

# POLITECNICO

## MILANO 1863

### CodeKataBattle

Requirements Analysis and  
Specifications Document

Software Engineering 2 project  
Academic year 2023 - 2024

20 October 2023  
Version 0.0

*Autors:*  
Tommaso Pasini  
Elia Pontiggia  
Michelangelo Stasi

*Professor:*  
Matteo Camilli

# Revision History

Date	Revision	Notes
23/10/2023	v.0.0	Document creation
	v.1.0	First release

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Purpose . . . . .	4
1.1.1	Purpose of the product . . . . .	4
1.1.2	Goals . . . . .	4
1.2	Scope . . . . .	5
1.2.1	World Phenomena . . . . .	6
1.2.2	Shared Phenomena . . . . .	6
1.3	Definitions, Acronyms, Abbreviations . . . . .	9
1.3.1	Definitions . . . . .	9
1.3.2	Acronyms . . . . .	9
1.3.3	Abbreviations . . . . .	10
1.4	Reference Documents - qui ci mettiamo i link . . . . .	10
1.5	Document Structure . . . . .	10
<b>2</b>	<b>Overall Description</b>	<b>12</b>
2.1	Product Perspective . . . . .	12
2.1.1	Scenarios . . . . .	12
2.1.2	Creating Game badges . . . . .	12
2.1.3	Class Diagram . . . . .	14
2.1.4	State Diagrams . . . . .	15
2.2	Product Functions . . . . .	18
2.2.1	Registrar function . . . . .	18
2.2.2	Creating tournament function . . . . .	18
2.2.3	Creating battles function . . . . .	18
2.2.4	Partecipating in tournament and battles function . . . . .	18
2.2.5	Score visualization function . . . . .	18
2.2.6	Gamification function . . . . .	18
2.3	User characteristics . . . . .	18
2.4	Assumptions, Dependencies and Constraints . . . . .	19
2.4.1	Domain Assumptions . . . . .	19
2.4.2	Dependencies . . . . .	20
2.4.3	Constraints . . . . .	20

<b>3 Specific Requirements</b>	<b>21</b>
3.1 External Interface Requirements . . . . .	21
3.1.1 User Interfaces . . . . .	21
3.1.2 Hardware Interfaces . . . . .	26
3.1.3 Software Interfaces . . . . .	26
3.1.4 Communication Interfaces . . . . .	26
3.2 Functional Requirements . . . . .	27
3.2.1 Use cases Diagrams . . . . .	27
3.2.2 Use cases Description . . . . .	27
3.2.3 Use cases Sequence Diagrams . . . . .	38
3.2.4 List of functional requirements . . . . .	38
3.2.5 Traceability matrices . . . . .	40
3.3 Performance Requirements . . . . .	42
3.4 Design Constraints . . . . .	42
3.5 Software System Attributes . . . . .	42
3.5.1 Reliability . . . . .	42
3.5.2 Availability . . . . .	43
3.5.3 Security . . . . .	43
3.5.4 Maintainability . . . . .	43
3.5.5 Portability . . . . .	43
<b>4 Formal Analysis Using Alloy</b>	<b>44</b>
<b>5 Effort Spent</b>	<b>46</b>

# 1. Introduction

## 1.1 Purpose

Software development is one of the most sought-after skills globally, and its growing importance is set to persist in future years. The versatility of this ability allows solving many problems in different fields, including science, art, and entrepreneurship. The acquisition of software development skills improves both creativity and innovation, as well as fosters the development of fundamental skills such as problem-solving, logical thinking, and effective communication.

To understand and master software development, it is essential to have the ability to design, write, and test software programs. Dedication and patience are necessary for the gradual process of learning programming. Theory is a foundational element, but it is only the start of developing your skills. Consistent practice is necessary to master concepts and programming techniques.

### 1.1.1 Purpose of the product

The purpose of the product is to provide a solution to the problem previously highlighted. The **CodeKataBattle** project, or CKB, is a platform that assist students in enhancing their software development abilities through battles. Students in teams participate in programming exercises where they complete software projects following a test-first approach. Creating battles, setting rules, and evaluating students' performance are tasks that educators do. CKB promotes skill development, competition, and evaluation, fostering a collaborative and skill-building atmosphere.

### 1.1.2 Goals

The following table provides an aggregate list of the specific goals that must be accomplished by CodeKataBattle system.

---

<b>G1</b>	Educators can create tournaments that involve coding battles to challenge students.
<b>G2</b>	Provides educators with the ability to track student software development knowledge.
<b>G3</b>	Students can improve software development skills by taking part in coding tournaments and battles where they must write programs.
<b>G4</b>	Coding battles enable students to enhance their soft skills, such as communication, collaboration, and time management, by creating team and collaborating with the members.

---

Table 1.1: Goals

## 1.2 Scope

CodeKataBattle (CKB) is a platform that allows students to improve their software development skills by participating in coding challenges. Educators create coding battles within specific tournaments, encouraging students to improve their programming proficiency. Students compete in coding battles to solve programming exercises in their chosen language while adhering to the test-first approach.

Educators create a battle by uploading a programming exercise (also known as Code Kata), specifying the group size, registration and submission deadlines, and scoring parameters. Once created, students can form teams, a team could be composed of just one person or at most the number specified by the group size. Each team can be changed between different battles in the same tournament, register for the battle, and receive access to a GitHub repository containing the code kata. They are required to fork the repository and automate their workflow through GitHub Actions.

The battle score ranges from 0 to 100 and is determined by both mandatory automated evaluation (test case pass rate, timeliness, source code quality) and optional manual evaluation (personal scores assigned by educators). The platform updates the battle score as students commit their code, in this way students and educators can keep track of the battle's ranking. Following the deadline for submission, educators perform an optional manual evaluation before sharing the final battle rank with participants.

All students on the CKB platform have a personal tournament score, which is the sum of their battle scores in that tournament. This score is determined by CKB when educators close a battle, and it is accessible to all platform users.

