tournament. The "TournamentManager" is responsible for checking if the educator can add their battle in that tournament on the DBMS. If they have permission, they can then use a form to upload the code kata, with a description and the software project, including test cases and build automation scripts, to set the minimum and maximum number of admitted students per group, the registration deadline, the final submission deadline and any additional configurations for scoring. All data are checked by the

system and then inserted into the DB by a logic unit named “BattleManager”, that communicates with the DBMS.

Name	Creation of a battle
Actors	Educator, Subscribed students
Entry Condition	Educator has already logged in and has the permission to create a battle in that tournament
Event Flow	• Educator opens Battle screen • Educator writes name and description of the battle • Educator selects a tournament between the available ones • Educator selects registration deadline and submission deadline • Educator clicks on the "Upload" button to upload the code kata, which includes test cases and build automation scripts • Educator clicks on the "Configure" button to set group parameters and any additional configurations for evaluating • Educator clicks on the "Create" button • System inserts battle’s new data into DB • System shows the new battle to Battles screen of students that are subscribed to that tournament • System sends a notification to subscribed students
Exit Conditions	The battle is successfully created if students can see it and join it
Exceptions	Battle is not created if: • In that tournament already exists a battle with the same name • The selected tournament is already closed • Educator has not the permission to add a battle in that tournament • Deadlines are inconsistent • Code kata doesn’t compile or tests are missing

Table 14: UC7: Battle Creation

8. **Tournament subscription:** A student can subscribe to a tournament. They must be logged into the platform and the tournament must be open. The "Tournament-Manager" inserts this information into the DBMS, so that after the subscription the student can be notified of all upcoming battles created within that tournament.

Name	Subscribing to a tournament
Actors	Student
Entry Condition	Student has already logged in
Event Flow	• Student opens Tournament screen • Student clicks on the "Register" button for the correct tournament • System inserts subscription information into DB • System puts the student into the notification list for that tournament's battle • System shows the tournament in the student's Tournament screen
Exit Conditions	Subscription went well when the student can now see the tournament they're subscribed to in his Home screen and can participate in its battles
Exceptions	Subscription fails if: • Tournament registration's deadline has expired • Student is already subscribed to that tournament

Table 15: UC8: Tournament subscription

9. **Battle Join:** A student can join a battle if and only if they are subscribed to the tournament it belongs to and logged in. The "Tournament Manager" is responsible for checking if the student is subscribed to the tournament. When joining a battle, the student can choose between using the platform to form a team with other students (by sending them an invitation) or doing it on their own. If the number of students in the formed team respects the set values then it's accepted and the students composing the team can participate in the competition. The "BattleManager" is the component that does this check. When the registration deadline expires, the platform creates a GitHub repository containing the code kata and then sends the link to all students who joined. At this point, students can start working on the project.

Name	Join a battle
Actors	Students
Entry Condition	Student has already logged in and is subscribed to the battle's tournament
Event Flow	• Student opens the Battle screen • Student clicks on the "Create team" button for the correct battle • Student can choose between inviting other subscribed students using the form shown by the system or joining it on their own • System sends any team invitation and elaborates responses (see UC10) • System checks if battle's team constraints are respected • System inserts team's information into DB • Team's students automatically join the battle • System creates a GitHub repository containing the code kata and sends the link to teams • System shows the battle in the student's Battle screen
Exit Conditions	Students can see the battle in their Battle screen and can start working on the project
Exceptions	Students can't join the battle if: • Team is not accepted • Battle registration's deadline has expired

Table 16: UC9: Battle Join

10. **Invitation accepting:** A student can receive the invitation to a team if and only if they are subscribed to the tournament it belongs to (this is checked when the invitation is sent by "TournamentManager"). The student can also choose to not accept the invitation.

Name	Accepting an invitation to a team
Actors	Student
Entry Condition	Student has received an invitation to a team
Event Flow	• Student logs in • System shows to the student the invitation • Student clicks on the "Accept" button
Exit Conditions	The invitation is accepted, the student joins the team that needs to be checked (see UC9)
Exceptions	The student does not join the team if: • Student is not subscribed to the tournament • Battle registration's deadline has expired • Student doesn't accept the invitation

Table 17: UC10: Invitation accepting

11. **Code Manual Evaluation:** An educator can choose to manually evaluate the code of teams participating in a battle. To do so they must be logged in and they must be the creator of the battle. This is checked by the "BattleManager" component. The manual evaluation can be done only after the submission deadline has expired, in the consolidation stage. The educator uses the platform to go through each team's sources.

Name	Evaluate manually the code of teams
Actors	Educator
Entry Condition	Educator has already logged in and has created at least one battle that is in the consolidation stage but not closed yet
Event Flow	• Educator opens the Battle screen • Educator clicks the button "Evaluate" for the correct battle • System shows each team's sources so that educators can go through them and evaluate them • Educator inserts all grades • System closes the battle
Exit Conditions	The manual evaluation went well if the battle is closed and every team has a grade given by the educator that created the battle
Exceptions	The manual evaluation has failed if: • Educator is not the creator of the battle • Tournament is already closed • Battle is already closed

Table 18: UC11: Code Manual Evaluation

12. **Tournament Closure:** An educator can close a tournament, but they must be not just logged into the platform, but they must be also the creator of that tournament and all the contained battles must be finished. The "TournamentManager" and the "BattleManager" are responsible for checking that. After the tournament is closed, the final tournament rank becomes available and the system notifies every participating student.

Name	Close a tournament
Actors	Educator, Subscribed students
Entry Condition	Educator has already logged in and has created at least one tournament that is not closed yet
Event Flow	• Educator opens Tournaments screen • Educator clicks the button "Close" for the correct tournament • System calculates final rank, registers it, and publishes it • System notifies all subscribed students
Exit Conditions	The tournament has been successfully closed, the final rank has been registered and published, and the subscribed users are correctly notified
Exceptions	The tournament is not successfully closed if: • Educator is not the creator of the tournament • Tournament is already closed • One or more battles in the tournament have not been finished or completely evaluated yet

Table 19: UC12: Tournament Closure

13. **Badges Integration:** An educator, while creating a tournament, can also define gamification badges associated with it. They use a form on the platform that permits the inserting of the title and one or more rules that need to be fulfilled to obtain the badge. Rules can be chosen between predefined ones or created from scratch in JavaScript language. Since they are part of the tournament's properties, the badges are inserted into the DB by the "TournamentManager" component.

Name	Create and add a badge
Actors	Educator
Entry Condition	Educator has already logged in and is creating a tournament
Event Flow	• Educator clicks the button "Add badge" • System opens the form to create a badge • Educator writes the title in the text box • Educator writes rule(s) selecting correct variables to complete the equation they want or, if not present, writes a JavaScript code from scratch • System saves the badge in DB
Exit Conditions	The new badge is correctly saved in the DB and is syntactically correct
Exceptions	The badge is not accepted if: • Exists another badge with the same title for that tournament • Exists another badge with the same rule-set for that tournament • JavaScript code is not syntactically correct • Rules are mutually exclusive • One or more rules are not achievable

Table 20: UC13: Badges Integration

14. **Badges Obtaining:** A student who is subscribed to a tournament can obtain the gamification badges of that tournament. They can win the badge by fulfilling all the corresponding rules. Each badge is assigned to one or more students at the end of the tournament. The set of badges corresponding to a tournament can be retrieved via the "TournamentManager" component. After obtaining a badge, it can be seen on the student's profile, to do so the collected badges are inserted into the DB by the "UserManager" component.

Name	Student obtains a badge
Actors	Student
Entry Condition	Educator closes the tournament and the student has fulfilled all corresponding rules of the badge
Event Flow	• System inserts the collection of the badge into the DB • System updates student's profile adding the new collected badge • If not already logged in, the student opens the web application and logs in • Student opens the Home screen and sees their updated badge collection
Exit Conditions	Student can see their new badge on their profile

