Requirement Analysis and Specification Document

Christian Rossi
Kirolos Sharoubim

Academic Year 2023-2024

Contents

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.

1.2 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 procedes 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.

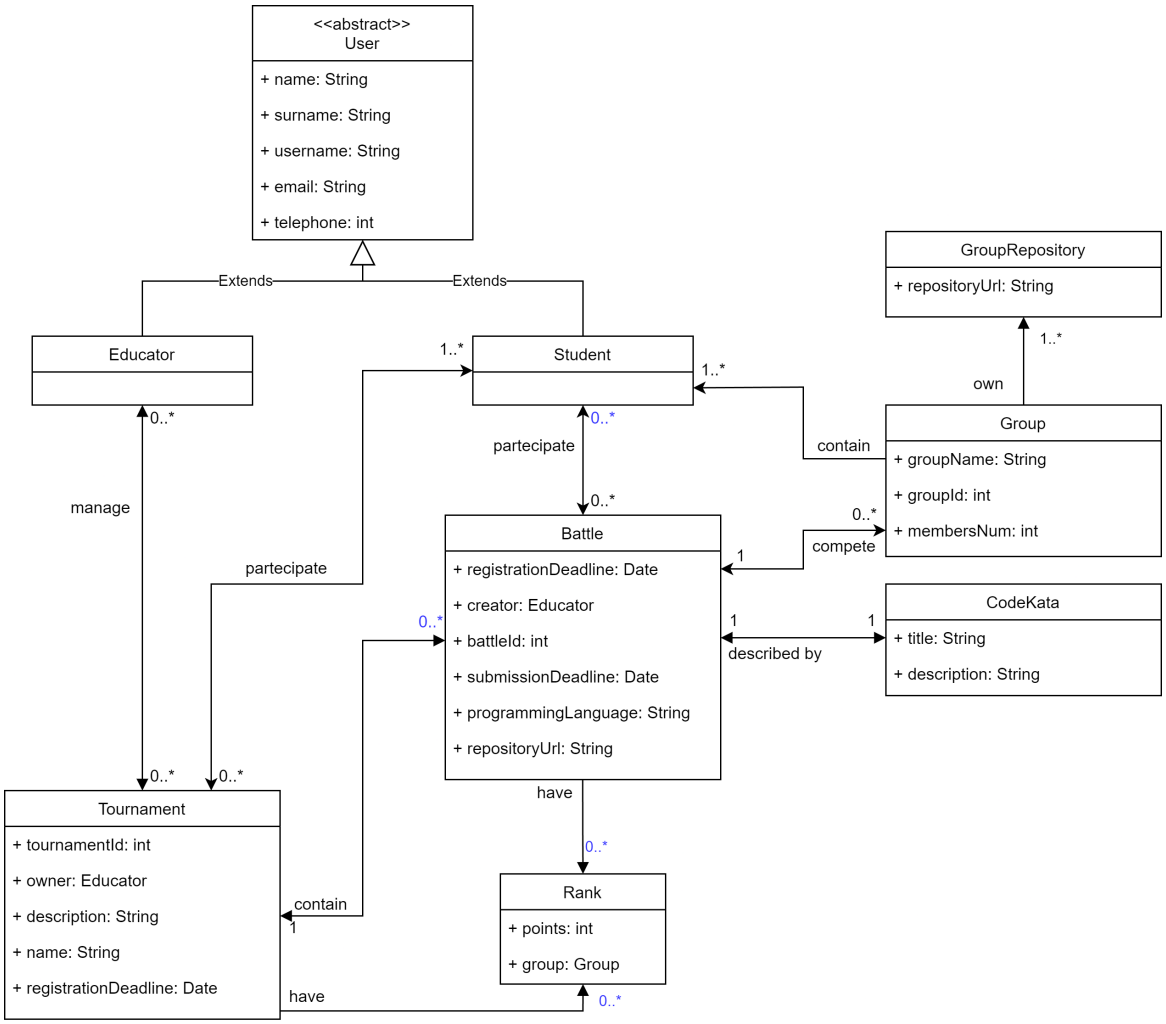
4. *Student joins a group for a specific CK in a tournament*
   Donald is a car wash employee, even if things are going great his dream was to become a web developer. Donald already learned a lot JavaScript and is always in search of new challenges. Donald knows that in programming environment team working 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 were 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 CodeKata-Battle 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 partecipate 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 partecipate 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 partecipate 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 *Educator*s that manage the *Tournament* in cooperation with the owner of it. The *Tournament* hase a relation with *Battle* and *Student* it is composed by some *Battle*s and one or more students are enrolled. Finally, it has also a general *Rank*, that specifies the rankings of all battles combined.