When educators close a tournament, the final tournament rank remains available to all platform users. Gamification badges, which educators can define when creating a tournament, reward students for their performance or achievement.

CodeKataBattle fosters collaborative coding challenges, helping students improve their software development skills while introducing competition and gamification elements to enhance engagement and recognition.

### 1.2.1 World Phenomena

ID	Description
<b>WP1</b>	Students partecipating in a tournament can decide whether doing a battle or not by subscribing or not to that battle.
<b>WP2</b>	Students choose for each battle if coding alone or forming a team.
<b>WP3</b>	The student who formed the team fork the GitHub repository of the code kata
<b>WP4</b>	Students set up an automatic workflow in the GitHub repository.
<b>WP4</b>	Student invites team member to its repository.
<b>WP4</b>	Students work and complete the code kata battle with their code.
<b>WP4</b>	The students push their work to the GitHub repository.
<b>WP4</b>	An educator creates a correct code kata, defining a coherent description of the project within its test cases and the configuration for automation scripts.
<b>WP4</b>	An educator checks the work done by students in order to evaluate them.

Table 1.3: World Phenomena

### 1.2.2 Shared Phenomena

ID	Description	Controlled by
<b>SP1</b>	A user registers its personal datas in CodeKataBattle system specifying if it is a student or an educator	
<b>SP2</b>	A registered user insert its credentials to get in CodeKataBattle environment	
<b>SP3</b>	An educator creates a tournament, defining all necessary details	
<b>SP5</b>	The educator who created a specific tournament grants other colleagues permission to create battles inside it	

<b>SP4</b>	An educator that has permission creates battles, defining all necessary details, within a tournament	
<b>SP7</b>	Students are notified of an upcoming tournament	
<b>SP8</b>	Students join a tournament	
<b>SP9</b>	Students are notified of an upcoming battle within a tournament they are subscribed to	
<b>SP10</b>	Student joins a battle	
<b>SP13</b>	Student invites other students, which are subscribed in the same tournament, to join its team respecting the boundaries imposed	
<b>SP14</b>	Student joins a team and therefore it gets enrolled to a battle via an invite by another student	
<b>SP16</b>	The platform sends the link of the GitHub repository to the students who created the team for the battle	
<b>SP17</b>	Students who received the GitHub repository link are asked to fork it and set up an automated workflow	
<b>SP18</b>	The forked repository's workflow notifies the platform of a new GitHub push action	
<b>SP19</b>	The platform updates the battle score of the students whenever their repository gets pushed	
<b>SP20</b>	Educator and student subscribed to the battle can monitor the battle ranking among other participants	
<b>SP21</b>	The educator uses the platform to go through the sources produced by each team	
<b>SP6</b>	The educator evaluates manually the work done by students in his own battle	
<b>SP23</b>	The platform notifies students of the end of a battle when the final battle rank becomes available	

<b>SP25</b>	An educator that has permission to create battles within a tournament, can also close the tournament	
<b>SP26</b>	The platform notifies all students involved in the tournament about its end when the tournament ranking is available	
<b>SP27</b>	All users can see the list of ongoing and ended tournaments as well as the corresponding tournament rank at any time	
<b>SP28</b>	An educator creates a gamification badge within the CKB platform and associates to it a rule and a variable	
<b>SP29</b>	An user checks the personal badges gained by anyone subscribed to the CKB platform	

Table 1.4: Shared Phenomena

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

Term	Definition
User	Can be a Student or a User
Educator	Identifies a person who provides instruction or education, such as a teacher.
Student	Identifies a person who is studying at a school or college.
Kata	A training exercise system for karate where you repeat a form multiple times, making small improvements to each one.
Test-first approach	A software development process that is based on converting software requirements into test cases before creating the software, and then tracking the entire development process by repeatedly testing the software against those test cases.

Table 1.5: Definitions

### 1.3.2 Acronyms

Acronyms	Term
CKB	CodeKataBattle
EDU	Educator
STU	Student

Table 1.6: Acronyms

### 1.3.3 Abbreviations

Abbreviation	Term
$G_i$	i-th goal
$WP_i$	i-th World Phenomena
$SP_i$	i-th Shared Phenomena
$DA_i$	i-th Domain Assumption
$Dep_i$	i-th Dependencie
$R_i$	i-th Requirement

Table 1.7: Abbreviations

## 1.4 Reference Documents - qui ci mettiamo i link

- Assignment RDD A.Y 2023-2024
- Course slides on WeeBeep
- RASD review by Prof. M. Camilli
- ISO/IEC/IEEE 29148 dated 2018, Systems and software engineering - Life cycle processes - Requirements engineerings

## 1.5 Document Structure

The structure of this RASD document follows six main sections:

1. **Introduction:** provides an overview of the problem at hand, the purpose of the document and the project, the scope of the domain, and introduces the main goals of the system as a solution.
2. **Overall Description:** gives a general description of the system, going into more details about its main functions. The description is assisted with the help of UML diagrams, such as class, activity and state diagram. The domain assumptions of the examined world are then explained along with any dependencies and constraints.
3. **Specific Requirements:** specifies the functional and non-functional requirements of a software system. It includes use cases diagrams, descriptions of each use case, and related sequence diagrams. Finally, it provides a mapping of the requirements to both goals and use cases.

4. **Formal Analysis Using Alloy:** contains Alloy models which are used for the description of the application domain and his properties, referring to the operations which the system has to provide and some critical aspects of the system.
5. **Effort Spent:** keeps track of the time spent to complete this document. The first table defines the amount of hours used by the whole team to get important decisions and to make reviews, the other tables contains the individual effort spent by each team member.
6. **References:** lists all the documents used and that were helpfull in drafting the RASD.