Table 21: UC14: Badges Obtaining

Student Use Case Diagram

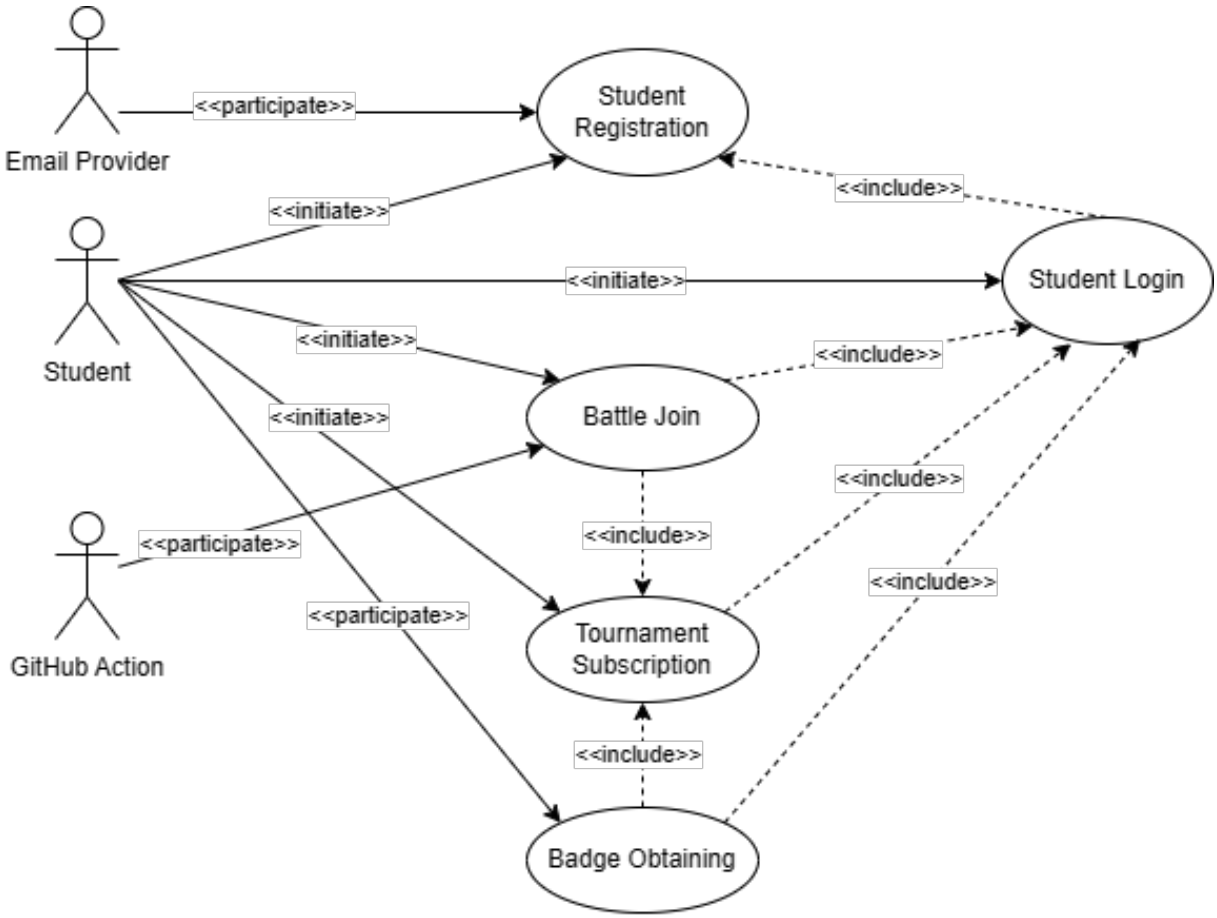

Figure 15: Student UC Diagram

Educator Use Case Diagram

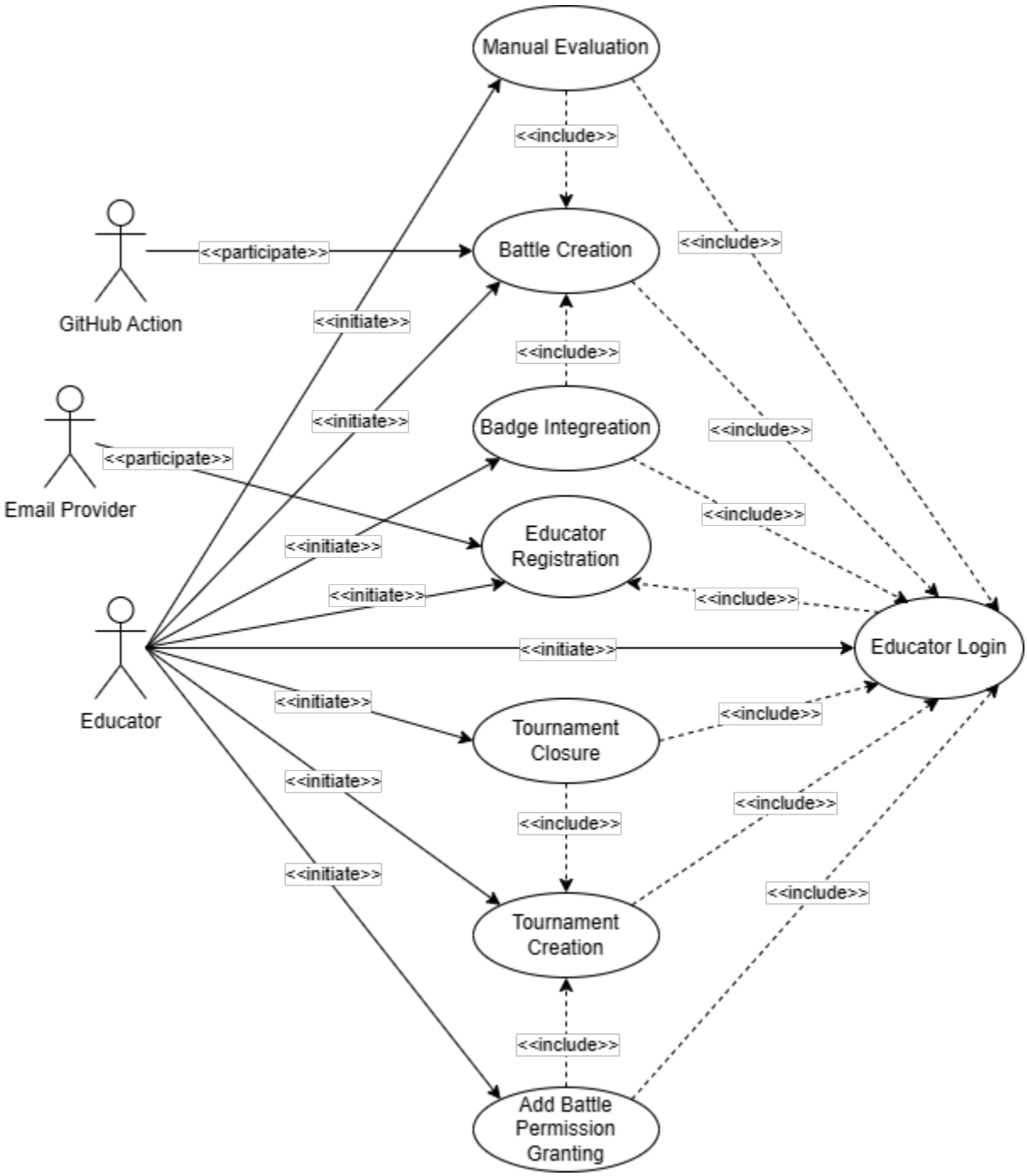

Figure 16: Educator UC Diagram

3.2.2 Sequence diagrams

[UC01] - Student Registration

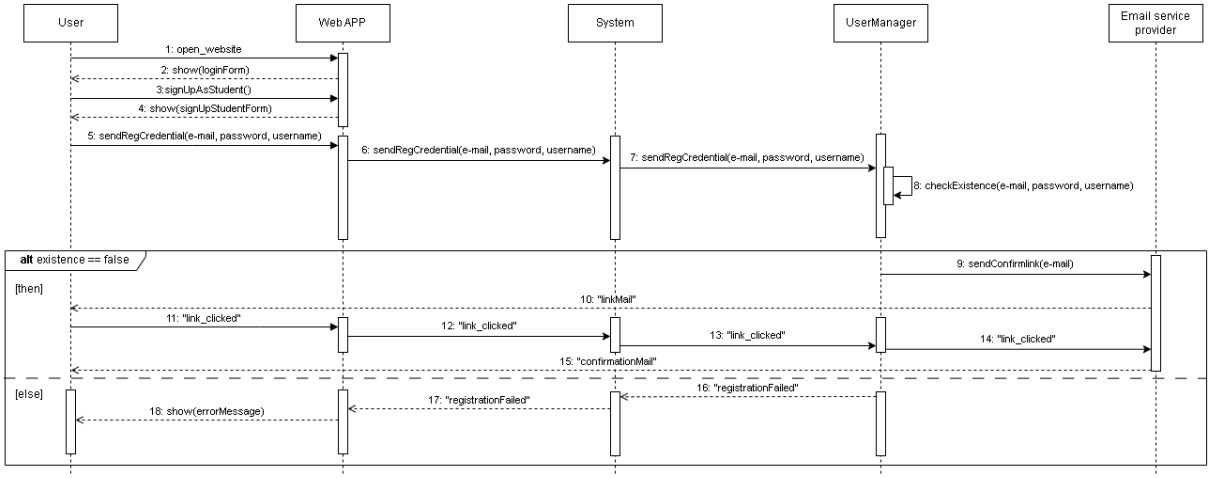

Figure 17: UC01 sequence diagram

[UC02] - Educator Registration

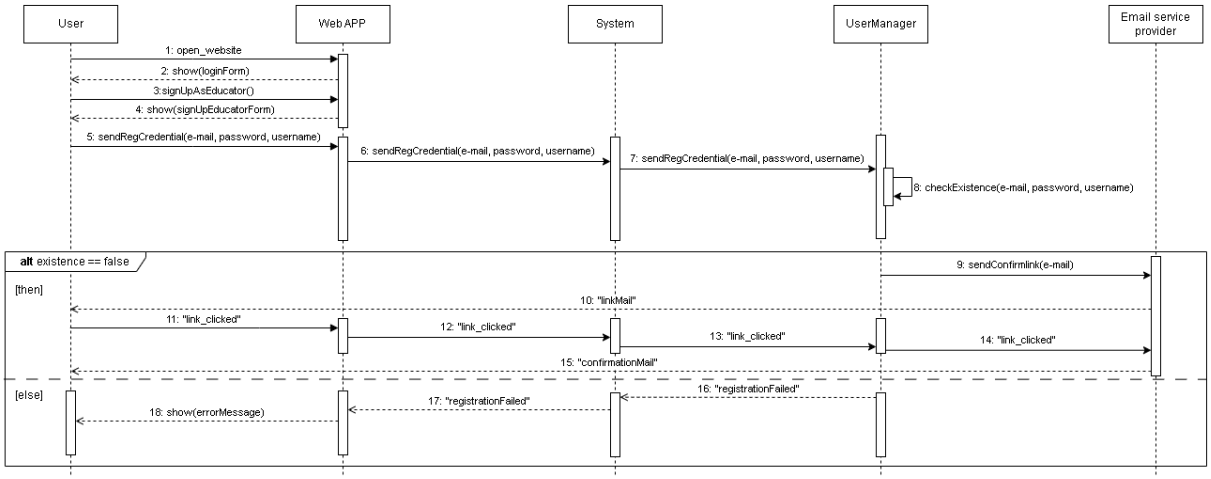

Figure 18: UC02 sequence diagram

[UC03] - Student login

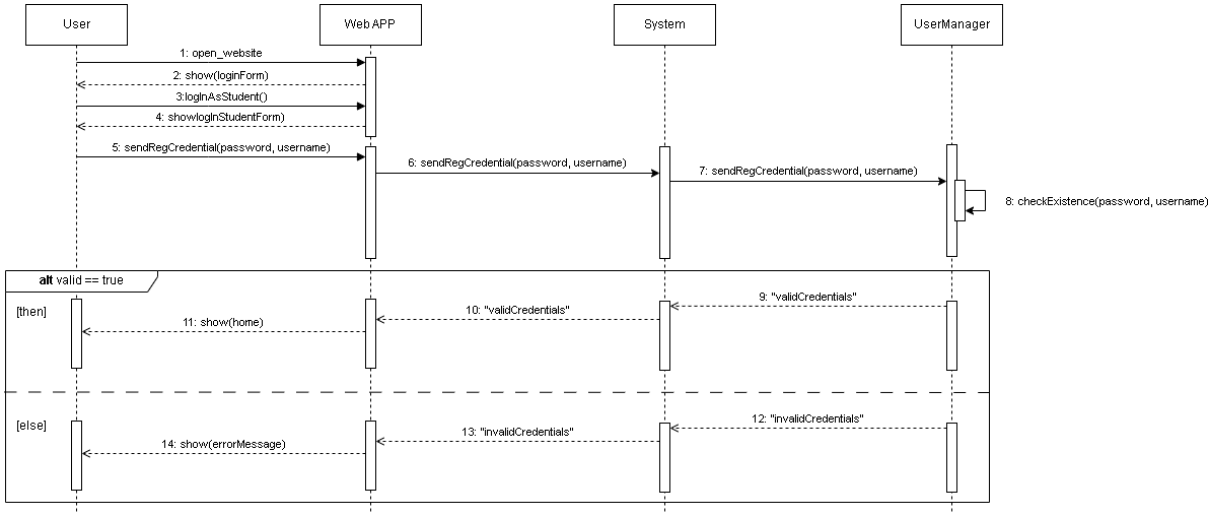

Figure 19: UC03 sequence diagram

[UC04] - Educator login

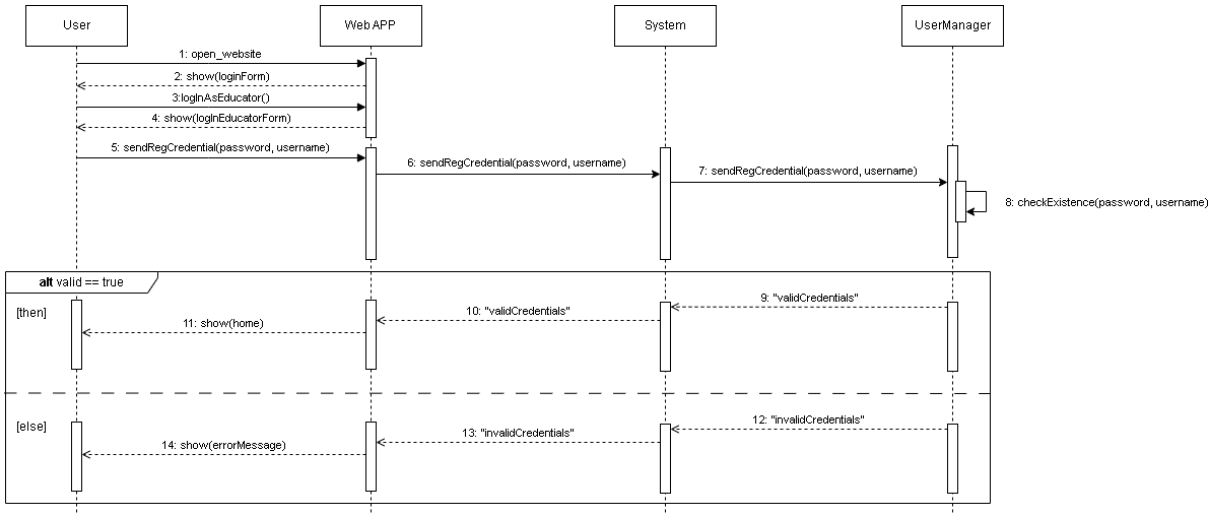

Figure 20: UC04 sequence diagram

[UC05] - Tournament Creation

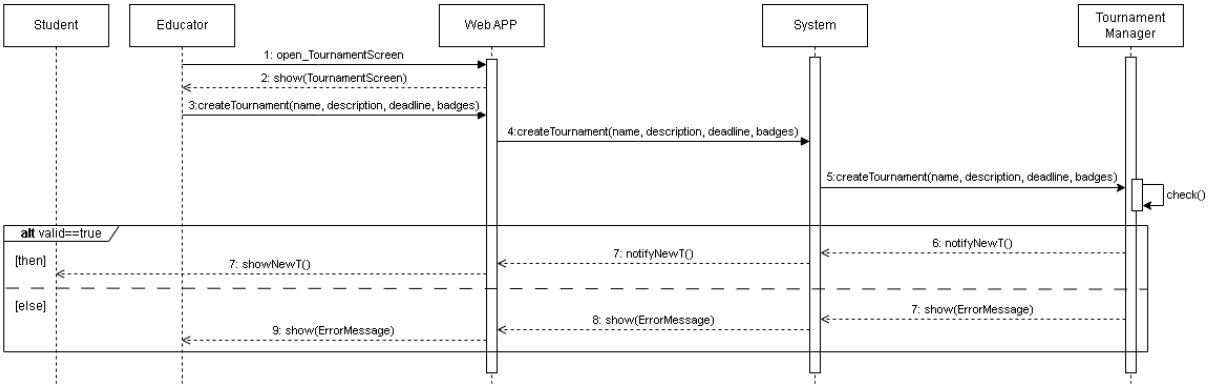

Figure 21: UC05 sequence diagram

[UC06] - Add Battle Permission Granting

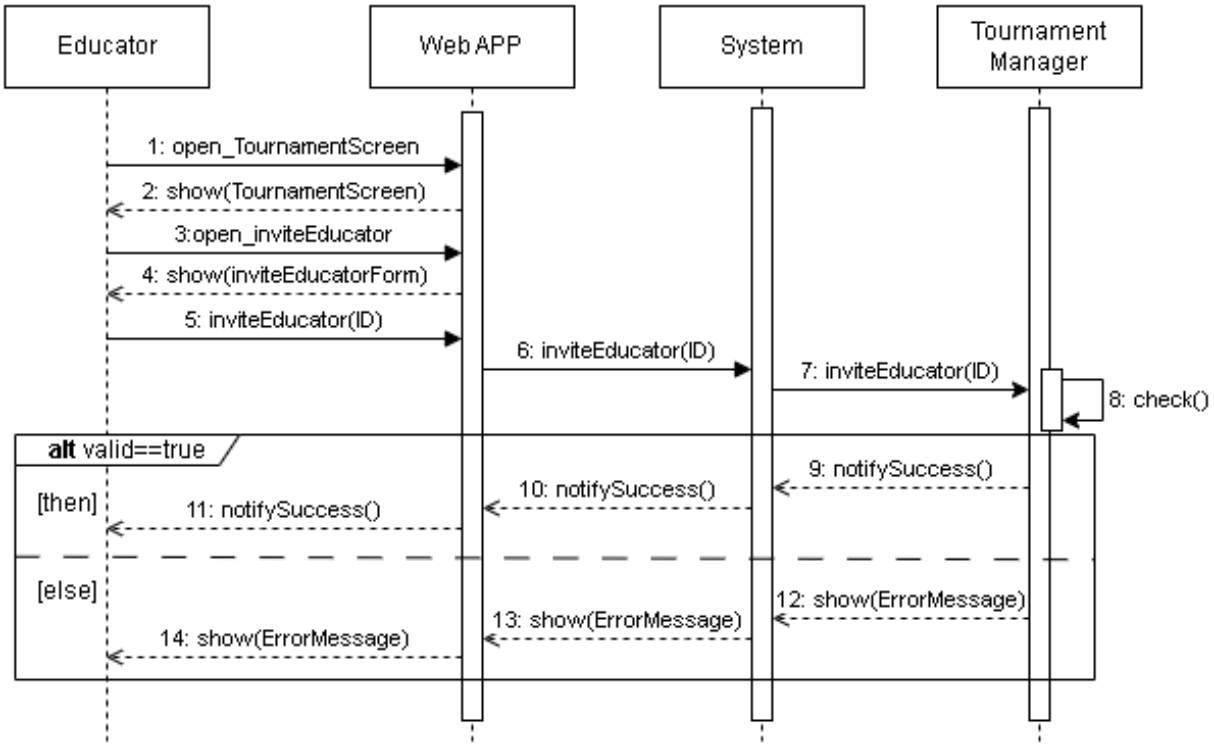

Figure 22: UC06 sequence diagram

[UC07] - Battle Creation

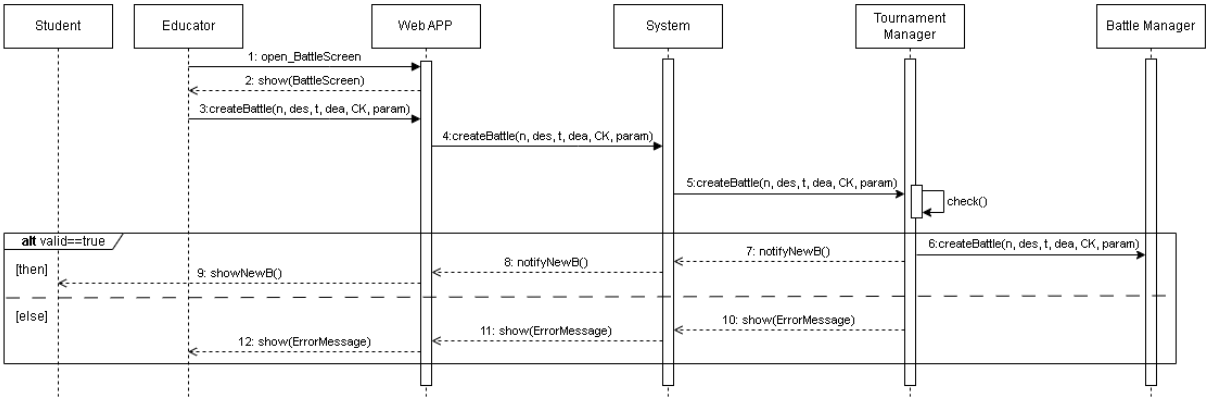

Figure 23: UC07 sequence diagram

[UC08] - Tournament subscription

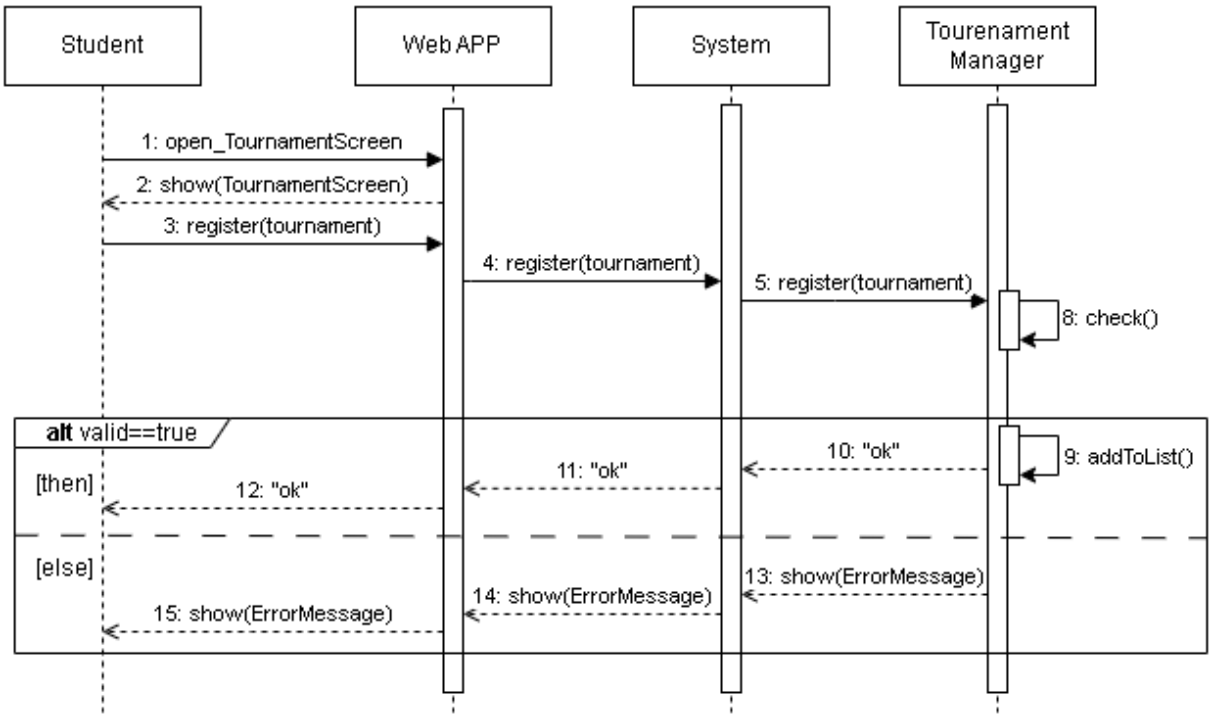

Figure 24: UC08 sequence diagram

[UC09] - Battle Join

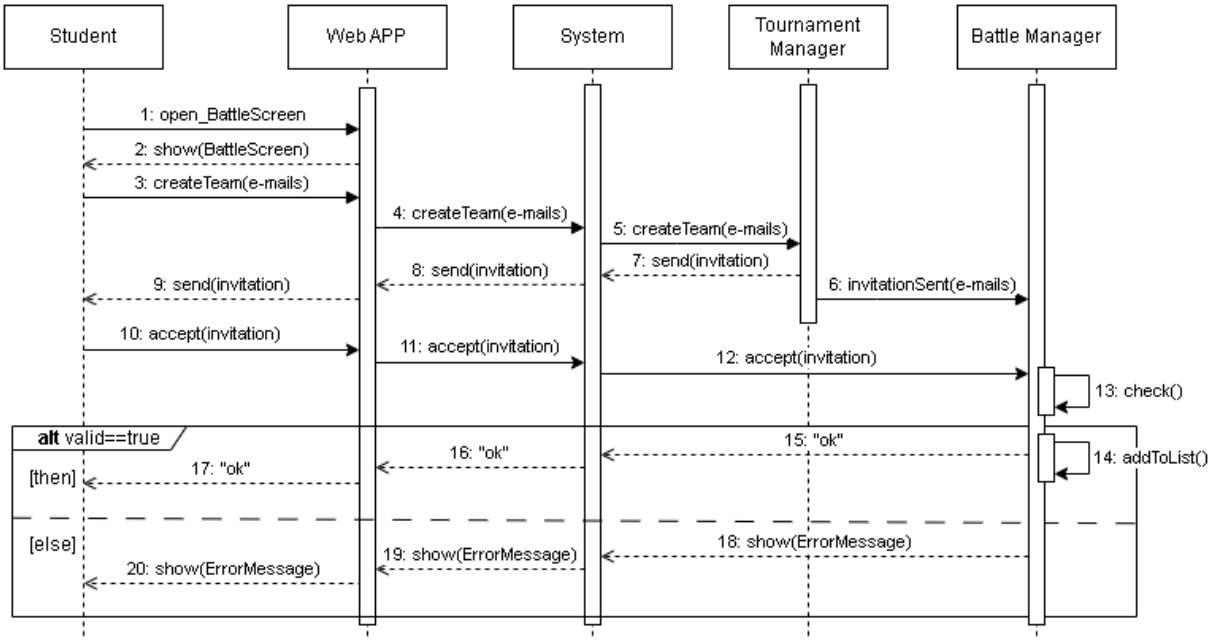

Figure 25: UC09 sequence diagram

[UC10] - Invitation accepting

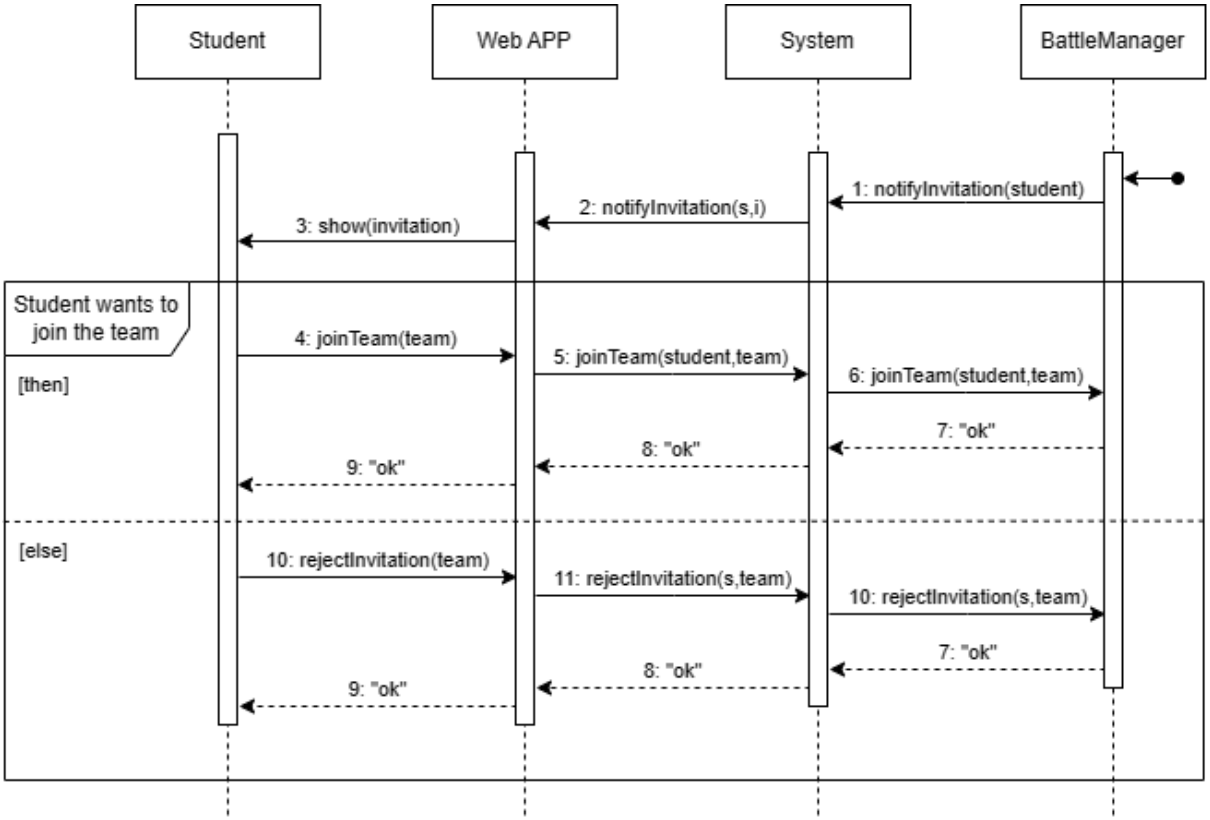

Figure 26: UC10 sequence diagram

[UC11] - Code Manual Evaluation

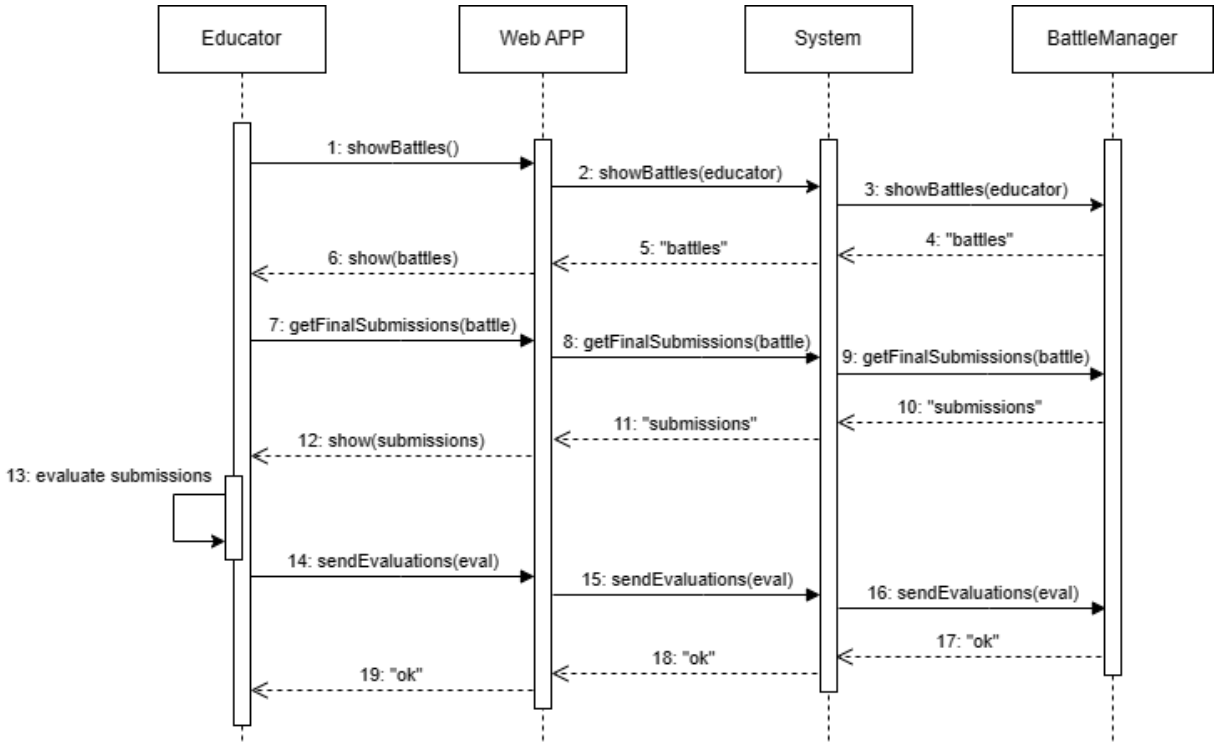

Figure 27: UC11 sequence diagram

[UC12] - Tournament Closure

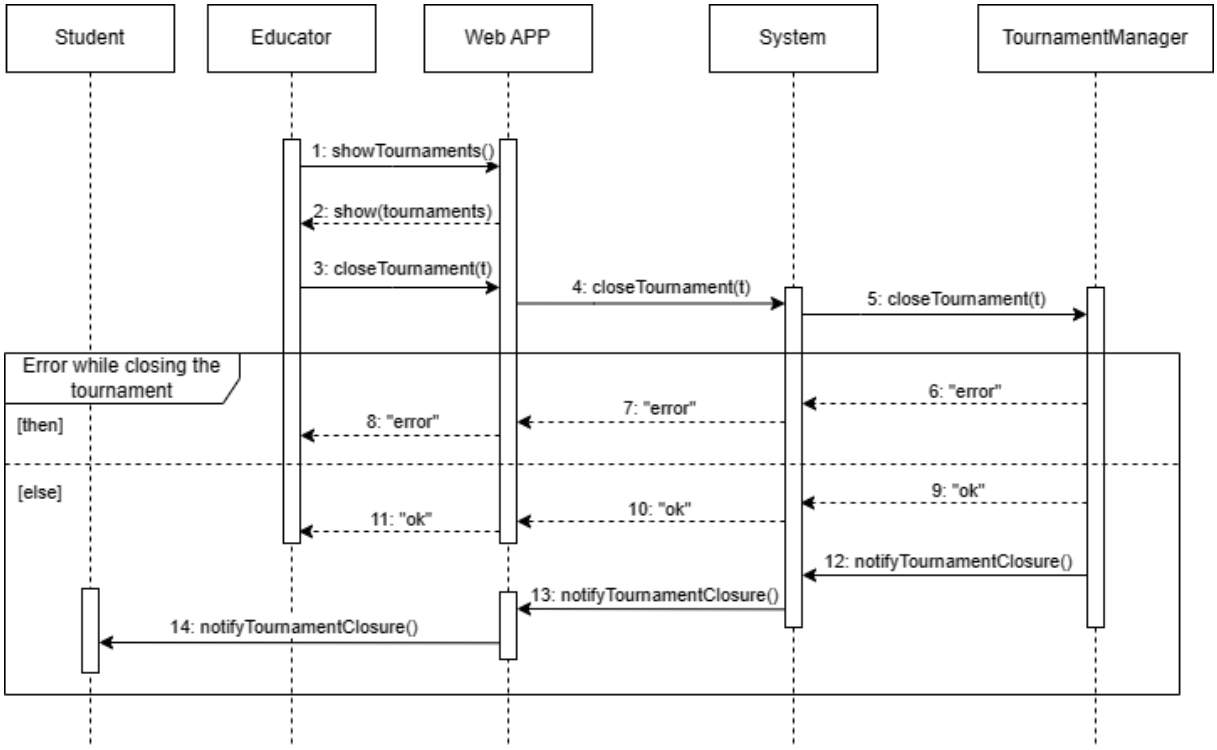

Figure 28: UC12 sequence diagram

[UC13] - Badges Integration

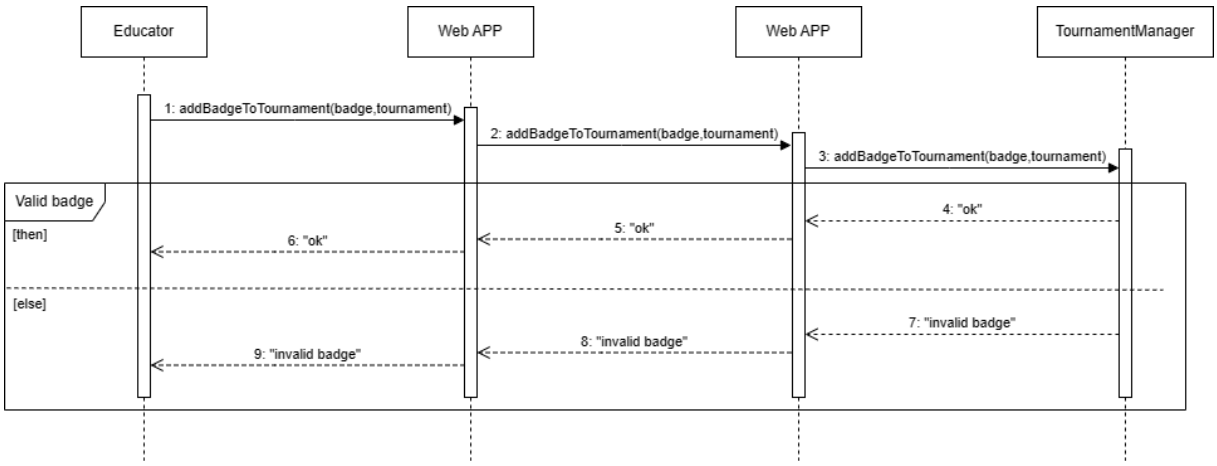

Figure 29: UC13 sequence diagram

[UC14] - Badge Obtaining

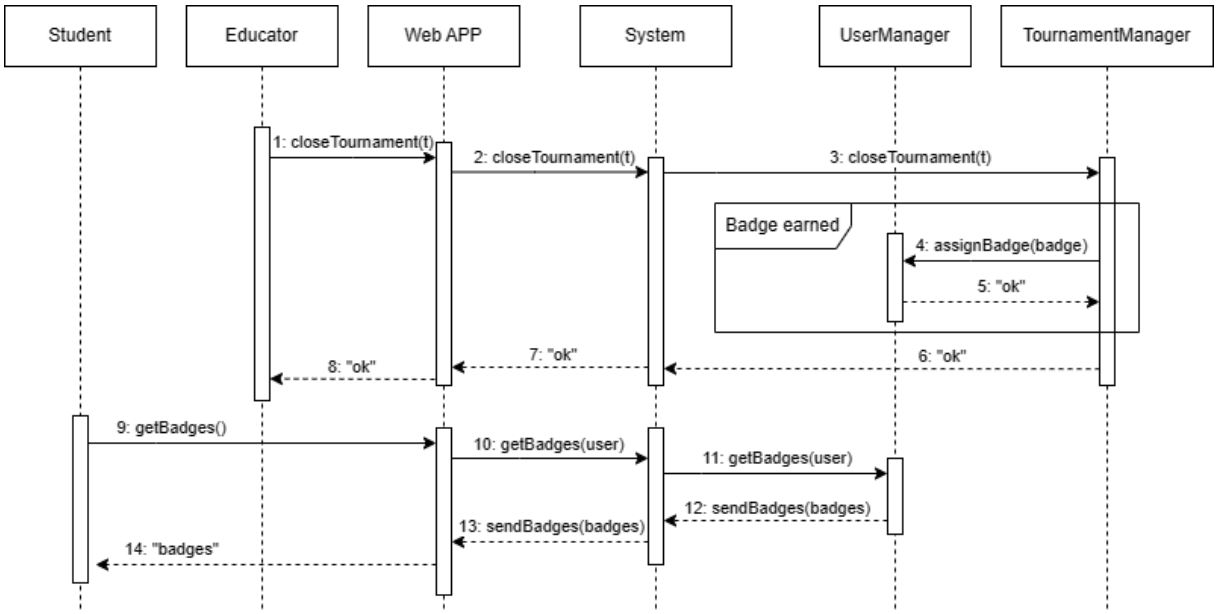

Figure 30: UC14 sequence diagram

3.2.3 Requirements mapping

[G1] Educator can create tournaments and battles setting all the necessary parameters and badges	
R1: The system must allow users to register on the platform R2: The system must allow users to authenticate themselves and log in securely R3: The system must allow educators to create new tournaments R4: Tournament creation must include specifying a description of the tournament R5: The system must allow authorized educators to create code kata battles within a tournament R6: Creation of CKB must include kata description, software project, build scripts, and scoring configurations R8: The system automatically creates GitHub repositories for battles R14: The system must allow educators to create badges associated with specific tournaments R16: The system automatically notifies students and