POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

CodeKataBattle
Requirements Analysis and Specification Document

Authors:

Gigante Davide (Personal code: 11018245)
Topazio Federica (Personal code: 10985760)

Academic Year: 2023-24

Contents

Contents **i**

1 Introduction **1**
1.1 Purpose . . . 1
1.1.1 Goals . . . 3
1.2 Scope . . . 4
1.2.1 World phenomena . . . 5
1.2.2 Shared phenomena . . . 5
1.3 Definitions, Acronyms, Abbreviations . . . 6
1.3.1 Acronyms . . . 6
1.3.2 Definitions . . . 7
1.3.3 Revision history . . . 8
1.3.4 Reference Documents . . . 9
1.4 Document Structure . . . 9

2 Overall Description **11**
2.1 Product perspective . . . 11
2.1.1 Scenarios . . . 11
2.1.2 Class Diagram . . . 21
2.1.3 State Chart Diagrams . . . 21
2.2 Product functions . . . 26
2.3 User characteristics . . . 28
2.4 Assumptions, dependencies and constraints . . . 29
2.4.1 Domain assumptions . . . 29

3 Specific Requirements **31**
3.1 External Interfaces Requirements . . . 31
3.1.1 User Interfaces . . . 31

3.1.2 Hardware Interfaces . . . 37
3.1.3 Communication Interfaces . . . 37
3.2 Functional Requirements . . . 38
3.2.1 Use cases diagrams . . . 38
3.2.2 Use cases . . . 40
3.2.3 Sequence Diagrams . . . 58
3.2.4 Functional Requirements . . . 74
3.2.5 Mapping Use Cases - Requirements . . . 79
3.2.6 Traceability Matrix . . . 83
3.3 Performance Requirements . . . 85
3.4 Design Constraints . . . 85
3.4.1 Standard Compliance . . . 85
3.4.2 Hardware Limitations . . . 85
3.5 Software System Attributes . . . 86
3.5.1 Reliability . . . 86
3.5.2 Availability . . . 86
3.5.3 Security . . . 87
3.5.4 Maintainability . . . 88

4 Formal Analysis using Alloy **91**
4.1 Definitions . . . 91
4.1.1 Enumerations . . . 91
4.1.2 Signatures . . . 92
4.1.3 Facts . . . 96
4.2 First World . . . 99
4.3 Second World . . . 101
4.4 Third World . . . 102

5 Effort Spent **105**
5.0.1 Effort-Topazio . . . 105
5.0.2 Effort-Gigante . . . 106

List of Figures **107**

List of Tables **109**

1 Introduction

The Requirements Analysis and Specification Document (RASD) aims to explain both the application domain and the system to be developed and it is a baseline for project planning and estimation of its size, cost, and schedule. The RASD is also the inception of the software evaluation by supporting system testing, verification, and validation activities, and for change control by keeping track of the software evolution.

1.1. Purpose

Informatics is a multidisciplinary field that lies at the intersection of computer science, information technology, and various domains of knowledge, ranging from healthcare and business to social sciences and beyond. It revolves around the collection, storage, analysis, and dissemination of information to make informed decisions, solve problems, and enhance the efficiency and effectiveness of various processes.

In today's digital age, informatics has become an integral part of our lives, profoundly shaping how we communicate, work, learn, and access information. Its impact is felt in nearly every facet of modern society. One of the fundamental aspects of informatics is the utilization of programming languages to harness the power of computers for various applications.

Hence, **CodeKataBattle (CKB)** aims to be a platform that wants to help *students* how to reason and how to solve problems related to the paradigm of programming. This platform is based on the concept of **katas**. Katas are an essential component of traditional martial arts training and serve various purposes, such as developing technique, muscle memory, balance, and mental focus.

The idea is to use the same approach of katas, but applied to programming exercises:

- **Problem-Solving Skills**: Just as katas teach how to respond to specific martial arts scenarios, programming challenges and projects teach problem-solving skills. You analyze a problem, break it down into smaller parts, and apply your knowledge of programming to find a solution.

- **Repetition and practice**: Karate practitioners repeat katas to develop muscle memory and precision. Similarly, in programming, practice is crucial. Repeatedly working on coding exercises, projects, and challenges helps reinforce your understanding and improve your coding skills.

These soft skills are obtained through training with peers, challenging each other using **code kata battles**. A code kata battle is a coding exercise, that belongs to a specific *tournament*. When a tournament is created, the platform automatically notifies all the subscribed students, who must enroll in it before the specified subscription deadline, if they intend to participate. The students, who have chosen to participate in a tournament, will be notified for each battle published in it.

The *educator*, who initiates the tournament, chooses also the programming language of all battles that will be created within it. He has the authority to grant other instructors to create battles within it. When defining a battle, the educator must specify on CKB platform:

- a **code kata**, defined as a brief textual description and a software project, including test cases and build automation scripts;
- minimum and maximum number of students per group, formed by students themselves;
- a registration deadline;
- a final submission deadline;
- eventual additional configurations for scoring.

Therefore, in order to join a battle, Students sometimes will need to form a team. Since a Student can join a battle either on his own, or by invitation of a peer, this is the first case by which it is possible to join a battle which requires a minimum team size bigger than one. Otherwise, the system will be in charge of forming groups, providing the possibility to students to put their names down in waiting list. This list will be processed by the system after the expiration date for the subscription to teh tournament. Such that, the system will form groups of max size, until possible, than for the remaining ones, it will check if they satisfy the dimension constraints, notifying all students of the enrollment or not to the battle. In the same way, also in case the team is orchestrated by the Student, who sends out the invites, it will always be the system in the end, that by the deadline will process and notify the Students of the creation or not of the team, based on the constraints. Once the battle registration deadline expires, CKB created a GitHub repository, which link is sent to all the subscribed groups of students. Students begin by

forking the code kata's GitHub repository and setting up a GitHub Actions automated workflow to notify the CKB platform when they push new commits to the main branch. Each push triggers the CKB platform to update the team's battle score. The score, between 0 and 100, considers various automated factors:

- **functional aspects**, measured in terms of number of test cases that pass out of all test cases (the higher the better);
- **timeliness**, measured in terms of time passed between the registration deadline and the last commit (the lower the better);
- **quality level of the sources**, extracted through static analysis tools that consider multiple aspects such as security, reliability, and maintainability (the higher the better). Aspects are selected by the educator at battle creation time. This tools evaluates on the code the aspects selected by the educator, and provide a score that needs to be added to the other mechanisms.

and optional manual evaluation factors:

- after the submission deadline, **personal score assigned by the educator**, who checks and evaluates the work done by students (the higher the better).

The platform continuously updates the battle scores as students push changes, allowing both students and educators to track the evolving rank. Once the consolidation is complete, students are notified of the final battle rank.

At the end of each battle, students' personal tournament scores are updated, reflecting their overall performance in that tournament. The students' overall performance is given by the sum of all battle scores, received during the tournament itself. This information is accessible to all platform users, showing ongoing tournaments and their corresponding ranks. Educators can close tournaments, and when the final rank is available, notifications are sent to all participating students.

1.1.1. Goals

G1 Allow *Educators* to go through the sources produced by each team to assign a manual evaluation, if needed

G2 Allow *Students* to commit their work in the Github repository connected to a specific battle

G3 Allow *Students* to track their personal evolving rank as they push changes on the platform

G4 Allow *Users* to be identified either as *Students* or as *Educators*.

G5 Allow *Educators* to create tournaments

G6 Allow *Educators* to create a code kata battle with all its related features

G7 Allow *Students* to enroll in an existing tournament

G8 Allow *Students* to enroll in a code kata

G9 Allow *Educators*, creators of tournaments, to grant privileges to other *Educators* of creating battle in the related tournament

G10 Allow *Students* to work on the project being given by the platform the link of the Github repository containing the code kata

G11 Allow *Educators* involved in a battle to see the current rank evolving during the battle

G12 Allow *Users* to overview the list of ongoing tournaments

G13 Allow *Users* to overview the list of corresponding tournament rank, measuring how a student's performance compares to other students in a tournament

G14 Allow *Students* to create a team

G15 Allow *Students* to invite other students to a code kata

G16 Allow *Students* to deliver their solution of the battle to the platform

1.2. Scope

The considered stakeholders in this document are the students, who participate in tournaments and battles, and the educators, who create them. Both will be referred as Users. This means that the following phenomena are meant to be possible by any person who subscribes on the CKB platform, no matter the age, country, working employment of the users, as long as they have a device through which they can have access to the platform.

The phenomena affecting the scope of this project can be divided into World and Machine phenomena. The former are the ones occurring in the environment, namely the portion of the real world affected by the machine, but the Machine cannot observe them. The latter are phenomena occurring inside the machine, which is the portion of the system to be developed, but the World cannot observe them.

The intersection between World and Machine phenomena results in Shared phenomena

in turn divided into the ones controlled by the World and observed by the Machine and the ones controlled by the Machine and observed by the World.

1.2.1. World phenomena

World phenomena are events that take place in the real world and taken by themselves do not have a direct impact on the System.

WP1 A Student does not check the System's notifications

WP2 A Student works on the project

WP3 A User runs out of computer battery

1.2.2. Shared phenomena

Shared phenomena are world phenomena that are shared with the Machine. These are further divided in two categories: phenomena controlled by the World and observed by the Machine and phenomena controlled by the Machine and observed by the World.

World controlled-machine observed

SP1 A User subscribes to the CKB platform

SP2 A User inputs to the System all data needed for registration.

SP3 A User logs in to the CKB platform

SP4 An educator creates a tournament with a specific programming language, a code kata, a minimum and maximum students per group, two deadlines: one for registration and one for final submission, and eventual additional configurations for scoring.

SP5 An educator closes a tournament

SP6 A student enrolls in a battle, before the deadline

SP7 A student enrolls in a tournament, before the deadline

SP8 An educator grants other educators the possibility to create battles inside his tournament.

SP9 A student pushes his work on the respective repository

Machine Controlled-World Observed

SP10 The System notifies all students subscribed to the CKB platform of the creation of a new tournament

SP11 The System notifies all students subscribed to a tournament of the creation of new battles within it

SP12 The System shows the available battles within a tournament to subscribed users

SP13 The System shows the available tournaments to users subscribed to the platform

SP14 The System sends the link of the Github repository containing the code kata created for each team to all students members of a subscribed team

SP15 The System askes the students to fork the GitHub repository, and set up an automated workflow through GitHub Actions

SP16 The System pulls the latest sources after each new push; analyses them, runs the tests on the corresponding executables to calculate and updates the battle score of the related team

SP17 The System notifies all students participating in the battle of the final battle rank once available

SP18 The System shows to users involved in a battle to see the current evolving rank of it

SP19 The System updates the personal tournament score of each student at the end of each battle

SP20 The System shows to all users of the platform the list of ongoing tournaments

SP21 The System shows to all users of the platform the list of ongoing tournaments ranks.

SP22 The System notifies the final tournament rank to all students involved in it, once available

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Acronyms

Acronym	Meaning
CKB	Code Kata Battle

RASD	Requirements Analysis and Specification Document
DD	Design Document
UML	Unified Modelling Language
WPX	World Phenomenon number X
SPX	Shared Phenomenon number X
GX	Goal number X
DAX	Domain assumption number X
REX	Requirement number X
API	Application Programming Interface
MTTF	Mean Time To Failure
MTTR	Mean Time To Recovery
HTTP	Hypertext Transfer Protocol
XDR	eXternal Data Representation

1.3.2. Definitions

Users

User A generic User: can be either a guest, a participant, a student, or an educator.

Guest An Unregistered User who can register to CKB as either a student or an educator

Participant A generic registred user: can be either a student or an educator.

Student A registered user, who typically engage in code kata battles, enroll in tournaments, and interact with their peers to practice and improve their programming skills.

Educator A registered user, who initiates a coding tournament, defines its parameters, has the authority to grant other instructors the ability to create coding battles within that tournament, and can create battles himself.

GitHub definitions

GitHub A web-based platform that provides version control and collaboration features for software development. In CKB, GitHub is used to host repositories for code katas and facilitate collaboration among students.

Commit A commit in the context of CKB refers to a specific snapshot or version of code changes made by students. These commits are pushed to the GitHub repository's

main branch and are used to track progress and trigger automated scoring.

Fork To fork a GitHub repository means to create a personal copy of that repository under your GitHub account. In CKB, students fork the code kata's repository to work on their solutions independently.

Push Pushing code refers to the action of uploading code changes made by students to the main branch of the GitHub repository. When students push new commits, GitHub Actions are triggered to notify the CKB platform, and battle scores are updated accordingly.

GitHub Repository A digital space on the GitHub platform where code katas and related software projects are stored and managed. Each tournament's coding exercises have their own dedicated repositories.

GitHub Actions Automated workflows set up by students in their GitHub repositories to notify the CKB platform when they push new code commits to the main branch. This triggers the platform to update the team's battle scores and track their progress.

Main Branch The primary branch in the GitHub repository where students push their code changes. Updates to the main branch trigger GitHub Actions and score updates in CKB.

Platform definitions

Kata Japanese word describing detailed patterns of movements practiced either solo or in pairs. Katas are executed as a specified series of a variety of moves, with stepping and turning, while attempting to maintain perfect form.

Code Kata A brief textual description and a software project, including test cases and build automation scripts.

Code Kata Battle A coding exercise created by an instructor for a specific tournament, that includes code kata, number of members per group, registration and final submission deadlines, and eventual additional configurations.

1.3.3. Revision history

Relase Date	Description
22/12/23	First Release

1.3.4. Reference Documents

- Specification Document: Assignment RDD A.Y. 2023-2024
- Prof. Matteo Camilli's Software Engineering course
- IEEE Standard on Requirement Engineering (ISO/IEC/IEEE 29148)
- `https://docs.github.com/en/get-started/quickstart/github-glossary`
- `https://en.wikipedia.org/wiki/Karate_kata`
- Github Actions documentation: `https://docs.github.com/en/actions`

1.4. Document Structure

1. **Introduction**: A general introduction to the purpose and scope of the project, defining the main goals and phenomena. It also contains the definitions, acronyms, and abbreviations used throughout the document to clarify its reading, as well as the revision history and the reference documents to better point out how the document has been developed.
2. **Overall Description**: A general description of the product to be containing scenarios and further details on the shared phenomena given through class diagrams and state charts. There are also pieces of information about the users in order to clarify their needs. Finally, the domain assumptions as descriptive assertions assumed to hold in the environment of the project are included.
3. **Specific Requirements**: A definition of both functional and non-functional requirements, with the help of use case diagrams and sequence diagrams. There are also requirements on the external interfaces and the design constraints.
4. **Formal Analysis using Alloy**: The model designed with Alloy and the corresponding metamodel generated from the code, in order to formally describe the application and verify its correctness.
5. **Effort spent**: The amount of time spent by each team member for each section.

2 Overall Description

2.1. Product perspective

2.1.1. Scenarios

Ben, the new subscriber

Ben is a Computer Science Engineering student and he struggles with studying courses that involve programming. In college, he hears from his peers about the CKB platform, which allows students to participate in challenges on certain programming languages of their choice.

Consequently, he decides to register to the platform as a Student, filling out a form in which he specifies his personal information, which is necessary for the creation of his account. These include first name, last name, email, a password, a username and any preferred programming languages.

By subscribing and allowing notifications, the platform will send Ben notifications about battles involving the use of those chosen languages. Once the form is filled out, CKB sends a confirmation email and once confirmed, Ben is properly registered and he is ready to benefit from the offered services of the platform. Indeed, he will receive notifications by the platform for every new tournament added to the platform.

Alice, the passionate instructor

Alice is a computer science educator at a university, and she's looking for a platform to enhance her students' programming skills. After hearing about the CKB platform from her colleagues, she decides to sign up as an educator. Alice visits the CKB website and initiates the registration process. Therefore, she clicks on "Log in/Sign up", and after that she chooses to register as an Educator. She can then fill out a registration form with her personal information, including her first name, last name, email, and a secure password.

Thus, Alice provides her educational background and institution affiliation. This information is essential for verifying her eligibility to become an educator on the platform.

After submitting the registration form, the CKB platform processes her request. It verifies her credentials and ensures that she meets the criteria for educator registration. Once her educator status is confirmed, CKB sends Alice a confirmation email, and she becomes a registered educator on the platform.

Now, Alice can create tournaments and battles, define programming challenges, and mentor her students in improving their programming skills through code katas and coding exercises.

Hannah, the wannabe R programmer

Hannah, newly employed and facing an imminent project that demands R programming skills, seizes an opportunity on her day off to enhance her proficiency in this language. Eager to bridge the knowledge gap, she embarks on a mission to master R programming.

Having previously subscribed to the CKB platform, Hannah opts for this valuable resource to achieve her goal. She logs in and begins her search for R programming tournaments, with the aim of honing her capabilities.

Navigating through the CKB platform, Hannah diligently explores various available tournaments, seeking the one that aligns perfectly with her skill level and project requirements. Her determined pursuit eventually leads her to the ideal tournament.

If the submission deadline remains open, Hannah subscribes to this selected tournament, ensuring her active participation. This subscription guarantees that she receives timely notifications for every battle created within this specific tournament.

Once she has enrolled in, she decides to enter the first battle she comes across (which requires a minimum number of participants equal to one). However, if it is her first battle, the CKB platform shows Hannah a popup screen, where Hannah can decide whether:

- to view a tutorial on how GitHub works and how to set up the automatic workflow via GitHub Actions, or
- to skip it

In the same pop-up screen, there is also an option 'Do not display any more'. If Hannah ticks this option, then at the next battle the tutorial will no longer be displayed, otherwise yes. Once this is done, Hannah can actually start the battle.

Alex and the notifications

Alex, a computer science student, is an active participant in the CKB platform, having enrolled in a C-focused tournament. He's eager to improve his coding skills.

As a registered student in this tournament, Alex receives timely notifications for every new battle created within it. When a new battle is introduced, he's instantly alerted.

With a quick glance at the notification, Alex knows that there's a new coding challenge awaiting him. He accesses the CKB platform to review the details, understand the requirements, and check the submission deadline. Ready to tackle the challenge, he gets to work, all thanks to the platform's effective battle notifications.

Group Formation and Enrollment in CKB Battle

Alex, a diligent computer science student, is actively engaged on the CKB platform and currently enrolled in a Java-focused tournament. He's determined to enhance his coding skills, but this time, there's a twist in the tournament dynamics.

In this particular tournament, one of the created battles has minimum number of group participants higher than one. Therefore, this means that Alex can't go solo and must find teammates before he can enroll in this battle. However, the platform also offers a collaborative solution.

As the tournament's enrollment deadline approaches, the platform gives students like Alex the opportunity to declare their availability to form or join a group.

The platform diligently accumulates these group enlistments up to the maximum number of participants per group, ensuring efficient utilization of team resources. If, by the enrollment deadline, a group reaches the required minimum, and stays within the maximum, the platform promptly forms the group and notifies the students.

However, if the minimum number of students per group isn't reached before the deadline, the platform notifies the students of its incapability to form a group, and they are rejected from that battle.

This dynamic approach ensures that participants like Alex have the opportunity to collaborate and form groups while maintaining flexibility in their enrollment, ultimately enriching the learning experience in the CKB tournament.

Sara the team searcher

Sarah, a passionate programmer registered on the CKB platform. She's intrigued by a specialized coding battle focusing on algorithmic challenges. The battle permits solo participants but also encourages group formation if others express interest.

Eager to embrace the challenge and perhaps collaborate if the opportunity arises, Sarah enthusiastically enrolls herself and indicates her openness to joining a group. Despite her readiness to collaborate, the enrollment period concludes without any other participants expressing interest in forming a group for this particular battle.

As a result, the system acknowledges Sarah's enrollment but notifies her that due to the absence of additional participants, no group was formed. She receives confirmation that she'll be engaging in the battle as a solo participant, despite her initial willingness to team up.

Granting privileges

When creating a tournament on the CKB platform, John realizes he doesn't have enough time to create and manage all the battles in the tournament. For this reason, he writes to two of his coworkers at his workplace, Rita and Leonard, seeking their assistance. They accept his request, so John accesses the platform to continue the creation of the tournament he had in mind.

On the tournament's main screen, he clicks the 'Invite other educators...' button. After clicking this button, a popup window opens where he can specify the email addresses of the instructors to whom he wants to send the invitation. John then enters Rita and Leonard's email addresses and confirms.

At this point, CKB sends both an email notification and a platform notification to the mentioned colleagues. Thus, now both can create battles inside the tournament John created.

At this point, CKB sends both an email notification and a platform notification to the mentioned colleagues, containing John's request, which they can either accept or decline. If they accept, they will be able to create battles within the tournament; otherwise, they won't. Regardless of their choice, John will be notified of the decision made by the two invitees.

Claudia's new device

After the change to a new device, Claudia has to access again to all the software tools she normally uses, including CKB. Then, she logs in with her credentials by inserting her email and her password. She can now re-use all the functionalities of CKB like she used to do.

Julie, the busy professor

Julie works as a professor at ETH Zurich and in her spare time enjoys helping her students by assisting them with programming challenges. That's why she joined the CKB platform, where she creates weekly challenges.

During her lunch break, Julie starts writing a code kata, which she cannot finish because she has to rush to class. She then decides to save a draft of her work, which she will then finish as soon as she clears her schedule.

Once class is over, Julie logs into the platform and on the page showing her profile there is a section where all the drafts are put, hence she picks up the one she had set aside, being able to resume her work.

The Python tournament

John is a data analyst at the company *Product Analytics* and an educator on the CKB platform. Recently, at his workplace, he hired some young people, who have just graduated from university, therefore do not have much experience in Python matplotlib library. Hence, John decided to help them by suggesting them to take part in a tournament that he would create on CKB in those days, focusing precisely on data analysis using Python.

Consequently, John logs into the CKB platform, and as an educator, he creates a tournament. The creation simply involves entering a name for the tournament and a date by which to register. Once the confirmation button is clicked, a unique id is associated with the tournament, allowing those wishing to participate (in this case, by the company's freshmen) to search for it in the platform and be able to register by the deadline.

Sarah's Anticipation of New Tournaments on CKB

Sarah, a dedicated student pursuing her passion for coding, is an active member of the CodeKataBattle platform. She's constantly seeking opportunities to expand her programming skills, and CKB is her go-to resource.

Having previously subscribed to the platform expressing her preferences for specific programming languages, she's very eager to engage in coding challenges that align with her interests and expertise.

As she diligently checks her email and the platform, Sarah eagerly awaits notifications from CKB about newly uploaded tournaments that match her chosen programming languages. She's confident that the platform will keep her informed about the latest opportunities to participate and enhance her skills.

With every notification, Sarah is ready to explore these new tournaments. The CKB platform, with its tailored notifications, plays a crucial role in keeping her informed and motivated on her path to programming excellence.

Luke's Quest for a New Challenge on CKB

Luke, a curious and eager-to-learn student, finds himself at a moment of boredom, seeking a new intellectual challenge to conquer. As an active participant in the CodeKataBattle platform, he knows it's the perfect place to quench his thirst for knowledge.

With a few moments to spare, Luke decides to explore the notifications on the platform, he has been ignoring recently of the new battles inserted in the tournaments he joined.