## 2. Overall Description

### 2.1 Product Perspective

#### 2.1.1 Scenarios

##### **Creating a tournament**

Chip, a professor of Algorithm and Data Structures at Mouseton Institute of Technology, prepared to teach the chapter on strings, launching the "Strings Operations" coding tournament on CKB. To expand participation, he allowed his colleague Dale to create challenges for his software engineering class. Students across classes would compete in string manipulation tasks, ranging from basic concatenation to advanced text analysis, fostering collaboration and learning. To make the tournament more interesting, Chip decided to award badges to the best performing students, so he adds th badge for the student who participated in the most tournaments, the one for the student who won the most battles and the one for the student that wrote the most lines of code. All students already subscribed to CKB were notified of the new tournament, and they could join it from the tournament page.

##### **Creating a battle**

In order to get the students familiar with the CKB platform and its features, Chip created an easy battle for his students to practice on, called "wordcheck", that basically requires the student to implement the game wordle, to be implemented in c language. He decides that the battle will last 2 weeks and that the students will be able to work in teams of 2 or 3 people; they will be able to join the battle until the last day of the battle. In addition, he wants to give extra points to the code cleanup, so he will have to revise the code of each team at the end of the battle and assign extra points to the teams that wrote clean code. He sets all this information in the battle creation form and then he creates the battle.

#### 2.1.2 Creating Game badges

Scrooge, a professor of Algorithm and Data Structures at Duckburg University, thinks that, in addition to the badges that are already present in the CKB platform, it would

be nice to have also a badge for the student who reaches the perfect score in a battle and a badge for the student who reaches the perfect score in a tournament. Since the CKB platform allows educators to create new badges, he creates the two badges and from now on, all educators will be able to include such badges in their tournaments and battles.

### **Joining a battle**

Huey and Dewey, two students of Chip's class, are notified of a new battle and decide to join it. Since the more the merrier, they decide to invite their friend Louie to join them in the battle. Louie receives the invitation mail and decides to join the battle in their team. After the registration deadline, they are notified that the battle is about to start and they are given the link to the GitHub repository of the battle to fork and set up the automated workflow in order to be able to link their GitHub account to the CKB platform. After the automated workflow is set up, they are ready to start working on the battle.

### **Improving the score and obtaining a badge**

Donald is a warrior of the "wordcheck" battle and he is working on the battle alone. After a first commit, he logs in to the CKB to check his score. He sees that he is in the 3rd position and that he is 10 points behind the leader group, that is composed by Huey, Dewey and Louie. Fortunately, the battle is still open and the CKB platform allows him to improve his score by pushing new commits to the GitHub repository, so he decides to work on the battle for a couple of days and then push his work to the GitHub repository. After checking his score again, he sees that he is now in the 1st position and moreover he obtained a badge for being the first to reach 100 points in the battle and now both students and professors can see this badge when they visit Donald's profile.

### **Closing a battle**

When the deadline for the battle created by Scrooge is reached, all participants are notified that the battle is closed and that they can't push new commits to the GitHub repository. Scrooge is notified that the battle is closed and he can now evaluate the code of each team and assign extra points for the clarity of the comments and the code, as he decided when he created the battle. After the evaluation, the final rank of the battle is available to all participants and the students are notified that they can now see the final rank of the battle.

### **Closing a tournament**

Chip decides to close the "Strings Operations" tournament, even if the deadline is not reached yet. In order to do so, he logs in to the CKB platform and he closes the tournament. All participants are notified that the tournament is closed and that they can't join it anymore, in the end, the CKB platform makes the final rank of the tournament available to all participants.

### Accessing the scores of the players

Huey wishes to enroll to the class of Advanced Algorithms and Data Structures held by professor Pippo, so he applies for the class. Pippo, who wants to make sure that Huey is a good student, comes to know that Huey is a very active user of the CKB platform and he decides to check his profile. He sees that Huey has a very high score in the "Strings Operations" tournament and that he has a badge for being the most active user of the platform and he also notes that Huey is involved in more than one tournament simultaneously. Thanks to the CKB platform, Pippo has now a complete overview of Huey's skills and he can decide whether to accept his application or not.

#### 2.1.3 Class Diagram

In figure 2.1 is represented the class diagram of the software. In particular, the most important details are :

- There are only two possible types of users : the Student type and the Educator type;
- The Educator can create tournaments and battles, he can grants other educators to his own tournament;
- The Student can invite other students, he can achieves some badges, he can subscribes to a tournament and he will use GitHub. But the most important detail is the one about subscribing to a battle : in fact, a student can subscribe to a battle by creating his own team (a team is composed by at least one person) or by joining an existing team. In this way we don't use a "Team" class because a team doesn't exist without a battle;
- The Tournament and Battle classes use the MailAPI to notify students about events.

