

Requirement Analysis and Specification Document

Christian Rossi
Kirolos Sharoubim

Academic Year 2023-2024

Contents

1	Introduction	3
1.1	Purpose	3
1.1.1	Goals	3
1.2	Scope	3
1.3	Definitions, acronyms and abbreviations	5
1.3.1	Definitions	5
1.3.2	Acronyms	5
1.3.3	Abbreviations	5
1.4	Revision history	5
1.5	Reference documents	5
1.6	Document structure	5
2	Overall description	6
2.1	Product perspective	6
2.1.1	Scenarios	6
2.1.2	Domain class diagrams	7
2.1.3	State diagrams	9
2.2	Product functions	10
2.2.1	Student's functionalities	10
2.2.2	Educator's functionalities	11
2.3	User characteristics	11
2.3.1	Student	11
2.3.2	Educator	12
2.4	Assumptions, dependencies and constraints	12
2.4.1	Domain assumptions	12
2.4.2	Dependencies	13
2.4.3	Constraints	13
3	Specific requirements	14
3.1	External interface requirements	14
3.1.1	User interfaces	14
3.1.2	Hardware interfaces	21
3.1.3	Software interfaces	21
3.1.4	Communication interfaces	21
3.2	Functional Requirements	21
3.2.1	Use case diagrams	21
3.2.2	Use cases	25
3.3	Performance requirements	42

3.4	Design constraints	43
3.4.1	Standards compliance	43
3.4.2	Hardware limitations	43
3.5	Software system attributes	43
3.5.1	Reliability	43
3.5.2	Availability	43
3.5.3	Security	43
3.5.4	Maintainability	43
3.5.5	Portability	44
4	Formal analysis with Alloy	45
5	Effort spent	46
6	References	47

Chapter 1

Introduction

1.1 Purpose

CodeKataBattle is a platform employed by educators to challenge students on code Katas, problems meant to be tackled using a programming language selected by the organizers of the challenge.

Specifically, educators have the capability to establish a code Kata within a particular tournament by defining the following parameters: the problem to be solved (with some test cases), the minimum and maximum number of students per group, and the deadlines for code Kata registration and solution submission.

After the creation of the tournament, students can form groups and start their work on the solution, employing a test-first approach. When the ultimate deadline approaches, the system autonomously calculates the final rankings and unveils the winners.

1.1.1 Goals

The goals, which are the objectives to accomplish are the following

1. The system should allow students to join tournaments.
2. The system should allow students to join battles as groups or individually in a specific tournament.
3. The system should allow educators to manage a tournament.
4. The system should allow educators to manage a battle in a specific tournament.
5. The system should allow students and educators to see battles and tournaments rankings.

1.2 Scope

To ensure the proper development of this application, it is crucial to examine all imaginable phenomena linked to the system.

These phenomena can be categorized into three distinct categories: world phenomena (occurring in the external environment beyond the machine's control), machine phenomena (taking place within the machine and beyond the influence of the external world), and shared phenomena (interactions and events involving both the system and the external world).

Regarding CodeKataBattle, we encounter the following external phenomena:

1. The educator generates a code Kata problem description.
2. The educator formulates a series of test cases for the code Kata problem.
3. The students develop a program to solve the code Kata.
4. The educator assigns a personal score based on code quality.

Additionally, the following internal phenomena arise within the system:

1. The system creates a GitHub repository.
2. The system establishes an automated workflow through GitHub Actions to notify CKB about new commits.
3. The system analyzes the repositories at every commit.
4. The system conducts automated evaluation and updates the score according to the following criteria:
 - (a) Number of passed test cases.
 - (b) Timeliness between the start of the challenge and the last commit on the main branch.
 - (c) Static code analysis for security, reliability, and maintainability.

Lastly, the following shared interactions and events emerge:

1. The educator initiates a battle on the platform, including the following steps:
 - (a) Uploading the code Kata.
 - (b) Specifying the minimum and maximum number of students per group.
 - (c) Setting a registration deadline.
 - (d) Setting a final submission deadline.
 - (e) Configuring additional scoring parameters.
2. Students submit their implementations to the platform via GitHub commits.
3. The educator creates a tournament.
4. The educator grants permissions to other educators to create code Kata within a specific tournament.
5. Students subscribe to specific tournaments.
6. The student joins an already existing group.
7. The student forms a new group and extends invitations to other students.
8. The educator concludes the tournament, and the system notifies the students.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

Code Kata: adaptation of the concept of karate katas, where you repetitively refine a form, to the realm of software development, fostering iterative practice and improvement.

Test-first approach: software development process relies on the transformation of software requirements into test cases before the software is completely developed, and it involves monitoring the entire software development by iteratively testing the software against all these test cases.

1.3.2 Acronyms

CKB: CodeKataBattle

CK: Code Kata

1.3.3 Abbreviations

1.4 Revision history

Version 1.0 - Release - date TBD

1.5 Reference documents

Document 1 - Presentation about RASD structure

Website 1 - <http://codekata.com>

Website 2 - <https://en.wikipedia.org>

1.6 Document structure

This document contains the following elements:

1. **Introduction:** this section offers a general overview of CodeKataBattle and its objectives.
2. **Overall description:** this section provides comprehensive information about the system, including interfaces, constraints, domain assumptions, software dependencies, and user characteristics.
3. **Specific requirements:** this section outlines the system's functionalities through scenarios and use case diagrams.
4. **Formal analysis with alloy:** this section contains the Alloy employed to verify the platform's correctness.
5. **Effort spent:** this section details the amount of time (in hours) contributed by each group member.
6. **References:** this section lists the tools used in the document's development.

Chapter 2

Overall description

2.1 Product perspective

2.1.1 Scenarios

1. *New user joins the platform*

Maria is a Computer science student. Maria wrote a simple program in Java for a school project a month ago, and realised she really enjoys it. Thus, she is looking for a chance to improve her Java skills as she didn't feel always confident about all the code she wrote. While searching online, Maria finds the CodeKataBattle platform and, as some challenges were exactly what she was looking for, she signs up: she inserts her email and a password and proceeds as a student. A verification email is sent and, once the mail is verified by clicking the link in the verification email, she can start browsing the available tournaments looking for an interesting battle.

2. *Educator creates a new tournament*

Anne is a professor in an engineering university. She teaches a course about optimization of problems' solution. One day she hears from a colleague of her, who's a computer science professor, about a platform called CodeKataBattle, which allows students to compete solving programming problems. Anne enjoys writing some python scripts to solve simple problems of everyday life and, while writing the scripts, she noticed that they required a deeper understanding of the solution. Since she knows that her students enjoy programming, as herself, Anne decides to create a tournament on CodeKataBattle, with the objective to make her students implement algorithms for the methods of problem-solving taught in the course. At first, she was not sure that she could make something so complex but the fact that the platform could assign automatically the ranking needing only some test cases and that multiple programming languages were supported made Anne change her mind. Thus, Anne creates a new tournament by setting a name and a deadline. All the students subscribed to CodeKataBattle are notified by the platform and can now subscribe.

3. *Educator adds a CK to a specific tournament*

Jamie is a high school programming teacher; he is a Python and JavaScript expert but also knows Java. Jamie has been using CodeKataBattle for a while, since it helps his students to challenge themselves and provides an interactive programming exercise. Every year a CodeKataBattle tournament is held by the professor and based on the final ranking the students get some extra points in the exam grade.

For this year, Jamie just found out an interesting problem that can be solved in various ways, each of them exploiting different proprieties of python. Thus, he decides to upload the problem in the tournament, not allowing students to join in groups since he wants to test each student individually.

Therefore, Jamie creates five test cases, sets the deadline date and proceeds to upload the CodeKata, the registration deadline, the submission deadline and an additional scoring criteria on cleanliness of the code.

4. *Student joins a group for a specific CK in a tournament*

Donald is a car wash employee, even if things are going great his dream was to become a web developer. Donald already learned a lot JavaScript and is always in search of new challenges. Donald knows that in programming environment team working is a must. Since he always studied and exercised alone on CodeKataBattle, he now wants to practice some team work. So Donald opens CodeKataBattle and looks for some tournaments where battles in numerous groups are allowed. Once he found some battles about topics he's interested in, where groups are allowed, he creates a group and waits for other students to join.