As he browses through his notifications, Luke comes across a tantalizing invitation to a new battle in one of his enrolled tournaments. The description intrigues him, and the battle aligns perfectly with his aspirations for skill development.

Without hesitation, Luke enrolls in this new battle, ready to embark on a learning journey that promises to elevate his programming expertise.

Linda, the continuously improving student

Linda has been using the CKB platform for three months to enhance her programming skills in Swift and Ruby. After participating in numerous tournaments, she has realized she gained a deep understanding of Swift, so she's no longer interested in excercising this programming language.

Hence, she decides to log in to the CKB platform. After logging-in it, she enters her personal account area, where she can have access to all her account information, including her name, surname, email, username, password, and the programming languages she has chosen as her favorites.

Here, she can edit any of these fields and decides to update the list of languages by

removing Swift and adding JavaScript, HTML, and CSS, which she wishes to explore further. She then goes on and confirms these changes, as a consequence, the platform will no longer send her notifications about Swift tournaments but will focus on JavaScript, HTML, CSS, and, of course, Ruby, which she has retained.

Missed Opportunity at the Eleventh Hour

David, a diligent student with a passion for coding, is on the CKB platform, exploring the details of a captivating tournament he's been eyeing. The tournament seems tailor-made for his skill level and interests, and he's eager to join.

As he meticulously reviews the tournament information, David finds himself immersed in the description, reading about the challenges, rules, and deadlines. Enthusiastically, he decides that this is the tournament he wants to enter and takes a deep breath before reaching for the "Enroll" button.

However, as he clicks the "Enroll" button, an unexpected and disheartening message pops up on the screen: "Tournament Enrollment Deadline Expired." In that very instant, he realizes that he missed the opportunity to join the tournament he was so excited about.

Aurora creates a battle

Aurora is an educator for the CKB platform and has been granted permissions to create battles in a specific tournament on the platform. As a result, Aurora decides to create a battle.

To do so, from the main screen, she navigates to 'Your tournaments' section and selects the relevant tournament. On the tournament main screen she clicks the 'Create a new battle' button.

Once this button is clicked, a pop-up screen appears where she must specify the minimum and maximum number of participants per group allowed in that challenge, registration and final submission deadlines, any additional score configurations, and the code kata for the battle.

In defining the code kata, Aurora provides a brief description of the challenge to be tackled and includes software project in the programming language specified by the tournament for automatic testing of solutions uploaded by participating students.

After specifying all these parameters and clicking the 'Confirm' button, if everything goes well, the battle is created, and all registered participants are notified, both via email and

through the platform.

Micheal, step by step

Micheal has just finished a first version of his work, so he decides to commit the code to the main branch of the repository.

The commit triggers the platform thanks to the GitHub Actions mechanism. Thus, the CKB pulls the newly updated code, analyzes it, and runs the tests (defined by the educator who created the battle) on the corresponding executables to update the score of the team.

At this point, Micheal can view his updated battle rank.

The end of a tournament

Emily, an educator passionate about fostering programming skills, initiated a challenging tournament on the CKB platform. She curated battles, engaged students, and witnessed an array of remarkable coding accomplishments within her tournament.

As the battles within the tournament conclude and evaluations are completed, Emily meticulously reviews the outcomes. Satisfied with the students' progress and the success of the battles, she discerns that the tournament has served its purpose admirably.

In a logical decision, Emily determines that the tournament has reached its conclusion. Acknowledging that the learning objectives have been achieved and the students have showcased their programming prowess, she decides to close the tournament on the CKB platform.

Upon closure, the platform swiftly initiates the computation of the final tournament rankings. It diligently processes the accumulated scores, considering the students' performances across all battles. Once the final tournament rank is determined, the platform promptly notifies all participating students.

Olivia's team and their personal score

Olivia is participating with her team in a tournament. So far they have concluded two battles and are participating in a third. After committing, the team decides to view the battle rank for the current battle to see how positively their work has been evaluated.

After doing this, to get an overview, they check their tournament score and compare it with that of all the other participating teams. They notice that they are positioned in the middle of the leaderboard, so they decide to get busy and improve the code loaded in

the current battle. It's time to work hard!

Dr. Rodriguez the manual evaluator

Dr. Rodriguez, an educator passionate about nurturing coding talent, organizes a challenging battle on the CKB platform. In this battle, teams of students are tasked with solving intricate coding challenges.

As the submission deadline draws to a close, Dr. Rodriguez observes an impressive array of submissions from the participating teams. Recognizing the need for a deeper evaluation beyond automated scoring, Dr. Rodriguez decides to conduct manual evaluations to provide personalized feedback and assign a personal score to each team's work.

With the submission deadline expired, Dr. Rodriguez navigates through the CKB platform's interface designed for manual evaluation. Accessing the sources produced by each team, Dr. Rodriguez meticulously reviews their code, assessing its quality, creativity, and efficiency.

Employing expertise and experience, Dr. Rodriguez meticulously evaluates each team's submission, offering constructive feedback and assigning a personalized score based on the thorough assessment of their work.

Once the consolidation stage of manual evaluations concludes, the CKB platform updates the final battle ranks. All participating students eagerly anticipate the notification that signals the availability of the final battle rank.

The nosy Professor

Professor Smith, an enthusiastic educator and a catalyst for programming excellence, is deeply involved in the CodeKataBattle (CKB) platform. He not only creates tournaments but also actively monitors and evaluates the performance of students across various tournaments.

Taking a moment to assess the ongoing battles within the tournaments, Professor Smith navigates to the platform's to examine the tournament ranks, which measure students' performances relative to their peers in the context of specific tournaments.

Professor Smith doesn't limit his look through to just the tournaments he has created. He also delves into the tournaments initiated by his colleagues, navigating through the apposite competition rank interface, keen on understanding the progress and rankings achieved by students in different challenges across the platform.

Driven by a passion for the students success, Professor Smith follows the evolution of their current ranks within the battles. He marvels at the real-time updates, appreciating how the battle scores of each team are automatically adjusted as they perform new push actions on GitHub.

Witnessing the dynamic nature of the evolving ranks during the battles, Professor Smith gains valuable insights into students' performances, enabling him to offer guidance and support where needed.

Coding allies

Nicole and her best friend Alex had made a pact to learn programming before starting their journey into university with a focus on Computer Science. They were determined to sharpen their coding skills and decided to enroll in the renowned coding platform, CKB.

As they embarked on their coding adventures together, one ordinary morning, Nicole stumbled upon an intriguing tournament on the platform. Excitedly, she noticed that among the participants was none other than her close friend, Alex. So, Nicole decided to subscribe to the tournament and to enter the first battle created for the tournament, which required a minimum of two participants.

Nicole, with her eyes set on victory, quickly formed a team. In the interface where it was needed to enter a team, she clicked on the 'Invite Other Students...' button. A new popup screen appeared, offering Nicole the choice to either enter her friend's username or email. Intriguingly, she decided to go with the email option, thinking it would be a fun surprise for Alex.

Shortly after, a notification appeared in Alex's email box. Curious, he opened his email to find an invitation from Nicole to join the coding battle on CKB. The message explained the details of the tournament and gave him the option to accept or decline the invitation, all within the registration deadline.

Excitement brewed as Alex read the invitation. He wasted no time in accepting the challenge, knowing that together, they would make a formidable coding team.

2.1.2. Class Diagram

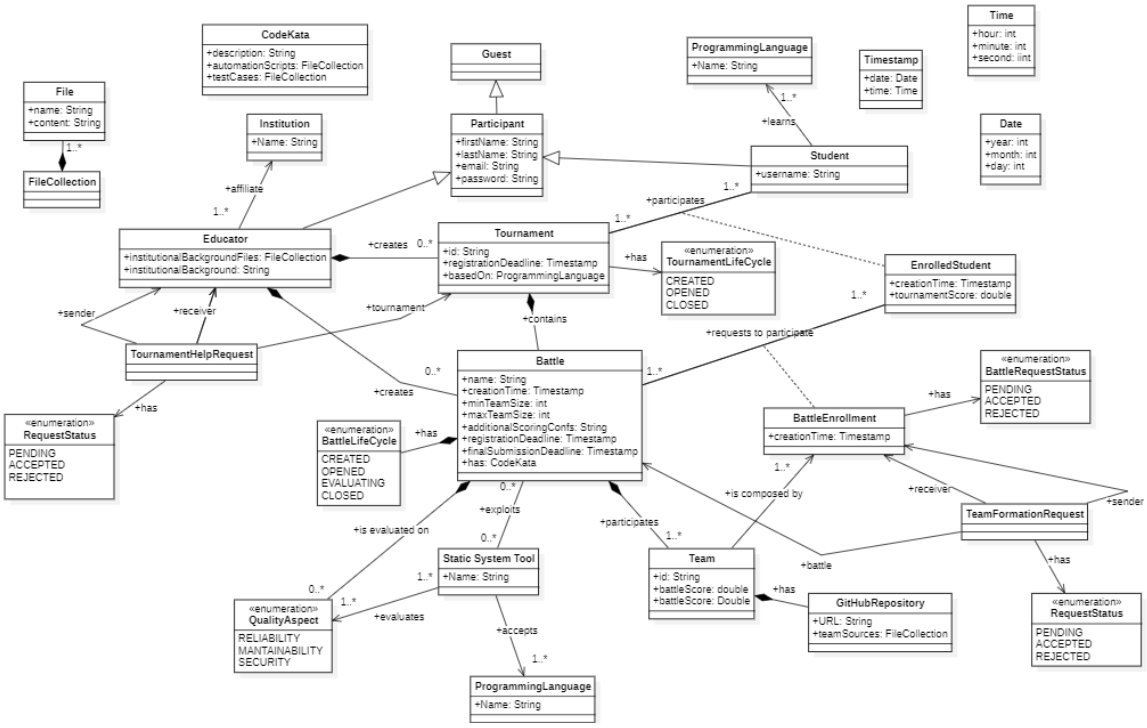

Figure 2.1: CKB CLass Diagram

2.1.3. State Chart Diagrams

Battle lifecycle

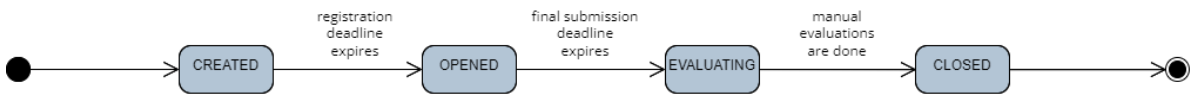

Figure 2.2: Battle lifecycle State Chart Diagram

The state chart diagram represents the lifecycle of a battle on the CKB platform. The process begins with the educator creating a battle, marking the system's entry into the *CREATED* state.

Subsequently, the platform moves to the *OPENED* state when the registration deadline expires, signifying the commencement of the coding challenge.

After the final submission deadline elapses, the system transitions to the *EVALUATION* state, where the educator personally evaluates the submitted projects.

Upon completion of the evaluation process, the system moves to the *CLOSED* state, indicating the conclusion of the battle.

Tournament lifecycle

Figure 2.3: Tournament lifecycle State Chart Diagram

The state chart diagram for tournaments in the CKB platform encapsulates the essential stages of a tournament's lifecycle.

The process begins with an educator that creates the tournament, triggering a transition to the *CREATED* state. During this phase, students can enroll.

Once the registration deadline expires, the system goes to the *OPENED* state. Here the battles are initiated, and the overall competition is in progress. The educator holds the authority to manage and control the tournament, making critical decisions, as asking for other educators' help.

Finally, when the educator decides to conclude the tournament, the system transitions to the *CLOSED* state. This marks the end of the tournament, preventing further student enrollments or participation.

Student Request Lifecycle

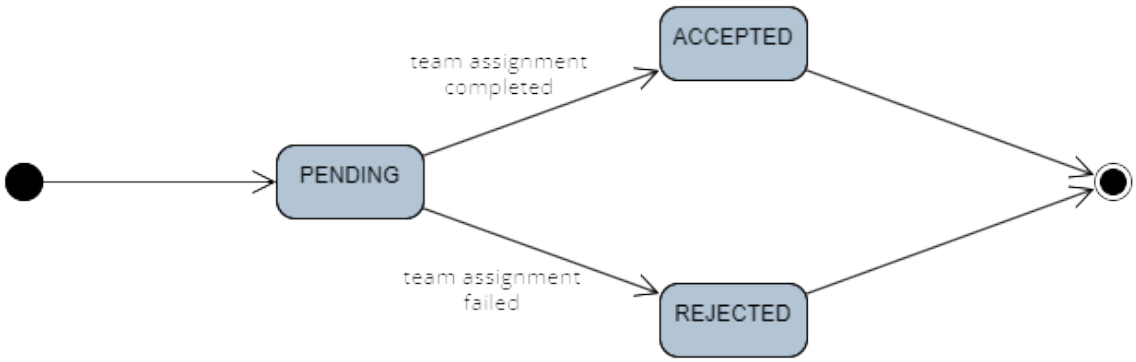

Figure 2.4: Student Request Lifecycle State Chart Diagram

The State Machine Diagram delineates the process of handling a student's request to participate in a battle on the CKB platform. This process begins when a student submits their participation request, moving the system into the *PENDING* state. In this state, the request is in a holding pattern, awaiting action until the registration deadline.

As the deadline approaches, the system initiates the team formation process. This crucial phase involves evaluating and grouping students into teams based on the `maxTeamSize` parameter.

The team pairing mechanism operates by grouping students into sets of `maxTeamSize`. When the number of *PENDING* students becomes insufficient to form a complete team, their fate is decided based on their count relative to `minTeamSize`. If the remaining number of *PENDING* students modulo `maxTeamSize` equals or exceeds `minTeamSize`, they are formed into an additional team, moving their status to *ACCEPTED*. This indicates their successful inclusion in a team.

However, if the remaining student count is below `minTeamSize`, their requests transition to the *REJECTED* state. This reflects their inability to participate in the battle due to an unsuitable team structure.

The *ACCEPTED* and *REJECTED* states mark the final stages in the student request lifecycle, symbolizing the end of the team formation process. The *ACCEPTED* state confirms the student's participation in a team, while the *REJECTED* state signifies their non-participation due to team size constraints.

Tournament Help Request Status

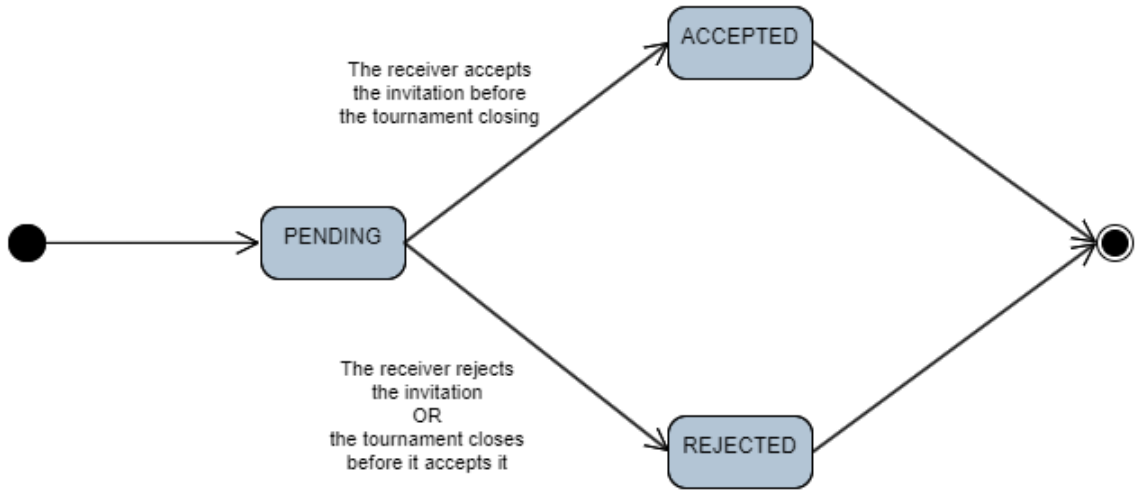

Figure 2.5: Tournament Help Request Lifecycle State Chart Diagram

The State Machine Diagram delineates the process of a tournament help request on the CKB platform, where an educator invites another educator to assist in a tournament they have created

The process initiates when an educator sends an invitation to another educator to join and assist in a specific tournament, particularly in creating battles within that tournament. At this point, the system is in the *PENDING* state. This state indicates that the invitation has been dispatched and is awaiting a response from the recipient educator.

If the invited educator accepts the invitation, the system transitions from the *PENDING* state to the *ACCEPTED* state. This transition signifies that the help request has been successfully accepted, and the invited educator is now part of the tournament team, ready to assist in battle creation and other related tasks.

Alternatively, if the invited educator either rejects the invitation or does not respond to it, and the tournament concludes before they accept, the system then shifts to the *REJECTED* state. This outcome denotes that the help request for the tournament has been unsuccessful, either due to explicit rejection or inaction on the part of the invited educator.

Team Creation Request Status

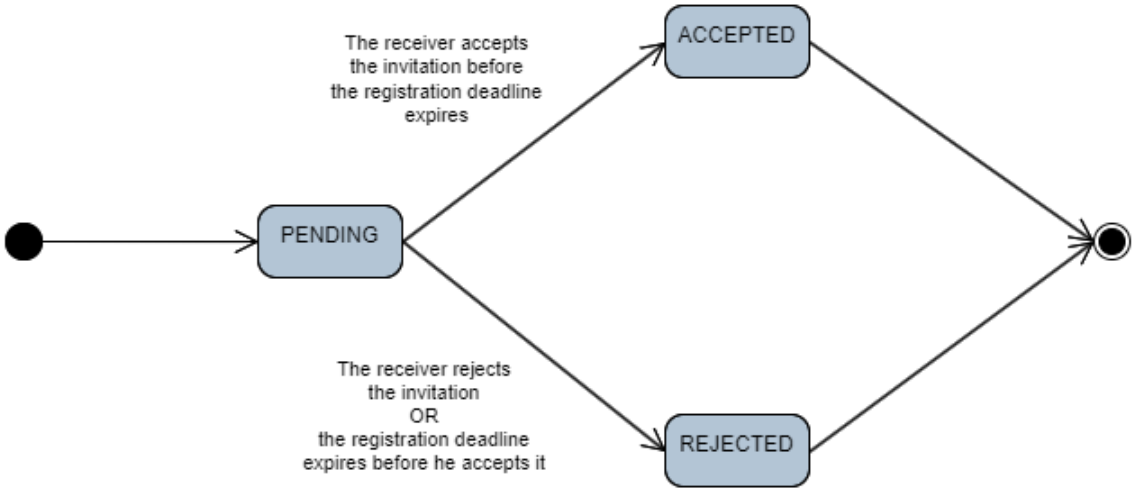

Figure 2.6: Team Creation Request Lifecycle State Chart Diagram

The State Machine Diagram delineates the process of team creation by invitation, where a student invites another to join their team for a battle.

The process begins when a student sends an invitation to another student to join their team. At this moment, the system enters the *PENDING* state. This state signifies that the invitation has been sent and is awaiting a response.

If the invited student accepts the invitation before the registration deadline, the system transitions from the *PENDING* state to the *ACCEPTED* state. This transition indicates that the team creation request has been successfully accepted, and the team is now officially formed for the battle.

Conversely, if the invited student rejects the invitation, or if the registration deadline expires before they accept it, the system then moves to the *REJECTED* state. In this scenario, the team creation request is considered unsuccessful, and the team is not formed.

2.2. Product functions

Create a tournament

The creation of a tournament is certainly one pivotal function of CKB platform. As such, it is guaranteed only to users registered in the System as educators. Through this function an Educator is allowed by the System of the function "Create a new tournament", which will bring to a dedicated interface where he can enlist all the characteristics of the new tournament. When creating a tournament, the System allows the Educator to enter a name and a deadline, before which interested students must subscribe. Additionally, the creator is allowed by the system, in the dedicated interface, to "Invite other educators..." through the related button. The subsequent pop-up windows that appears, where the system allow the educator to enter the email addresses of the educators he want to grant privileges to create new battle inside his tournament and go on.

The System, once the tournament is created, will send an e-mail and platform notification to all students subscribed to the CKB platform in order to give students an head-up on the newly tournament. A well as, sending an e-mail and platform notification to the educators granted the "creation" privilege.

Create a battle

Another key feature offered by the System is the creation of a battle. The System allows the Educators creators of a tournament and the ones, who were granted the privilege to do so, to create new battles. They can create a battle through the button "Create a new code kata". The System will then allow the Educator, through a pop-up screen to specify all the needed information about the the code kata: minimum and maximum number of participants per group, registration and final submission deadlines, additional score configurations and the code kata for the battle. Additionally, the System will allow these users to save a draft of the code kata they have been working on. To which they can come back whenever they desire, through the dedicated section in the user personal profile interface.

Inspect the list of all the available tournaments and respective rank

CKB platform allow all users subscribed to the platform to visualize all the ongoing competitions and the respective rank. This rank contains the personal tournament score of each student, that is, the sum of all battle scores received in that tournament. This is done by retrieving all information about the ongoing competitions, plus the competition rank, which the system will ensure to continuously keep up-to-date with respect to the ending of each battle in the tournament. In particular, the System allows the users to search for a particular tournament, in the interface containing this list. Additionally, a

user is allowed to select a tournament to explore the affiliated rank, as well as further information on it. Consequently, in the interface containing the ranking of the students, the system allow the users to look for a specific nickname, if interested. Therefore, the system guarantees a student (subscribed to the tournament) to easily check for his personal score.

Team formation

Team formation is certainly one pivotal function offered by the platform, fundamental to subscribe to a battle, with a lower bound of group participants greater than one. Furthermore, it is guaranteed only to users who had previously subscribed to the related tournament.

A user can either join a battle by himself or by invitation of a friend. In the first case, if the minimum number of participants is greater than one, than the System allow students to declare their availability to form or join a group. The platform diligently accumulates these group enlistments up to the maximum number of participants per group, ensuring efficient utilization of team resources. Therefore, periodically the system checks the number of requests, forming groups with max number of participants. Once the enrollment deadline is reached, if the number of pending students is sufficient to form a group, the platform promptly forms the group and notifies the students. However, if the minimum number of students per group isn't reached by the deadline, the platform notifies the students of its incapability to form a group, and they are rejected from that battle. In the second case, the Students can declare a group to code with, indeed, the System allows them to 'Invite other friends...' (cannot invite more than the maximum number). Hence, their request of joining a battle remains pending, until friends, who are enabled to accept/decline their invitation by the System, respond. The same logic as before applies, if the number of the invited friends, who have accepted the invitation, is bigger than the minimum needed, a group is formed, and all the participants are notified. Otherwise, the are notified of the rejection of their enrollment in the battle.

2.3. User characteristics

Guest

An Unregistered User who can access the platform and log in, or sign up, in case he never did so, until then he can't have access to any of the feature of the platform.

Participant

A registered User who has logged in the platform, and can access some features of the platform such as view the list of all tournaments available, and the score per each tournament.

Student

A registered user who have provided his personal data to the system, namely: first name, last name, email, a password, a username and any preferred programming. Therefore, he can join a tournament. Once in a tournament, he gain access to visualize the list of all the available battles in a tournament which he can join. He can decide to enter a battle by an invitation received by another, or/and form a team to participate with, if the minimum number of people required in a battle is greater than 1. A student in a battle, can work on the project of the code kata, push submits, and then can visualize the battle rank of his team.