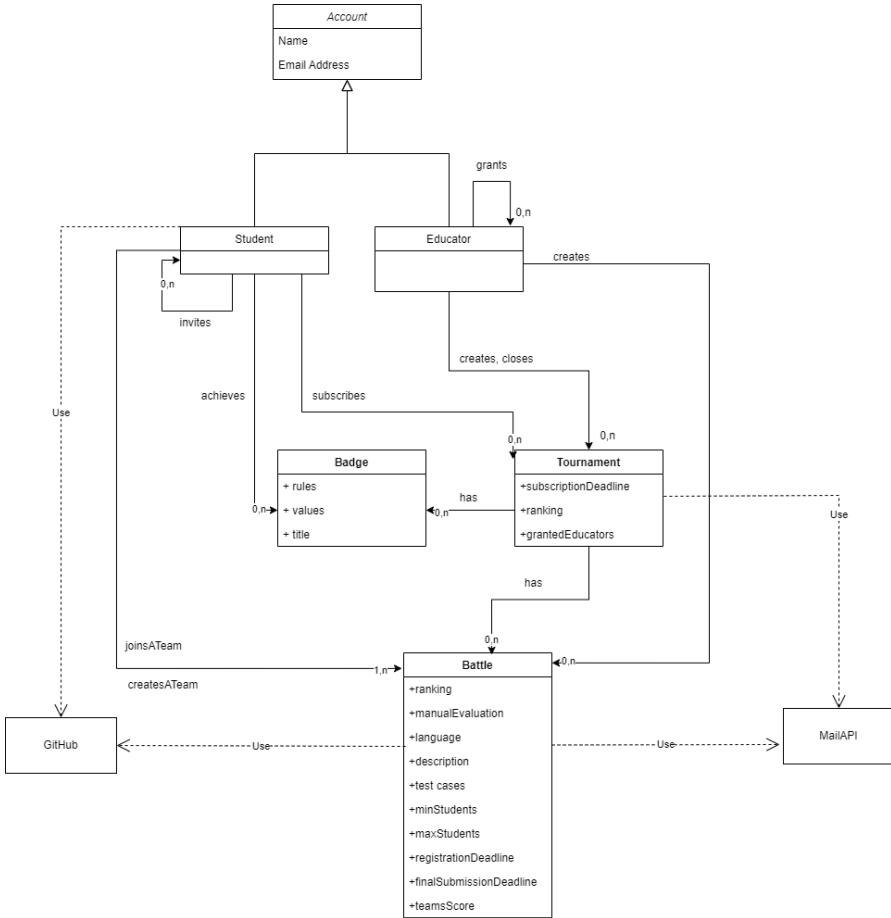


Figure 2.1: Software Class Diagram

### 2.1.4 State Diagrams

The following state diagrams describe the life cycle of the main entities of the system. Moreover, they also specify the sequence of states that an object goes through during its lifetime in response to stimuli from the environment. We want to focus on the events that cause a transition from one state to another and the actions that result from a state change.

#### Tournament

After an educator creates a tournament, it is both in the *registration open* and *tournament open* states.

In the *registration open* state, students can join the tournament, while in the *tournament open* state, educators with the right permissions can create battles within the tournament, and that leads the tournament to the *battling* state.

When the deadline for the registration is reached, the tournament moves to the *registration closed* state and no more students can join it.

When the deadline for the registrations is reached, no more students can join the tournament and it moves permanently to the *registration closed* state.

During the *battling* state educatos can start multiple parallel battles and, if and only if all battles are ended, the educator can finally close the tournament.

The diagram is shown in figure 2.2.

## Battle

The battle evolves in a linear way, starting from the *registration open* immediately followed by the *registration closed* state.

After the registration deadline is reached, the github repository of the battle is created and thus the battle moves to the *coding* state, allowing the students to fork the repository and start working on the battle.

When the deadline for the battle is reached, the educators can start evaluating the code of the students, if previously enabled (*consolidation* state).

After the evaluation is completed, the battle can be closed and the final rank is available to all participants.

The diagram is shown in figure 2.3.

## Score evaluation

The score evaluation of a battle is a process that is triggered by the end of a battle and it is composed by multiple steps.

First, three aspects can be automatically evaluated: functional aspects (the higher the better, +), timeliness (the lower the better, -) and quality level of the sources, extracted through static analysis tools (+).

Finally, if the educator enabled the manual evaluation, he can assign extra points.

The diagram is shown in figure 2.4.