5. *Student does a commit*

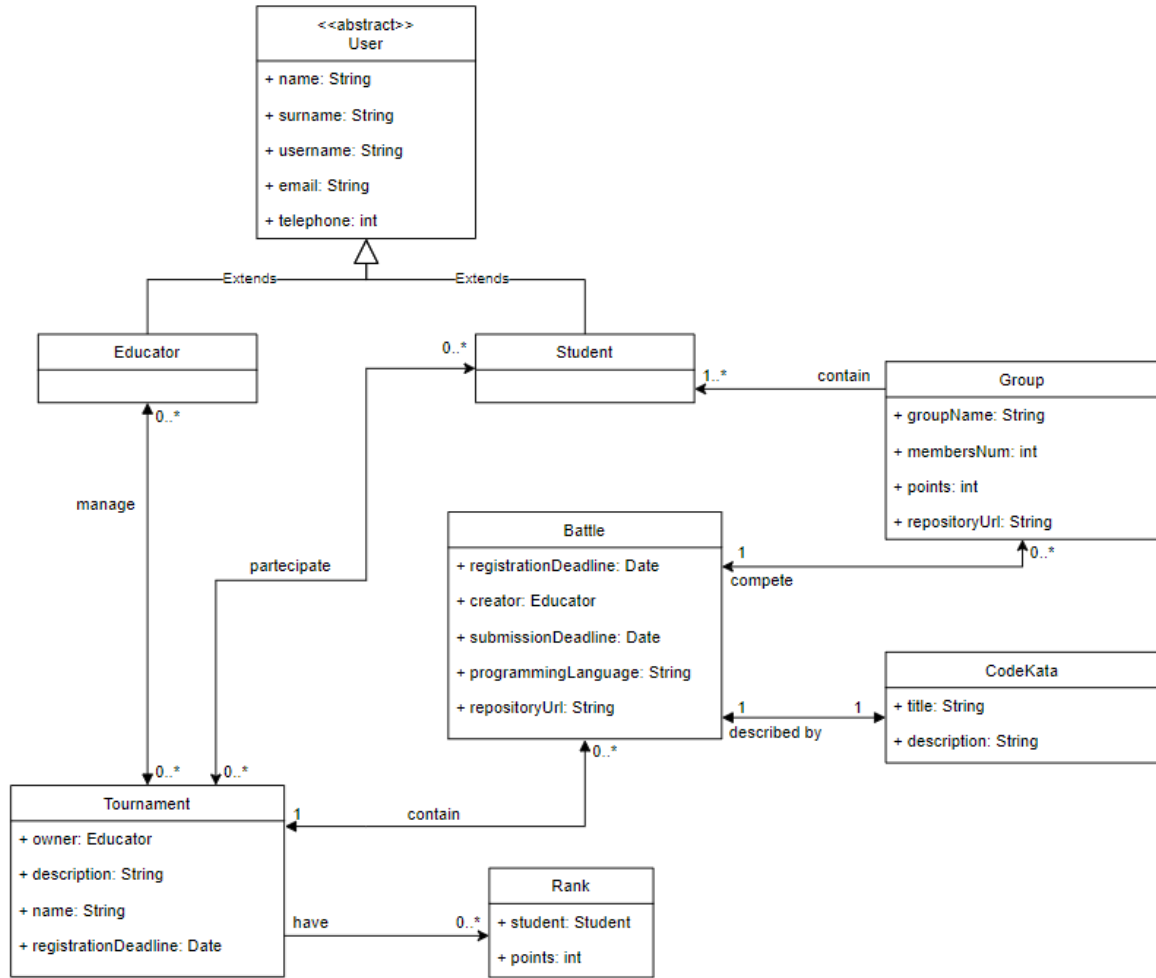
Charlotte and Matt are working on the solution of a battle in their teacher tournament. They thought every test was passed, but then, they realised there might be a particular case in which their algorithm fails. Thus, they fix the bug and, once they think the algorithm is done, the changes are committed and pushed. At this point the CodeKataBattle platform performs the tests, the static analysis and the time ranking, giving the two students a new higher rank, confirming that there was a bug in their code.

6. *Educator closes the tournament*

Jeremiah is an Educator on CodeKataBattle, each month he creates a tournament about a simple problem, with a couple battles, one is usually a simple exercise, while the second is a more complex variation of the first one. As the month is coming to its end Jeremiah opens CodeKataBattle platform and checks how the participants are doing and decides to close the tournament. The platform then proceeds to assign the final ranking and score of each student subscribed to the tournament and makes them available for everyone to consult.

2.1.2 Domain class diagrams

The domain class diagram for CodeKataBattle is presented below, covering all the elements within the system's operational environment and illustrating their interactions.



The main elements of the UML Class diagram presented above are:

- *Educator*: every *Educator* has a name, surname, username, email, and telephone. The username must be unique. The *Educator* is able to manage multiple *Tournament*.
- *Student*: every *Student* has a name, surname, username, email, and telephone. The username must be unique. The *Student* is able to participate in multiple *Battle*, that is part of a certain *Tournament*. The *Student* is part of a *Group* for a given *Battle*.
- *Group*: every *Student* that participate in a *Battle* must set up a group with the number of *Student* fixed by the rules. Each group is created for a specific *Battle* within a *Tournament*. Different *Battle* within the same *Tournament* may have different *Group*. The *Student* that creates a group have to set up a groupName. The system automatically set the groupId (univoque) and the membersNum based on the *Battle*'s rule. It has a relation with *Student* and *Battle*, since it is formed by one or more *Student* and is created in the context of a specific *Battle*. It has also a relation with *GroupRepository* since every *Group* needs a repository to participate in a given *Battle*.
- *GroupRepository*: this class contains the URL of the repository associated to each group. This is used by the system to know every repository that is subscribed to a given *Battle*. The system checks (after a notification by GitHub Actions) the repository and the code and updates the *Rank* points.

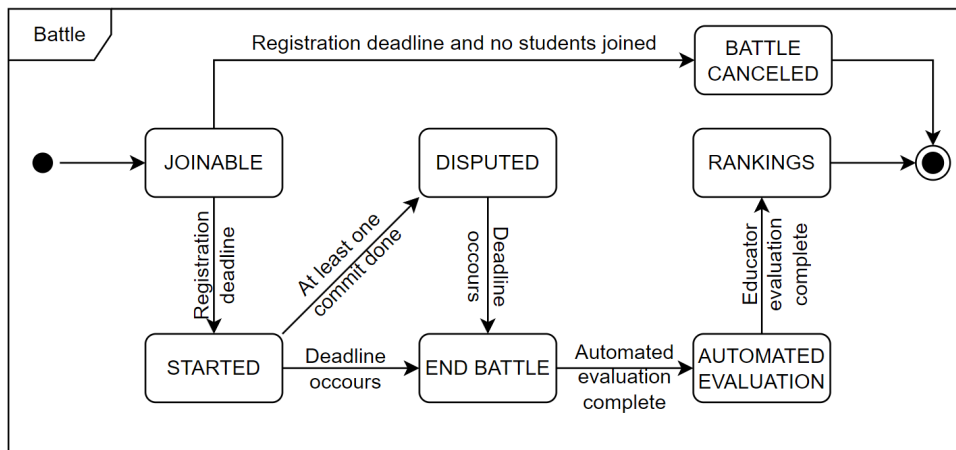
- *Battle*: this class contains a single *Battle* within a *Tournament*. The *Battle* has a registration and submission deadline, an univoque ID, the requested programming language, and the URL of the main repository from which users have to fork their own. The *Battle* is created by a single *Educator*, so it specifies who created it with the creator attribute. Each *Battle* may have multiple *Group* and, consequently, multiple *Student*. Each battle is part of a specific *Tournament* and has a *CodeKata* that describes the problem. Every battle has a *Rank*, that updates during the *Battle* itself.
- *CodeKata*: is the description of a problem within a *Battle*. It contains the name of the problem and a textual description of it. Each *Battle* has exactly one *CodeKata*.
- *Tournament*: each *Tournament* is created by an *Educator* that becomes the owner of it. It has a unique ID, a textual description on the general topic, and the name. It also has a deadline for the whole *Tournament*. The manage relation indicates the *Educators* that manage the *Tournament* in cooperation with the owner of it. The *Tournament* have a relation with *Battle* and *Student* it is composed by some *Battles* and one or more students are enrolled. Finally, it has also a general *Rank*, that specifies the rankings of all battles combined.
- *Rank*: it comprises a points attributes (that ranges between 0 and 100) and the group linked to this value. Each *Tournament* and *Battle* has a list of *Rank* that, ordered by the points attributes return the actual standings.

2.1.3 State diagrams

They are now presented the UML State diagrams that defines the possible states for battle and a tournament. These two scenarios are analyzed as they are the core part of the whole application. In particular, an educator can create a tournament by setting all the rules and its battles. Then the tournament has his own lifecycle shown in the diagram. In each tournament there can be multiple battles. At a certain point of the tournament there can be battles in different states.

Battle state diagram

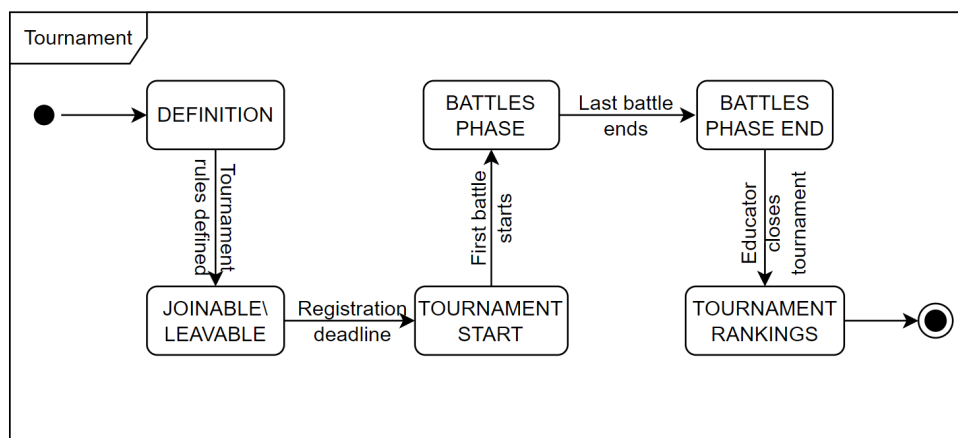
The diagram that represents the possible states of a battle is depicted below:



Description After the creation of a battle, the students enrolled in the battle's tournament can join alone or in groups (*Joinable*). At registration deadline there are two possibilities: there are at least one student enrolled or there are none. In the former scenario the battle starts (*Started*), while in the latter the battle ends immediately without final ranking (*Battle canceled* and *Final state*). When the battle starts each student or group can create commits and gain points for the battle (*Disputed*). However, when the deadline occurs the battle is closed and both groups and student with and without commits are evaluated (*End battle*). Obviously, the groups without commits will gain zero points. The solutions undergo the automated evaluation (*Automated evaluation*) and then the educator's evaluation (*Rankings*). After all this process the final rankings are shown and the winner is revealed.