Educator

A registered user who have provided his personal data to the system: first name, last name, email, a password, a username and any preferred programming, and in addiction also his educational background as well his institution affiliation. The educator, can create tournament, grant privileges to create battle in a tournament he created, close a tournament he created, create battles. In the consolidation stage the educator can provide a manual evaluation. Moreover, they can access the source code of the students to personally evaluate them, as well as the list of all the available tournaments, and respective ranks.

2.4. Assumptions, dependencies and constraints

2.4.1. Domain assumptions

DA1 Users must own a working device with an Internet connection to access and use the platform.

DA2 The personal information provided by users is assumed to be reliable and accurate.

DA3 Students are required to own a GitHub account, which is integral to the platform's functionality.

DA4 Students should have knowledge about GitHub Actions APIs, as this is crucial for the platform's operations.

DA5 Students are expected to make periodic pushes to their GitHub repository to allow the system to update the competition rank accurately.

DA6 The information given by GitHub, which the system relies on, is assumed to be accurate.

DA7 Each user is expected to create only one account on the platform to maintain integrity and prevent duplicity.

DA8 The system will have access to the necessary GitHub APIs for its functionalities.

DA9 GitHub Actions will trigger the system every time a push is executed by a user involved in a battle, ensuring real-time updates and interactions.

DA10 Participants will always receive email notifications, which is critical for timely communications and updates

DA11 The results retrieved from the queries to Static Analytic Tools are accurate

DA12 Static analysis tools used are always compatible with the programming languages of the code they are intended to analyze

3 Specific Requirements

3.1. External Interfaces Requirements

In this section will be given a detailed description regarding the hardware and software specifications and regarding all the communication interfaces of the CodeKataBattle Platform, considering both guests and participants (students and educators).

3.1.1. User Interfaces

The User Interface of CodeKataBattle is designed as a web browser application, and this choice is particularly advantageous over mobile devices for several reasons.

Firstly, a web application provides a larger and more immersive interface, which is crucial when dealing with complex coding exercises and projects.

Educators need the space and precision offered by a web browser to conduct in-depth manual evaluations of students' code, as coding requires a detailed examination of code structure and logic.

Additionally, web browsers offer a superior platform for visualizing leaderboards, tournament updates, and overall performance, benefiting both students and educators.

The nature of programming exercises demands a robust and feature-rich interface that web applications can offer, making it the ideal choice.

Mockup

Figure 3.1: User Interfaces

Figure 3.2: User Interfaces: Student Personal Area

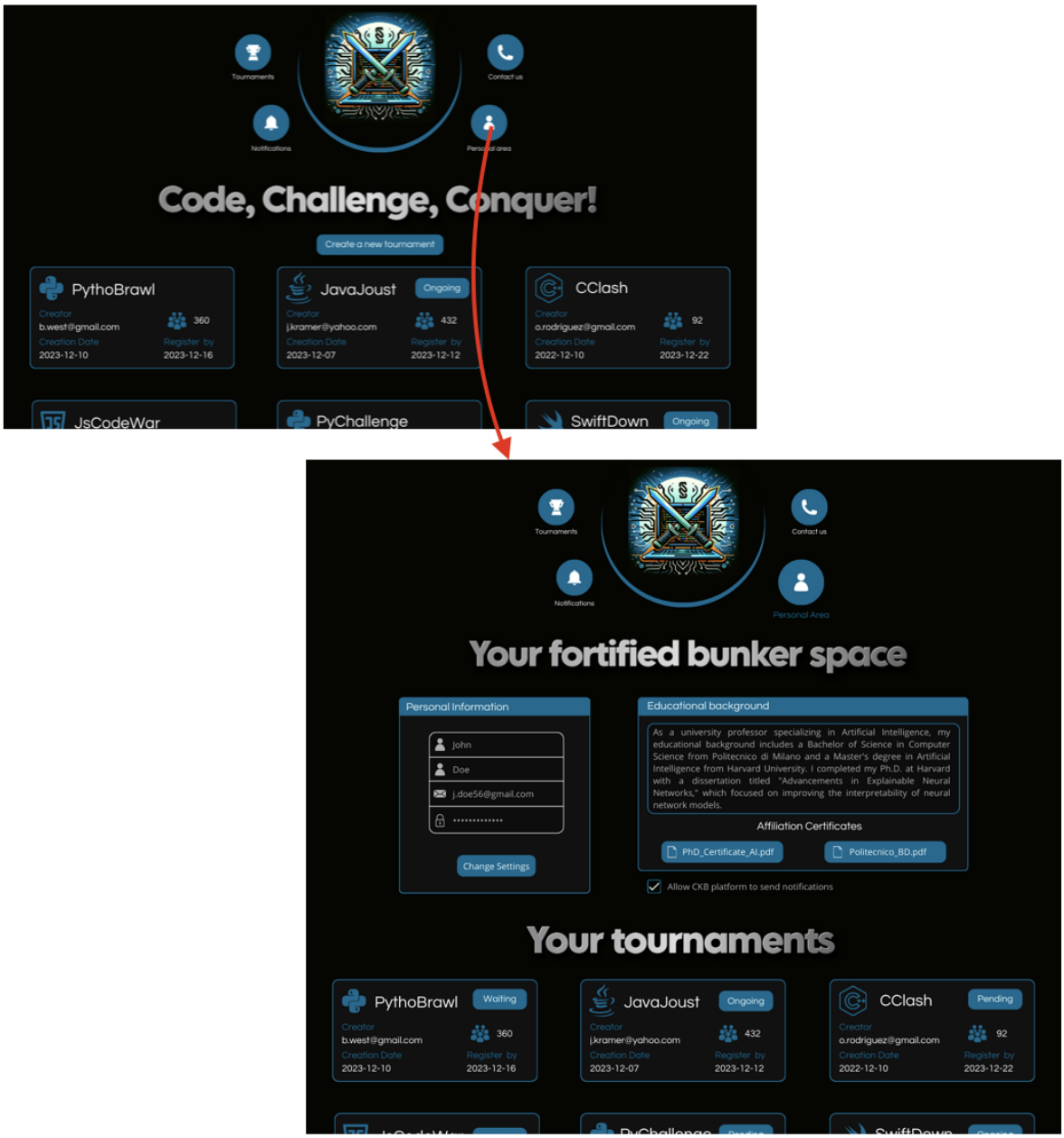

Figure 3.3: User Interfaces: Educator Personal Area

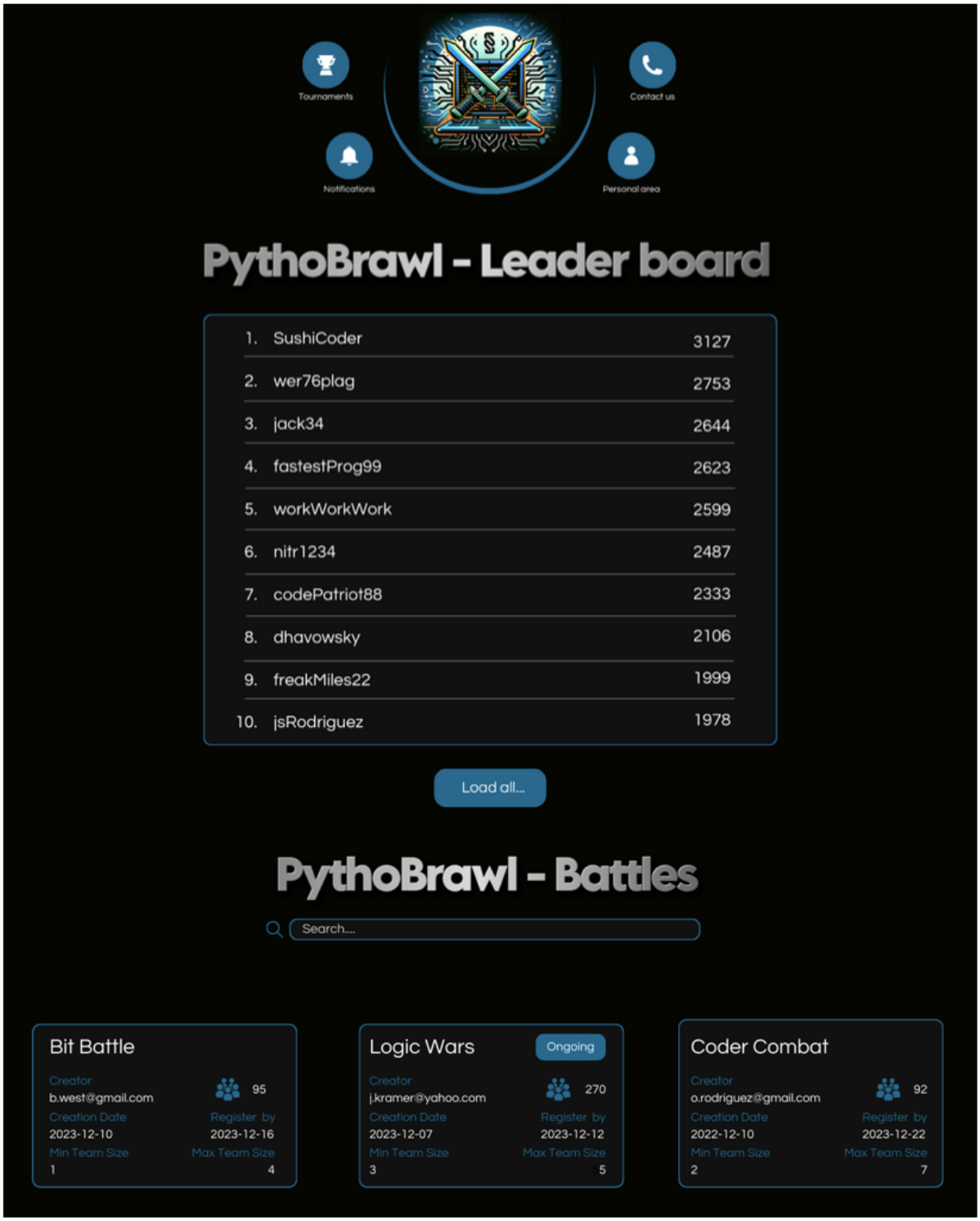

Figure 3.4: User Interfaces: Student Tournament Page

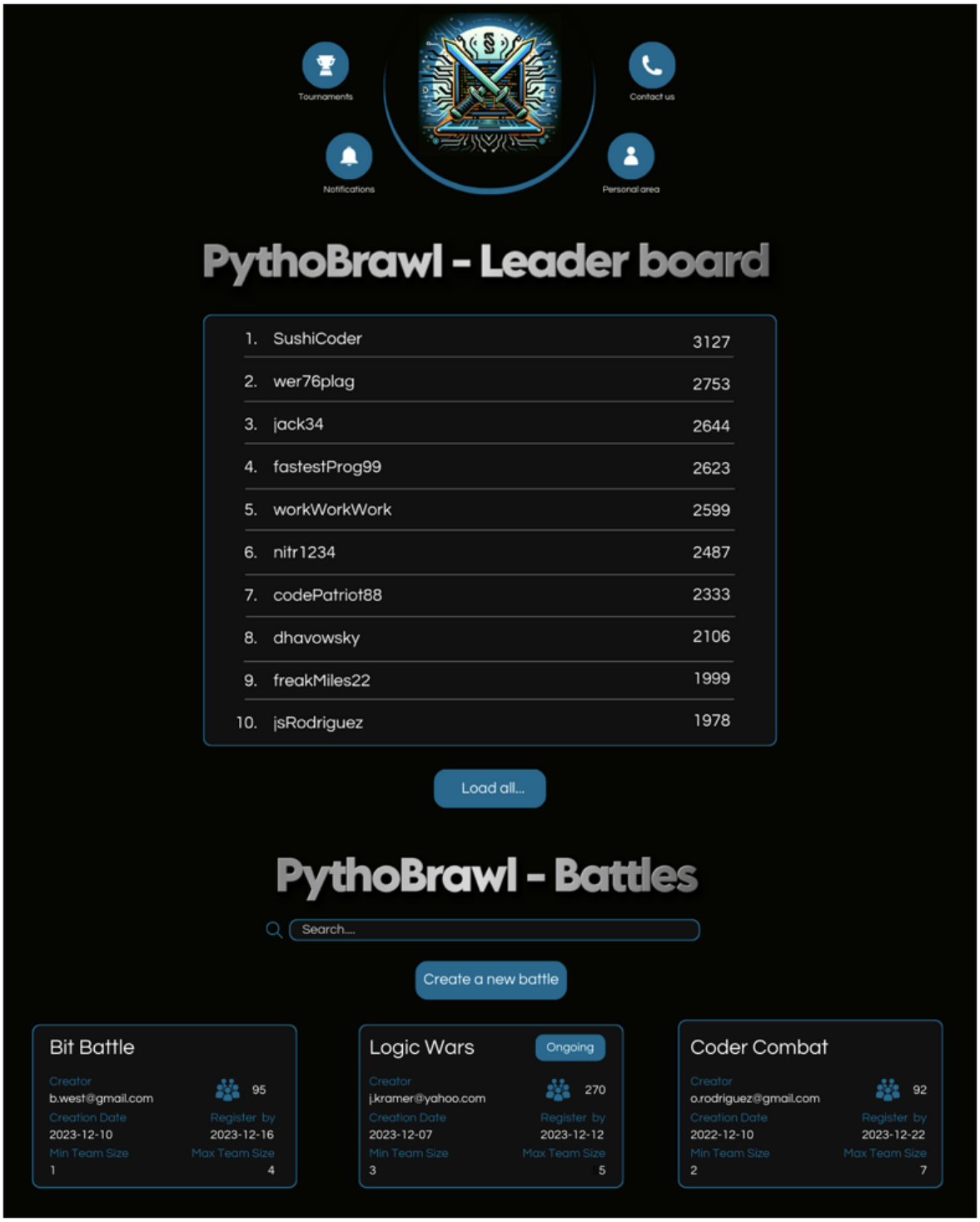

Figure 3.5: User Interfaces: Educator Tournament Page

3.1.2. Hardware Interfaces

The hardware interface for CodeKataBattle is primarily designed to be used on laptops or desktop computers, and this choice is directly aligned with the web-based user interface.

Indeed, the web app's feature-rich interface, is best experienced on a laptop or desktop computer due to their larger screens, precision input devices (keyboard and mouse), and better multitasking capabilities.

3.1.3. Communication Interfaces

GitHub The communication interface between GitHub and the CodeKataBattle platform is one of the main aspects of the system, enabling an automated and seamless interaction that drives the code kata battle process. This interaction begins when a student pushes their code changes to a GitHub repository, a trigger event defined within a GitHub Actions workflow. This workflow is specified in a YAML file within the repository, detailing the steps to be executed upon such an event. One of these steps includes making an API call to the CKB platform, designed to transmit relevant data for the scoring machanism. The data sent can include details of the push (like commit ID and timestamp), results of preliminary tests, and links to the updated code. When the CKB platform receives this API call, it triggers a series of automated processes on its end. These include validating the data, associating it with the correct battle and team, and initiating any further automated evaluations as per the battle's criteria. This streamlined communication interface ensures that each student's code submission is promptly and accurately reflected in the ongoing code kata battle, allowing for real-time updates and scoring.

Static Analysis Tools The integration of static analysis tools in the CKB platform forms another essential communication interface, focusing on the automated evaluation of code quality metrics. Once the student's code is pushed and processed through the initial GitHub Actions workflow, the CKB platform receives it and takes over for further analysis. At this stage, the platform communicates with various static analysis tools to evaluate the code based on predefined quality metrics. These metrics, which include maintainability, security, and reliability, are selected by the educator during the battle setup. CKB sends the code to these static analysis tools and receives a detailed report on these metrics. This report is then processed to translate the findings into quantifiable and normalized scores as part of the team's overall performance evaluation.

3.2. Functional Requirements

3.2.1. Use cases diagrams

Student

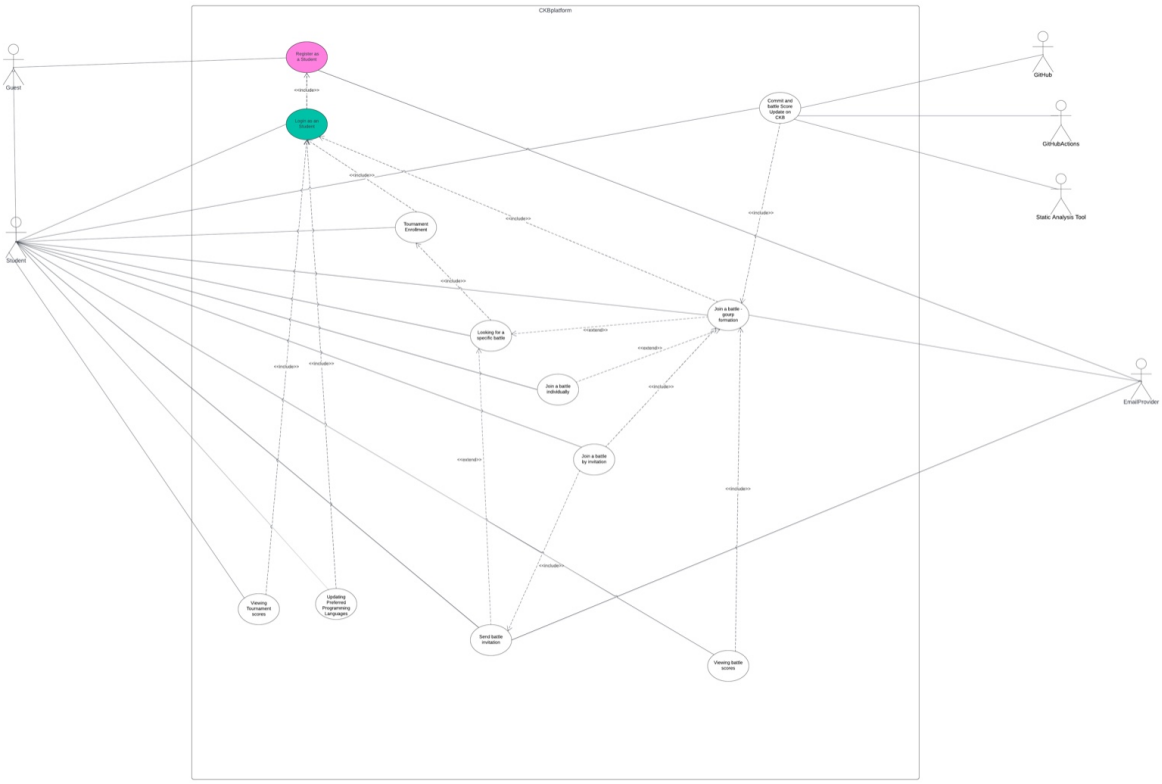

Figure 3.6: Student Use Case Diagram

Educator

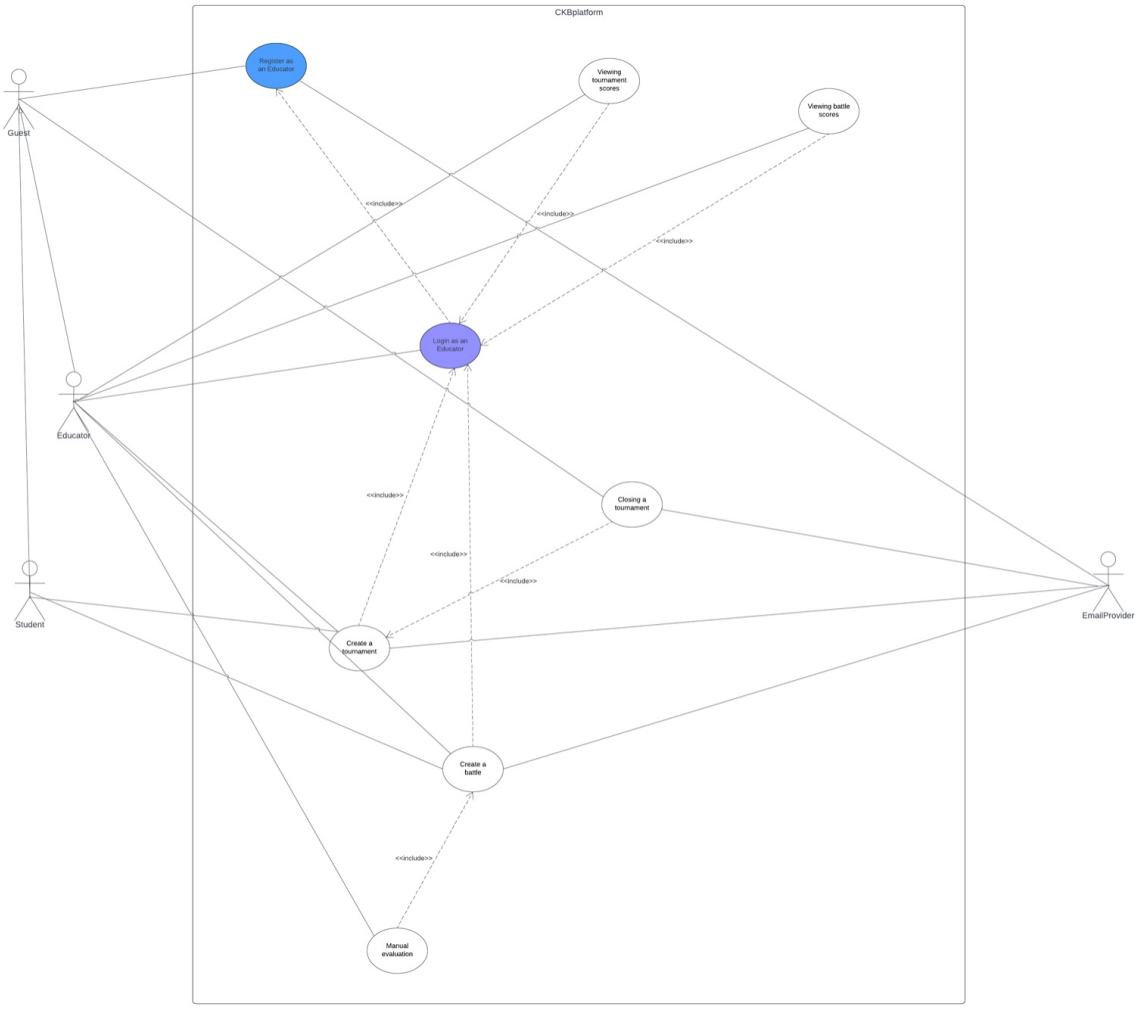

Figure 3.7: Educator Use Case Diagram

3.2.2. Use cases

Name	Register as a Student
Actors	Guest E-mail provider
Entry Condition	Guest is on the homepage and wants to sign up
Event Flow	1. The Guest clicks on the "Log in/Sign up" option; 2. The System opens a popup window with two options "Log in" or "Sign up" 3. The Student selects the option "Sign up" 4. The System asks which type of account the Guest wants to create 5. The Guest selects the button corresponding to the creation of a Student account 6. The Guest inserts his personal data in a form: his first and last name, an email, and a password, a username and any preferred programming languages; 7. Guest checks the "allow CKB platform to send notifications" checkbox; 8. The Guest clicks on the "Sign-up" button; 9. The System saves the Guest's information; 10. The System sends a request to the E-mail provider to send an e-mail to the provided e-mail address; 11. The E-mail provider sends the confirmation e-mail; 12. Guest checks the email and clicks on the link to confirm his registration;
Exit Condition	Guest has successfully created a Student account
Exceptions	• The email does not exist • The email is already registered In these situations, the System shows an error message saying that the email is invalid. • The Guest inputs a password that doesn't respect security constraints (e.g., minimum number of characters, presence of numbers, etc.) • If not all required fields are filled in, the System shows an error message saying that some fields are missing • If the "Allow CKB platform to send notifications" checkbox is not checked, the System shows an error message saying that the check is mandatory In all the above cases, the System restarts from point 3.
Special Requirements	• The pop-up must appear after no more than 2 seconds