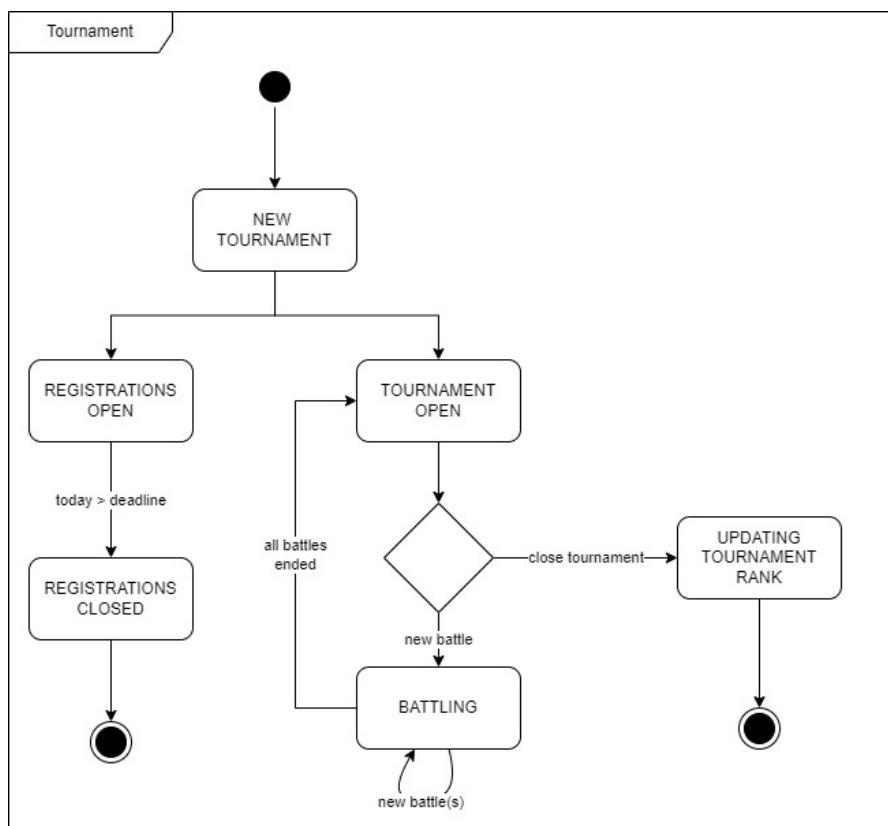


Figure 2.2: Tournament state diagram

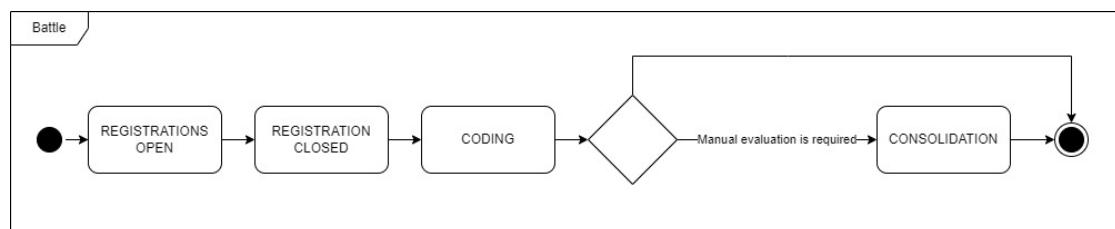


Figure 2.3: Battle state diagram

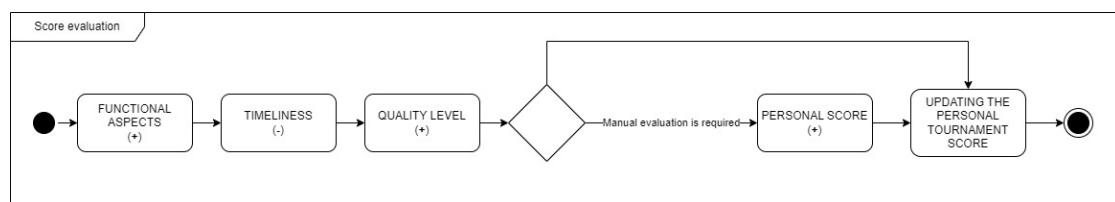


Figure 2.4: Score evaluation state diagram

## 2.2 Product Functions

### 2.2.1 Registrater function

- Studente o educatore - Educatore ha un livello di account maggiore rispetto allo studente  
-> Crea battaglie o tornei e specifico i dettagli per ogni cosa

### 2.2.2 Creating tournament function

- Come si crea il torneo molto in generale

### 2.2.3 Creating battles function

- Come si crea la battaglia molto in generale

### 2.2.4 Partecipating in tournament and battles function

- Come si iscrive a battaglie e tornei molto in generale

### 2.2.5 Score visualization function

- Score, ranking e badges vederli

### 2.2.6 Gamification function

## 2.3 User characteristics

Users of the system fall into one of the following categories: student or educator.

### Student

Student participates actively in code kata tournaments and battles to enhance their software development skills. During a battle, students develop solutions following the "test-first" approach and use GitHub to manage the code. The CKB platform automatically evaluates student progress based on the number of tests passed, timeliness, quality of code, providing scores updated in real-time. Students aim to achieve high scores and accumulate gamification badges defined by educators. In addition to participating in battles, students can view their tournaments' rank and receive notifications on final results. The student's primary goal is to improve programming skills, obtain competitive scores, and earn badges through active participation in the collaborative context of the CKB platform.

## Educator

The educator takes a central role in directing students toward improving software development skills through programming battle competitions. Its duties include creating tournaments and competitions, finalizing challenge details, and managing student registrations. The educator assigns specific tasks, manually evaluates students' solutions, and contributes to the automatic evaluation of projects, considering functional, temporal, and code quality aspects. Furthermore, it can create gamification badges, motivating students with personalized rewards based on rules it establishes. The main objective is to improve students' skills, ensuring fair and efficient assessment and encouraging active involvement. The educator plays a key role in educational innovation, exploring new possibilities through the definition of personalized rules and badges that stimulate growth and collaboration. In summary, the educator acts as a promoter of an engaging and competitive training challenge on CKB.

## 2.4 Assumptions, Dependencies and Constraints

### 2.4.1 Domain Assumptions

- D1.** STUs code with the programming language setted for the battle to which they are participating
- D2.** EDUs upload the code kata with the description and the correct software project, including test cases and build automation scripts
- D3.** STUs fork the GitHub repository of the code kata and set up an automated workflow through GitHub Actions that informs the CKB platform (through proper API calls) as soon as STUs push a new commit into the main branch of their repository
- D4.** EDUs manual evaluation range from 0 to 100<sup>1</sup>
- D5.** The information inserted at registration moment of all users are truthful
- D6.** GitHub and the tool for static analysis work properly
- D7.** A team is formed by at least one person
- D8.** All users subscribed to the CKB platform have a GitHub account

---

<sup>1</sup>The full evaluation will be given by the average of all the four aspects evaluated by the CKB platform, the three automatic ones and the manual one

### 2.4.2 Dependencies

---

<b>Dep1</b>	The system will require internet connection to interact with CKB and other users
<b>Dep2</b>	The system will integrate an external API in order to compile the code written by students
<b>Dep3</b>	The system will integrate a GitHub API in order create repository for each battle

---

Table 2.1: Dependencies

### 2.4.3 Constraints

- The software must follow local laws and rules, especially when it comes to handling user data, like letting users access their data when they want.
- The software should only collect the data it really needs, like just the user's email address.
- To keep users' important info safe, like passwords and personal data, it must be stored in SHA256 encoding in the database.
- When choosing external APIs, especially those that are crucial for it to work properly, we should pick the ones that are the most dependable and always available.

# 3. Specific Requirements

## 3.1 External Interface Requirements

In the following, the interfaces, both hardware and software, of the system are described.

### 3.1.1 User Interfaces

In the following it is shown the mockup of the main pages of the system.