Tournament state diagram

The diagram that represents the possible states of a tournament is shown below:



Description After the creation of a tournament, the educator can invite other educator to manage it together (*Definition*). While doing this the educator has also to set up the rules for the tournament and the termination date (*Joinable/leavable*). Note that the educators can create battles also after the creation of the tournament. When the registration deadline occurs the tournament starts and students not enrolled cannot join the tournament anymore (*Tournament start*). After the tournament starts, the battles can also commence. When the first battle starts (*Battle phase*) the students can compete in the open battles and have to wait the opening date for the ones not yet started. When the last battle ends (the deadline must be anterior to the tournament ending) (*Battle phase end*), it is possible to compute the scores for all the students. When the tournament's deadline occurs, the educator closes the tournament itself and can finally reveal the final winner.

2.2 Product functions

The functionality offered by CodeKataBattle differs based on the user that is going to interact with it. Thus, the functionalities are presented based on the user.

2.2.1 Student's functionalities

The educator is the user that participates to the tournaments and the battles. After the login the system must redirect the student to the page that offers the following functionalities:

- *Join tournaments*: the user must be able to access a list of opened tournament that can be joined. This list has also to show if the user is eligible for the opened tournaments.
- *Active tournaments*: the user can view all the tournaments in which he is enrolled, with the relative termination dates and the active battles within it.
- *Tournament details*: the user can view the details of a tournament. It can see the status of all battles and the relative rankings.
- *Battles details*: the user can view the details of each battle. The details consist, if the battle is open, in the problem description and test files. It can check the actual ranking, the points it has received, and the details about commits it has done. If the battle is closed it can see the final ranking. If the battle is not yet open he can view only the topics and the other user enrolled, but not the problem itself.

2.2.2 Educator's functionalities

The educator is the user that manages the tournaments and the battles. After the login the system must redirect the educator to the page that offers the following functionalities:

- *Create tournaments*: the user can create a new tournament by setting the rules and adding other educator to help it manage the competition. After setting up collaborators and rules, it (and the collaborators) can add battles to the tournament. When this phase ends the tournament is ready to be seen by the students.
- *Add battle*: the user must have the possibility to add a new battle also after the creation of the tournament. This functionality allow, by selecting a managed tournament, to create a new battle with all the rules and add it to the tournament.
- *Manage tournament*: after the creation of the tournament the educator can check the status until the end and close it when the time expires.
- *Manage battle*: when a battle in a certain managed tournament is active the user can see the actual rankings and the commit status of each participant. If specified by the rules, when the time expires he must check the final submissions for each group and assign points based on the quality and finally close the battle. If the manual check is not in the rules he can directly close the battle when the time expires.

2.3 User characteristics

The platform accommodates interaction from two user categories: students and educators. The initial user category engages in Code Kata tournaments, while educators, on the other hand, serve as the organizers of these tournaments.

2.3.1 Student

The student, as an individual aiming to secure victory in a specific battle, may need to either join an existing team or establish one with fellow students, considering that each battle has a predetermined group size.

In particular, a student requires access to the following features within the system:

- Student login.
- Personalized user interface.
- Access to the list of available tournaments.
- The ability to join a desired tournament if it's open for registration.
- Capability to create and join groups.
- Access to the GitHub repository link for a specific tournament.
- Access to both interim and final tournament rankings.

2.3.2 Educator

The educator's role encompasses the creation and management of tournaments, with the potential involvement of other educators designated by the tournament's owner, who is typically the initiating educator. All educators should possess the capability to create code Katas for the tournaments they manage and provide additional points for evaluation.

To facilitate these responsibilities, educators need access to the following features within the system:

- Educator login.
- Personalized user interface.
- Access to a list of tournaments under their management.
- The possibility to create new tournaments.
- The ability to create new battles with customized rules.
- Within a managed tournament:
 - The option to invite other educators to participate.
 - The capability to add new code Katas.
 - The authority to close the tournament.
 - The ability to assign extra points if specified in the battle rules.
- Access to the GitHub repository for every student in the active tournament.
- Access to both interim and final tournament rankings.

2.4 Assumptions, dependencies and constraints

2.4.1 Domain assumptions

The domain assumptions that we have to make are:

- All the users must register with an email associated with a GitHub account.
- The educators must not commit on student's repositories.

- The students do not share their code with students of other groups.
- A student submits a code that is not malicious.
- Every user register itself to the website with the right role (student or educator).
- GitHub and the API always works correctly.
- The test cases uploaded by the educators are correct.
- The CodeKata problem proposed has a solution.
- The students fork the repository and set up an automated workflow GitHub Actions.

2.4.2 Dependencies

The dependencies that we have to make are:

- The system requires an internet connection to retrieve all the data.
- The system will use an external API to interact with GitHub to check for commits and to create the repositories.
- The system will use an external API to send notifications to the user.

2.4.3 Constraints

The system shall be compliant to local laws and regulations, in particular users data should be treated according to the GDPR. The user must be able to retrieve the personal data given to the site at the time he needs them. The data collected in the registration phase must be the strictly necessary ones.

All information given from the user must be encrypted to protect them from malicious users. The APIs used by the application must be chosen based on security and reliability.

Chapter 3

Specific requirements

3.1 External interface requirements

In this section are presented all the interfaces required by the system to work as intended.

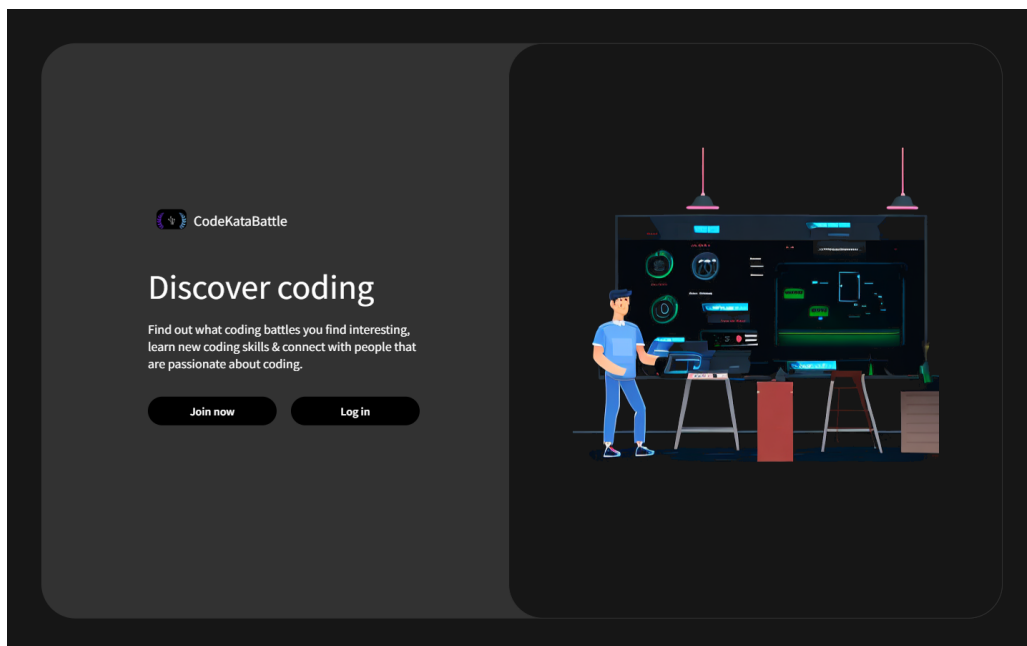
3.1.1 User interfaces

The platform will be implemented via a website that present a different interface based on the user type.

The website interface will be modified based on the type of hardware used (mobile or personal computer).

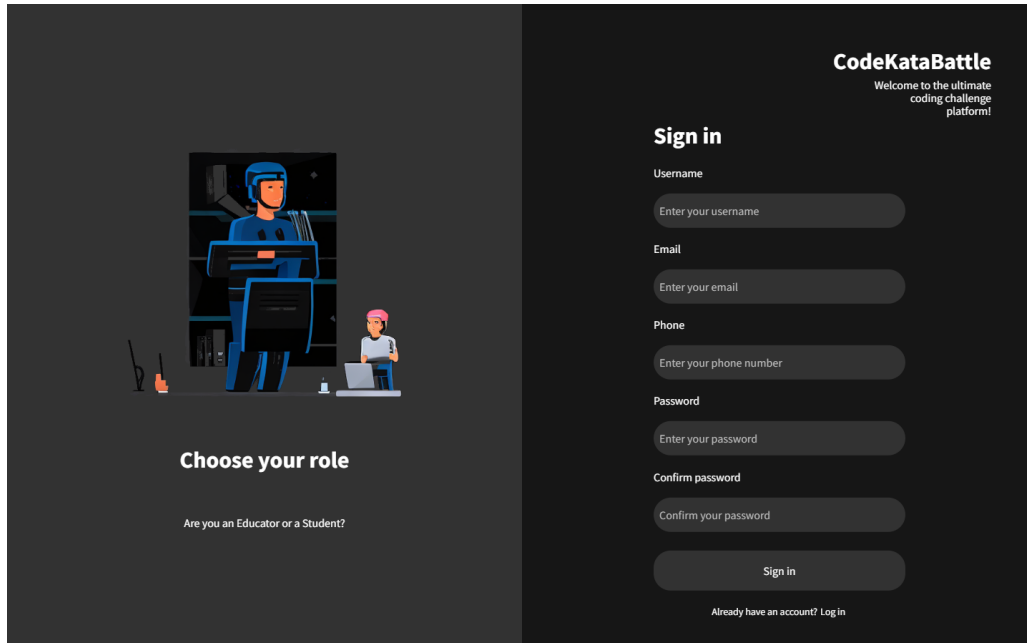
Welcome page

This is the page that is shown when the user is not registered or logged in. There is a brief description of the website and two buttons to let the user enter in the system.