Table 3.1: Use Case 1 - Register as a Student

Name	Register as an Educator
Actors	Guest E-mail provider
Entry Condition	Guest is on the homepage and wants to sign up
Event Flow	1. The Guest clicks on the "Log in/Sign up" option; 2. The System opens a popup window with two options "Log in" or "Sign up" 3. The Educator selects the option "Sign up" 4. The System asks which type of account the Guest wants to create 5. The Guest selects the button corresponding to the creation of an Educator account 6. The Guest inserts his personal data in a form: his first and last name, an email, and a secure password. 7. The Guest inserts her educational background and istitutional affiliation 8. Guest checks the "Allow CKB platform to send notifications" checkbox; 9. The Guest clicks on the "Sign-up" button; 10. The System verifies the credentials inserted, ensuring the data provided meet the criteria for an educator account; 11. The System saves the Guest's information; 12. The System sends a request to the E-mail provider to send an e-mail to the provided e-mail address; 13. The E-mail provider sends the confirmation e-mail; 14. Guest checks the email and clicks on the link to confirm his registration;
Exit Condition	Guest has successfully created an Educator account
Exceptions	• The email does not exist • The email is already registered In these situations, the System shows an error message saying that the email is invalid. • The Guest inputs a password that doesn't respect security constraints • If not all required fields are filled in, the System shows an error message saying that some fields are missing • If the "allow CKB platform to send notifications" checkbox is not checked, the System shows an error message saying that the check is mandatory • Information about educational background and institutional affiliation is incorrect, he System shows an error message saying that the this information is mandatory to register as an Educator. In all the above cases, the System restarts from point 3
Special Requirements	• The pop-up must appear after no more than 2 seconds

Table 3.2: Use Case 2 - Register as an Educator

Name	Login as a Student
Actors	Student
Entry Condition	The Student has already successfully registered, he is on the homepage and wants to log in.
Event Flow	1. The Student selects the "Log in/Sign up" option 2. The System opens a popup window with two options "Log in" or "Sign up" 3. The Student selects the option "Sign up" 4. The Student inserts his credentials, that is, his email or username and password; 5. The Student clicks the "Login" button; 6. The System checks the Student's credentials;
Exit Condition	The Student has successfully signed in
Exceptions	• The email is not associated to any registered account • The username is not associated to any registered account • The password is incorrect In these situations, the System shows an error message saying that the credentials are invalid
Special Requirements	• The pop-up must appear after no more than 2 seconds

Table 3.3: Use Case 3 - Login as a Student

Name	Login as an Educator
Actors	Educator
Entry Condition	The Educator has already successfully registered, he is on the homepage and wants to sign in.
Event Flow	1. The Educator selects the "Log in/Sign up" option 2. The System opens a popup window with two options "Log in" or "Sign up" 3. The Educator selects the option "Sign up" 4. The Educator inserts his credentials, that is, his email and password; 5. The Educator clicks the "Login" button; 6. The System checks the Educator's credentials;
Exit Condition	The Educator has successfully signed in
Exceptions	• The email is not associated to any registered account • The password is incorrect In these situations, the System shows an error message saying that the credentials are invalid
Special Requirements	• The pop-up must appear after no more than 2 seconds

Table 3.4: Use Case 4 - Login as an Educator

Name	Create a tournament
Actors	Educator Student E-mail Provider
Entry Condition	The Educator has successfully logged in to the System, is on the main page and wants to create a tournament
Event Flow	1. The Educator selects the "Create a new tournament" option in the homepage; 2. The System shows a form to be filled in with the information of the new tournament; 3. The Educator enters a name and a deadline; 4. The Educator selects the button "Invite other educators..."; 5. The System displays a pop-up window where the Educator can enter the email of other educators, granting them privileges to create battles within his tournament 6. The System displays a pop-up window where the Educator can review the tournament details before final submission 7. The Educator clicks the "Create tournament" button 8. The System saves information about the new tournament 9. The System sends out platform notifications to the invited educators 10. The System sends a request to the E-mail provider to send an e-mail to the provided e-mail address; 11. The E-mail provider sends e-mails to the invited Educators; 12. The System sends out notifications to all Students subscribed to the platform, to inform about the new available tournament 13. The System sends a request to the E-mail provider to send e-mails to all subscribed Students to the platform; 14. The E-mail provider sends e-mails to the provided e-mail addresses;
Exit Condition	The tournament has been successfully created and is now available on the platform All the students of the platform have been notified about the new tournament
Exceptions	• The educator does not invite any educator to the tournament In this case, step 5 and 9, 10, 11 are skipped
Special Requirements	• The notifications must be sent within 5 minutes from the creation of the tournament • The pop-up must appear after no more than 2 seconds

Table 3.5: Use Case 5 - Create a tournament

Name	Tournament Enrollment
Actors	Student
Entry Condition	The Student is logged in the CKB platform
Event Flow	1. The Student goes on the 'Available tournaments' page. 2. The System retrieves all the tournaments that with the programming language matching to the ones the Student has selected as preferred 3. The Student navigates through the list of the available tournaments or search for a specific through the search bar. 4. The Student reviews the tournament details, including battles, rules, and deadlines. 5. The Student clicks the "Enroll" button.
Exit Condition	The Student has successfully subscribed to a tournament
Exceptions	• The input the Participant inserts does not meet any available tournament In this case the System does not load any tournament in the page, showing an empty list • The Student press the "Enroll" button after the subscription deadline. In this case, the Student cannot enroll, and the System brings him back to the page with the list of the available tournaments • No tournaments are retrieved by the System, for the preferred inserted programming languages. In this case, the System displays an alert stating no tournaments are available for the preferred languages
Special Requirements	• The alert must be shown within 1 second • The page of the available tournaments, after one of these exceptions, is reloaded after no more than 5 seconds

Table 3.6: Use Case 6 - Tournament Enrollment

Name	Looking for a specific battle
Actors	Student
Entry Condition	Student has already logged into the System, has already subscribed to a tournament and wants to look for a specific battle. The Student is on the specific page of the tournament he subscribed to.
Event Flow	1. The Student is on the main page of the tournament he subscribed to and selects the search bar 2. The Student inserts keywords for the battle he is looking for 3. The System retrieves all the information regarding all battles that matches the information inserted by the Student 4. The System loads a list containing all the battles meeting the inserted criteria 5. The Student selects the battle he is interested in 6. The System retrieves all information about that battle
Exit Condition	The System loads the dedicated page of that specific battle
Exceptions	• The input inserted by the Student in the search bar does not meet any battle inside the System In this case the System shows a message under the search bar, suggesting to re-enter information, because the one inserted does not match any of the battles inside that tournament
Special Requirements	• The message must be shown within 1 second • The page of the specified battle must be loaded within 2 seconds from the selection of the User

Table 3.7: Use Case 7 - Looking for a specific battle

Name	Join a battle individually
Actors	Student
Entry Condition	The Student has already logged in the platform, and has subscribed to a tournament, and has already selected the battle he is interested in (UC7)
Event Flow	1. The Student goes through the code kata (description and software project) 2. The Student checks the availability to still enroll in the battle (deadline has not expired yet) 3. The Student checks the minimum number of participants needed to join a battle 4. The Student selects the "Join battle individually" option (in this case available, because the minimum is greater or equal than 1) 5. The System saves this information, and successfully enrolls the Students to that battle
Exit Condition	The Student has successfully joined a battle on his own
Exceptions	• The deadline has expired • The minimum number of students per group must be at least 2 In these cases, the System displays an alert stating the battle cannot be joined and the motif.
Special Requirements	• The alert must be shown within 1 second • The popup screen must be displayed after no more than 2 seconds

Table 3.8: Use Case 8 - Join a battle individually

Name	Join a battle - group formation
Actors	Student E-mail Provider
Entry Condition	The Student has already logged in the platform, and has subscribed to a tournament, and has already selected the battle he is interested in (UC7)
Event Flow	1. The Student goes through the code kata (description and software project) 2. The Student checks the availability to still enroll in the battle (deadline has not expired yet) 3. The Student checks the minimum number of participants needed to join a battle 4. The student select the option "Join a battle" 5. The Student selects the option "Available to join a group" 6. The System saves this information 7. The Student's enrollment to the battle remains pending based on whether a group will be created 8. The System creates groups among the available students once the deadline has expired 9. The System saves the created teams 10. The System sends out notifications to all Students who were pending/waiting, to inform whether they joined the battle 11. The System sends a request to the E-mail provider to send e-mails; 12. The E-mail provider sends e-mails to the provided e-mail addresses;
Exit Condition	The Student has successfully joined a battle with the created group
Exceptions	• The deadline has expired In this case, the System will create a group, if the min-max criteria are met, and will follow the same flow of events from step 8. Otherwise, if the minimum number per group is not met, the System notifies to the students whose enrolment was pending of the failure of their subscription to the battle • The System is unable to create a group (min criteria is not met) In this case, the System sends a notification to all the students who had a pending enrollment request, rejecting their enrollment • By the deadline, there is only one student enlisted, but the minimum number of students is one In this case, the system will form a group composed by him alone (following step 8 and 9)
Special Requirements	• Every kind of notification must be sent by the System within 5 minutes from the formation of the group, the expiration of the deadline or the rejection of the Student's enrollment request

Table 3.9: Use Case 9 - Join a battle - group formation

Name	Send battle invitation
Actors	Student E-mail Provider
Entry Condition	The Student has already logged in the platform, and has subscribed to a tournament, and has already selected the battle he is interested in (UC7)
Event Flow	1. The Student go through the code kata (description and software project) 2. The Student checks the availability to still enroll in the battle (deadline has not expired yet) 3. The Student checks the minimum number of participants needed to join a battle 4. The Student selects the "Join battle" option 5. The Student selects the option 'Invite Other Students...'. 6. The System loads a popup screen allowing the Student to enter the email or the username of other Students 7. The Student enters the emails/usernames of students to invite 8. After the selection, he selects the "Send invitation" button 9. The Student closes the popup windows 10. The System saves the requests of the Student as pending waiting on the response of others 11. The System brings the Student back to the battle enrolment page
Exit Condition	The Student successfully invites other Students to join him in the battle
Exceptions	• The email/username entered do not correspond to any existing account of the CKB platform • The invited student must have already subscribed to that tournament In these case the System will present an alert, asking the Student enter an email/username of a suitable user
Special Requirements	• The invitation notification must be sent by the System within 5 minutes from the request • The popup screen must be loaded within 2 seconds from the request • The enrolment page must be reloaded within 5 seconds, after the invitations have been sent out

Table 3.10: Use Case 10 - Send battle invitation

Name	Join a battle by invitation
Actors	Student
Entry Condition	The Student has already logged in the platform, and has subscribed to a tournament. The Student is on the main page
Event Flow	1. The Student receives a platform 2. The Student navigates to part of the page dedicated to notifications 3. The System retrieves all the notifications related to that user 4. The System loads the notification page 5. the Student opens the battle invitation notification to see further information 6. The System retrieves all the information needed to correctly represent the notification in a dedicated page, plus information about the battle 7. The System asks the Student to either accept or decline the request 8. The Student selects the option "Join battle" 9. The System saves the Student choice
Exit Condition	The Student successfully accepted an invite and his enrolment remains "pending/waiting" The Student is taken to the battle main page
Exceptions	
Special Requirements	• The alert must be loaded within 5 seconds from the click of the "Join battle" button • The popup screen must be loaded within 2 seconds from the request • The enrolment page must be reloaded within 5 seconds, after the invitation is accepted and criteria are met

Table 3.11: Use Case 11 - Join a battle by invitation

Name	Updating Preferred Programming Languages on CKB
Actors	Student
Entry Condition	Student is already logged in and is on his main page. Student wants to change his preferred programming languages.
Event Flow	1. The student navigates to his personal account area. 2. The System retrieves student's personal information. 3. The System loads student's personal area. 4. The student clicks the 'Change settings' button in his personal page 5. The System makes the student able to modify the editable settings. 6. The student updates his preferred programming languages, removing one or more of them and adding others. 7. The student confirms his changes through the 'Save changes' button. 8. The System processes student's choices and updates his account settings.
Exit Condition	Student's preferred programming languages are successfully updated in the CodeKataBattle platform and the personal area is correctly reloaded.
Exceptions	• The student tries to remove all programming languages. In this case the System shows him an alert where it specifies to select at least one language as a favorite to receive relevant notifications. After that, it refreshes the page and goes back to step 3.
Special Requirements	• The system must load student's personal area within 1 second. • The system shows the personal area as editable within 1 second.

Table 3.12: Use Case 12 - Updating Preferred Programming Languages

Name	Creating a battle on CKB
Actors	Educator E-mail Provider Student
Entry Condition	The Educator is logged into her CKB educator account. Educator has permissions to create battles in a specific tournament. Educator is on the main page of CKB platform.
Event Flow	1. The Educator navigates on "Your tournaments" section. 2. The systems retrieves all the tournaments the educator is involved in and displays them as a list. 3. The educator selects one of the available tournaments or search for its name. 4. The System retrieves all the information about that tournament, and loads the dedicated page. 5. On the tournament main screen, the Educator clicks the 'Create a new battle' button. 6. The system shows a pop-up screen, prompting the Educator to specify some information: • Minimum and maximum number of participants per group. • Registration and final submission deadlines. • Additional score configurations (if any). • The code kata for the battle, including a brief description and software project for automatic testing. 7. The Educator fills in all the required parameters and details for the new battle. 8. The Educator clicks the 'Confirm' button to create the battle. 9. The system processes the educator's choices. 10. The System saves the information of the new battle. 11. The System notifies all the students of that specific tournament of the creation of that new battle 12. The System sends a request to the E-mail provider to send e-mails to all subscribed Students to that tournament; 13. The E-mail provider sends e-mails to the provided e-mail addresses;
Exit Condition	The new Code Kata Battle is successfully created in the specified tournament. All registered participants are notified of the new battle.
Exceptions	• The input the Participant inserts does not meet any available tournament In this case the System does not load any tournament in the page, showing an empty list • The educator does not specify mandatory parameters. In this case the system prompts the educator to complete all required fields before confirming the creation. • The educator specifies weird parameters (e.g., setting the minimum number of participants to 999). In this case the system prompts the educator to complete the fields with "normal" values. • The educator decides to save a draft of the battle he is creating through the button "Save draft" available on the pop-up screen where all the parameters of the battle are specified In this case, the system stores the incomplete battle, and gives the educator the possibility to complete it, through the dedicated section ("Drafts") in the Educator personal profile page. Additionally, all the steps from 7 and/or 8 are skipped, bringing the Educator back to his personal account page.
Special Requirements	• 'Your tournaments' page must be loaded within 2 seconds. • The tournament main screen must be loaded within 2 seconds. • The notification to the Students must be send within 5 minutes from the creation of the battle.

Table 3.13: Use Case 13 - Creating a battle on CKB

Name	Commit and battle Score Update on CKB
Actors	Student GitHub GitHubActions
Entry Condition	Student is already logged in. Student is enrolled in a tournament. Student is enrolled in a battle with his team. Student and his team have completed a first version of their work. Student is on the GitHub page related to the battle.
Event Flow	1. The student, using GitHub, commits his code changes to the main branch of the repository. 2. The commit triggers GitHub Actions, configured for the repository. 3. GitHub Actions notifies the System of the code update. 4. The system pulls the newly updated code from the repository. 5. The system analyzes the code and runs the tests defined by the educator who created the battle. 6. The system updates the score of the student's team based on the test results. 7. The student and his team view their updated battle rank on the battle page.
Exit Condition	The commit is successfully processed by GitHub Actions and CKB. Student's team score is updated based on the test results. The student can view his updated battle rank on the battle page.
Exceptions	
Special requirements	• The updated score must be available within 5 minutes.

Table 3.14: Use Case 14 - Commit and battle Score Update on CKB

Name	Closing a Tournament on CKB
Actors	Educator E-mail Provider Student
Entry Condition	Educator is already logged in on the platform Educator has organized a tournament on the CodeKataBattle platform. The educator is on the tournament page.
Event Flow	1. The educator clicks on the 'Close tournament' button. 2. The system shows a pop-up message that acts: "Are you sure?", with two possible options: • Close tournament. • Cancel. 3. The Educator chooses "Close tournament". 4. The System computes the final tournament rankings. 5. The System notifies all participating students of the results 6. The System sends a request to the E-mail provider to send e-mails to all Students in that tournament; 7. The E-mail provider sends e-mails to the provided e-mail addresses;
Exit Condition	The tournament is successfully closed on the CKB platform. The platform computes and finalizes the tournament rankings. All participating students receive notifications regarding the final tournament results.
Exceptions	• There are ongoing battles when the Educator tries to close the tournament. In this case, the System shows an error message bringing the Educator back to the tournament page
Special requirements	• The System must send notifications with final tournament ranks within 5 minutes from the closure of the tournament itself. • The System must show the error message within 2 seconds.

Table 3.15: Use Case 15 - Closing a Tournament on CKB

Name	Viewing battle score on CKB
Actors	Participant
Entry Condition	The Participant is already logged in on the platform The Participant is involved in a tournament on the CKB platform. The Participant is on the main page of the Platform
Event Flow	1. The Participant navigates to 'Your tournaments' section in his personal area. 2. The System retrieves all the tournaments the Participant is involved in and displays them as a list. 3. The Participant selects the tournament of interest or search for its name. 4. The System retrieves all the information about that tournament, and loads the dedicated page. 5. On the tournament main screen, the Participant goes to the 'Ongoing Battles' section and search for the specific battle. 6. The System retrieves all the information about that battle, including the leader board with the battle score of each team, and loads the dedicated page.
Exit Condition	The Participant successfully views the battle leader board.
Exceptions	• The input the Participant inserts does not meet any available tournament In this case the System does not load any tournament in the page, showing an empty list
	• There are no ongoing battles in the selected tournament In this case the system, shows an empty 'Ongoing Battles' section. Therefore, step 6 is skipped, and no leader board can be viewed
Special Requirements	• 'Your tournaments' page must be loaded within 2 seconds. • The tournament main screen must be loaded within 2 seconds. • The battle main screen must be loaded within 2 seconds.

Table 3.16: Use Case 16 - Viewing battle scores

Name	Viewing tournament score on CKB
Actors	Participant
Entry Condition	The Participant is already logged in on the platform
Event Flow	1. The Participant clicks on the 'Available tournaments' section in the header. 2. The System retrieves all the tournaments and displays them as a list. 3. The Participant can either select the tournament of interest or search for a specific tournament in the search bar. 4. The System retrieves all the information about that tournament, including the leader board with the tournament score of each participant, and loads the dedicated page.
Exit Condition	The Participant successfully views the tournament ranks.
Exceptions	• The input the Participant inserts does not meet any available tournament In this case the System does not load any tournament in the page, showing an empty list
Special Requirements	• 'Available tournaments' page must be loaded within 2 seconds. • The tournament main screen must be loaded within 2 seconds.

Table 3.17: Use Case 17 - Viewing tournament scores on CKB

Name	Manual Evaluation on CKB
Actors	Educator
Entry Condition	The Educator is already logged in on the platform. The educator has organized a battle in a tournament. The submission deadline for the battle has expired. The Educator has received a notification stating the closing of his battle. The Educator is on the main page.
Event Flow	1. The Educator navigates to the notification section. 2. The System retrieves all the pending notifications referred to the Educator. 3. The System enlists all the pending notifications. 4. The Educator reads the notification about the closing of the battle. 5. The Educator can choose between two different buttons related to that battle: • Conduct manual evaluation. • Close the battle. 6. The Educator chooses the option "Conduct manual evaluation". 7. The System retrieves all the uploaded projects related to the just closed battle. 8. The System loads a dedicated page where the Educator can evaluate the projects. 9. Navigating through the manual evaluation page, the educator accesses the sources produced by each team. 10. The educator reviews the code submitted by each team. 11. The educator assigns a personalized score to each team based on the thorough assessment of their work. 12. Once the manual evaluations are complete, the educator clicks the 'Confirm' button to save the corrections. 13. The System saves the additional scoring, and computes the definite score 14. The system updates the final battle ranks. 15. The system sends a notification to all the enrolled students, signaling the availability of the final battle rank.
Exit Condition	Educator has successfully conducts manual evaluations for the participating teams. The CKB platform updates the final battle ranks based on the manual evaluations. Participating students receive notifications regarding the availability of the final battle rank.
Exceptions	
Special requirements	• The system must retrieve all the uploaded projects related to the just closed battle within 1 minute. • The system must update the final battle ranks within 1 minute. • The system must send notifications to all the students within 3 minutes.

Table 3.18: Use Case 18 - Manual Evaluation on CKB

3.2.3. Sequence Diagrams

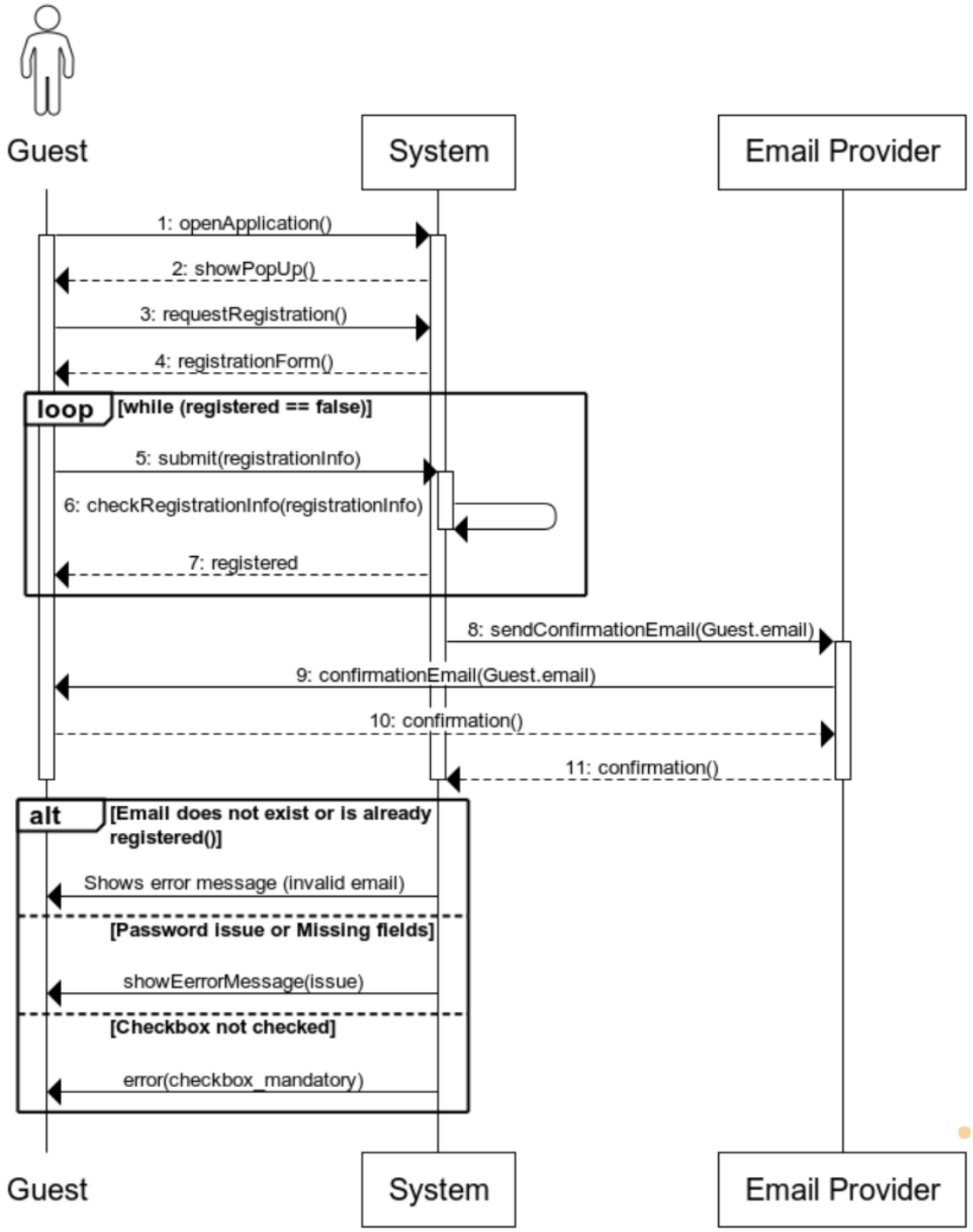

Figure 3.8: Sequence Diagram 1: User Registration

It is only shown a general use case for both the registration of a Student and an Educator, since they involve the same steps, if not for the fields required when registering.