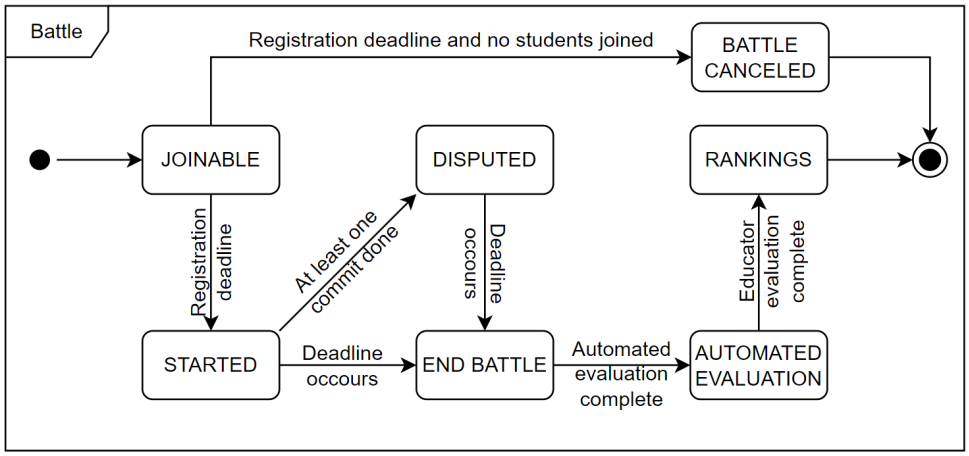
- *Rank*: it comprises a points attributes (that ranges between 0 and 100) and the group linked to this value. Each *Tournament* and *Battle* has a list of *Rank* that, ordered by the points attributes return the actual standings.

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

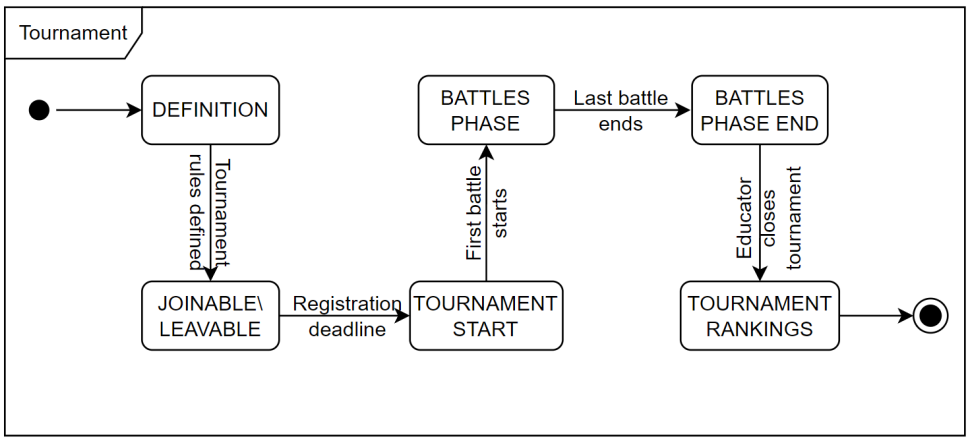
The diagram that represents the possible states of a battle is depicted below:



Description After the creation of a battle, the students enrolled in the battle's tournament can join alone or in groups (*Joinable*). At registratione 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 canceled0* 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 occours 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 doind 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 occours 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 stared. 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 occours, the educator close 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 spresented based on the user.