Sign in page

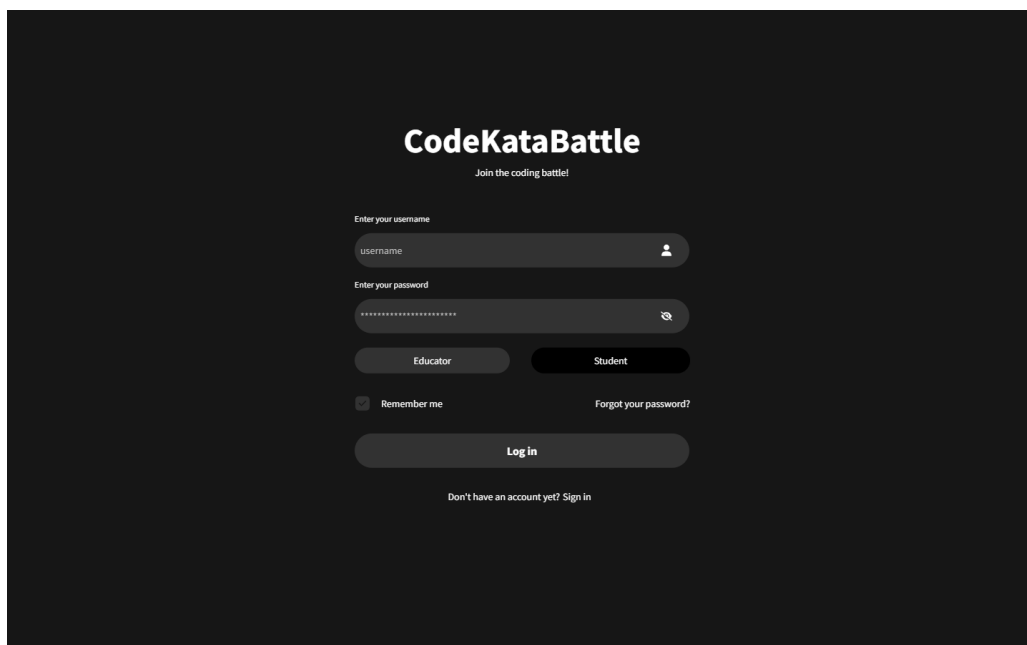
In this page the user can select the corresponding role on the left and add the information required to register. After completing all the information the user must click on the sign-in button to finally register.



The sign in page is split into two main sections. On the left, there is a dark-themed illustration of a person in a blue suit and helmet standing next to a desk with a laptop. Below the illustration, the text 'Choose your role' is displayed, followed by the question 'Are you an Educator or a Student?'. On the right, the 'CodeKataBattle' logo is at the top, with the tagline 'Welcome to the ultimate coding challenge platform!'. Below the logo, the heading 'Sign in' is followed by a series of input fields: 'Username' (with placeholder 'Enter your username'), 'Email' (with placeholder 'Enter your email'), 'Phone' (with placeholder 'Enter your phone number'), 'Password' (with placeholder 'Enter your password'), and 'Confirm password' (with placeholder 'Confirm your password'). A 'Sign in' button is located below these fields. At the bottom right, there is a link that says 'Already have an account? Log in'.

Login page

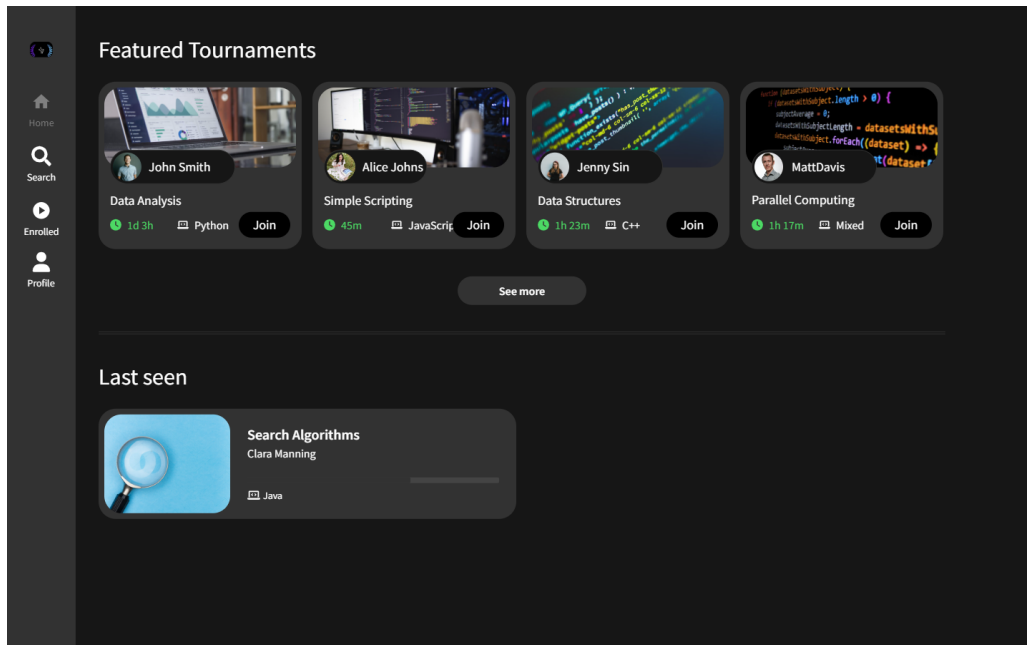
In this page the user must select the role, and insert the credentials if already registered in the system. If the credentials are correct the user can access the functionalities associated with him.



The login page features a dark background with the 'CodeKataBattle' logo at the top center, accompanied by the tagline 'Join the coding battle!'. Below the logo, there are two input fields: 'Enter your username' (with placeholder 'username' and a user icon) and 'Enter your password' (with a password icon). Below these fields are two buttons: 'Educator' and 'Student'. A 'Remember me' checkbox is located below the 'Educator' button, and a 'Forgot your password?' link is to its right. A 'Log in' button is positioned below the 'Remember me' checkbox. At the bottom center, there is a link that says 'Don't have an account yet? Sign in'.

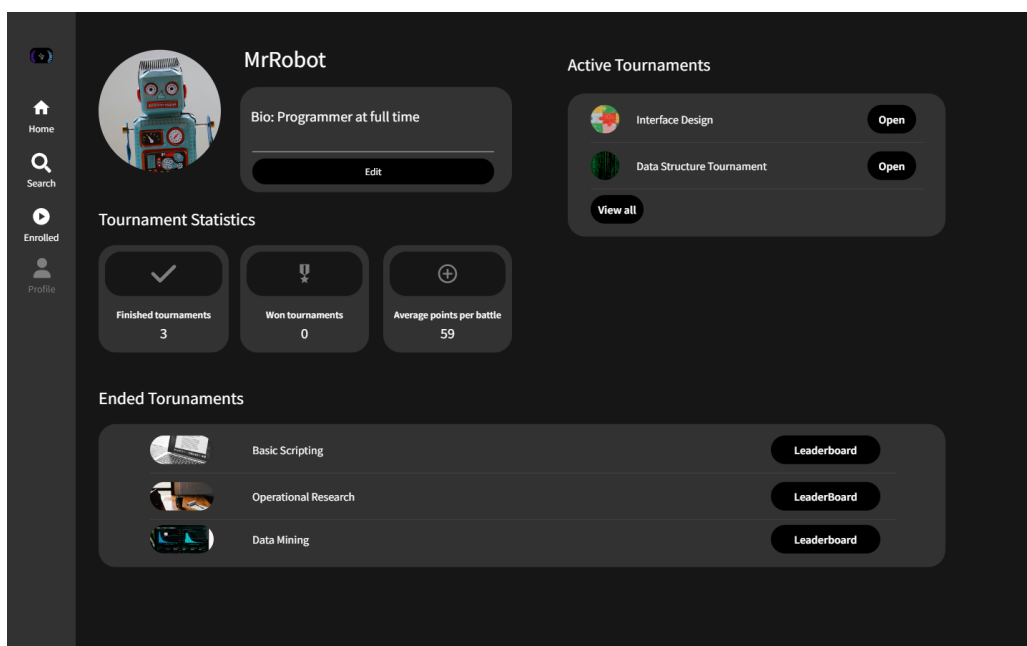
Home page

The home page shows all active tournament and the tournament checked before by the current user. If the home page is accessed by the Educator the buttons are for viewing details instead of joining the tournament.