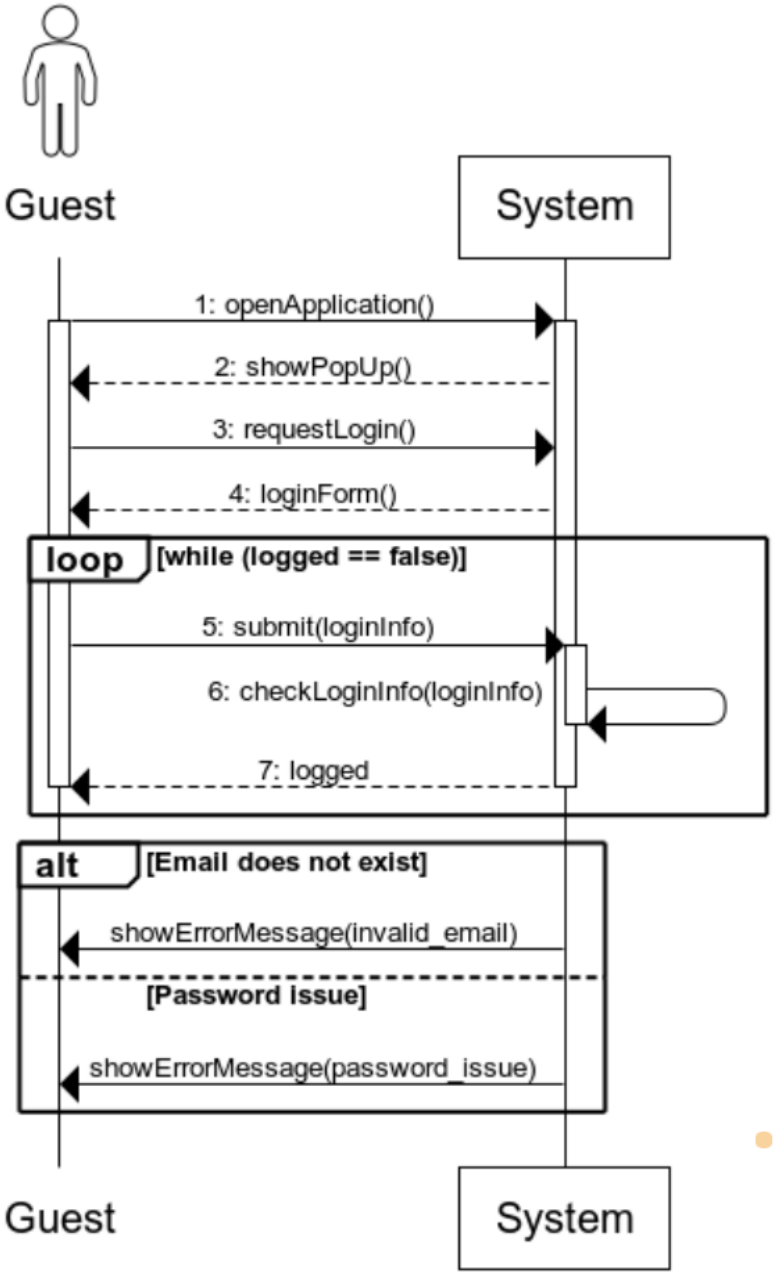

Figure 3.9: Sequence Diagram 2: User login

It is only shown a general use case for both the registration of a Student and an Educator, since they involve the same steps, if not for the fact that the Student can login either with his email or username.