educators about new tournaments, battles, and tournament closures R17: Notifications must include deadlines and updates	D1: Users (educators and students) have basic proficiency in using web-based platforms and are familiar with version control systems, such as Git D2: Educators have the necessary knowledge to create meaningful code kata battles, including defining test cases and scoring criteria D3: Educators and students can connect to the Internet with their devices D4: Notification must arrive to connected users in one minute D6: Users dispone of an e-mail address

Table 22: Requirement mapping on goal 1

[G3] Educator that creates a tournament can allow other educators to add battles to the tournamen	
R5: The system must allow authorized educators to create code kata battles within a tournament R6: Creation of CKB must include kata description, software project, build scripts, and scoring configurations R18: The system must allow the educator owning the tournament to authorize other educators to create CKB	D3: Educators and students can connect to the Internet with their devices D4: Notification must arrive to connected users in one minute D6: Users dispone of an e-mail address

Table 24: Requirement mapping on goal 3

[G2] Student can complete battles on their own or in groups by providing their code	
R1: The system must allow users to register on the platform R2: The system must allow users to authenticate themselves and log in securely R5: The system must allow educators to create code kata battles within a tournament R7: The system must allow students to register for individual battles or form teams based on specified team size limits R9: Students must be able to fork the repository and set up automated workflows using GitHub Actions R12: The system must allow authorized educators to add battles to tournaments R13: The closure triggers updates to personal tournament scores for each student	D1: Users (educators and students) have basic proficiency in using web-based platforms and are familiar with version control systems, such as Git D3: Educators and students can connect to the Internet with their devices D5: Students are required to have a GitHub account for participation, and the platform assumes that students have the necessary permissions to fork repositories D6: Users dispone of an e-mail address D7: Users need a reliable internet connection to interact with the CKB platform

Table 23: Requirement mapping on goal 2

[G4] Students and educators can see the list of ongoing and terminated tournaments and the corresponding rank	