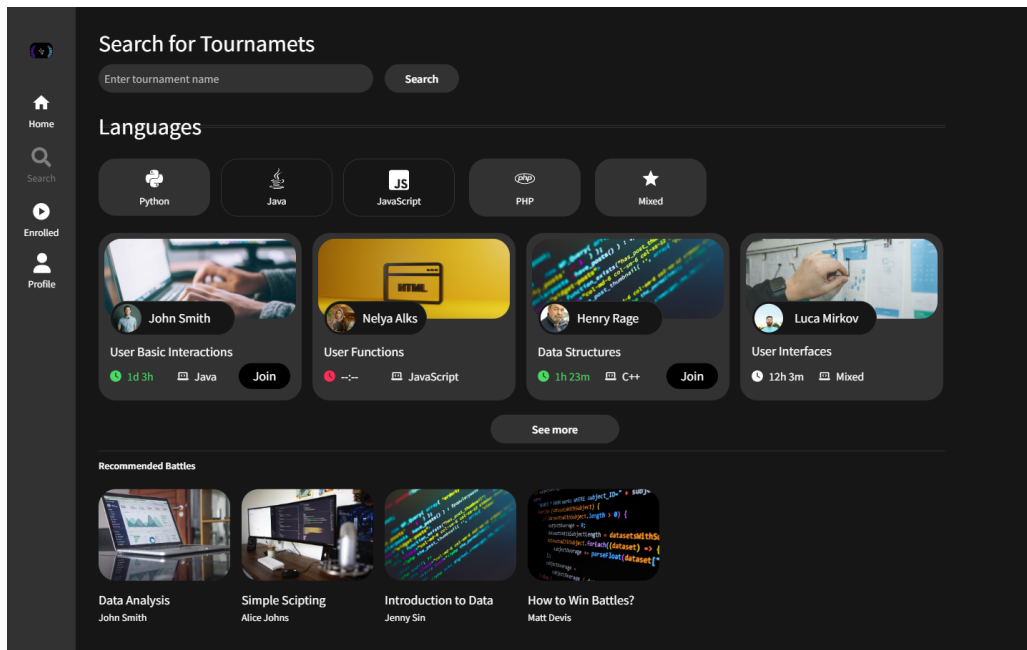
Profile page

In the profile page the users can modify the personal description and the image related to the account. Other functionalities for the Student are: check active tournaments, check personal statistics and view the details of ended tournament in which the student participated. The Educator has the same page (without statistics) and the tournament listed are the one who he is managing.



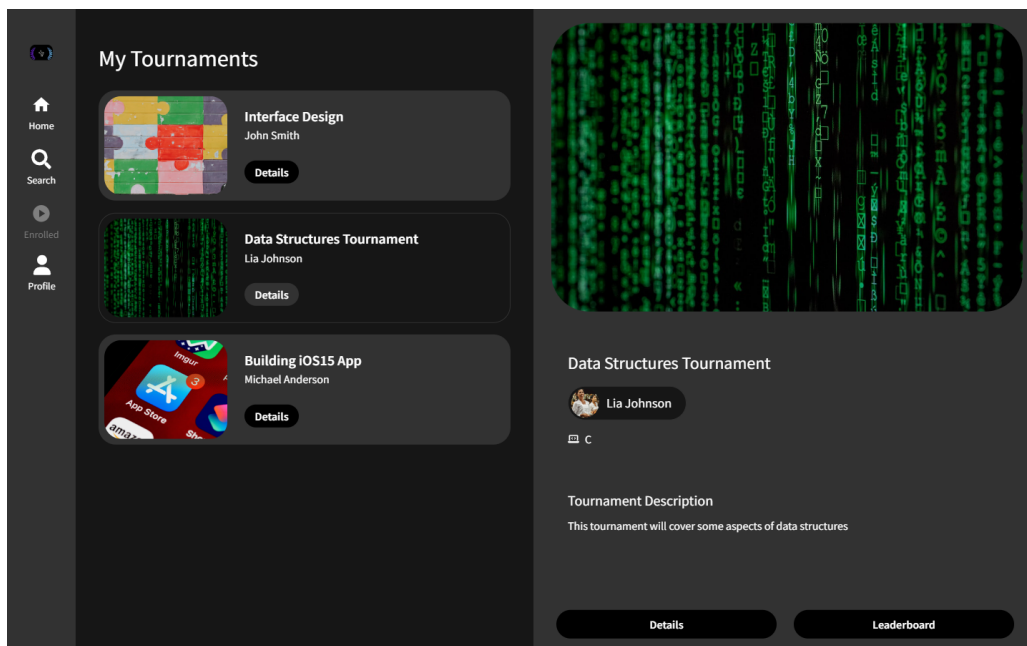
Search page

In this page the user can filter by choosing the desired language or by name. The first action can be done by selecting one of the button below "Languages", while the second is done with the bar at the top of the page. Also in this page the join button becomes a details button for the Educator. Finally, at the bottom of the page there is a section with recommended battles based on the previous researches.



Enrolled page

This page is only for Students. Here they can view the active tournaments where they are enrolled, and check the details (the battles within a tournament) and the current leaderboard.



Tournament details page

This page is for the Student and shows all the battles within a tournament. Here the student can join the single battles with a group made of one or more people.

Data Structures Tournament

Battles

- Sorting Alghoritm**
Ended | 60pt
- Dictionary implementation**
5h 12m
In progress | 12pt
- Binary Research Tree**
2d 5h
Starting Soon

Binary Research Tree

Starts in 2d 5h
C++
1 - 4
Code cleanliness

Battle Description
This battle is designed for coders who want to test their skills and compete against other talented programmers in a challenging coding tournament.

Join a group

Name	Creator	Participants
Winners_Ox	WarioMaster	1
TheCoders	CodingGod99	2
RobotArmy	JhonnyCode	3
W.C	XX_XX	1

Create your group

Group Name

Create and join battle

Join on your own

Leaderboard page

The leaderboard page shows the current ranking of the tournament selected with the details button in other pages.

Data Structures Tournament

1st Battle **2nd Battle** **3rd Battle** **Overall**

Group / Username	Platform Evaluation	Educator Evaluation	Points Earned	Rank	Final Score
Liam Harrison	100	90	190	1	95
Group 2	98	85	183	2	91
MrRobot	80	59	139	3	69
Programming_King_21	-	79	79	4	39
Group 13	26	48	74	5	37
Annabelle Ralph	20	26	46	6	23
Jonathan Bolt	13	25	38	7	19

Sorting Alghoritm | 1st battle
Ended | 60pt

Dictionary implementation | 2nd battle
5h
In progress | 12pt

Binary Research Tree | 3rd battle
2d 5h
Starting Soon

Create tournament page

In this page the Educator can set all the items needed to create a new tournament.

Tournament Name

Description

Enter tournament description

Language

Select a Language

Multilanguage

To add an image drag and drop it or choose a file from your computer

Upload

Add battles

Subscription deadline

January 2024

12:00

AM

PM

Invite other educators

Username or email

Invite

Create

Create battle page

In this page the Educator can set all the items needed to create a new battle within a tournament.

Battle Name

Description

Enter battle description

To add an image drag and drop it or choose a file from your computer

Upload

Language

Select a Language

Upload .fhware project with build automation scripts and test cases (compressed folder)

Upload

✓ Allow groups

Limit groups size

Enter the minimum group size to Enter the maximum group size

Manual evaluation

Enter manual evaluation criteria

Subscription deadline

January 2024

12:00

AM

PM

Commit deadline

January 2024

12:01

AM

PM

Create

Manage tournament page

In the manage tournament page the Educator can select the tournament that he manages and modify them by addin a battle, invite collaborators or close. It can also check the shared tournaments (where he is not the owner, but a collaborator).

My Tournaments

Numerical Approximation
In progress
Add battle Add coll... Close

Physics modelization
In progress
Add battle Add coll... Close

Expand

Shared Tournaments

Linear algebra problems
John Smith
Starts 12h 5m
Add battle

Physics modelization
Matlab
In this tournament we'll modelize real word scenarios using matlab

Battles

MTU visualization
12m : 6s
! Manual evaluation
Manage

RPS visualization
6h : 36m
Manage

Collaborators

AlbertEinstein
Jonn Oliver

Invite other educators
Username or email
Invite

Close Leaderboard

Manage battle page

In this page the Educator can check the status of the battles within a specific tournament. Here, he can check the final rankings and add the manual evaluation when needed.

Physics modelization

MTU visualization
12m : 6s
Details

RPS visualization
6h : 36m
Details

Matlab intro
Ended
Missing manual evaluation
Details Evaluate

Matlab intro
This fist battle will allow us to become more familiar with matlab. I will also give an evaluation in order to make you more counscious about mistakes

Add manual evaluation
📄 Readable code

Name	Number of	Auto rank	Repo link	Evaluation
CodingCrew	3	59	Link	39/100
FellowProgra	1	54	Link	36/100
JohnsAnne	2	89	Link	.../100
Turbo_coders	3	62	Link	.../100
Lukas_tesla	1	48	Link	.../100

Finish later Publish

3.1.2 Hardware interfaces

Both user must have a mobile device or a personal computer. Since the system will be implemented as a website the personal computer will be a better device for visualize better all the information.

Additionally, the challenge is to solve some problems using programming language, so the personal computer is necessary to compete (it is possible also with mobile devices but certainly more difficult).

3.1.3 Software interfaces

The system is a web application. Thus, the only software needed for both users is a modern web browser.

3.1.4 Communication interfaces

To work as intendend the system requires a reliable internet connection. The backend of the application will expose a unified RESTful API. This API is used to communicate with all users using HTTPS and TCP/IP protocols.

Furthermore, the system uses other two application program interfaces to ensure all its functionalities:

- Create a repository: CodeKataBattle needs to interact with GitHub in order to create a repository for each group participating to starting battles.
- Check repositories commits: CodeKataBattle needs to interact with the repositories linked to active battles in order to check all the new commits.
- Notification: CodeKataBattle needs to be able to send notification to the user to alert the users about important facts (e.g., the start or the end of a battle).