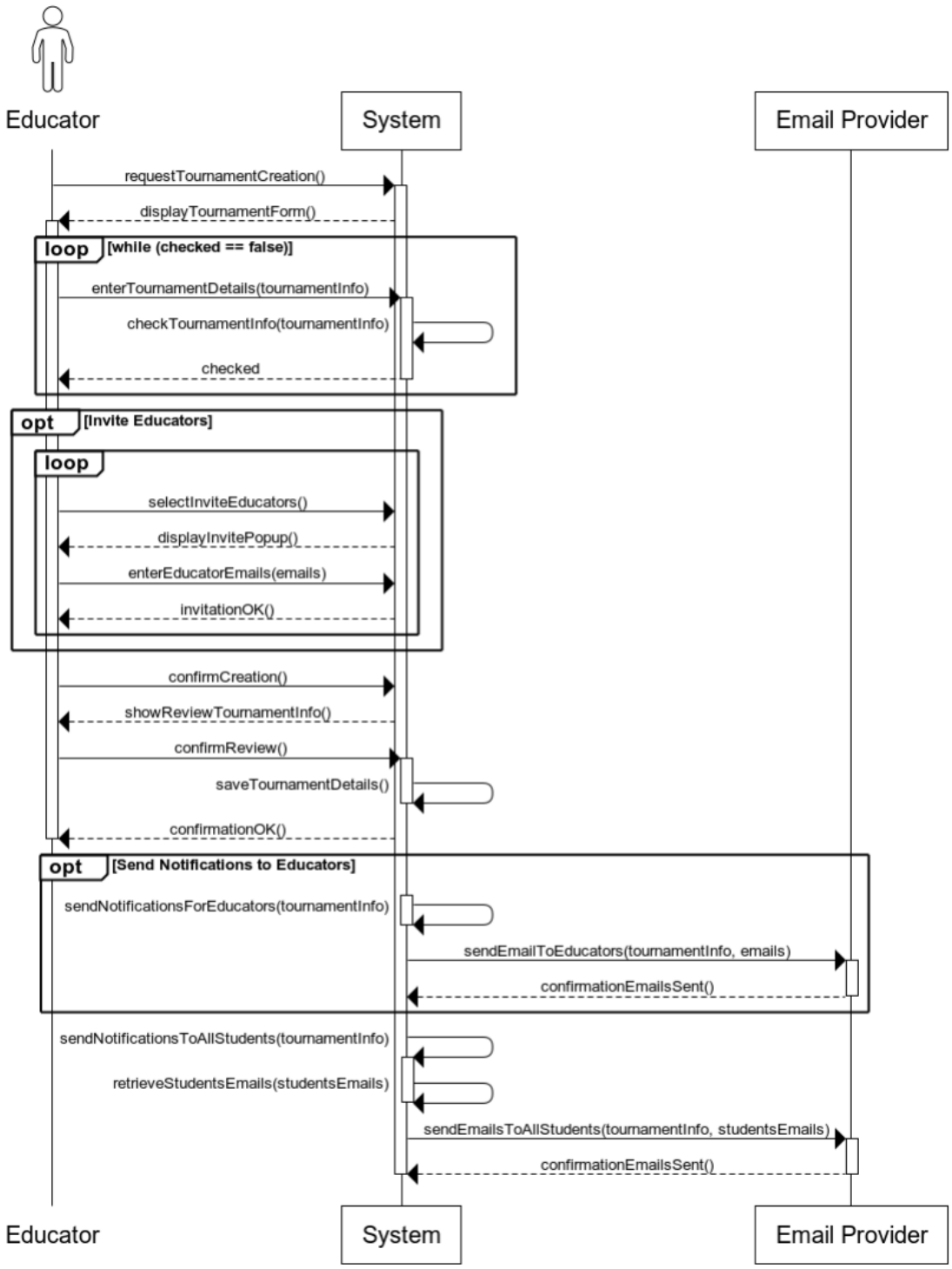

Figure 3.10: Sequence Diagram 3: Tournament Creation

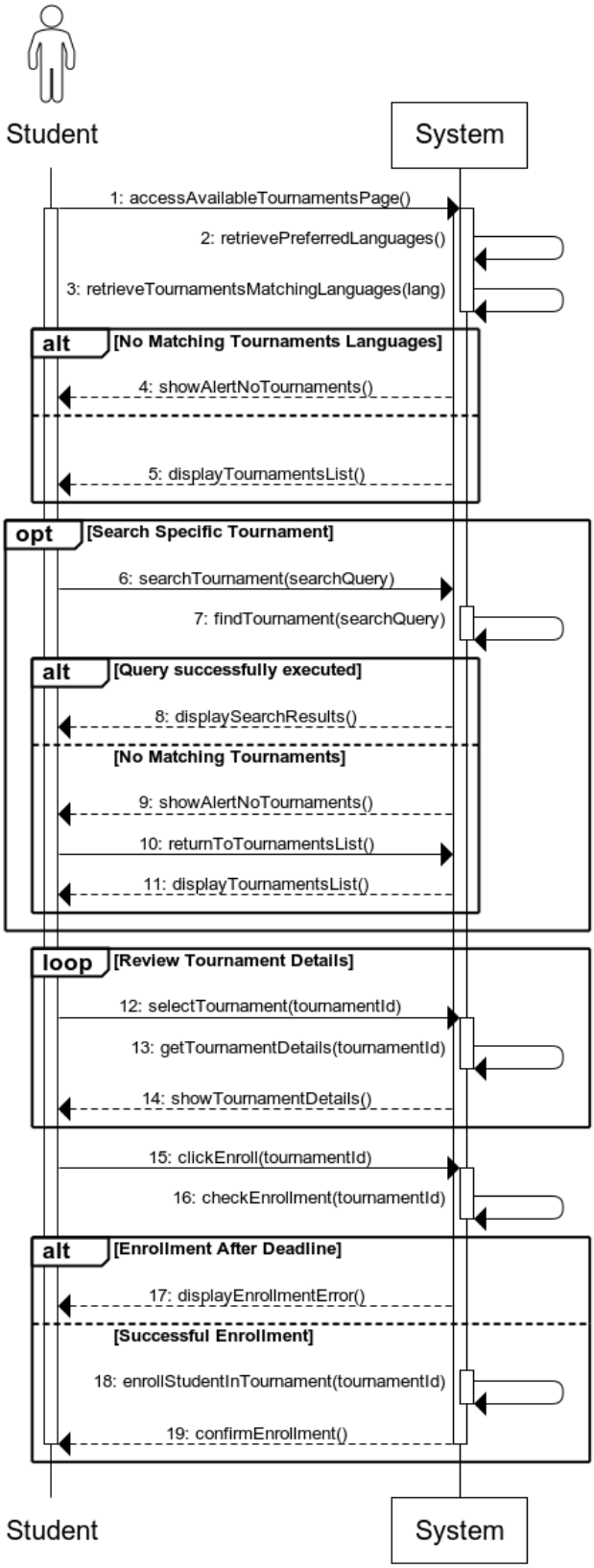

Figure 3.11: Sequence Diagram 4: Tournament Enrollment

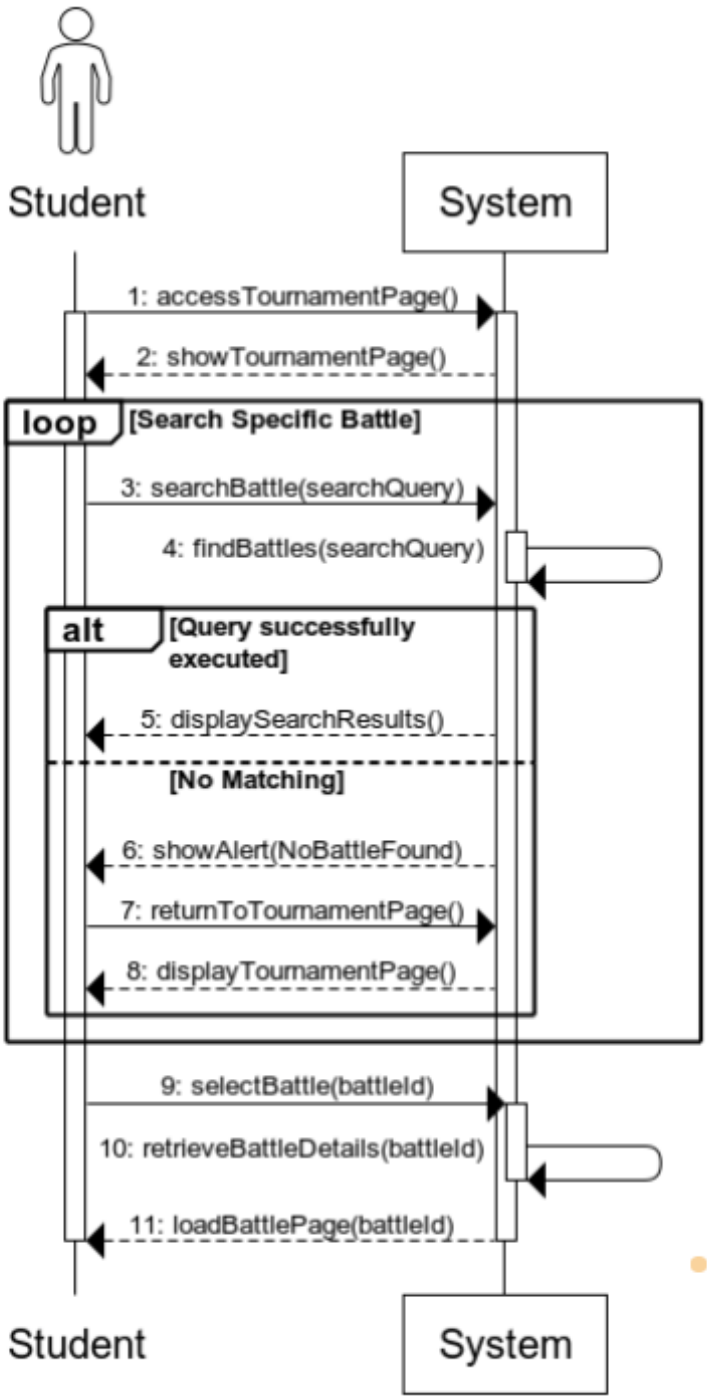

Figure 3.12: Sequence Diagram 5: Battle Search

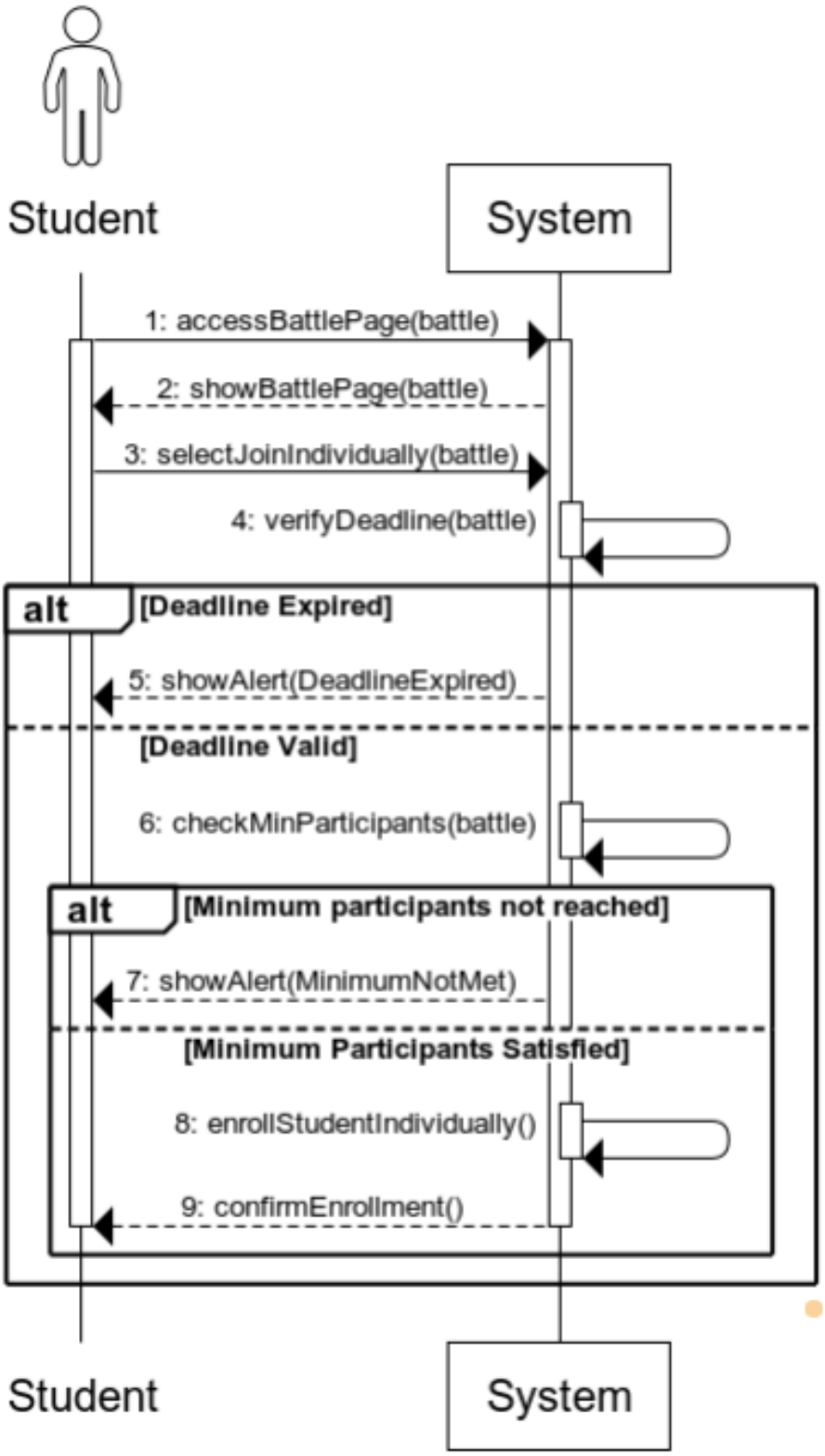

Figure 3.13: Sequence Diagram 6: Join battle individually

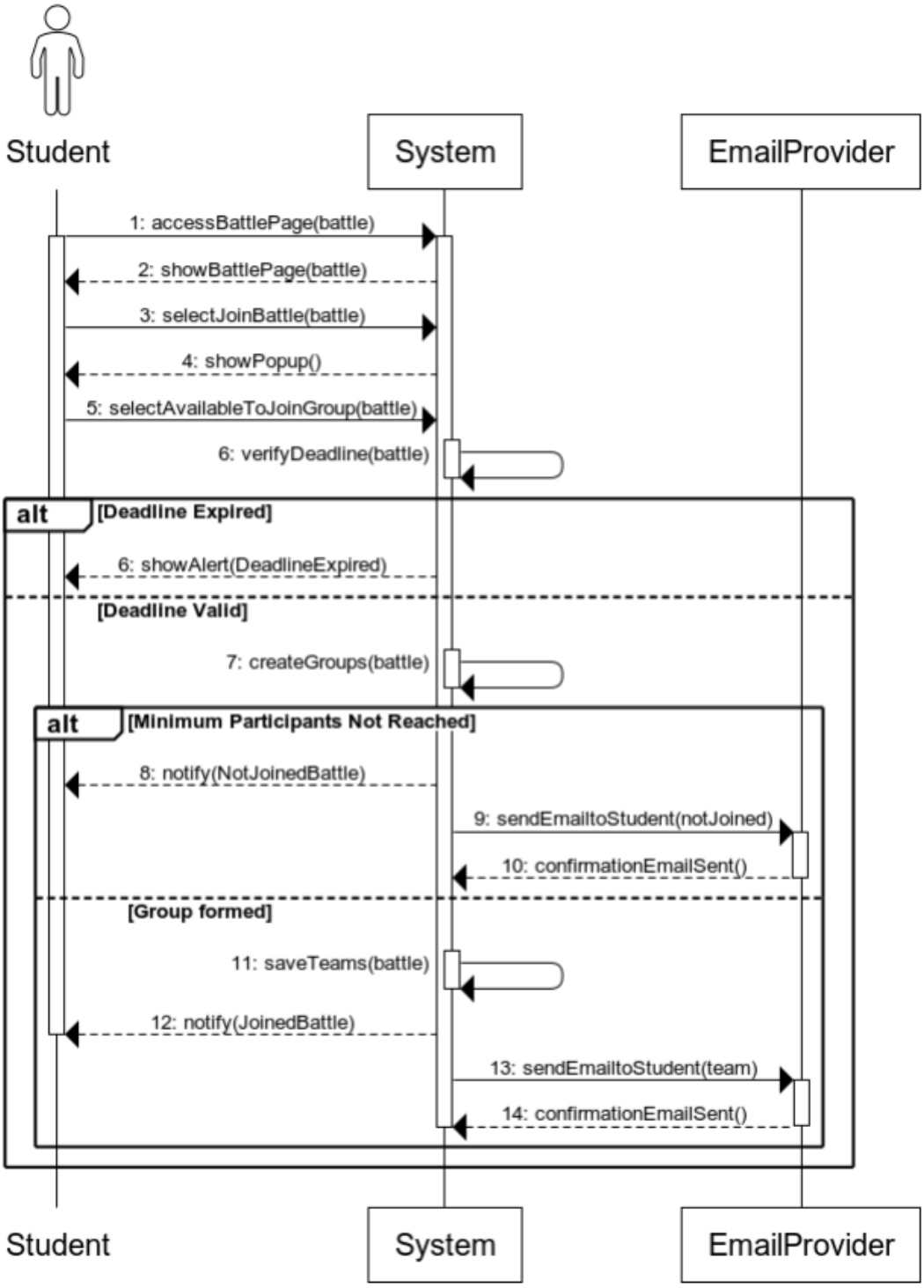

Figure 3.14: Sequence Diagram 7: Group formation

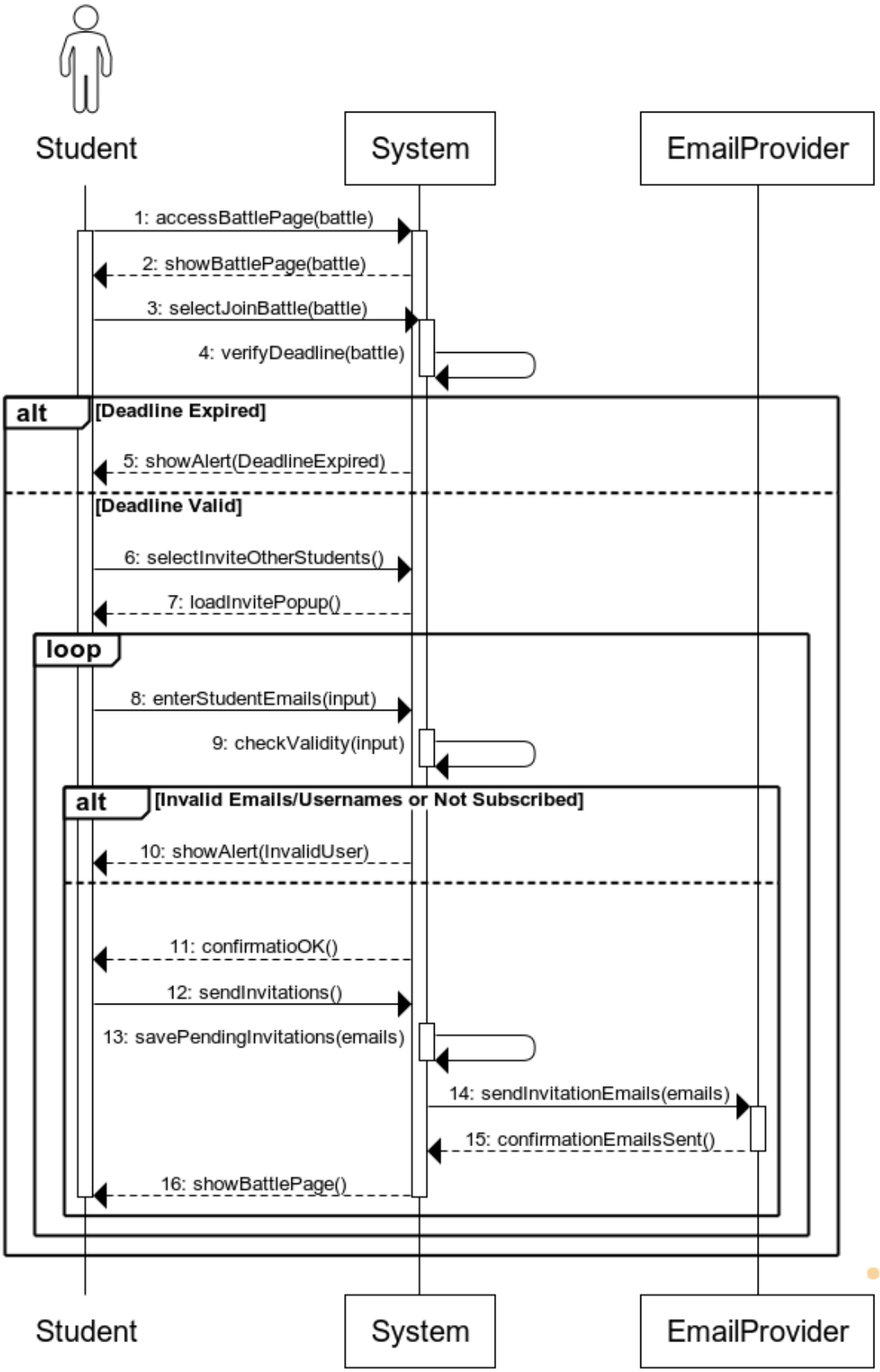

Figure 3.15: Sequence Diagram 8: Send Battle Invite

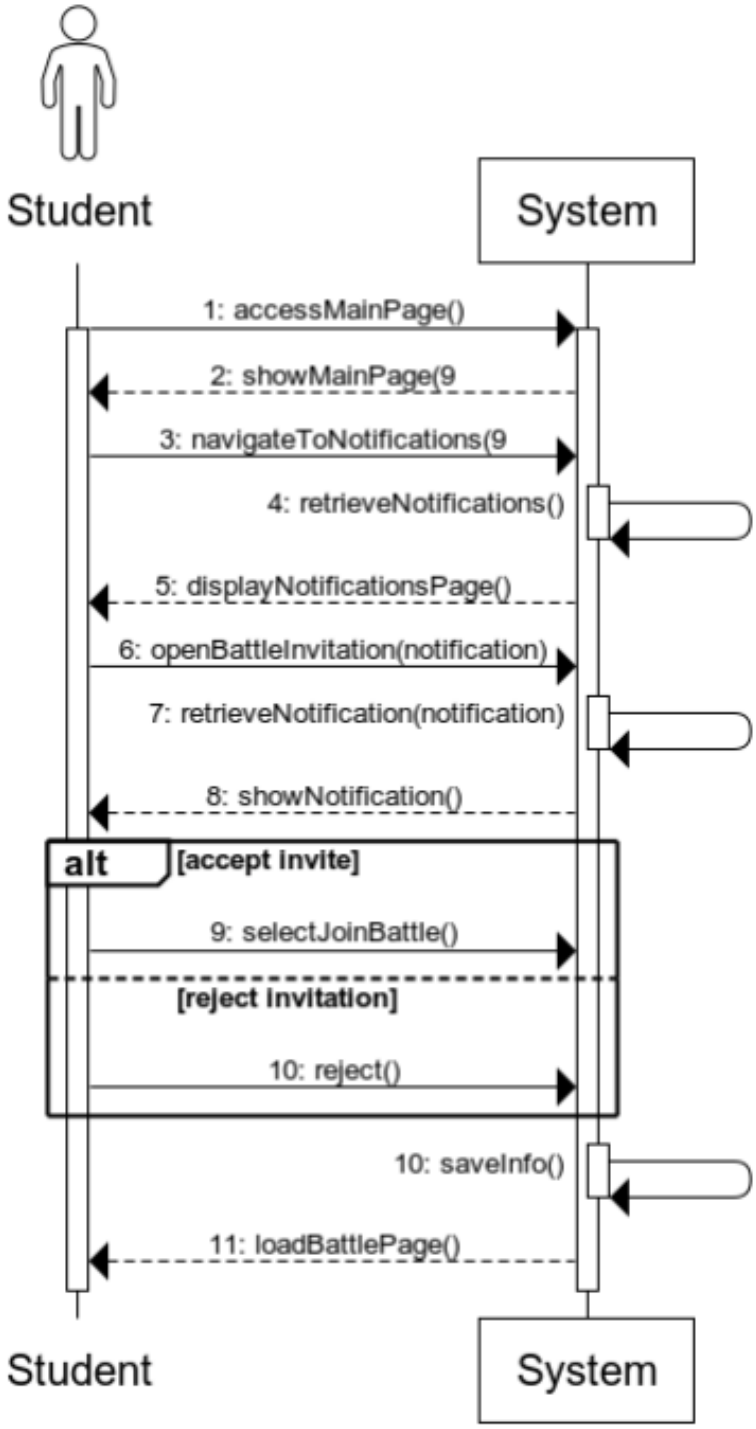

Figure 3.16: Sequence Diagram 9: Join Battle by Invitation

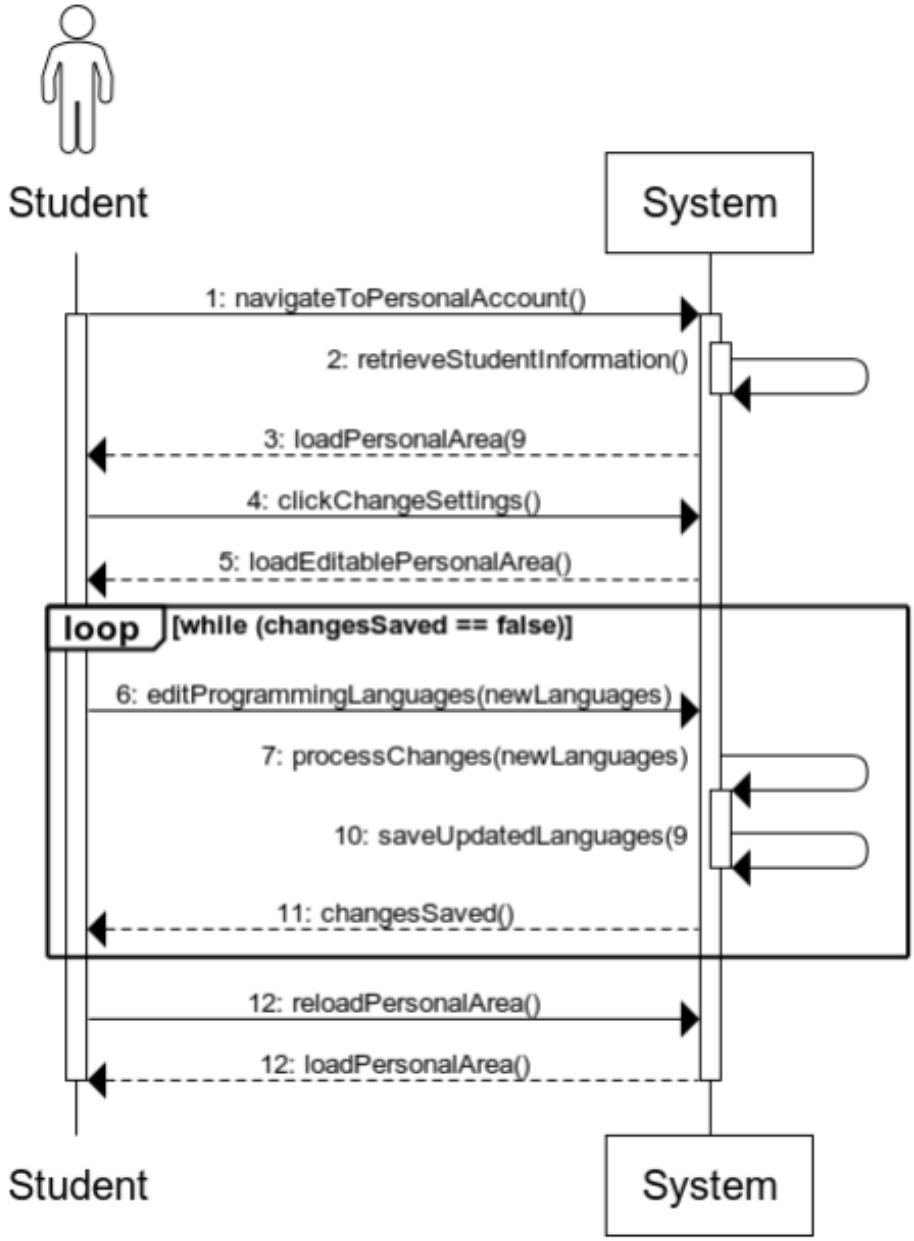

Figure 3.17: Sequence Diagram 10: Change Languages Settings

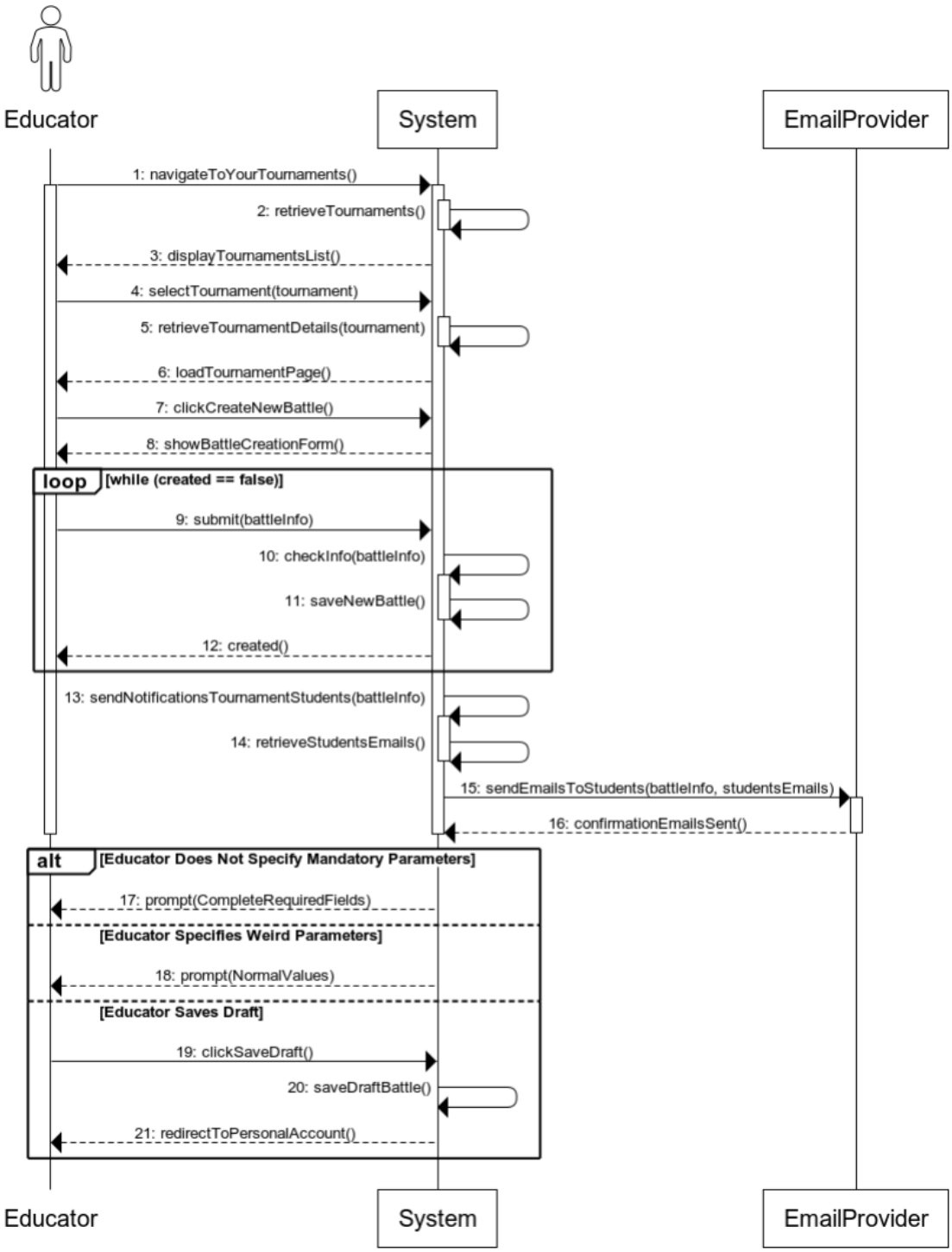

Figure 3.18: Sequence Diagram 11: Battle Creation

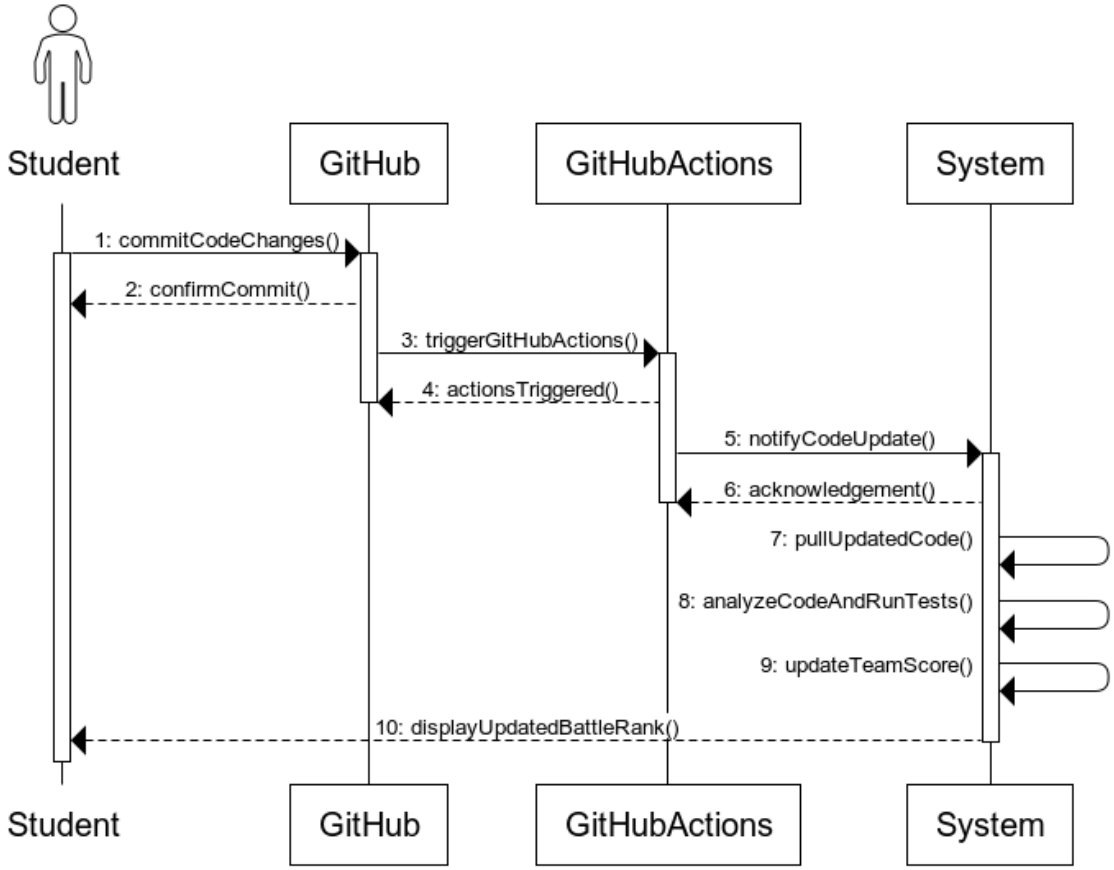

Figure 3.19: Sequence Diagram 12: Commit & Score Update

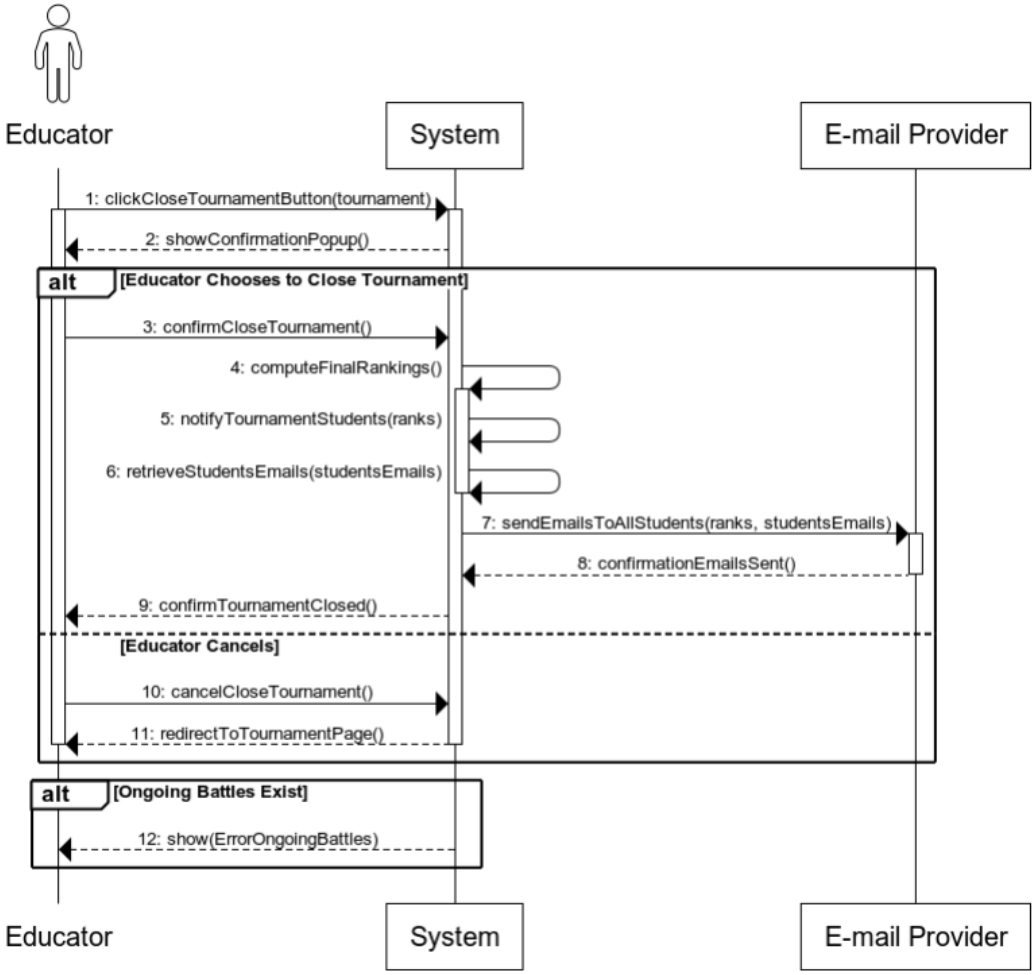

Figure 3.20: Sequence Diagram 13: Close Tournament

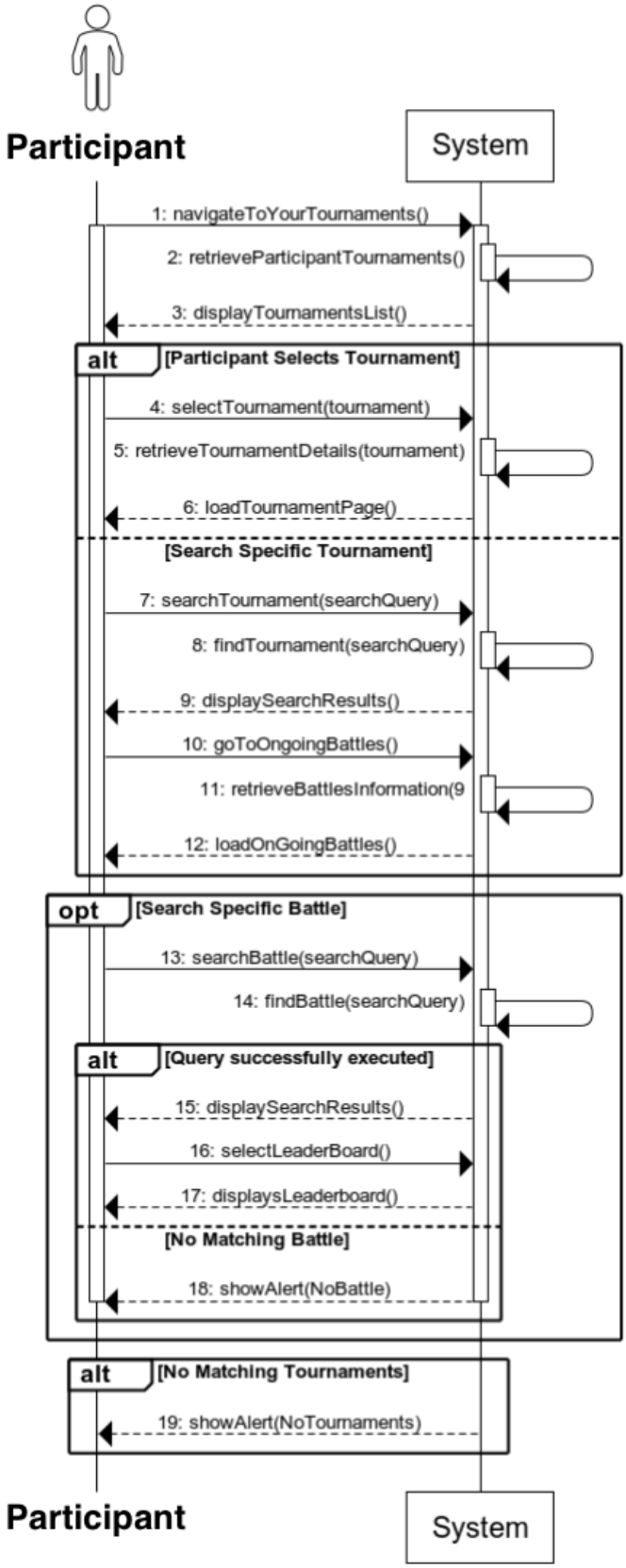

Figure 3.21: Sequence Diagram 14: View Battle Score

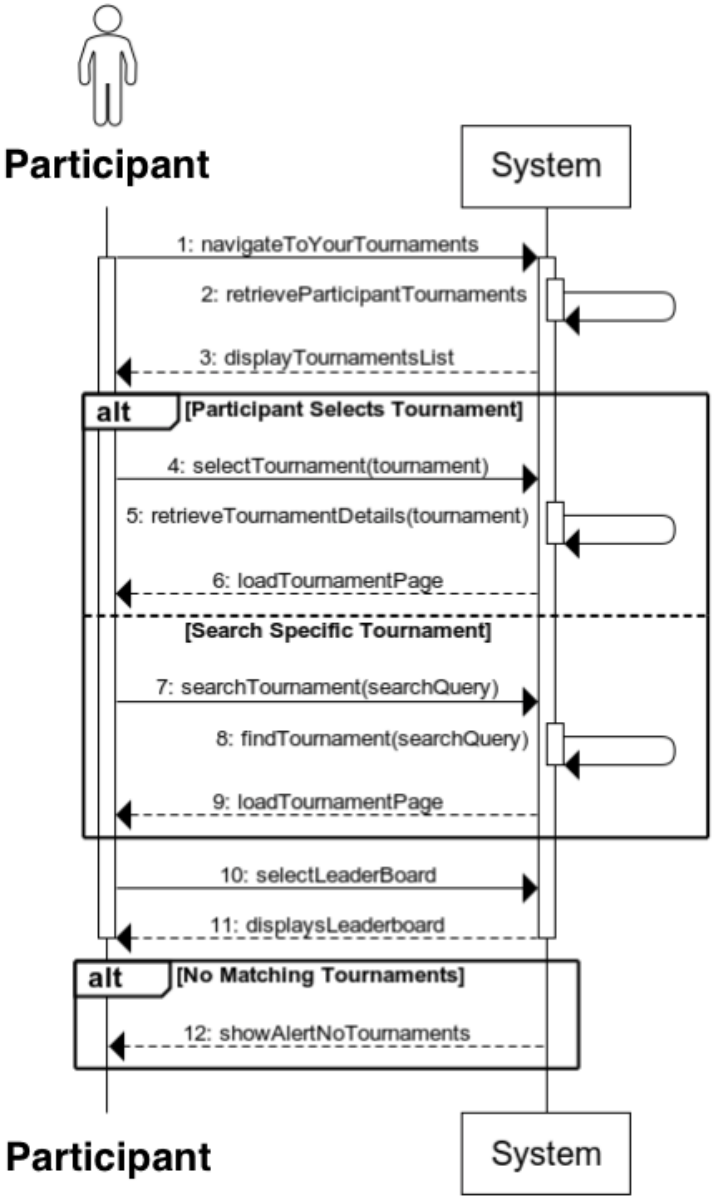

Figure 3.22: Sequence Diagram 15: View Tournament Score

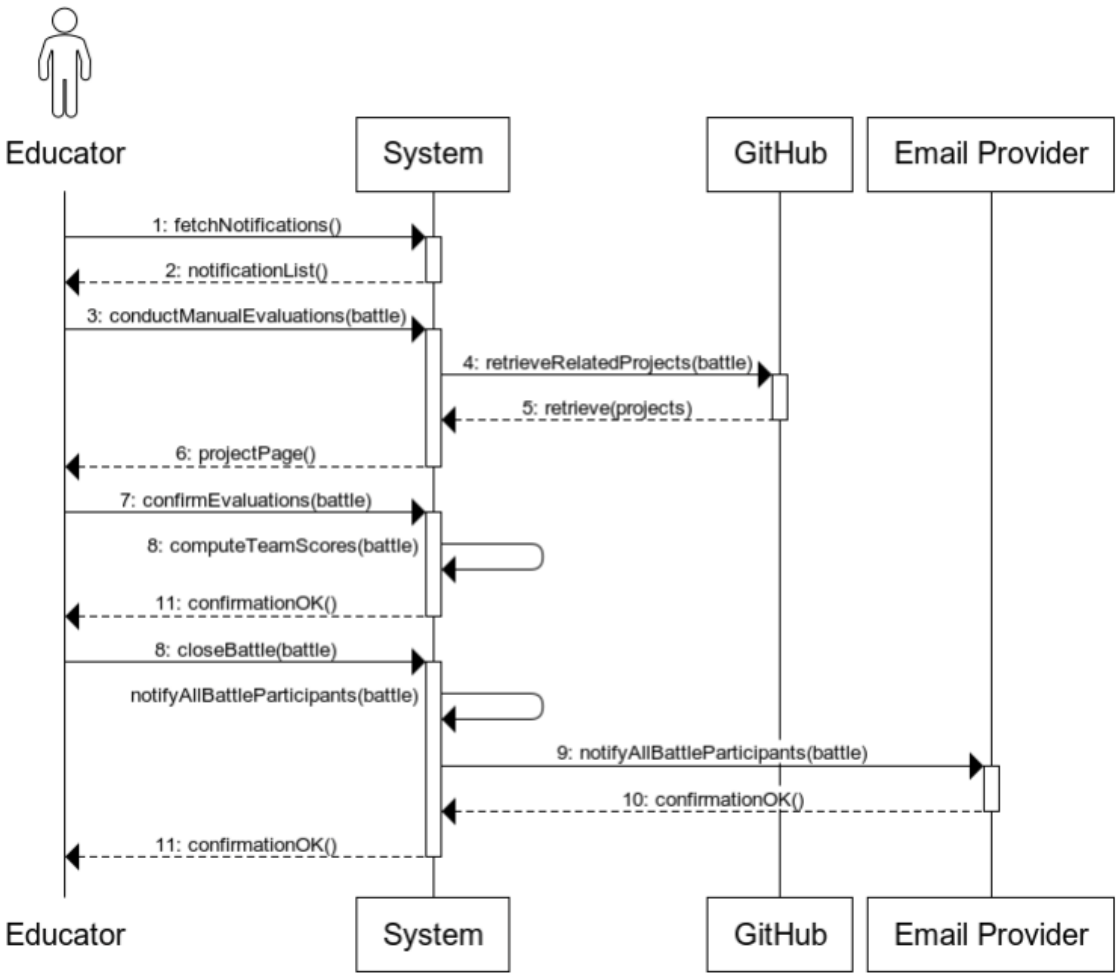

Figure 3.23: Sequence Diagram 16: Manual Evaluation

3.2.4. Functional Requirements

RE1 The System allows users to create either a Student or an Educator account.

RE2 An Educator account can be created if and only if he/she provides her educational background and institution affiliation.

RE3 The System must allow a User to register by filling in a form containing a set of fields

RE3.1 A User must be able to indicate his personal data: his first name, last name, an e-mail, and a password

RE4 The System must allow a Student to enter a username

RE5 The System must allow a Guest, when registering as an educator, to enter his educational background and institutional affiliation

RE6 The System must grant the possibility to create a tournament only to Educators

RE7 The System must store the registration data relating to the Users

RE8 The System must allow a user to login by entering his email and password

RE9 The System must allow a student to login by entering his email/username and password

RE10 The System must ask a Guest permission to send notifications

RE11 The System must process the Guest's choice about the notification permission

RE12 The System must allow a Participant to search the battles by name

RE13 The System must allow a Participant to search the tournaments by name

RE14 The System must allow a Student to register preferred programming languages

RE15 The System must allow the Educator that sets up a tournament to grant privileges to other educators to create battles within it

RE16 The system must allow Students to create a team per each battle (the team is single-use per each battle)

RE17 The System must allow Students to visualize the battles in a tournament if and only if they have subscribed to it.

RE18 The System must allow to all subscribed Users to visualize all the ongoing tournaments and their related rank

RE19 The System must allow Students to enroll in existing tournament, if and only if the expiration date is not met yet

RE20 The System must allow Educators to create tournaments on the CKB platform.

RE21 The System must allow Educators to define and publish code katas within a tournament

RE22 The System must allow an Educators to define and publish code katas within a tournament by filling in a form containing a set of fields

- **RE22.1** An Educator must be able to indicate in a code kata definition: programming language, project details, test cases, and automation scripts.
- **RE22.2** An Educator must be able to set minimum and maximum numbers of students per group for each battle.
- **RE22.3** An Educator must be able to set minimum and maximum numbers of students per group for each battle.
- **RE22.4** An Educator must be able to set additional scoring configurations
- **RE22.5** An Educator must be able to select aspects to take into consideration to evaluate quality level of the sources

RE23 The System must allow Students to invite other Students to join a battle with them, if and only if the invited Students already have an account

RE24 The System must not allow Students to invite more friends in a battle than the maximum number required to build a team

RE25 The System must allow invited students to either accept or reject the invitation of joining a battle

RE26 The System must not allow a Student to join a closed battle

RE27 The System must not allow an Educator without "creator" privileges in a tournament to create a battle within it

RE28 The System must not allow a Student to join a battle, if the expiration date has come

RE29 The system must automatically create a GitHub repository for the code kata, when the deadline expires.

RE30 The system must automatically notify enrolled student groups of a battle about the link to the GitHub repository for the code kata, when the deadline expires.

RE31 The System must be able to retrieve all the information from GitHub

RE32 The System must be able to update battle score, after each student's commit

RE33 The System must allow Educators to perform manual evaluation to the scoring of teams, after each battle has ended

RE34 The System must allow Educators to have access to all the source codes of each team in order to perform the manual evaluation

RE35 The System must guarantee to students and educators, a continuous update of battle scores and ranks

RE36 The System must guarantee Students are notified when the final battle rank is available

RE37 The System must guarantee the students' overall tournament scores is updated after each battle has ended.

RE38 The System must allow an Educator to close a tournament, if and only if he is the creator of it

RE39 The System must update the team's rank after a student, who is part of the team, executes a commit.

RE40 The System must send a notification to all subscribed Students when a new tournament is created.

RE41 The System must send a notification to all Students enrolled in a tournament, when a new battles are created within it.

RE42 The System must compute the automated evaluation after each push based on mandatory factors: functional aspects, timeliness, quality level of the sources.

RE43 The System to evaluate the functional aspects of the score must track the number of test cases that pass out of all test cases

RE44 The System to evaluate the timeliness of the score must track the time passed between the registration deadline and the last commit of a team

RE45 The System to compute the quality level of the sources must be able to extract through static analysis tools multiple aspects such as security, reliability, and maintainability

RE46 The System must allow Students to view their team battle score

RE47 The System must allow Students to view the battle team leaderboard, if and only if the students are enrolled in the battle

RE48 The system must notify Students when the final battle rank becomes available, if and only of the student is enrolled in that battle

RE49 The System at the end of each battle must be able to update the personal tournament score of each student

RE49.1 The System must be able to calculate the personal tournament score of each student by summing of all battle scores received in that tournament

RE50 The System must notify all Students involved all students involved in the tournament closed by an educator, as soon as the final tournament rank becomes available.

RE51 The System must send notifications by email, and through the platform

RE52 The System must set to "Created" the status of a tournament just created by an Educator.

RE53 The System must change to "Opened" the status of a tournament when its registration deadline expires.

RE54 The System must change to "Closed" the status of a tournament when its creator closes it.

RE55 The System must set to "Created" the status of a battle just created by an Educator.

RE56 The System must change to "Opened" the status of a battle when its registration deadline expires.

RE57 The System must change to "Evaluating" the status of a battle when its final submission deadline expires.

RE58 The System must change to "Closed" the status of a battle once battle scores have been calculated.

RE59 The System must guarantee secure login functionality for both students and educators.

RE60 The System must set to "Pending" the status of a Student's request to join a battle with a minimum group size greater than one

RE61 The System must set to "Accepted" the status of a Student's request to join a battle, if and only if he is part of a team that meets the min-max criteria group formation

RE62 The System must have the capability to interface with an external email service provider for sending automated emails. This includes generating email requests containing appropriate content (such as notifications or invitations) and transmitting these requests securely to the email provider for distribution to the intended recipients

3.2.5. Mapping Use Cases - Requirements

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11
RE1	•	•									
RE2		•									
RE3	•	•									
RE4	•										
RE5		•									
RE6					•						
RE7	•	•									
RE8			•	•	•	•	•	•	•	•	•
RE9			•		•	•	•		•	•	•
RE10	•	•									
RE11	•	•									
RE12							•				
RE13						•					
RE14	•										
RE15					•						
RE16								•	•	•	•
RE17								•	•	•	•
RE18											
RE19						•					
RE20					•						
RE21											
RE22											
RE23											
RE24										•	
RE25											
RE26											•
RE27											
RE28											
RE29											
RE30											
RE31					•						
RE32											
RE33											

Table 3.19: Use Case vs. Requirement Matrix

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11
RE34											
RE35											
RE36											
RE37											
RE38											
RE39											
RE40					•						
RE41						•					
RE42											
RE43											
RE44											
RE45											
RE46											
RE47											
RE48											
RE49											
RE50											
RE51					•				•	•	•
RE52					•						
RE53											
RE54											
RE55											
RE56											
RE57											
RE58											
RE59			•	•							
RE60								•	•	•	
RE61								•	•	•	
RE62	•	•	•								

Table 3.20: Use Case vs. Requirement Matrix

	UC12	UC13	UC14	UC15	UC16	UC17	UC18
RE1							
RE2							
RE3							
RE4							
RE5							
RE6							
RE7							
RE8	•	•	•	•	•	•	•
RE9	•		•		•	•	
RE10							
RE11							
RE12		•	•		•	•	
RE13		•		•	•	•	
RE14	•						
RE15		•					
RE16							
RE17							
RE18					•	•	
RE19	•	•					
RE20		•					
RE21		•					
RE22							
RE23							
RE24							