R1: The system must allow users to register on the platform R2: The system must allow users to authenticate themselves and log in securely R10: During the battle, the platform updates scores with every push made by students in real-time R11: The system must allow educators to evaluate manually submissions R13: The closure triggers updates to personal tournament scores for each student	D3: Educators and students can connect to the Internet with their devices D6: Users dispone of an e-mail address D7: Users need a reliable internet connection to interact with the CKB platform

Table 25: Requirement mapping on goal 4

[G5] Students and educators can see the current rank of every battle they are involved in	
R1: The system must allow users to register on the platform R2: The system must allow users to authenticate themselves and log in securely R10: During the battle, the platform updates scores with every push made by students in real-time R13: The closure triggers updates to personal tournament scores for each student	D3: Educators and students can connect to the Internet with their devices D6: Users dispone of an e-mail address D7: Users need a reliable internet connection to interact with the CKB platform

Table 26: Requirement mapping on goal 5

[G6] Educator can go through the sources produced by a team and manually assign a score	
R1: The system must allow users to register on the platform R2: The system must allow users to authenticate themselves and log in securely R11: The system must allow educators to evaluate manually submissions R13: The closure triggers updates to personal tournament scores for each student	D1: The users (educators and students) have basic proficiency in using web-based platforms and are familiar with version control systems, such as Git D3: Educators and students can connect to the Internet with their devices D5: Students are required to have a GitHub account for participation, and the platform assumes that students have the necessary permissions to fork repositories D7: Users need a reliable internet connection to interact with the CKB platform

Table 27: Requirement mapping on goal 6

[G7] Educators and students can visualize all badges and every student's collected badges	
R14: The system must allow educators to create badges associated with specific tournaments R15: Badges are automatically assigned based on predefined rules and student performance	D3: Educators and students can connect to the Internet with their devices D6: Users dispone of an e-mail address