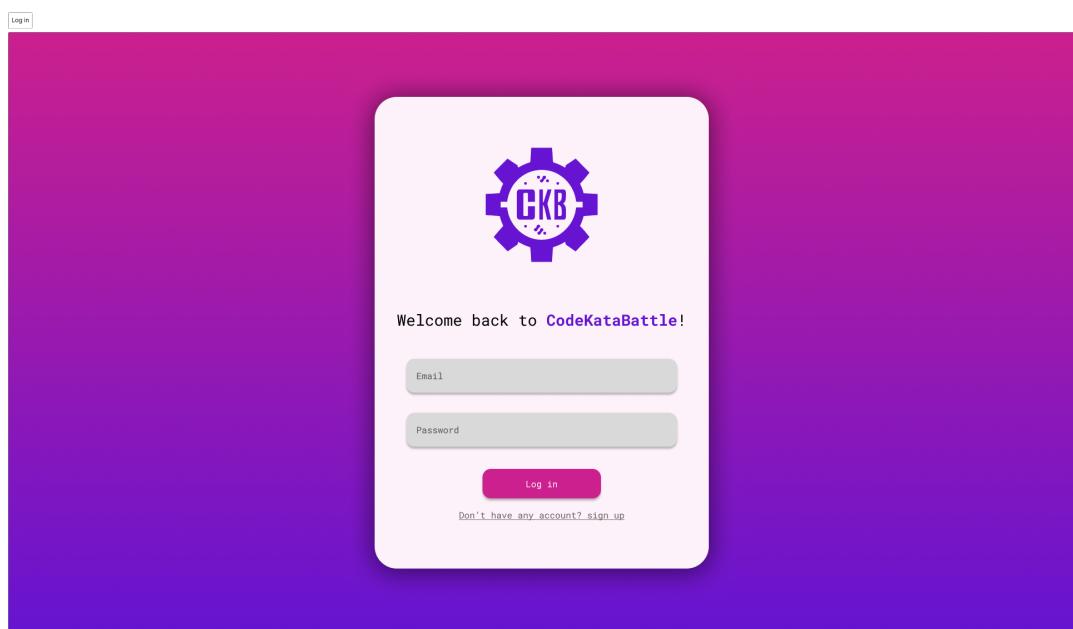


Figure 3.1: Log in page

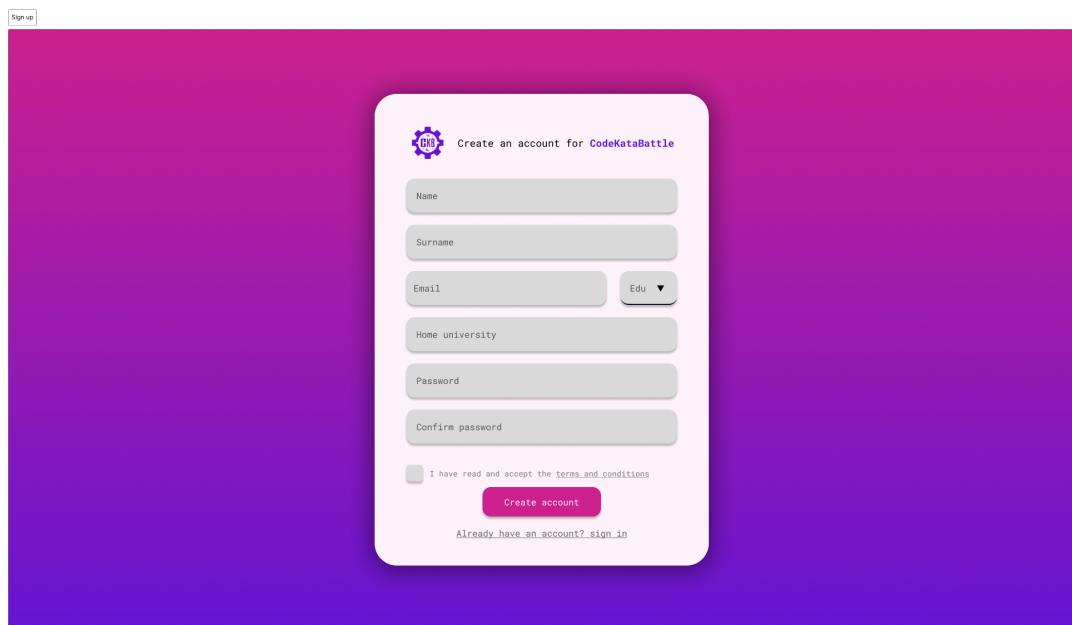


Figure 3.2: Sign up page

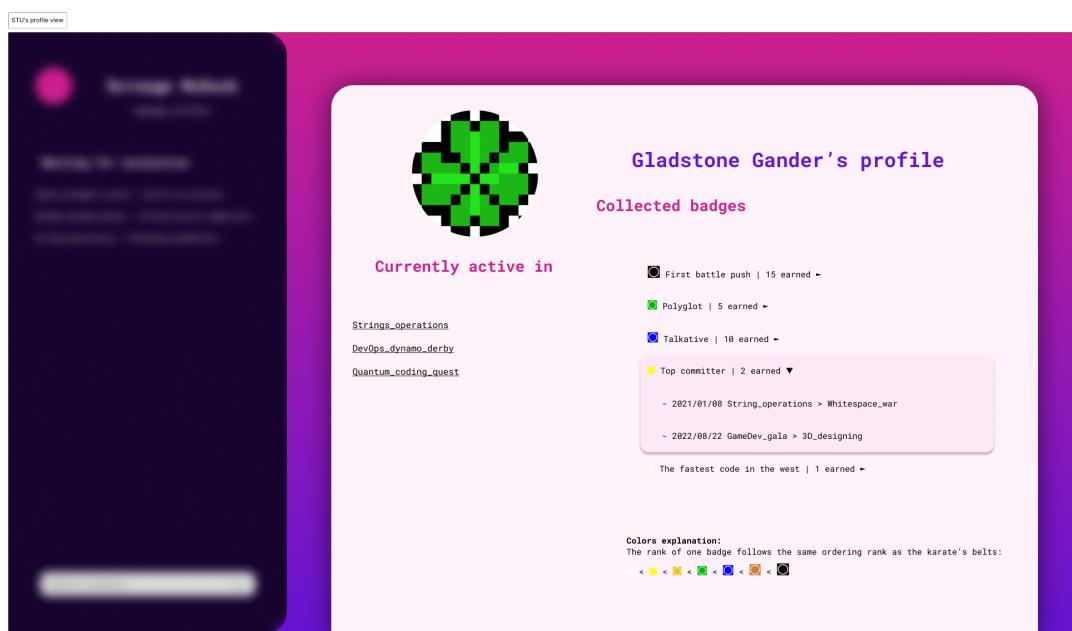


Figure 3.3: Page that shows badges acquired by a STU

## EDU's views

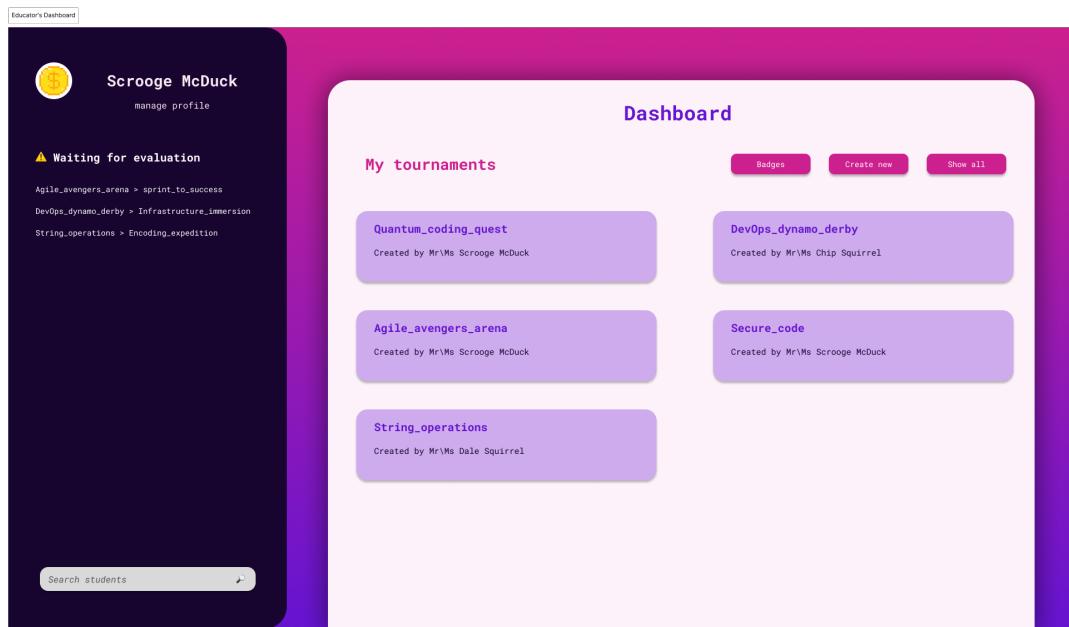


Figure 3.4: Dashboard

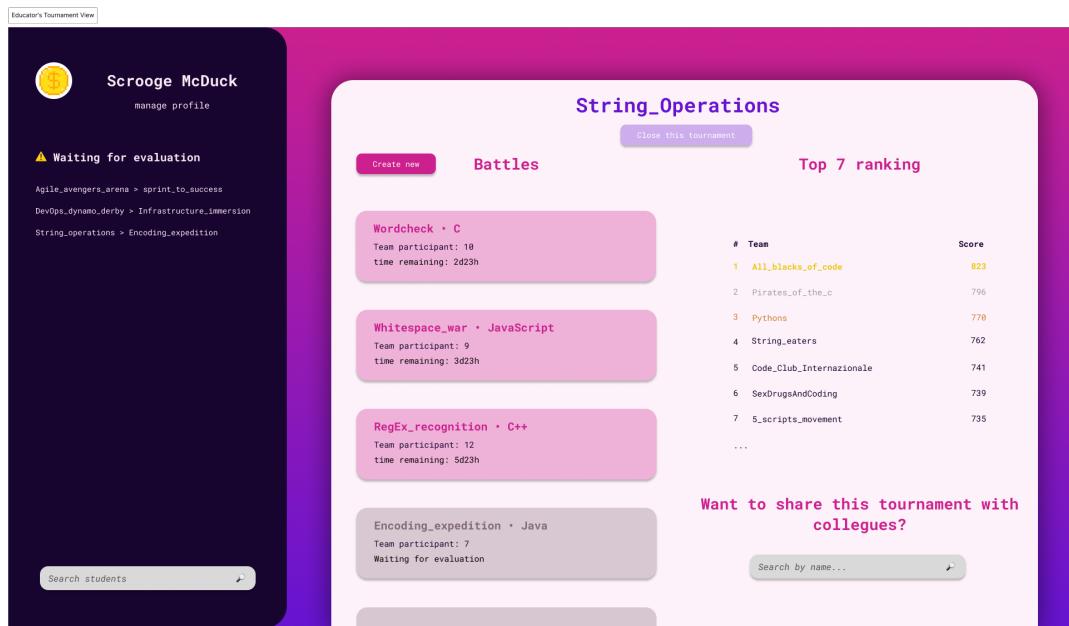


Figure 3.5: View of a tournament

The screenshot shows the 'Educator's Battle View (Manual evaluation stage)' interface. On the left, there's a sidebar with a user icon and the name 'Scrooge McDuck'. Below it, a message says 'Waiting for evaluation' with a warning triangle icon. A navigation bar at the top lists 'Agile\_avengers\_arena > sprint\_to\_success', 'DevOps\_dynamo\_derby > Infrastructure\_immersion', and 'String\_operations > Encoding\_expedition'. At the bottom of the sidebar is a search bar labeled 'Search students'. The main area is titled 'String\_Operations > Encoding\_expedition'. It contains a table with columns 'rk #', 'Team name', 'Score', and 'Action'. The table lists ten teams with their respective scores and actions (Edit or Evaluate). A button at the bottom right says 'Confirm grades and close battle'.

rk #	Team name	Score	Action
1	SexDrugsAndCoding	75/100	Edit
2	String_eaters	20/100	Edit
3	Code_Club_Internazionale	10/100	Edit
4	All_blocks_of_code	N/D	Evaluate
5	Insert_team_name_here	N/D	Evaluate
6	Mir_Kinder_vom_Milan_Zoo	N/D	Evaluate
7	5_scripts_movement	N/D	Evaluate
8	2001_a_code.odyssey	N/D	Evaluate
9	Mr_Jekyll_and_dr_pyde	N/D	Evaluate
10	The_three_musketeers	N/D	Evaluate