RE25							
RE26							
RE27							
RE28							
RE29						•	
RE30				•			
RE31							
RE32			•				
RE33							•

Table 3.21: Use Case vs. Requirement Matrix

	UC12	UC13	UC14	UC15	UC16	UC17	UC18
RE34							•
RE35					•		•
RE36							•
RE37							
RE38				•			
RE39			•				
RE40							
RE41							
RE42			•				
RE43			•				
RE44			•				
RE45			•				
RE46			•		•		
RE47			•		•		
RE48							
RE49							
RE50				•			
RE51		•		•			•
RE52							
RE53							
RE54			•				
RE55		•					
RE56							
RE57							
RE58							
RE59							
RE60							
RE61							
RE62							

Table 3.22: Use Case vs. Requirement Matrix

3.2.6. Traceability Matrix

Goals	Requirements	Domain Assumptions
G1: Allow *Educators* to go through the sources produced by each team to assign a manual evaluation, if needed	RE33, RE34	DA2
G2: Allow *Students* to commit their work in the Github repository connected to a specific battle	RE29, RE30, RE31, RE32	DA3, DA5, DA6, DA8, DA9
G3: Allow *Students* to track their personal evolving rank as they push changes on the platform	RE35, RE37, RE39, RE47, RE49	DA5, DA6, DA9
G4: Allow Users to be identified either as *Students* or as *Educators*	RE1, RE2, RE3, RE4, RE5	DA1, DA7
G5: Allow *Educators* to create tournaments	RE6, RE20, RE52	DA1, DA2
G6: Allow *Educators* to create a code kata battle with all its related features	RE21, RE22, RE55	DA1, DA2
G7: Allow *Students* to enroll in an existing tournament	RE19, RE53	DA1, DA7
G8: Allow *Students* to enroll in a code kata	RE17, RE19, RE23	DA1, DA7
G9: Allow *Educators*, creators of tournaments, to grant privileges to other *Educators* of creating battle in the related tournament	RE15, RE27	DA1, DA2

Goals	Requirements	Domain Assumptions
G10: Allow *Students* to work on the project being given by the platform the link of the Github repository containing the code kata	RE29, RE30, RE31	DA3, DA5, DA6, DA8, DA9
G11: Allow *Educators* involved in a battle to see the current rank evolving during the battle	RE35, RE37	DA5, DA6, DA9
G12: Allow *Users* to overview the list of ongoing tournaments	RE18, RE40, RE53	DA1, DA6
G13: Allow *Users* to overview the list of corresponding tournament rank, measuring how a student's performance compares to other students in a tournament	RE18, RE47, RE49, RE50	DA5, DA6, DA9
G14: Allow *Students* to create a team	RE16	DA1, DA7
G15: Allow *Students* to invite other students to a code kata	RE23, RE24, RE25, RE26	DA1, DA7
G16: Allow *Students* to deliver their solution of the battle to the platform	RE33, RE34, RE35, RE57, RE58	DA2, DA5, DA6

Table 3.24: Traceability Matrix for CKB Platform

3.3. Performance Requirements

The CodeKataBattle Platform is designed to meet high performance standards essential for a smooth and efficient user experience. Key performance requirements include real-time processing capabilities to ensure immediate updates and scoring following each code submission or update. The system must be scalable to handle a significant number of concurrent users and battles, maintaining performance integrity even during peak periods. Reliability is crucial, with the platform expected to operate continuously with minimal downtime. The response time should be rapid, particularly in updating scores and reflecting ranking changes, to enhance user engagement. Moreover, maintaining data integrity during the transmission of information between GitHub, static analysis tools, and the CKB platform is paramount, especially in the handling of scores and code evaluations.

3.4. Design Constraints

3.4.1. Standard Compliance

The web application component of the CodeKataBattle Platform must adhere to the latest web standards, including HTML5, CSS3, and JavaScript ES6, ensuring broad compatibility across various web browsers and devices.

Compliance with web accessibility standards, such as the WCAG, is essential to make the platform accessible to a diverse range of users, including those with disabilities.

Additionally, the platform should implement standard security protocols like HTTPS and OAuth for API interactions, safeguarding data transmission and preserving user privacy.

3.4.2. Hardware Limitations

The platform relies heavily on a stable internet connection for real-time data updates and synchronization with services like GitHub and static analysis tools.

Thus, the participants must have devices able to exchange data via internet.

3.5. Software System Attributes

3.5.1. Reliability

The reliability of the CodeKataBattle (CKB) platform is fundamental to its effectiveness as a learning tool. In this context, reliability pertains to the system's ability to function consistently and accurately, providing uninterrupted access to its resources and services. Achieving a reliability target of over 93% is ambitious yet essential for maintaining user confidence and ensuring the platform's utility.

To monitor reliability, CKB can implement advanced analytics and monitoring tools. These tools would track system performance in real-time, identifying any anomalies or performance dips that could indicate potential problems. By analyzing trends and historical data, the platform can predict and preemptively address issues before they escalate into significant downtimes.

The Mean Time To Failure (MTTF) metric plays a crucial role in this aspect. It represents the average time between system failures. By maximizing MTTF, CKB ensures that disruptions are rare and brief, thus minimizing their impact on the user experience. To do this effectively, regular maintenance schedules and updates are crucial. These maintenance windows can be used to apply patches, update software components, and perform hardware checks to prevent potential failures.

In the event of a system failure, a well-defined incident response plan is vital. This plan should outline the steps to be taken to quickly and effectively restore service. It includes identifying the problem, mobilizing the response team, implementing a fix, and communicating with users throughout the process. Post-incident reviews are equally important, as they provide insights into what went wrong and how similar issues can be prevented in the future.

3.5.2. Availability

For a learning platform like CKB, availability is about ensuring that students and educators can access the platform whenever they need it. High availability, quantified as "2-nines" or 99% uptime, translates to the platform being down for no more than four days a year. This level of availability is critical for maintaining a seamless educational experience, especially during peak usage times like examination periods or project deadlines.

To achieve this, CKB should implement robust server infrastructure with failover capa-

bilities. Load balancing can distribute the traffic evenly across multiple servers, ensuring that no single server becomes a bottleneck. In case one server fails, the load balancer redirects traffic to the remaining operational servers, minimizing the impact on users.

Redundancy is another key strategy for improving availability. By having redundant components, such as duplicate databases or servers, CKB can ensure that if one component fails, another can take over without interrupting service. This approach is particularly effective in handling hardware failures, network issues, or power outages.

Rapid recovery from incidents is another aspect of availability. This involves not just restoring service quickly but also ensuring data integrity and security during the recovery process. Regular backups, preferably stored in multiple locations, allow for quick restoration of data in the event of data loss. Additionally, having a well-practiced disaster recovery plan ensures that the team can respond effectively in the event of a significant system failure.

3.5.3. Security

Security is a paramount concern for any platform handling personal data, and CKB is no exception. The security measures implemented should protect against unauthorized access, data breaches, and other cyber threats. This involves multiple layers of security, including physical security of servers, network security, application security, and data encryption.

Data encryption is essential for protecting sensitive information such as personal details, coding submissions, and educational records. Encryption should be applied both in transit (as data moves across the network) and at rest (when stored on servers). This ensures that even if data is intercepted or a server is compromised, the information remains unreadable and secure.

Securing API endpoints is crucial to prevent unauthorized access to the platform's backend systems. Technologies like eXternal Data Representation (XDR) can provide robust security measures to protect against various forms of cyberattacks, including "man in the middle" attacks. Regular security audits and updates are also vital for staying ahead of new vulnerabilities and threats. This includes patch management, updating software components, and revising security protocols as needed.

Additionally, implementing robust authentication and authorization mechanisms ensures that only legitimate users can access their accounts and that they can only access the information and functionalities relevant to their role. This might include multi-factor

authentication, role-based access control, and regular reviews of user privileges.

3.5.4. Maintainability

Maintainability in the context of CKB relates to the ease with which the platform can be updated, modified, and improved over time. This is crucial for adapting to the evolving needs of students and educators, as well as for integrating new technologies and educational methodologies.

Good maintainability starts with a well-architected system. This means using modular design principles, where different functions of the platform are divided into discrete, loosely coupled components. Such a design makes it easier to update or replace individual parts of the system without affecting the whole. It also facilitates scalability, allowing the platform to grow and evolve in response to increasing user numbers or changing requirements.

Code quality is a significant factor in maintainability. High-quality, clean, and well-documented code is easier to understand, debug, and modify. It should follow established coding standards and best practices. This includes using meaningful variable names, writing clear and concise comments, and structuring code in a logical and organized manner.

Regular code reviews and testing are also essential components of maintainability. Code reviews help ensure that new code meets quality standards and is in line with the overall design of the system. Testing, including unit tests, integration tests, and user acceptance tests, helps identify and fix bugs early in the development process.