2.2.1 Student's functionalities

The educator is the user that partecipates to the torunaments 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 torunaments*: 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 catual 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 torunaments 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 torunament. 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 regiter 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 retrive the personal data given to the site ache 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 choosen 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).

Register page The register page is the same for educators and students.
In this page they have to select their role.

Login page The login page is the same for educators and students.
They have to choose if they want a student login or educator login based on their role.

Student interface The student is able to check all the tournament and to enroll in the available ones.

Educator interface The student is able to create and manage tournaments and battles.

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.
Additionaly, 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 partecipating 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., partecipate 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:

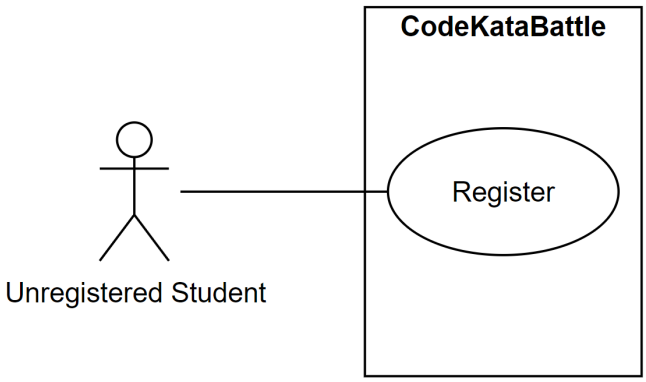


Figure 3.1: Use Case diagram for unregistered student

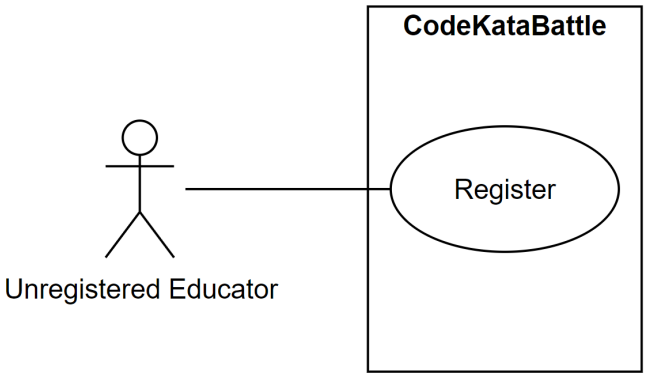


Figure 3.2: Use Case diagram for unregistered educator

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

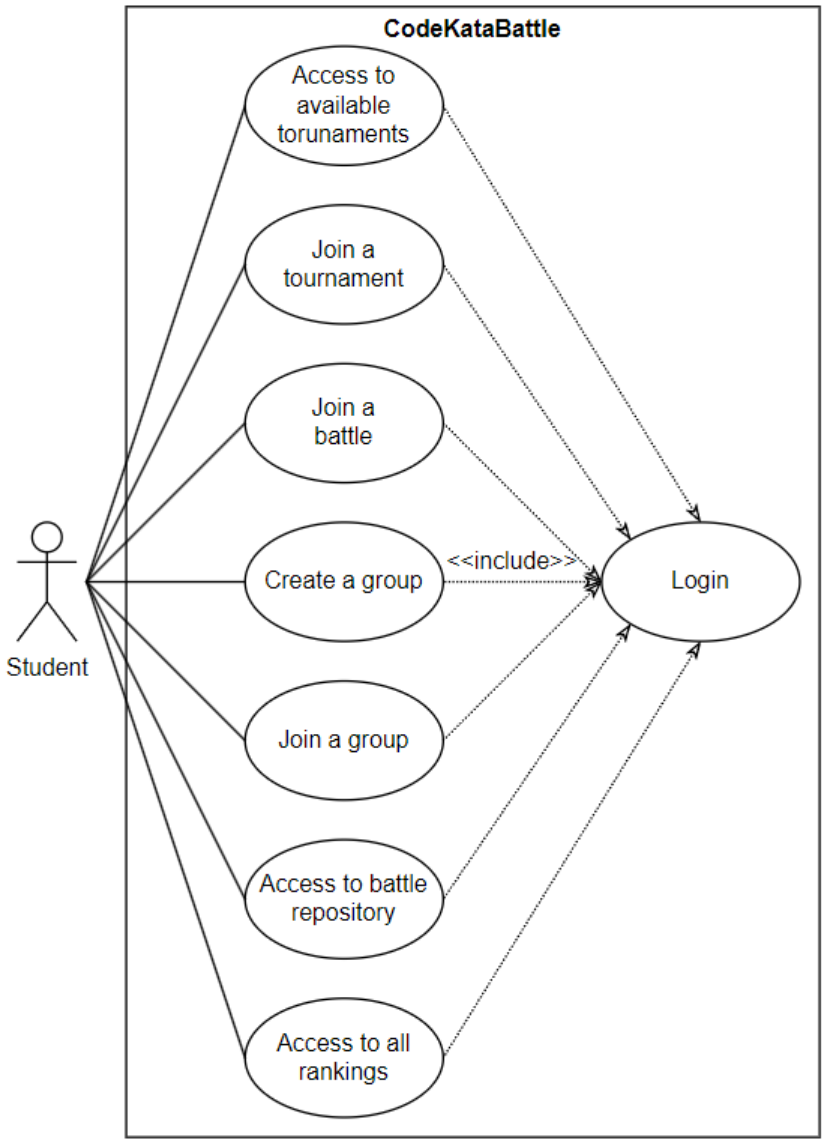