3.2 Functional Requirements

Initially we show the possible use cases for the interaction between the user and the application with the relative description and use case diagram. In the second part we list all the functional requirements as a mapping between the goals and the defined use cases.

3.2.1 Use case diagrams

An unregistered user must be able to sign up. The registration phase is different according to the role of the user. If it is a student it has to get the corresponding functionalities (i.e., participate in tournaments). If it is an educator it has to get the corresponding functionalities (i.e., create and manage tournaments). The corresponding Use Case Diagrams are shown below:

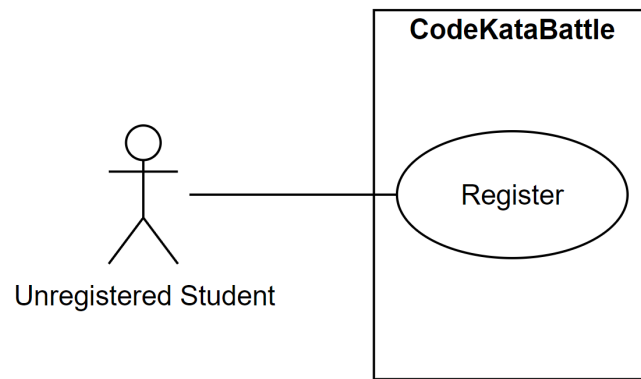


Figure 3.1: Use Case diagram for unregistered student

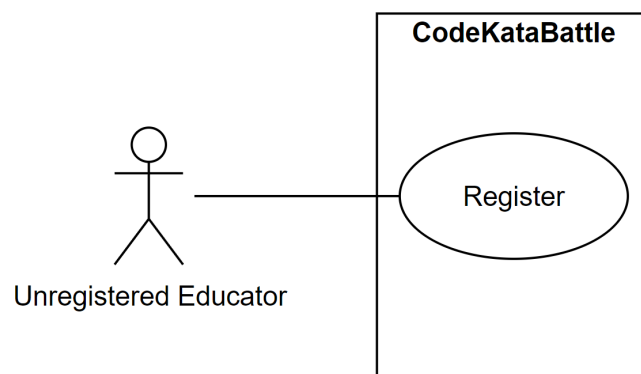


Figure 3.2: Use Case diagram for unregistered educator

Now, we analyze all the functionalities needed for both the students and the educators after the registration in the flowing two Use Case diagrams:

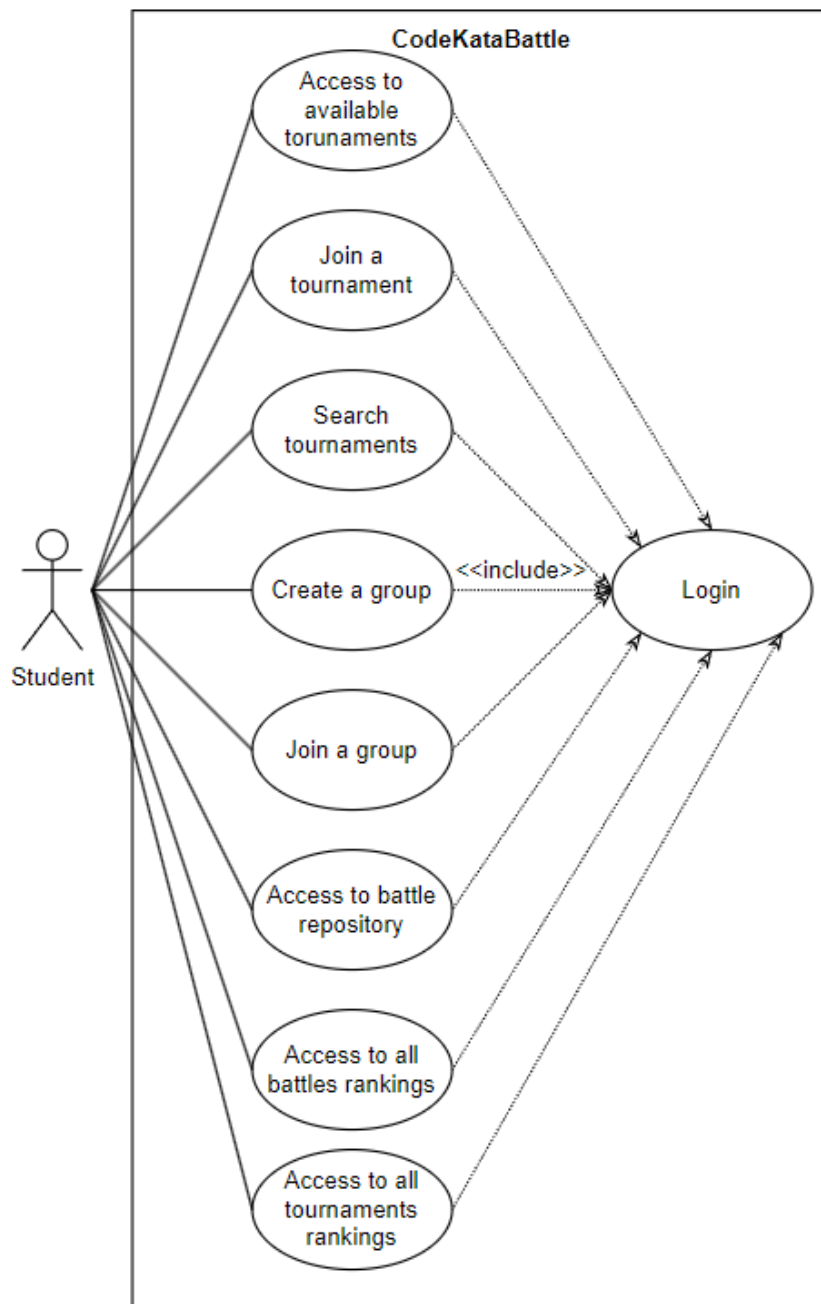


Figure 3.3: Use Case diagram for unregistered educator

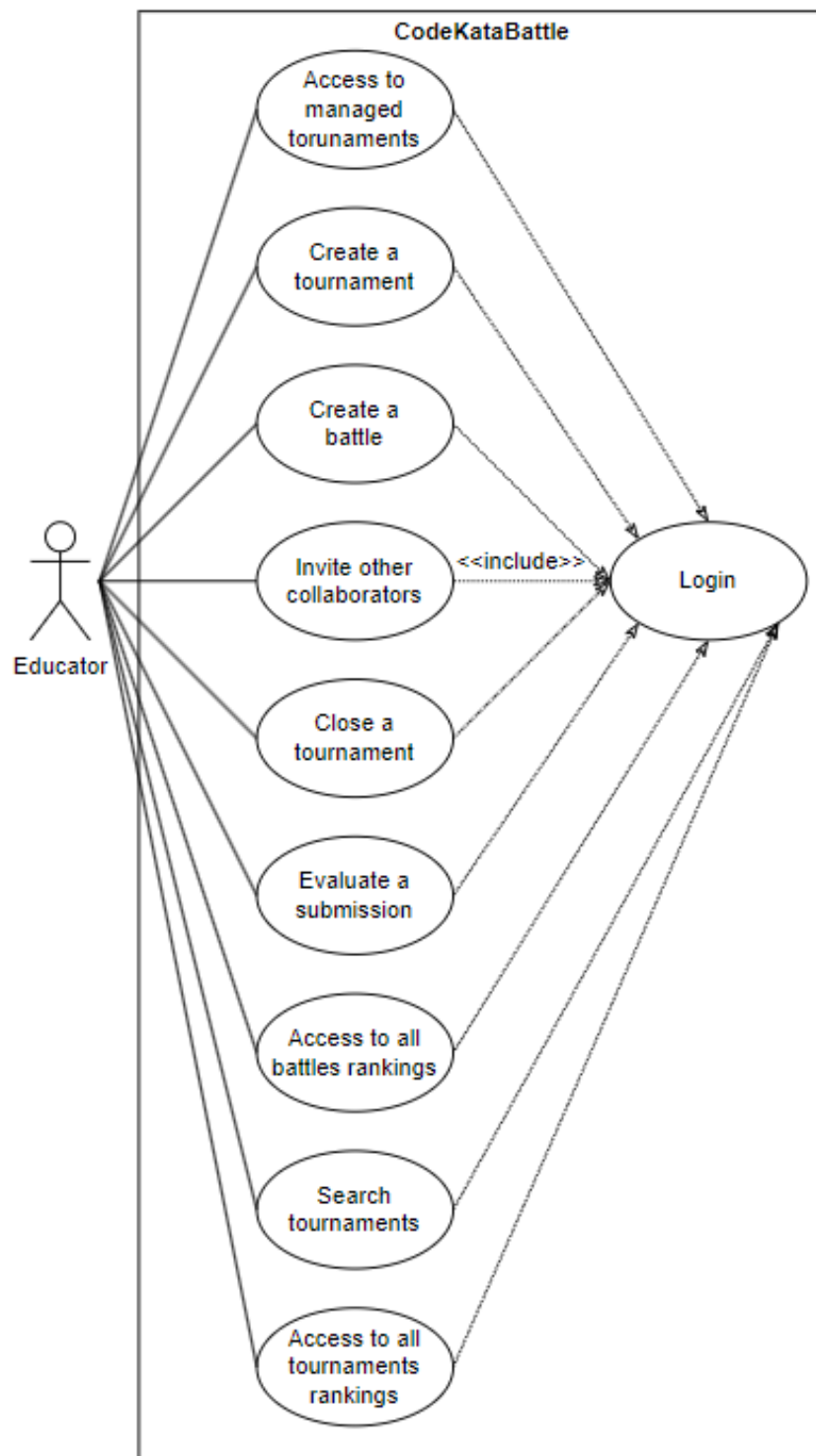


Figure 3.4: Use Case diagram for unregistered educator

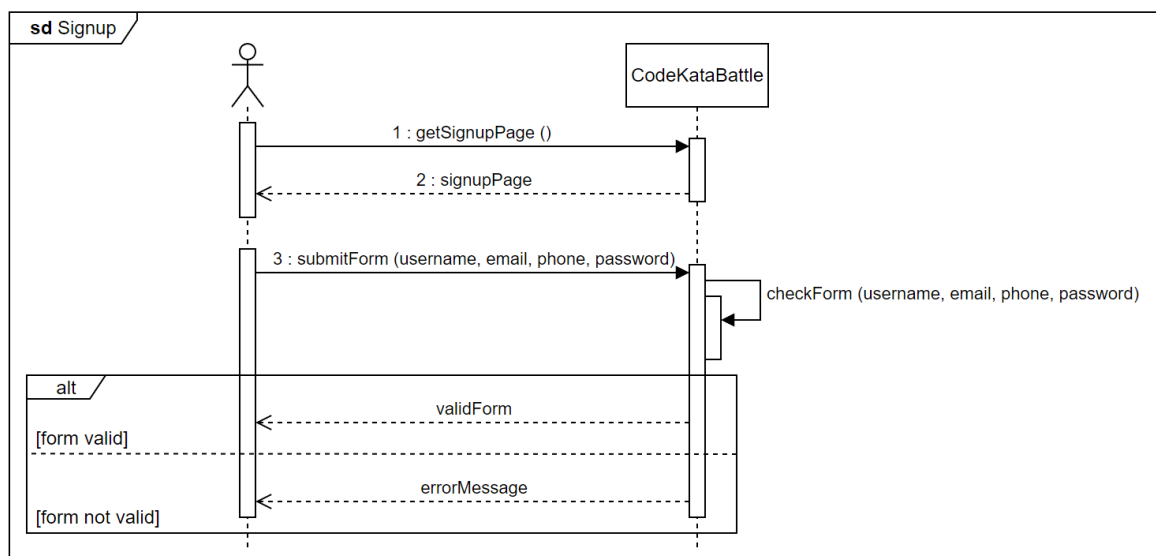
3.2.2 Use cases

Use case [U1]: Student registration

Actor(s)	Student
Entry Condition	The actor clicks on the register button
Event Flow	<ol style="list-style-type: none">1. The actor requires the sign-in page by clicking the join now button in the welcome page2. The system shows the sign-in page to the actor3. The actor inserts required personal information and sends it to the system4. The system processes the information and shows a success message redirecting the user to the login page
Exit Condition	The actor is registered correctly, and the login page is displayed
Exceptions	<ul style="list-style-type: none">• An already existing username, email or phone number are submitted• A required registration field is missing or invalid when the form is submitted
Notes	In case of exception the system will notify user with a human-readable message; the system is not able to check if the correct role was selected

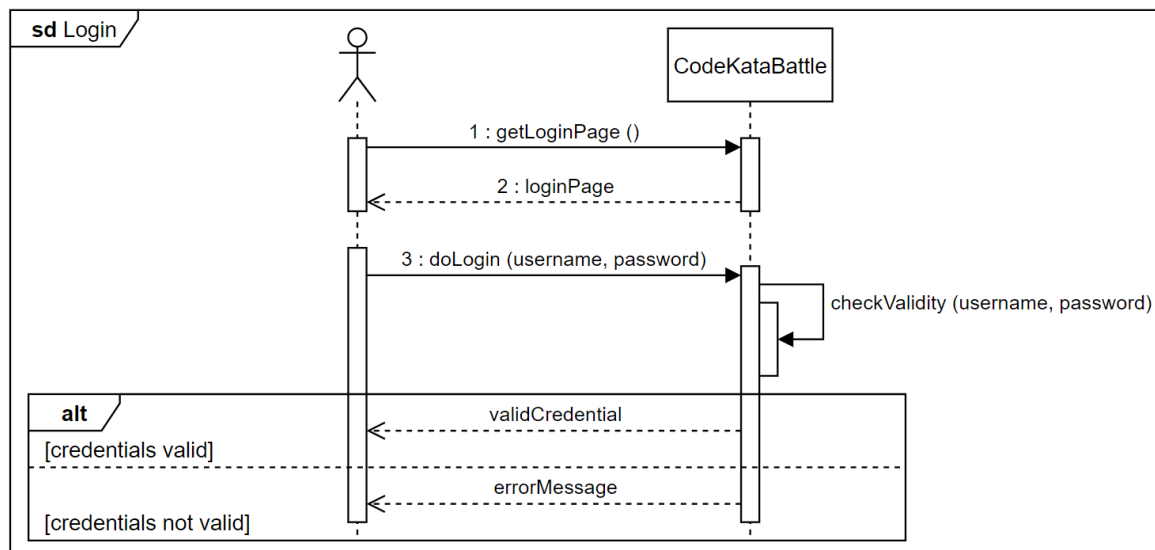
Use case [U2]: Educator registration

Actor(s)	Educator
Entry Condition	The actor clicks on the sign-in button
Event Flow	<ol style="list-style-type: none"> 1. The actor requires the sign-in page by clicking the join now button in the welcome page 2. The system shows the sign-in page to the actor 3. The actor inserts required personal information and sends it to the system 4. The system processes the information and shows a success message redirecting the user to the login page
Exit Condition	The actor is registered correctly, and the login page is displayed
Exceptions	<ul style="list-style-type: none"> • An already existing username, email or phone number are submitted • A required registration field is missing or invalid when the form is submitted
Notes	In case of exception the system will notify user with a human-readable message; the system cannot check if the right role is select during registration



Use case [U3]: Login

Actor(s)	Student, educator
Entry Condition	The actor clicks on the login button
Event Flow	<ol style="list-style-type: none"> 1. The actor requires the login page by clicking the log-in button in the welcome page 2. The system shows the login page to the actor 3. The actor inserts credentials and send it to the system 4. The system processes the information and shows a success message redirecting the user to the home page
Exit Condition	The actor is logged, and the homepage is displayed
Exceptions	<ul style="list-style-type: none"> • A wrong username or password is submitted • A student tries to log in as educator and viceversa
Notes	In case of exception the system will notify user with a human-readable message



Use case [U4]: Checking the profile

Actor(s)	Educator
Entry Condition	The actor clicks on profile button
Event Flow	<ol style="list-style-type: none">1. The actor clicks on profile button2. The system shows the profile page3. The actor can check his profile and eventually change personal information with edit button
Exit Condition	The actor is in the profile page
Exceptions	<ul style="list-style-type: none">• The actor is not signed-in
Notes	In case of exception the system will notify user with a human-readable message

Use case [U5]: Access to available tournaments

Actor(s)	Student
Entry Condition	The actor just logged in or clicked on the home button
Event Flow	<ol style="list-style-type: none">1. The actor requires the home page2. The system shows the home page to the actor3. The actor can now check all the available tournaments
Exit Condition	The actor is on the home page
Exceptions	<ul style="list-style-type: none">• There are no active tournaments
Notes	In case of exception the system will notify user with a human-readable message; the system cannot check if the right role is selected during registration

Use case [U6]: Join a specific tournament

Actor(s)	Student
Entry Condition	The actor clicks on the join tournament button
Event Flow	<ol style="list-style-type: none">1. The actor enters the home page2. The system shows the home page to the actor3. The actor clicks on the join button of the desired tournament4. The system shows the enrolled page with the new tournament
Exit Condition	The actor is on the enrolled page
Exceptions	<ul style="list-style-type: none">• The tournament selected by the actor is closed• The actor cannot join the selected tournament
Notes	In case of exception the system will notify user with a human-readable message

Use case [U7]: Search a specific tournament

Actor(s)	Student, Educator
Entry Condition	The actor goes to tournament details page
Event Flow	<ol style="list-style-type: none">1. The actor clicks the search tab on the left of any page2. The system shows the search page to the actor3. The actor fills thge form with the name of the searched tournament4. The system shows the matching tournaments
Exit Condition	The actor is in the search page
Exceptions	<ul style="list-style-type: none">• The actor searches a wrong string (null or invalid characters)• There are no matching active tournaments
Notes	In case of exception the system will notify user with a human-readable message

Use case [U8]: Join a specific tournament

Actor(s)	Student
Entry Condition	The actor clicks on create and join battle button
Event Flow	<ol style="list-style-type: none">1. The actor enters the tournament details page2. The system shows the tournament details page to the actor3. The actor clicks on the desired battle in the selected tournament4. The actor clicks on the create and join battle button or join on your own if it is alone5. The system shows the tournament details page with the new group
Exit Condition	The actor is in the tournament details page
Exceptions	<ul style="list-style-type: none">• The selected battle is in a tournament where the actor is not enrolled• The battle is already closed• The educators that manages the battle does not accept the new group
Notes	In case of exception the system will notify user with a human-readable message

Use case [U9]: Join a specific tournament

Actor(s)	Student
Entry Condition	The actor clicks the join button
Event Flow	<ol style="list-style-type: none">1. The actor enters the tournament details page2. The system shows the tournament details page to the actor3. The actor clicks on the desired battle in the selected tournament4. The actor clicks on an already existing group5. The actor requires the access to the selected group by clicking on join6. The system shows the tournament details page with a message
Exit Condition	The actor is in the tournament details page
Exceptions	<ul style="list-style-type: none">• The selected battle is in a tournament where the actor is not enrolled• The battle is already closed• The group does not accept the new student• The group is already at maximum capacity
Notes	In case of exception the system will notify user with a human-readable message