Another aspect of maintainability is dealing with technical debt – the cost of rework caused by choosing an easy solution now instead of using a better approach that would take longer. Technical debt can accumulate over time, making the system harder to maintain and update. Regularly addressing technical debt, through refactoring and improving existing code, is crucial for maintaining the long-term health of the system.

Portability

Portability is the ease with which the CKB platform can be used across different hardware and software environments. Given the diverse nature of the devices students use – ranging from desktop computers to tablets and smartphones – it's vital that the platform is easily accessible and functional on all these devices.

Cross-platform compatibility is the first step toward ensuring portability. The platform should be designed to work seamlessly across different operating systems, such as Win-

dows, macOS, Linux, iOS, and Android. This might involve using cross-platform development frameworks or ensuring that web-based components of the platform are compatible with all major browsers.

Responsive design is another crucial aspect of portability. The user interface of the platform should automatically adjust to fit the screen size and resolution of the device being used. This ensures that the platform is not only accessible but also usable on devices with smaller screens, like smartphones and tablets.

Testing the platform across various devices and environments is essential to ensure portability. This includes not only testing on different operating systems and browsers but also on devices with different screen sizes and hardware configurations. This helps identify and fix any issues that might arise due to differences in how these devices and systems operate.

Finally, considering the device-agnostic features is important. This means designing functionalities that do not rely on specific hardware capabilities that might not be present in all devices. For example, avoiding features that require a high-end graphics card or a certain amount of RAM ensures that the platform is accessible to students with less powerful devices.

4 Formal Analysis using Alloy

The Alloy model depicted here provides limited yet representative snapshots of the project domain, ensuring compliance with the specified requirements. This model is fundamentally structured around the class diagram, encompassing signatures complete with their respective attributes and constraints. To verify the accuracy and robustness of the model, all primary functional requirements have undergone rigorous testing through assertions. Three notable instances of the model are showcased, each accompanied by corresponding predicates and visual representations in the form of graphs.

4.1. Definitions

4.1.1. Enumerations

```
enum TournamentLifeCycle {
        tCreated,
        tOpened,
        tClosed
}

enum BattleLifeCycle {
        bCreated,
        bOpened,
        bEvaluation,
        bClosed
}

enum QualityAspect {
        reliability,
        maintanability,
        security
}
```

```
enum RequestStatus {
        pending,
        accepted,
        rejected
}

enum BattleRequestStatus {
        brPending,
        brAccepted,
        brRejected
}
```

Listing 4.1: Enumeration code snippet

4.1.2. Signatures

```
sig Str, Timestamp, FileCollection {}

abstract sig Guest {}

abstract sig Participant extends Guest {
        firstName: one Str,
        lastName: one Str,
        email: disj one Str,
        password: one Str
}

sig ProgrammingLanguage {
        name: disj one Str
}

sig Student extends Participant {
        username: disj one Str,
        learns: some ProgrammingLanguage
}

sig Educator extends Participant {
```

```
        affiliate: some Institution,
        institutionalBackground: one Str,
        institutionalBackgroundFiles: disj one FileCollection
          ↪ ,
        createsT: set Tournament
}

sig Institution {
        name: disj one Str
}

sig Tournament {
        creator: one Educator,
        helpers: set Educator,
        name: disj one Str,
        basedOn: one ProgrammingLanguage,
        creationTime: one Timestamp,
        regDeadline: one Timestamp,
        status: one TournamentLifeCycle,
        contains: disj some Battle
} {
        creationTime != regDeadline
        creator not in helpers
}

sig Battle {
        createdIn: one Tournament,
        creator: one Educator,
        name: disj one Str,
        creationTime: one Timestamp,
        regDeadline: one Timestamp,
        finalSubDeadline: one Timestamp,
        minTeamSize: one Int,
        maxTeamSize: one Int,
        additionalScoringConfigs: one Str,
        evaluatedOn: set QualityAspect,
        has: disj one CodeKata,
```

```
        status: BattleLifeCycle,
        exploits: set StaticSystemTool
} {
        minTeamSize > 0
        maxTeamSize > 0
        minTeamSize < maxTeamSize
        #evaluatedOn > 0 implies #exploits > 0
        #evaluatedOn >= #exploits
}

sig CodeKata {
        description: disj one Str,
        testCases: disj one FileCollection,
        automationScripts: disj one FileCollection
} {
        testCases != automationScripts
}

sig StaticSystemTool {
        evaluates: some QualityAspect,
        accepts: some ProgrammingLanguage,
        monitors: set Battle
}

sig Team {
        id: disj one Str,
        participates: one Battle,
        creationTime: one Timestamp,
        isComposedBy: disj some BattleEnrollment,
        has: disj one GitHubRepository,
        battleScore: one Int
} {
        participates.minTeamSize <= #isComposedBy
        participates.maxTeamSize >= #isComposedBy
        battleScore > 0
}
```

```
sig GitHubRepository {
        url: disj one Str,
        sources: disj one FileCollection
}

sig EnrolledStudent {
        student: one Student,
        tournament: one Tournament,
        creationTime: one Timestamp,
        tournamentScore: one Int
} {
        tournamentScore >= 0
        creationTime = tournament.regDeadline
}

sig BattleEnrollment {
        enrolledStudent: one EnrolledStudent,
        battle: one Battle,
        battleScore: one Int,
        creationTime: one Timestamp,
        status: one BattleRequestStatus
} {
        battle in enrolledStudent.tournament.contains
}

sig TeamFormationRequest {
        sender: one BattleEnrollment,
        receiver: one BattleEnrollment,
        battle: one Battle,
        status: one RequestStatus
} {
        sender != receiver
        sender.battle = receiver.battle
        status = pending and battle.minTeamSize > 1
                implies sender.status = brPending and
                receiver.status = brPending
        status = rejected and battle.minTeamSize = 1
```

```
                implies sender.status = brAccepted and
                receiver.status = brRejected
        status = rejected and battle.minTeamSize > 1
                implies sender.status = brRejected and
                receiver.status = brRejected
        status = accepted implies
                sender.status = brAccepted and
                receiver.status = brAccepted
        (status = pending and battle.minTeamSize = 1)
        implies sender.status = brAccepted and
        receiver.status = brPending
}

sig TournamentHelpRequest {
        sender: one Educator,
        receiver: one Educator,
        tournament: one Tournament,
        status: one RequestStatus
} {
        sender != receiver
        sender = tournament.creator
        status = accepted implies
                receiver in tournament.helpers
        status = rejected or status = pending implies
                receiver not in tournament.helpers
}
```

Listing 4.2: Signatures code snippet

4.1.3. Facts

```
    fact everyProgLangAtLeastOneStudent {
        all p: ProgrammingLanguage | some s: Student | p in s
          ↪ .learns
}

fact everyInstitutionOneEducator {
```

```
        all i: Institution | some e: Educator | i in e.
          ↪ affiliate
}

fact everyTournamentOneEducator {
        all t: Tournament | one e: Educator | t in e.createsT
          ↪  and t.creator = e
}

fact everyBattleOneTournament {
        all b: Battle | one t: Tournament | b in t.contains
          ↪ and b.createdIn = t
}

fact everyAspectAtLeastBattle {
        all q: QualityAspect | some b: Battle | q in b.
          ↪ evaluatedOn
}

// There is only one tournament creator
fact {
        no disj e1, e2: Educator | one t: Tournament |
        t in e1.createsT and t in e2.createsT
}

fact coherenceSSTsBattles {
        all sst: StaticSystemTool | all b: Battle |
        b in sst.monitors implies
        b.createdIn.basedOn in sst.accepts
}

fact everySSTatLeastOneBattle {
        all sst: StaticSystemTool | one b: Battle |
        b in sst.monitors and sst in b.exploits
}

fact everyQualityAspectOfABattleIsMonitored {
```

```
        all b: Battle, q: QualityAspect |
        q in b.evaluatedOn implies
        one sst: StaticSystemTool |
        q in sst.evaluates and
        b in sst.monitors
}

// If a battle has a certain quality aspect,
// there is only one security system tool that can analyze it
fact {
        no disj sst1, sst2: StaticSystemTool | all b: Battle
           ↪ |
        b.evaluatedOn in sst1.evaluates and
        b.evaluatedOn in sst2.evaluates
}

fact SSTcoeherentWithTournamentProgLang {
        all sst: StaticSystemTool | one b: Battle |
        b.createdIn.basedOn in sst.accepts
}

fact {
        all t: Team, be: BattleEnrollment |
        be in t.isComposedBy implies be.status = brAccepted
}

fact {
        all tfr: TournamentHelpRequest |
        tfr.status = accepted implies tfr.receiver in tfr.
           ↪ tournament.helpers
}

fact {
        all tfr: TournamentHelpRequest |
        (tfr.status = rejected or tfr.status = pending)
        implies tfr.receiver not in tfr.tournament.helpers
}
```

```
fact {
        all e: Educator, t: Tournament |
        e in t.helpers implies
        (one thr: TournamentHelpRequest |
        thr.receiver = e and
        thr.status = accepted and
        thr.sender = t.creator)
}

fact disjointEnrolledStudents {
        no disj t1, t2: EnrolledStudent |
        t1.student = t2.student and
        t1.tournament = t2.tournament
}

fact disjointBattleEnrollments {
        no disj be1, be2: BattleEnrollment |
        be1.enrolledStudent = be2.enrolledStudent and
        be1.battle = be2.battle
}

fact disjointTournamentHelpRequests {
        no disj t1, t2: TournamentHelpRequest |
        t1.sender = t2.sender and
        t1.receiver = t2.receiver
}

fact onlyAcceptedOne {
        all t: Team, be: BattleEnrollment |
        be in t.isComposedBy implies be.status = brAccepted
}
```

Listing 4.3: Facts code snippet

4.2. First World

```
    pred w1 {
        #EnrolledStudent = 4
        #Educator = 1
        #Tournament = 1
        #Team = 2
        #Battle = 1
}
```

Listing 4.4: First World

In World 1, the system is set up in its simplest form. This world consists of only one Educator that creates a single Tournament. The Tournament features only one Battle.

The enrolled students could be competing individually or forming up to two Teams, as indicated by the model. This world is perfect for testing the system's basic functionality, ensuring that core features like tournament creation, student enrollment, and team formation are working correctly.

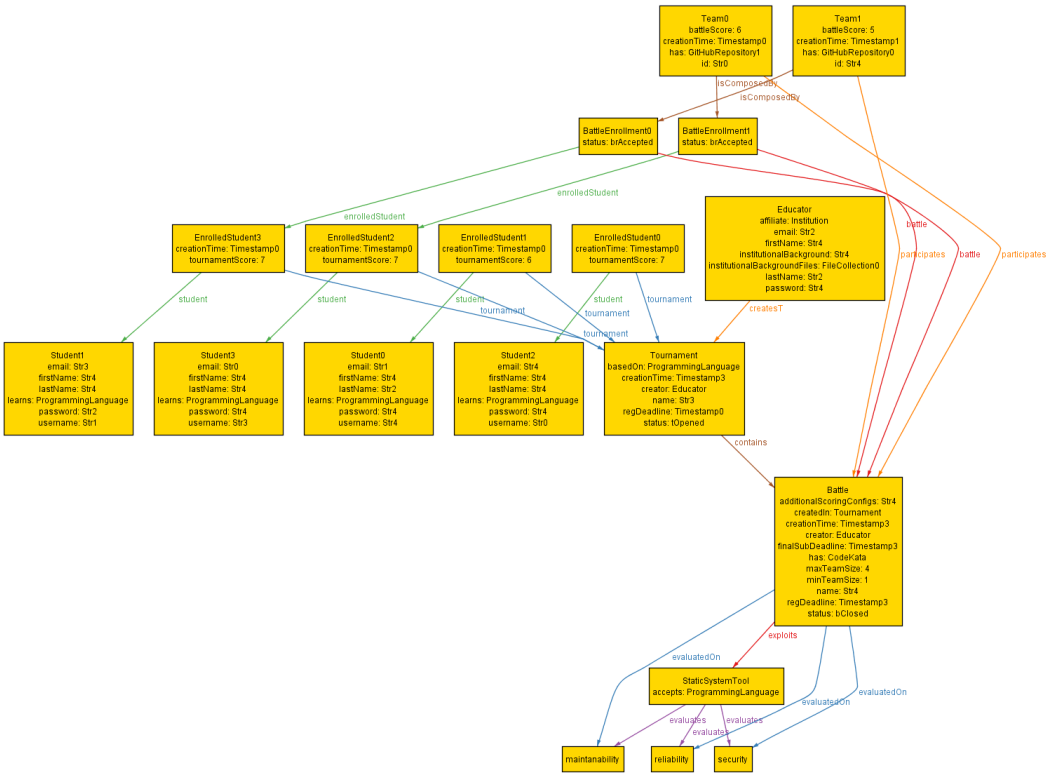

Figure 4.1: First World

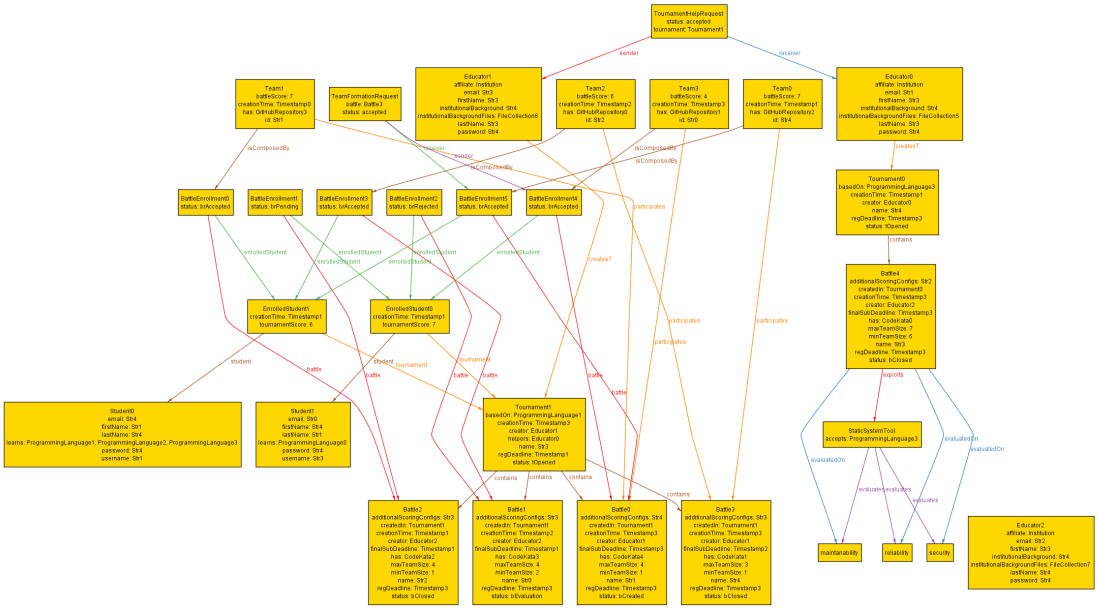

Figure 4.2: Second World

4.3. Second World

```
pred w2 {
        #Educator = 3
        #Tournament = 2
        #Battle = 5
        #Team = 4
        one t: TournamentHelpRequest | t.status = accepted
        one t: TeamFormationRequest | t.status = accepted
        one be: BattleEnrollment | be.status = brRejected
        one be: BattleEnrollment | be.status = brPending
}
```

Listing 4.5: Second world

In World 2, the complexity increases, representing a more mature and dynamic environment. There are three Educators, possibly indicating a collaboration among different instructors. also tournaments are increased.

The system now manages five Battles, indicating a variety of challenges or stages within the tournaments. The presence of different statuses for Tournament Help Requests and Team Formation Requests (one accepted and others in various stages) reflects a more dynamic interaction among participants.

4.4. Third World

```
pred w3 {
        #Educator = 3
        #Tournament = 4
        #Battle = 5
        #Team = 6
        one t: TournamentHelpRequest | t.status = accepted
        one t: TournamentHelpRequest | t.status = rejected
        one t: TeamFormationRequest | t.status = accepted
        one be: BattleEnrollment | be.status = brRejected
        one t: TeamFormationRequest | t.status = pending
}
```

Listing 4.6: Third world

World 3 presents the most complex and fully-featured scenario, suitable for a large-scale, multifaceted educational environment or a competitive programming league.

With five Battles included, the system demonstrates its ability to manage multiple challenges or stages within each tournament.

The model shows six Teams, highlighting significant student participation. This number suggests the formation of varied team structures, possibly encouraging collaboration, peer learning, and competition among a larger group of participants.

The inclusion of diverse statuses for Tournament Help Requests and Team Formation Requests (accepted, rejected, pending) illustrates a dynamic and interactive system. This aspect could involve complex decision-making processes, where educators and students negotiate roles, team compositions, and responsibilities. It represents a real-world scenario where logistics, personal preferences, and strategic considerations play a significant role.

Six Teams indicate a high level of student engagement and the potential for intricate team dynamics and strategies. The range of statuses for Tournament Help Requests and Team Formation Requests, including accepted, rejected, and pending, reflects a highly interactive and flexible system.

Finally, the presence of Battle Enrollments with different statuses (brRejected, brPending) adds another layer of complexity. It shows the system's capacity to manage individual student journeys through the tournaments, accounting for various outcomes and scenarios in the competitive process.

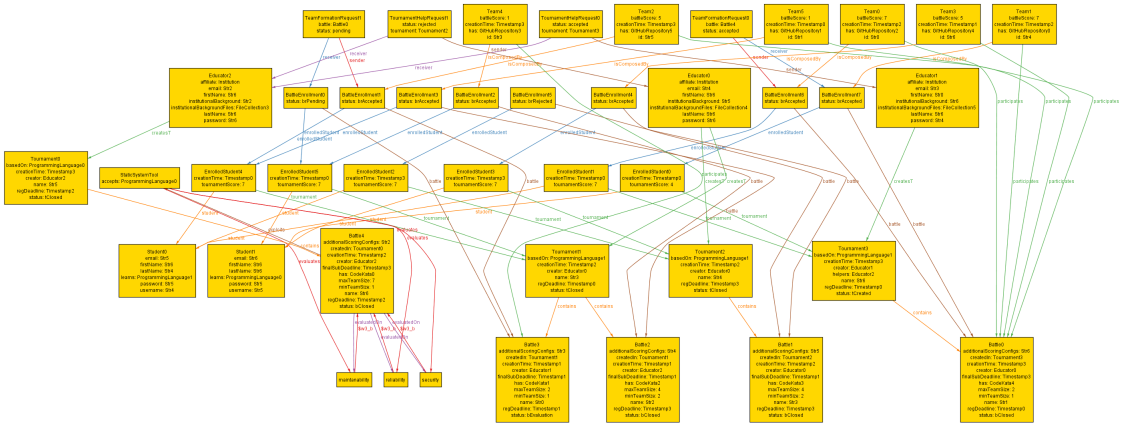

Figure 4.3: Third World

5 Effort Spent

5.0.1. Effort-Topazio

Hours	Description
8	Discussion about the Assignment 2023-2024
2	Document structure, Scope, and Purpose
6	Scenarios and Product Functions
8	Use Cases, Use Case Diagrams, Sequence Diagrams
3	User characteristics and Domain Assumptions
7	Class Diagram
2	Hardware, User, Communication Interfaces
10	Functional Requirements
4	Mapping Requirements
2	Performance Requirements and Design Constraints
2	Software System Attributes
10	Alloy

Table 5.1: Effort Spent - Topazio

5.0.2. Effort-Gigante

Hours	Description
8	Discussion about the Assignment 2023-2024
3	Document structure, Scope, and Purpose
4	Scenarios and Product Functions
8	Use Cases, Use Case Diagrams, Sequence Diagrams
5	User characteristics and Domain Assumptions
8	Class Diagram
2	Hardware, User, Communication Interfaces
8	Functional Requirements
2	Mapping Requirements
2	Performance Requirements and Design Constraints
2	Software System Attributes
15	Alloy

Table 5.2: Effort Spect - Gigante

List of Figures

2.1 CKB CLass Diagram 21
2.2 Battle lifecycle State Chart Diagram 21
2.3 Tournament lifecycle State Chart Diagram 22
2.4 Student Request Lifecycle State Chart Diagram 23
2.5 Tournament Help Request Lifecycle State Chart Diagram 24
2.6 Team Creation Request Lifecycle State Chart Diagram 25

3.1 User Interfaces 32
3.2 User Interfaces: Student Personal Area 33
3.3 User Interfaces: Educator Personal Area 34
3.4 User Interfaces: Student Tournament Page 35
3.5 User Interfaces: Educator Tournament Page 36
3.6 Student Use Case Diagram 38
3.7 Educator Use Case Diagram 39
3.8 Sequence Diagram 1: User Registration 58
3.9 Sequence Diagram 2: User login 59
3.10 Sequence Diagram 3: Tournament Creation 60
3.11 Sequence Diagram 4: Tournament Enrollment 61
3.12 Sequence Diagram 5: Battle Search 62
3.13 Sequence Diagram 6: Join battle individually 63
3.14 Sequence Diagram 7: Group formation 64
3.15 Sequence Diagram 8: Send Battle Invite 65
3.16 Sequence Diagram 9: Join Battle by Invitation 66
3.17 Sequence Diagram 10: Change Languages Settings 67
3.18 Sequence Diagram 11: Battle Creation 68
3.19 Sequence Diagram 12: Commit & Score Update 69
3.20 Sequence Diagram 13: Close Tournament 70
3.21 Sequence Diagram 14: View Battle Score 71
3.22 Sequence Diagram 15: View Tournament Score 72
3.23 Sequence Diagram 16: Manual Evaluation 73

4.1 First World 100
4.2 Second World 101
4.3 Third World 103

List of Tables

3.1 Use Case 1 - Register as a Student 40
3.2 Use Case 2 - Register as an Educator 41
3.3 Use Case 3 - Login as a Student 42
3.4 Use Case 4 - Login as an Educator 43
3.5 Use Case 5 - Create a tournament 44
3.6 Use Case 6 - Tournament Enrollment 45
3.7 Use Case 7 - Looking for a specific battle 46
3.8 Use Case 8 - Join a battle individually 47
3.9 Use Case 9 - Join a battle - group formation 48
3.10 Use Case 10 - Send battle invitation 49
3.11 Use Case 11 - Join a battle by invitation 50
3.12 Use Case 12 - Updating Preferred Programming Languages 51
3.13 Use Case 13 - Creating a battle on CKB 52
3.14 Use Case 14 - Commit and battle Score Update on CKB 53
3.15 Use Case 15 - Closing a Tournament on CKB 54
3.16 Use Case 16 - Viewing battle scores 55
3.17 Use Case 17 - Viewing tournament scores on CKB 56
3.18 Use Case 18 - Manual Evaluation on CKB 57
3.19 Use Case vs. Requirement Matrix 79
3.20 Use Case vs. Requirement Matrix 80
3.21 Use Case vs. Requirement Matrix 81
3.22 Use Case vs. Requirement Matrix 82
3.23 Traceability Matrix for CKB Platform 83
3.24 Traceability Matrix for CKB Platform 84

5.1 Effort Spent - Topazio 105
5.2 Effort Spect - Gigante 106