Figure 3.3: Use Case diagram for unregistered educator

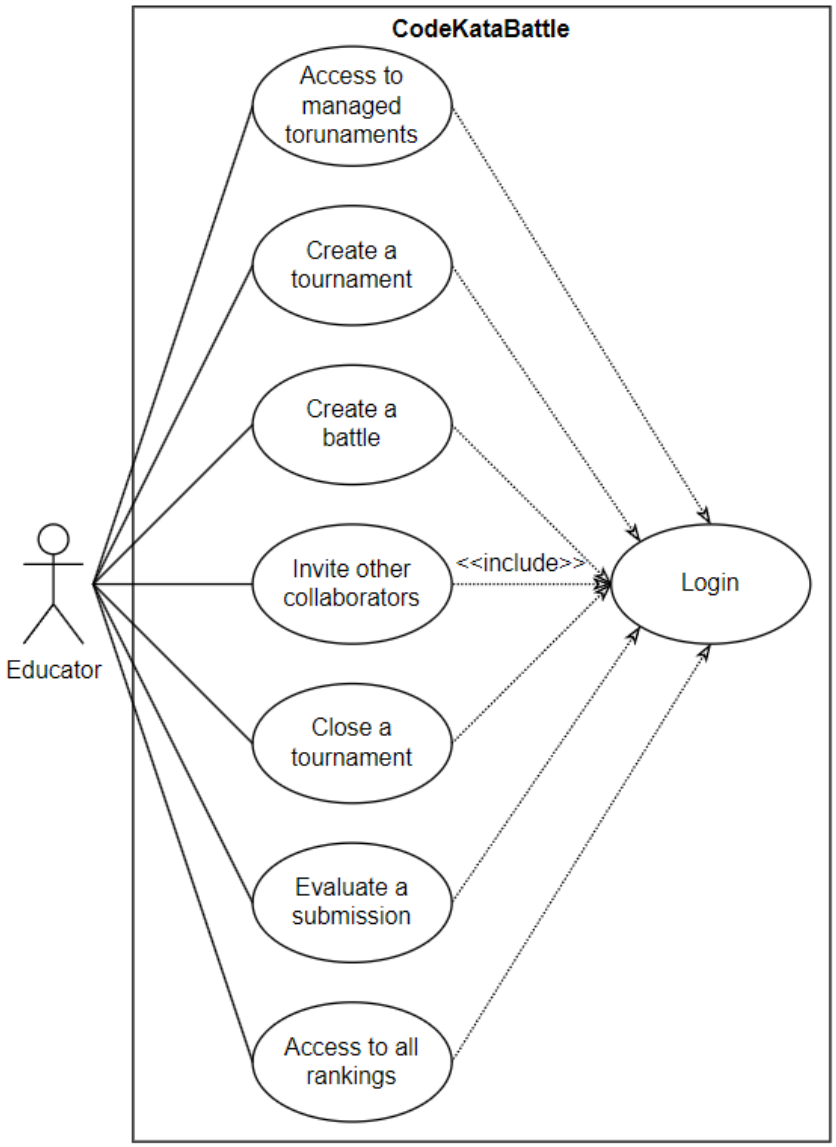
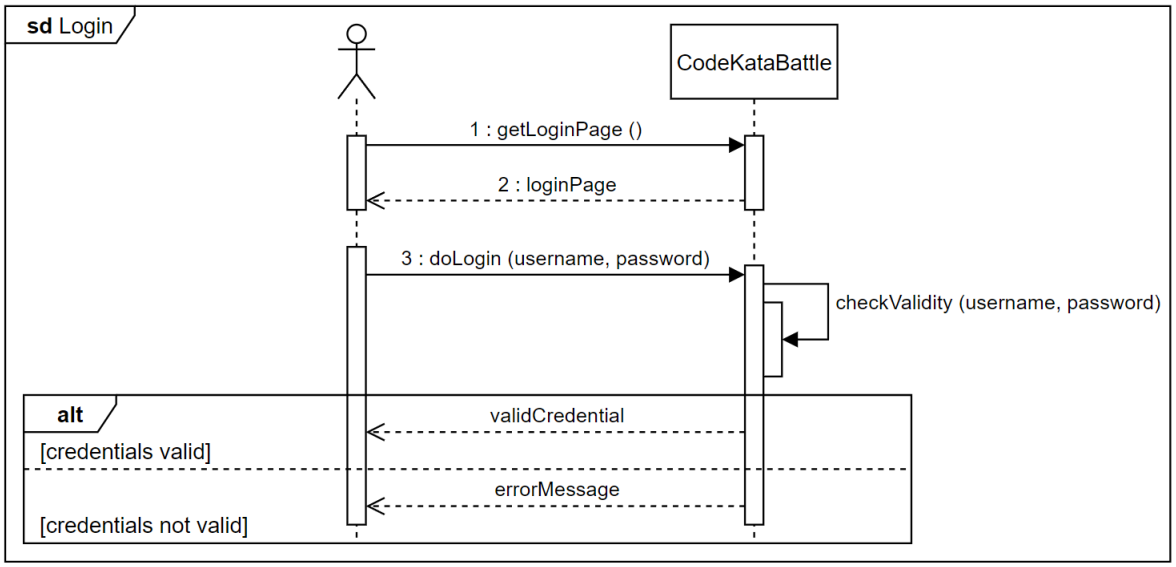


Figure 3.4: Use Case diagram for unregistered educator

3.2.2 Use cases

Use case [U1]: Login

Actor(s)	Student, educator
Entry Condition	The actor is already registered in the system
Event Flow	1. The actor requires the Login 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 homepage.
Exit Condition	The actor is logged, and the homepage is displayed
Exceptions	• 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



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

3.5.2 Availability

3.5.3 Security

3.5.4 Maintainability

3.5.5 Portability

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
xx-xx-xxxx	-	-	
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