Use case [U10]: Join a specific tournament

Actor(s)	Student
Entry Condition	The actor goes to tournament details page
Event Flow	<ol style="list-style-type: none">1. The actor enters the tournament details page2. The system shows the tournament details page to the actor3. The actor clicks on the repository button linked to the repository4. The actor is redirected to the GitHub repository linked to the battle selected
Exit Condition	The actor is in the tournament details page and has another tab with the requested repository
Exceptions	<ul style="list-style-type: none">• The selected battle is in a tournament where the actor is not enrolled• The battle is already closed
Notes	In case of exception the system will notify user with a human-readable message

Use case [U11]: Check the ranking of a specific battle

Actor(s)	Student, Educator
Entry Condition	The actor goes to tournament details page
Event Flow	<ol style="list-style-type: none">1. The actor enters the tournament details page2. The system shows the tournament details page to the actor3. The actor clicks on a battle in the selected tournament4. The actor can check the ranking in the details of the battle
Exit Condition	The actor is in the tournament details page
Exceptions	<ul style="list-style-type: none">• There are no participants
Notes	In case of exception the system will notify user with a human-readable message

Use case [U12]: Check the ranking of a specific tournament

Actor(s)	Student
Entry Condition	The actor clicks leaderboard button
Event Flow	<ol style="list-style-type: none">1. The actor requires the enrolled page2. The system shows the enrolled page3. The actor clicks on details of a tournament4. The system shows the details of the tournament5. The actor clicks on leaderbord button6. The system shows the leaderboard page7. The actor can check the ranking of the selected tournament
Exit Condition	The actor is in the leaderbord page of the selected tournament
Exceptions	<ul style="list-style-type: none">• There are no participants
Notes	In case of exception the system will notify user with a human-readable message

Use case [U13]: Check the ranking of a specific tournament

Actor(s)	Educator
Entry Condition	The actor clicks leaderboard button
Event Flow	<ol style="list-style-type: none">1. The actor search a tournament2. The system shows the corresponding tournament in the search page3. The actor clicks on detail on the desired tournament4. The system shows the details of the tournament5. The actor clicks on leaderbord button6. The system shows the leaderboard page7. The actor can check the ranking of the selected tournament
Exit Condition	The actor is in the leaderbord page of the selected tournament
Exceptions	<ul style="list-style-type: none">• There are no participants
Notes	In case of exception the system will notify user with a human-readable message

Use case [U14]: Create a tournament

Actor(s)	Educator
Entry Condition	The actor clicks on create button on create tournament page
Event Flow	<ol style="list-style-type: none">1. The actor enters the create tournament page2. The system shows the create tournament page to the actor3. The actor compile the form4. The actor clicks the create button to submit the form5. The system shows the manage tournament page to the actor with the new tournament
Exit Condition	The actor is in the manage tournament page
Exceptions	<ul style="list-style-type: none">• The actor tries to create a tournament with invalid data• The actor tries to create a tournament with invalid battles• The actor tries to create a tournament with invalid collaborator invitations
Notes	In case of exception the system will notify user with a human-readable message

Use case [U15]: Manage a tournament

Actor(s)	Educator
Entry Condition	The actor goes to manage tournament page
Event Flow	<ol style="list-style-type: none">1. The actor enters the manage tournament page2. The system shows the manage tournament page to the actor3. The actor selects the desired tournament4. The system shows the possible action for the selected tournament
Exit Condition	The actor is in the manage tournament page
Exceptions	<ul style="list-style-type: none">• The actor tries to access an unmanaged tournament• The actor tries to access a closed tournament
Notes	In case of exception the system will notify user with a human-readable message

Use case [U16]: Add a battle to a tournament

Actor(s)	Educator
Entry Condition	The actor clicks on add battle in the managed tournament page
Event Flow	<ol style="list-style-type: none">1. The system shows the create battle page2. The actor compiles all the elements required by the form3. The actor submits the form4. The system show the managed tournament page with the modified tournament
Exit Condition	The actor is in the manage tournament page on the modified tournament with the new battle
Exceptions	<ul style="list-style-type: none">• The form is not compiled correctly• The tournament selected is already closed
Notes	In case of exception the system will notify user with a human-readable message

Use case [U17]: Add a new educator to a tournament

Actor(s)	Educator
Entry Condition	The actor clicks on add collaborator in the managed tournament page
Event Flow	<ol style="list-style-type: none">1. The system shows the managed tournament page with the possibility to add collaborators2. The actor insert the email in the corresponding form3. The actor clicks on invite4. The system show the managed tournament page with the modified tournament
Exit Condition	The actor is in the manage tournament page on the modified tournament with the pending invitation
Exceptions	<ul style="list-style-type: none">• The email does not correpond to another educator• The tournament selected is already closed
Notes	In case of exception the system will notify user with a human-readable message

Use case [U18]: Close a tournament

Actor(s)	Educator
Entry Condition	The actor clicks on close button on a specific tournament in manage tournament page
Event Flow	<ol style="list-style-type: none">1. The actor enters the manage tournament page2. The system shows the manage tournament page3. The actor select the desired tournament4. The actor clicks on close button in the selected tournament5. The system shows the manage tournament page and the tournament is closed
Exit Condition	The actor is in the manage tournament page on the modified tournament with the closed tournament
Exceptions	<ul style="list-style-type: none">• The tournament is already closed
Notes	In case of exception the system will notify user with a human-readable message

Use case [U19]: Evaluate a battle

Actor(s)	Educator
Entry Condition	The actor enters the manage battle page
Event Flow	<ol style="list-style-type: none">1. The actor enters the manage battle page2. The system shows the manage battle page3. The actor clicks on selected battle4. The actor fill the form with the manual evaluation grades5. The actor clicks publish button6. The system shows the manage battle page and the manual evaluation is complete
Exit Condition	The actor is in the manage battle page
Exceptions	<ul style="list-style-type: none">• The battle is not yet closed• The form is not compiled correctly
Notes	In case of exception the system will notify user with a human-readable message

3.3 Performance requirements

The performance requirements needed to ensure the reliability and the availability of the system are:

- The backend must be scalable based on the number of active user, reactive. It must be able to magae a proper load-balancing to ensure performance also when numeorus users are conneccted togehter to the platform.
- The frontend must be modeled to be user-friendly, so that also educators and students that does not work often with technology can use the platform. The interface must also be fluid and handle asynchronous interaction to ensure a certain level of usability even when the connection is not too stable.
- The system must have a security level that ensure availability and protect the system and users' data from malicious pepole.
- Since the notifications are used to alert the unser mainly on deadlines, they must be delivered in a short time (in the order of some minutes).

- The system must also adapt when the number of user increase suddently (e.g., when the final rankings are published multiple users login into a single tournament page).

3.4 Design constraints

3.4.1 Standards compliance

The specifications described in this document must be followed during all the development process and the final product must adhere to these.

The source code of the system must be commented to ensure that all the specifications are met. The data used by the application must be treated accordingly to the GDPR.

3.4.2 Hardware limitations

The application requires a device that can be a mobile one or a personal computer.

In both cases the device needs an internet connection to retrieve data from the server.

3.5 Software system attributes

3.5.1 Reliability

To ensure enhanced reliability performance, conduct all scheduled system maintenance during nighttime. Additionally, be aware that certain system functionalities depend on external APIs; however, a failure in one of them should not result in a complete system failure. Prioritize the prevention of data loss by employing redundant storage methods.

3.5.2 Availability

The system must deliver its functionalities with an availability of 99.5% or higher, meaning it should be inaccessible for less than two days each year. To meet this objective, ensure high redundancy for the most crucial components of the system.

3.5.3 Security

To safeguard sensitive user data, the system should gather only essential information, storing it in an encrypted database. The connection between the application must adhere to contemporary standard protocols like HTTPS. We must encrypt email and passwords. The latter one also needs salt.

3.5.4 Maintainability

Source code and related documentation should have comprehensive comments and be consistently updated. Emphasize modularity, low coupling, and high cohesion between components during the design and development phases. Specifically, ensure that the modularity of the frontend and backend allows developers to update the backend seamlessly without users experiencing any disruptions.

3.5.5 Portability

The application should be accessible on prominent mobile operating systems, targeting at least the latest releases. The web application must be compatible with the most recent browsers. Optimal efficiency can be achieved by judiciously reusing a significant portion of the codebase across both mobile and web platforms through the use of cross-platform tools.

Chapter 4

Formal analysis with Alloy

Chapter 5

Effort spent

The table below offers a concise overview of the hours invested by each group member, along with a brief description of their contributions. Dates where the same amount of time was dedicated by all members usually indicates collaborative efforts.

Date	Rossi	Sharoubim	Description
22-10-2023	1	0	Repository setup and file structure
28-10-2023	3	1	First chapter initial content
29-11-2023	4	4	Part of second and third chapters
30-11-2023	2	2	Initial class diagram
05-12-2023	0	6	Graphical user interfaces
05-12-2023	2	2	Graphical user interfaces
06-12-2023	3	3	Class diagrams
09-12-2023	1	1	Use case diagrams
10-12-2023	2	2	Use case diagrams
XX-XX-XXXX	-	-	
XX-XX-XXXX	-	-	
XX-XX-XXXX	-	-	
XX-XX-XXXX	-	-	
Total	16	19	-

Chapter 6

References

Diagrams - <https://app.diagrams.net>

UI Mockup - <https://webflow.com>