Table 28: Requirement mapping on goal 7

3.3 Performance requirements

1. **Response time:**
   To ensure timely responsiveness for user interactions the platform should respond to user actions (e.g., loading pages, submitting forms) within a maximum of two seconds under normal load conditions.

2. **Scalability:**
   To ensure the platform can handle an increasing number of users and data the platform should support concurrent access by at least 1000 users without significant degradation in performance, for up to 1000 battles simultaneously.

3. **GitHub integration:**
   GitHub repository creation, including code kata and automation setup, should take no longer than two minutes, also automated workflow triggers from GHA should result in platform updates within 1 minute of code submission, otherwise, some students' submission may be lost.

4. **Automated assessment and consolidation stage:**
   To ensure fast evaluation of code submissions, automated assessments (including code analysis and test execution) should complete within two minutes for a battle. Also, the platform should handle simultaneous automated assessments from multiple battles without queuing delays.

5. **Notification system:**
   Notifications about battle and tournament updates need to be sent to every interested user in at most one minute.

6. **Badges and gamification:**
   To provide a continuous experience for badge assignment and visualization, they need to be assigned right after the end of each tournament and to the correct students. Also, their visualization on users' profile should load instantaneously.

7. **Security:**
   To ensure secure data handling and prevent unauthorized access user authentication and authorization processes should happen rapidly and the platform should do regular security inspections to identify and address potential vulnerabilities.

8. **Data storage and retrieval:**
   To optimize data storage and retrieval processes, database queries for common operations should be fully optimized. Frequent backups are done to prevent data losses.

9. **Reliability:**
   To ensure consistent and reliable performance, the platform should achieve 99.9% uptime.

These are criteria that CKB aims to achieve. Rigorous testing and continuous monitoring will be essential to meet and maintain these standards throughout the system's lifecycle.

3.4 Design constraints

These are the constraints related to the design of the system. They are divided in three categories: standards compliance, hardware limitations and other constraints.

3.4.1 Standard compliance

1. **Web standards:**
   User interface and interactions must adhere to modern web standards, ensuring compatibility with major web browsers such as Chrome, Firefox, Safari, and Edge.

2. **Coding standards:**
   Code written for the platform must stick to standard coding conventions, such as proper documentation and modular design.

3. **Security standards:**
   The platform must comply with standard security protocols to safeguard user data, prevent unauthorized access and protect against common web vulnerabilities, e.g. Cross-Site Scripting, File Inclusion Vulnerabilities or injection attacks.

4. **Data privacy regulations:**
   The platform must comply with data privacy regulations such as General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA), and ensure the secure handling of user data, in addition, explicit consent must be obtained from users regarding data collection and usage.

5. **Environmental sustainability for servers:**
   Servers hosting the platform need to adhere to the most recent environmental sustainability standards. This includes considerations for energy efficiency, eco-friendly practices, and responsible disposal of electronic waste.

3.4.2 Hardware limitations

1. **Server requirements:**
   The platform's server infrastructure must meet the specified minimum requirements for processing power, memory and storage to ensure optimal performance and responsiveness.

2. **Internet connection:**
   Users accessing the platform must have a reliable and high-speed internet connection to ensure efficient participation in battles, timely submission of solutions and access to real-time updates.

3.4.3 Any other constraints

1. **Educator access:**
   Educators must have appropriate permissions to create and manage tournaments, battles and associated configurations. Access control mechanisms must be in place via discretionary access control to prevent unauthorized access.

2. **GitHub integration:**
   Users must have a GitHub account to fully utilize the platform's features, especially the integration with GitHub repositories and automated workflows.

3. **GitHub API rate limits:**
   The platform must adhere to GitHub's API rate limits to avoid disruptions in GitHub integration. Excessive API requests may result in rate-limiting, affecting the automated workflow triggered by code submissions.

4. **Scalability considerations:**
   Scaling beyond a certain threshold may lead to additional costs or require adjustments to the hosting environment.

3.5 Software System Attributes

3.5.1 Reliability

The platform must consistently and accurately perform its functions without failures or errors. To do so is necessary to implement error handling mechanisms and use a redundant system to ensure continuous operation.
Also to not lose data in case of a failure it's necessary to frequently back up all the data in the database, keeping them in a different building and offline to their security.
It's also important to test the whole system, including unit testing and integration testing, to identify and correct bugs and malfunctioning.

3.5.2 Availability

The platform must be accessible to users when needed. A way to assure that is to implement a physical load balancing system that distributes incoming traffic through different server's replicas to prevent overload. Server's and data center's replicas also permit to minimize downtime in case of hardware failures. The usage of monitoring tools can also be very helpful to detect and respond to issues, ensuring high system uptime.
Since this system does not provide emergency services or services related to critical situations, it must provide availability of 99.9%. To have the best results, it is essential that the Mean Time To Failures (MTTF) is as long as possible and that the Mean Time to Repair (MTTR) is as short as possible.

3.5.3 Security

To safeguard the platform against unauthorized access, data breaches and malicious activities, it's necessary to employ an encryption mechanism to protect data during transmission (avoiding attacks like sniffing and spoofing) and storage in the database, also to guarantee users' privacy. Developers need to implement secure coding practices to mitigate vulnerabilities such as injection attacks (e.g. SQL injection), cross-site scripting (XSS) and cross-site request forgery (CSRF), at application level a Web Application Firewall (WAF) is employed to protect against this common attacks and also to log and alert on suspicious activities, providing insights for threat response.
Also, since the script uploaded can be potentially dangerous to execute on servers, it's necessary to implement a robust sandboxing mechanism to isolate and execute user-uploaded code securely, can also be very helpful the usage of static code analysis tools and threat intelligence feeds and pattern matching to check user-submitted code for potential security vulnerabilities or malicious code patterns before execution. Another important thing is applying restrictions on resource usage (CPU, memory) to prevent malicious activities from the users.
At physical level are necessary infrastructures such as: firewalls that filter and monitor incoming network traffic, Intrusion Detection System (IDS) to detect and respond to security threats or unusual activities.
Finally, frequently conduct regular checks and reviews for security and vulnerability and penetration testing to identify and address potential weaknesses is fundamental.

3.5.4 Maintainability

The platform should be easily modified, updated and widened over its lifecycle. Some approaches can be adopting modular code structures to facilitate code maintenance and utilizing version control systems for tracking changes and managing collaborative development, also it's important the incorporation of an automated testing routine that covers at least the 75% of the code and the continuous update of the software.

3.5.5 Portability

The platform needs to run on different environments and platforms. The principal solution is to implement platform-independent code by adhering to standard programming languages and frameworks, also it's necessary to address dependencies and ensure compatibility with different operating systems and web browsers, this can be done using containerization technologies (e.g. Docker) for packaging and deploying applications consistently across various environments.

4 Formal Analysis Using Alloy

In this chapter we report a formal analysis of the system performed with the specification language Alloy, along with the output of Alloy Analyzer, in order to check the correctness and coherence of the logical components of the system.

4.1 Signatures

In this section, we list the signatures used in the formal analysis. Note that the description of the various components includes only the elements that are at interest in the formal analysis.

```
open util/integer
open util/boolean

sig Email,Name {}
sig EducatorEmail,EducatorName {}

sig Badge{
    tournament: one Tournament
}

sig Student{
    name: one Name,
    email: one Email,
    //The set of earned badges
    var badges: set Badge
}

enum GroupStatus{
    //The group has not been created yet
    GroupNotCreatedYet,
    //The group has been created but it has not been accepted yet
    GroupPending,
    //The group has been accepted and its members can now solve the battle
    GroupValid,
    //The group has not been accepted
    GroupRejected
}

sig Group{
    var students: set Student,
    var status: one GroupStatus
}
```

```
sig Educator{
    name: one EducatorName,
    email: one EducatorEmail
}

enum TournamentStatus{
    //The tournament has not been created yet
    TournamentNotCreatedYet,
    //The tournament has been created and it accepts Registrations
    TournamentRegistration,
    //The tournament has started,
    //students can solve battles within the tournament
    TournamentStarted,
    //The tournament has been closed
    TournamentClosed
}

sig Tournament{
    creator: one Educator,
    var status: one TournamentStatus,
    //The set of students subscripted to the tournament
    var students: set Student
}

enum BattleStatus{
    //The battle has not been created yet
    BattleNotCreatedYet,
    //The battle has been created and it accepts Registrations
    BattleRegistration,
    //The battle has started, groups can submit their solutions
    BattleStarted,
    //The solutions are manually evaluated by the educator
    BattleConsolidation,
    //The battle has been closed
    BattleClosed
}

sig Battle{
    creator: one Educator,
    tournament: one Tournament,
    //Minimum and maximum students per group
    minStudentPerGroup: one Int,
```

```
    maxStudentPerGroup: one Int,
    //Groups partecipating to the battle
    var groups: set Group,
    //Groups enlisted for the battle but not yet confirmed
    var pendingGroups: set Group,
    var status: one BattleStatus,
    //True if the educator manually evaluates the solutions
    consolidationPhase: one Bool
}
```

4.2 Time-independent facts

Alloy facts allow the model to behave coherently with the real world. Facts can depend or not on the time evolution of the system: in this section, we report the static, time-independent facts, while the next sections will analyze time-dependent variables and their interactions.

```
//All Emails correspond to at least a Student
fact NoUnusedEmails{
    no e:Email | (no s:Student | s.email = e)
}
//All Names correspond to at least a Student
fact NoUnusedNames{
    no n:Name | (no s:Student | s.name = n)
}
//All EducatorEmails correspond to at least an Educator
fact NoUnusedEducatorEmails{
    no e:EducatorEmail | (no ed:Educator | ed.email = e)
}
//All EducatorNames correspond to at least an Educator
fact NoUnusedEducatorNames{
    no n:EducatorName | (no ed:Educator | ed.name = n)
}

//All the students have unique Emails
fact UniqueEmailForStudents{
    no disj s1,s2:Student | s1.email = s2.email
}
//All the educators have unique Emails
fact UniqueEmailForEducators{
    no disj e1,e2:Educator | e1.email = e2.email
}

//The minimum and maximum number of students per group must be coherent
```

```
fact CoherentGroupConstraintsInBattle{
    all b:Battle | b.minStudentPerGroup > 0
    and (b.maxStudentPerGroup >= b.minStudentPerGroup)
}
```

4.3 Tournaments

The following facts model the correct evolution of the state of each tournament, along with some considerations on the battles contained in the tournament and the students subscribed to it.

```
//Every Tournament is initially not created
fact BeginningNotCreatedYet{
    all t:Tournament | always(
    (t.status = TournamentNotCreatedYet implies
    historically t.status = TournamentNotCreatedYet) and
    (t.status != TournamentNotCreatedYet implies
    once t.status = TournamentNotCreatedYet))
}

//Transition predicates
pred TournamentNotCreatedYetToRegistration[t:Tournament]{
    t.status = TournamentNotCreatedYet
    t.status' = TournamentRegistration
}
pred TournamentRegistrationToStarted[t:Tournament]{
    t.status = TournamentRegistration
    t.status' = TournamentStarted
}
pred TournamentStartedToClosed[t:Tournament]{
    t.status = TournamentStarted
    t.status' = TournamentClosed
}

//All the changes in the tournament status follow the defined transitions
fact TransitionsFact{
    all t:Tournament | always(
    t.status'=t.status or
    TournamentNotCreatedYetToRegistration[t] or
    TournamentRegistrationToStarted[t] or
    TournamentStartedToClosed[t])
}

//For the tournament to close, all the battles have to be closed
```

```
fact TournamentsClose{
    all t: Tournament | always(
        t.TournamentStartedToClosed implies (
            all b:Battle | b.tournament = t implies b.status' = BattleClosed
        )
    )
}

//All the tournaments eventally close
fact TournamentsClose{
    all t: Tournament | eventually t.status = TournamentClosed
}

//The students can subscribe only in the Registration phase
fact RegistrationsInTournament{
    all t:Tournament | always(
    (t.status = TournamentNotCreatedYet implies #t.students = 0 ) and
    (t.students' != t.students implies t.status' = TournamentRegistration))
}
```

4.4 Student groups

The following facts model the correct formation of teams within students that want to solve together a specific battle, along with the evolution of the teams' status.

```
//The groups start from an initial state
fact GroupInitialState{
    all g:Group | once( g.status = GroupNotCreatedYet )
}

//Group state transitions
pred GroupNotCreatedYetToPending[g:Group]{
    g.status = GroupNotCreatedYet
    g.status' = GroupPending
}
pred GroupPendingToValid[g:Group]{
    g.status = GroupPending
    g.status' = GroupValid
}
pred GroupPendingToRejected[g:Group]{
    g.status = GroupPending
    g.status' = GroupRejected
}
```

```
//The group status must follow the possible transitions
fact GroupTransitions{
    all g:Group | always(
        g.status = g.status' or
        GroupNotCreatedYetToPending[g] or
        GroupPendingToValid[g] or
        GroupPendingToRejected[g]
    )
}

//A group must be initially created by at least one student
fact GroupsCreatedBySomeStudents{
    all g:Group | always(
        GroupNotCreatedYetToPending[g] implies #g.students' > 0
    )
}

//Students can join a group only when it is "pending"
fact StudentsJoinGroupsWhenCreated{
    all g:Group | always(
        (g.status = GroupNotCreatedYet implies #g.students = 0) and
        (g.students' != g.students implies g.status' = GroupPending)
    )
}

//A group is pending only if it is enlisted in a battle
fact PendingIfInBattlePendingGroups{
    all g:Group | always(
        g.status = GroupPending iff one b:Battle | g in b.pendingGroups
    )
}

//A group is valid only if it is accepted in a battle
fact ValidIfInBattleValidGroups{
    all g:Group | always(
        g.status = GroupValid iff one b:Battle | g in b.groups
    )
}

//No unused groups
fact NoUnusedGroups{
    all g:Group | eventually (g.status = GroupValid or g.status = GroupRejected)
}
```

```
//A group exists in the context of just one battle
fact NoGroupsInMultipleBattles{
    all g:Group |
    (no disj b1,b2:Battle |
        once (g in b1.pendingGroups) and
        once (g in b2.pendingGroups))
}
```

4.5 Code Kata Battles

These facts model the correct evolution in time of each battle, keeping into consideration all the context parameters such as the tournament they are in and the students that want to partecipate.

```
//The battles start from an initial state
fact BattleInitialState{
    all b:Battle | once( b.status = BattleNotCreatedYet )
}

//Battle state transitions
pred BattleNotCreatedYetToRegistration[b:Battle]{
    b.status = BattleNotCreatedYet
    b.status' = BattleRegistration
}
pred BattleRegistrationToStarted[b:Battle]{
    b.status = BattleRegistration
    b.status' = BattleStarted
}
pred BattleStartedToConsolidation[b:Battle]{
    b.status = BattleStarted
    b.status' = BattleConsolidation
}
pred BattleStartedToClosed[b:Battle]{
    b.status = BattleStarted
    b.status' = BattleClosed
}
pred BattleConsolidationToClosed[b:Battle]{
    b.status = BattleConsolidation
    b.status' = BattleClosed
}

//The battle status must follow the possible transitions
fact BattleTransitions{
```

```
    all b:Battle | always(
        b.status' = b.status or
        BattleNotCreatedYetToRegistration[b] or
        BattleRegistrationToStarted[b] or
        BattleStartedToConsolidation[b] or
        BattleStartedToClosed[b] or
        BattleConsolidationToClosed[b]
    )
}

//The consolidation phase happens only when necessary
fact ConsolidationPhaseWhenNecessary{
    all b:Battle | always(
        BattleStartedToConsolidation[b] implies b.consolidationPhase.isTrue and
        BattleStartedToClosed[b] implies b.consolidationPhase.isFalse
    )
}

//The battle can be created only when the tournament has started
fact BattleInStartedTournament{
    all b:Battle | always(
        BattleNotCreatedYetToRegistration[b] implies
        b.tournament.status' = TournamentStarted)
}

//Students can register to a battle only if they are registered
//to the tournament
fact AllGroupStudentsInTournament{
    all b:Battle | always(
        no s:Student | s not in b.tournament.students and
        s in b.(groups + pendingGroups).students
    )
}

//Student cannot register in two groups for the same battle
fact NoStudentInTwoGroupsSameBattle{
    all b:Battle | always(
        no disj g1,g2:b.(groups + pendingGroups) | (
            some s:Student | (
                (s in g1.students) and (s in g2.students) )))
}
```

4.6 Interaction between Groups and Battles

Since the interaction between Groups and Battles is crucial for the correctness of the model, this dedicated paragraph lists all the facts necessary for a correct time evolution of these entities.

```
//Groups can register only during the Battle Registration phase
fact GroupsRegistrationDuringRegistrationPhase{
    all b:Battle | always(
        (b.status = BattleNotCreatedYet implies
        #b.groups = 0 and #b.pendingGroups = 0) and
        (b.status = BattleRegistration implies #b.groups = 0) and
        ((b.status = BattleStarted or b.status = BattleConsolidation or
        b.status = BattleClosed) implies
        #b.pendingGroups = 0 and b.groups = b.groups'))
}

//Groups can partecipate to the battle only if they are valid
fact OnlyValidGroupsPartecipate{
    all b:Battle | always(
        all g:b.groups | g.status = GroupValid
    )
}

//Groups get accepted only if they have the correct number of students
fact OnlyGroupsWithCorrectStudentsGetAccepted{
    all b:Battle | always(
        all g:b.groups | (
            #g.students >= b.minStudentPerGroup and
            #g.students <= b.maxStudentPerGroup and
            once g in b.pendingGroups))
}

//Groups pending cannot invite more than maxStudentPerGroup students
fact NoMoreThanMaxStudentsPerGroup{
    all b:Battle | always(
        all g:b.pendingGroups | #g.students <= b.maxStudentPerGroup
    )
}

//Groups get rejected only if they have an incorrect number of students
//when the battle starts
fact RejectedGroupsHaveIncorrectStudents{
    all g:Group | always(
```

```
        GroupPendingToRejected[g] iff (
            one b:Battle | ( BattleRegistrationToStarted[b] and
                g in b.pendingGroups and (
                #g.students' < b.minStudentPerGroup or
                #g.students' > b.maxStudentPerGroup) )))
}
```

4.7 Badges

To conclude the formal analysis of the system, the following facts model the assignment of badges to the students when a tournament ends.

```
//The set of badges of each student can change
pred ChangedBadgesSet[s:Student,b:Badge]{
    b not in s.badges
    s.badges' = s.badges + b
}

//The set of badges increments only
fact NoSubtractedBadges{
    all s:Student | always( all b:Badge |
    (b in s.badges implies b in s.badges'))
}

//Badges can be earned only when a tournament they are subscribed to closes
fact BadgesWhenTournamentCloses{
    all s:Student | once #s.badges = 0 and always(
    all b:Badge | ChangedBadgesSet[s,b] implies
    (s in b.tournament.students and TournamentStartedToClosed[b.tournament]))
}
```

4.8 Resulting model

In this section, the model obtained with an execution of the following **run** command is presented:

```
pred show{
    //Test if these entities can exist
    #Student > 0
    #Educator > 0
    #Tournament = 1
    #Battle = 1
    //Test if the badges are assigned correctly
    some s:Student | eventually #s.badges > 0
    //Test if the groups and the battles interact correctly
    all b:Battle | b.maxStudentPerGroup > b.minStudentPerGroup
    some b:Battle | eventually #b.groups > 0
    some g:Group | eventually g.status = GroupRejected
}

run show for 6
```

Since Alloy allows for a time-dependent analysis, the result is divided into steps. To obtain a clear explanation, the presentation shows the correctness of the model one step at a time.

Initial state

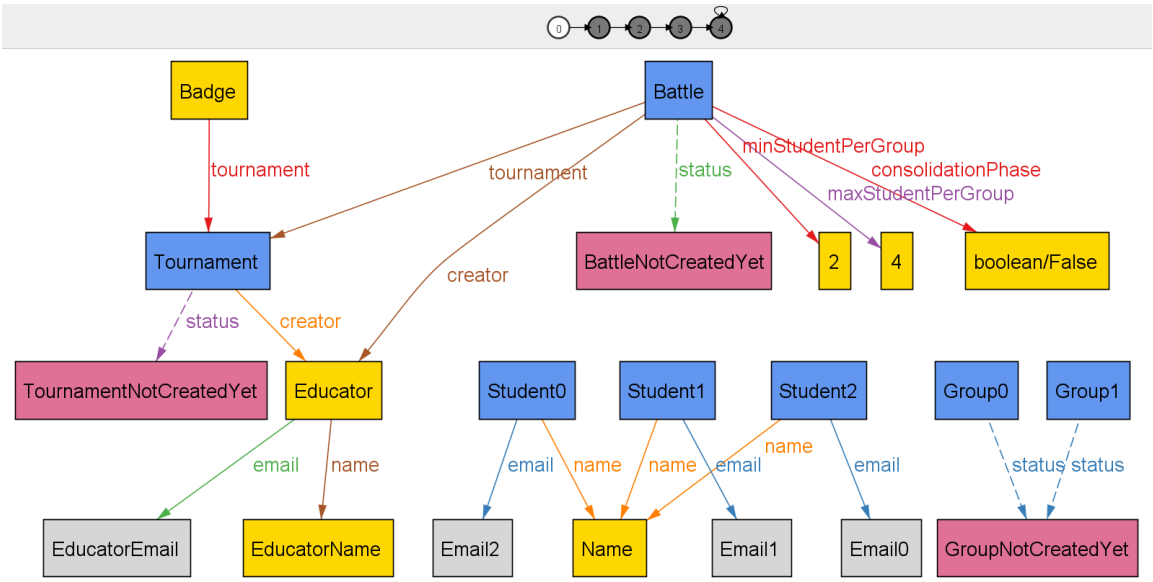

Figure 31: Initial state of the model

In Figure 31 we can see the initial state of the model. Starting with the time-independent constraints, we see that all the students have unique Emails (while having the same Name is allowed) and the Battle has coherent constraints: the number of students per group must be between two and four.

Concerning time-dependent constraints, we can see that the tournament, the battle and the two groups are in the *NotCreatedYet* states, and no registrations by students or groups have been made so far.

First step: tournament creation

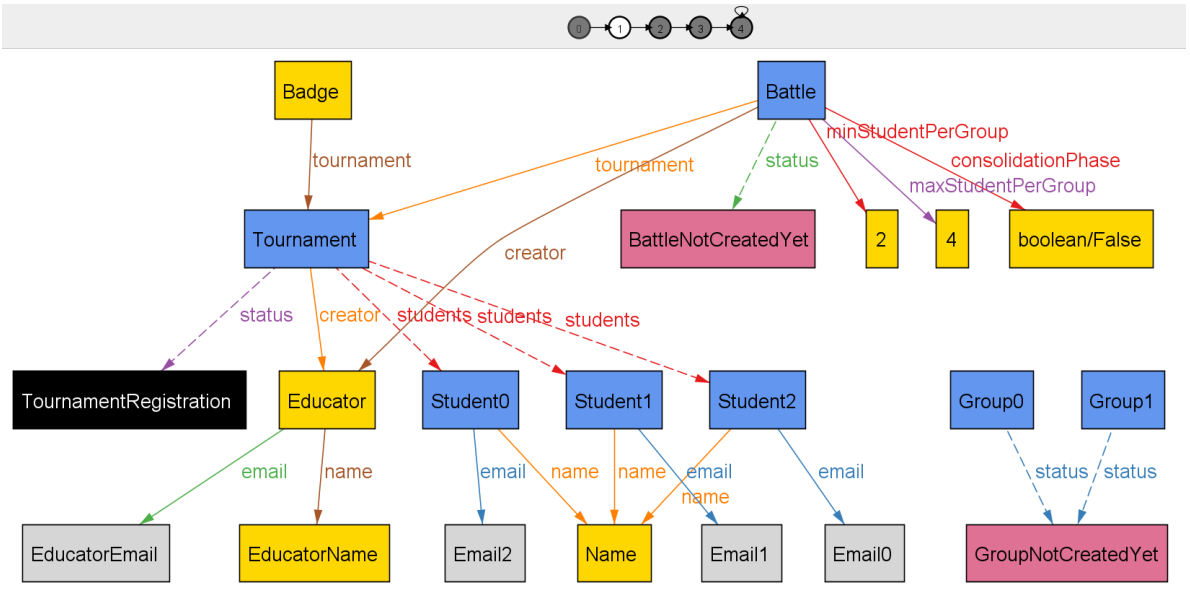

Figure 32: Model after 1 step

In Figure 32 we can see how the system evolves after one step: the tournament has been created and the status of the tournament is now *TournamentRegistration*. As shown by the red dashed lines, the three students subscribed to the tournament. We can see how, since the tournament has not started yet, no battle has been created in the context of this tournament, and no group has been formed yet.

Second step: battle creation

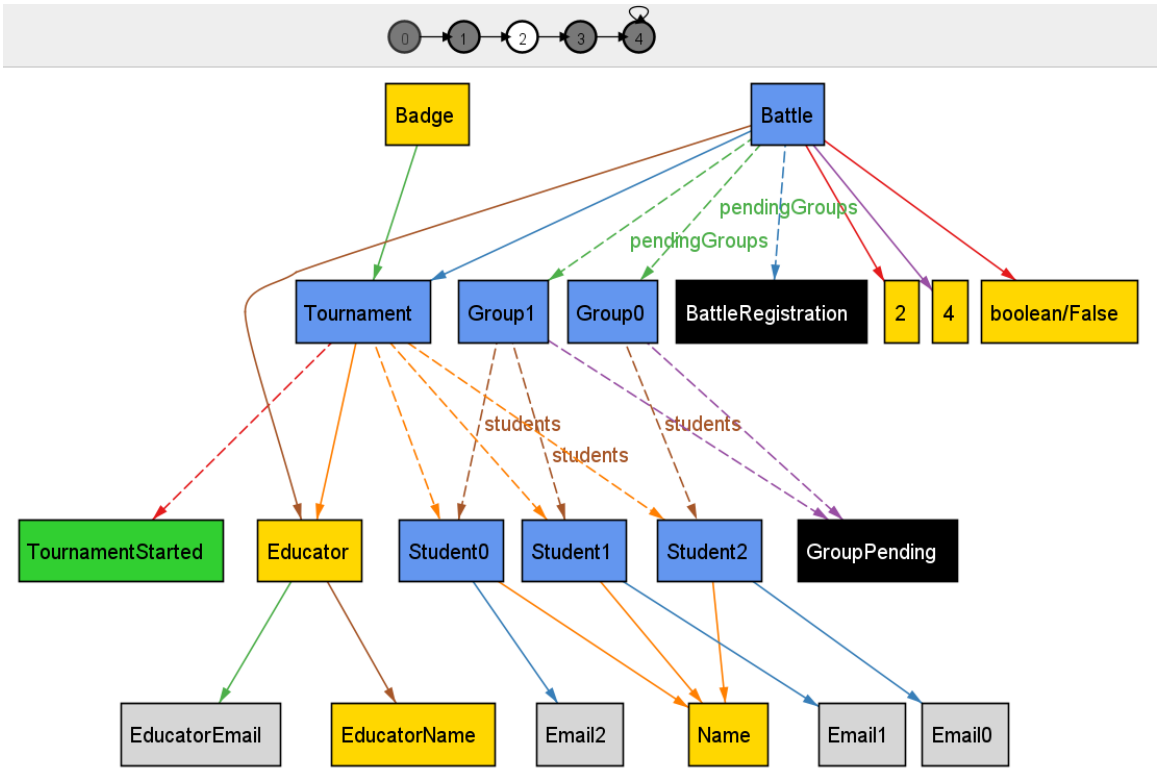

Figure 33: Model after 2 steps

In Figure 33 we can see how the system has evolved after the second step: the tournament has officially started. The Battle has been created and its state is *Registration*. Two groups enlisted for the Code Kata Battle: one group is formed by Student 0 and 1, the other is formed by Student 2 alone. The state of the two groups is *Pending* and they are in the list of *pendingGroups* of the battle.

Third step: group validation and start of the battle

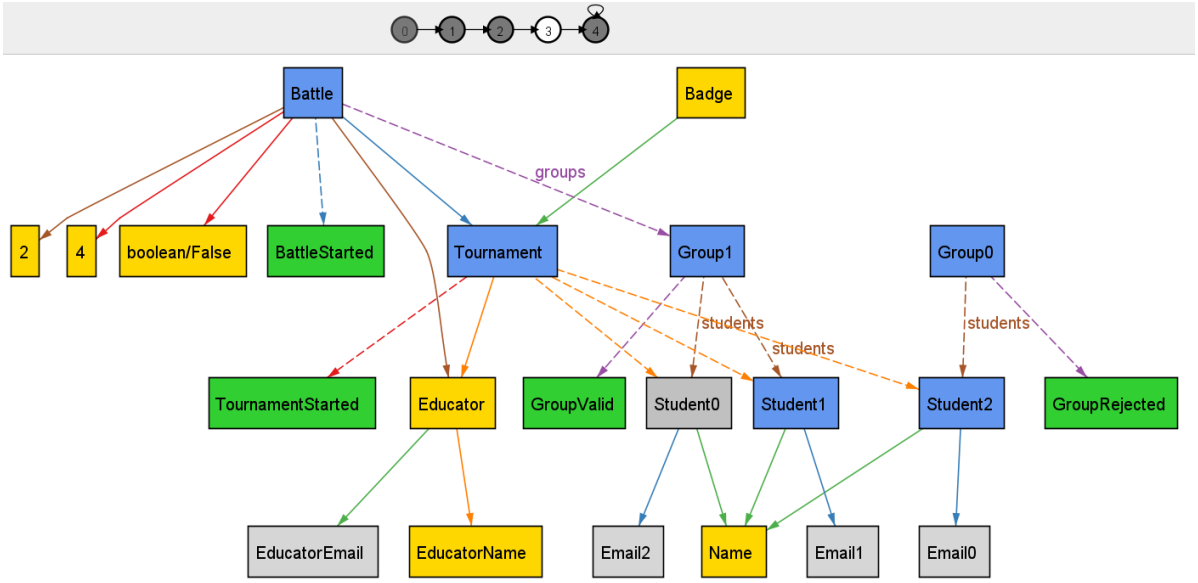

Figure 34: Model after 3 steps

In Figure 34 we can see how the system has evolved after the third step: the Battle has started. Since Group 0 had only one student it has been rejected (the minimum number of students per group in this battle is two), while Group 1 has been validated and inserted in the list of *groups* of the battle. Since the Battle has not finished yet, the Tournament is still in the *TournamentStarted* state.

Fourth step: tournament closure and badge awarding

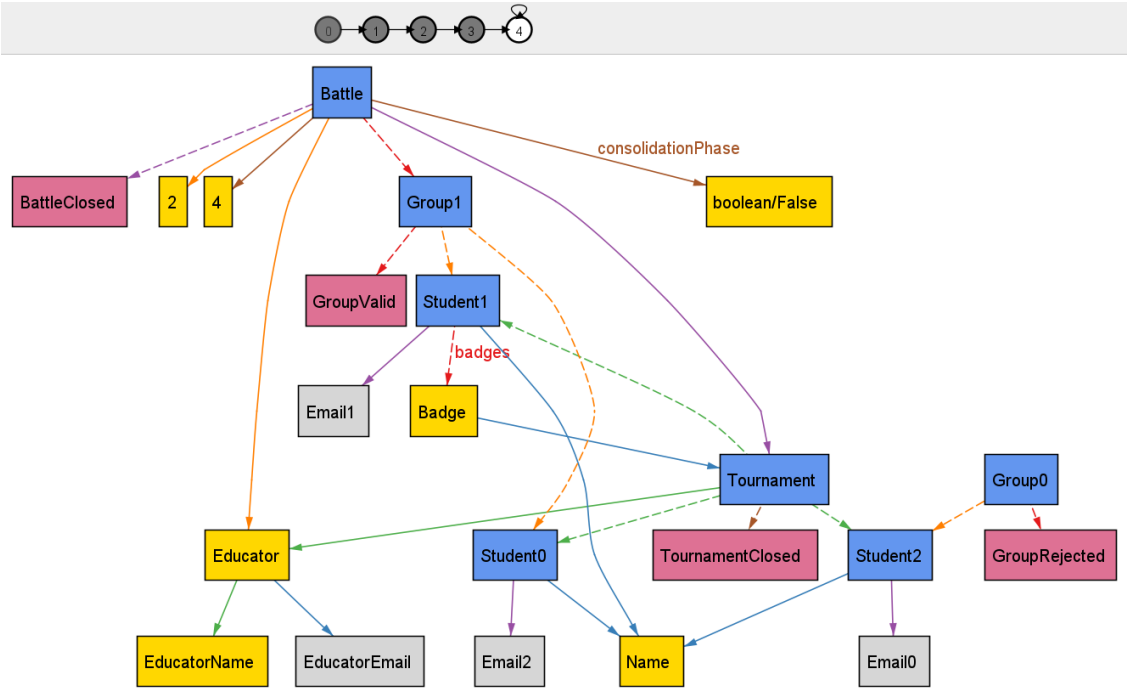

Figure 35: Model after 4 steps

In Figure 34 we can see how the system has evolved after the fourth and last step. Since the *consolidationPhase* parameter of the Battle is false, the Battle has closed without the need of manual evaluation by the educator. The tournament has closed too, and the badge associated with the tournament has been awarded to Student 1. This is the last step of the execution: all the time-steps presented in this section are coherent with the specification of the system and its requirements.

5 Effort Spent

The following tables contain an approximation of the number of hours spent on the creation of this document by each member of the group. In particular, each row represents the time invested in the designing and redaction of the respective chapter. These tables do not include the time spent on the collective review of the work done by each member.

Chapter	Hours spent
1	3
2	7
3	14
4	15

Table 29: Alessandro Annechini

Chapter	Hours spent
1	9
2	5
3	19
4	6

Table 30: Nicole Filippi

Chapter	Hours spent
1	4
2	15
3	16
4	4

Table 31: Riccardo Fiorentini

6 References

1. Inspirational sites similar to our web application:
    - `https://codecombat.com`
    - `https://www.codingame.com`
    - `https://codebattle.hexlet.io`
    - `https://www.codebattle.in`
2. DrawIO, tool used for equence diagrams, use case diagram and class diagram. URL: `https://www.drawio.com`
3. PlantUML, tool used for state charts. URL: `https://www.plantuml.com`
4. Moqups, tool used for web application's mockups. URL: `https://moqups.com`
5. Alloy specification and documentation, and alloy tools used to write and executr alloy model. URL: `https://alloytools.org/`
6. Specification document: `"Assignment RDD AY 2023-2024"`
7. LaTeX, used to redact the document. Documentation: `https://www.latex-project.org/help/documentation/`
8. Overleaf, LaTeX online editor. URL: `https://www.overleaf.com/`