Figure 3.6: Manual evaluation page

The screenshot shows the 'Creation of new badge' interface. On the left, there's a sidebar with a user icon and the name 'Scrooge McDuck'. Below it, a message says 'Waiting for evaluation' with a warning triangle icon. A navigation bar at the top lists 'Agile\_avengers\_arena > sprint\_to\_success', 'DevOps\_dynamo\_derby > Infrastructure\_immersion', and 'String\_operations > Encoding\_expedition'. At the bottom of the sidebar is a search bar labeled 'Search students'. The main area is titled 'Gamification Badges'. It has two sections: 'My badges' (listing three badges: 'Perfect score in a battle', 'Perfect score in a tournament', and 'First battle push') and 'Create a new badge' (with fields for 'Name', 'Descriptions', 'Rules' (variable dropdown), and 'Value'). Below these are sections for 'Top used badges' (listing 'Polyglot' and 'Talkative') and a 'Badges' button. A link '+ Add one more rule' is visible.

Figure 3.7: Page to create a new badge

## STU's views

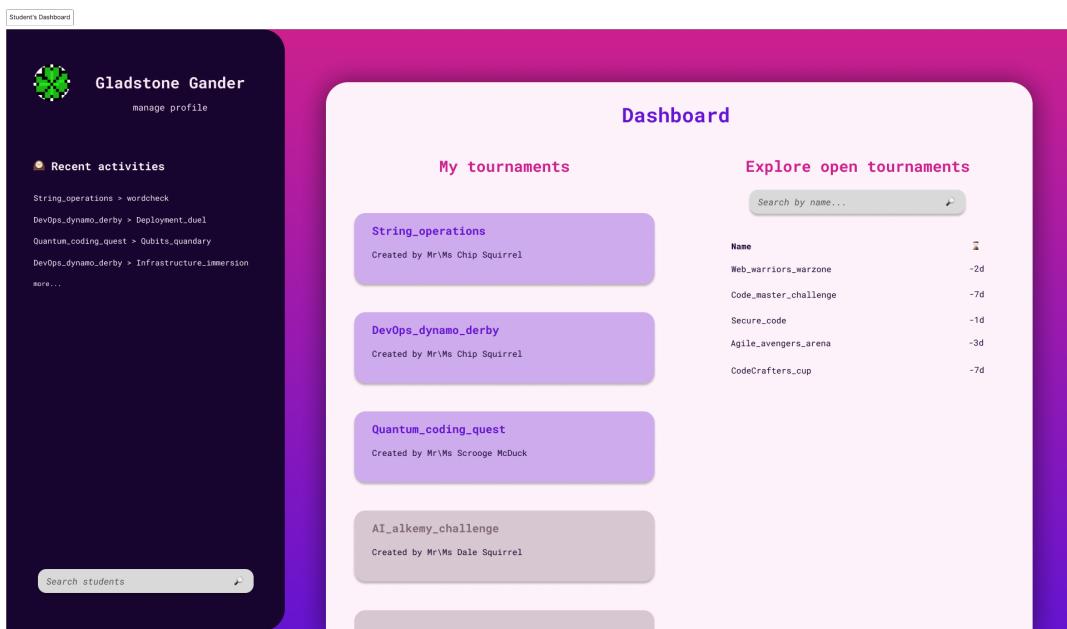


Figure 3.8: Dashboard

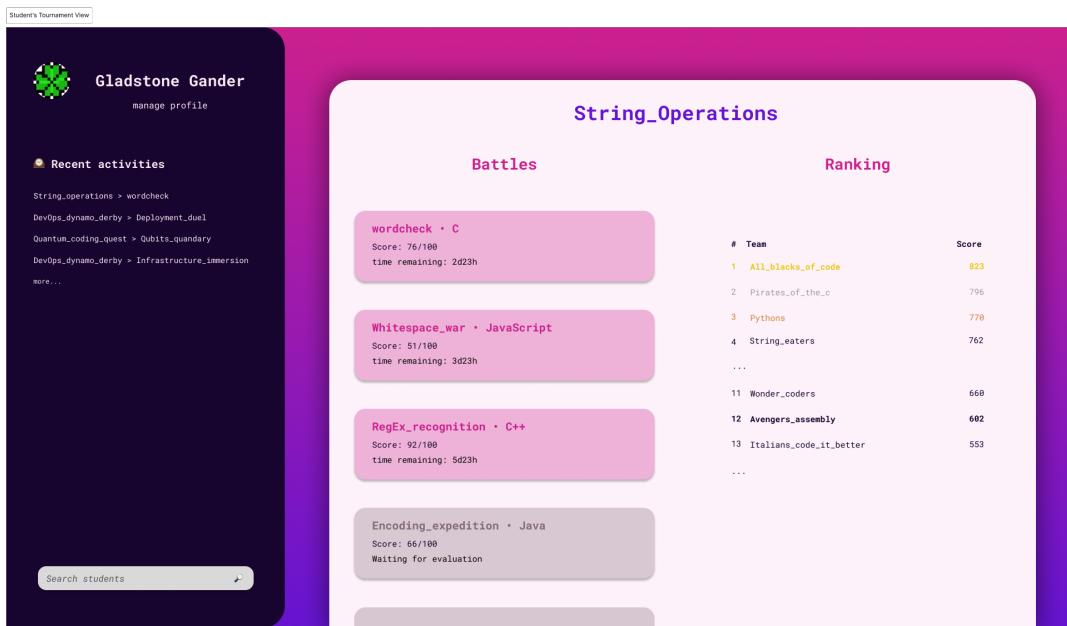


Figure 3.9: View of a tournament

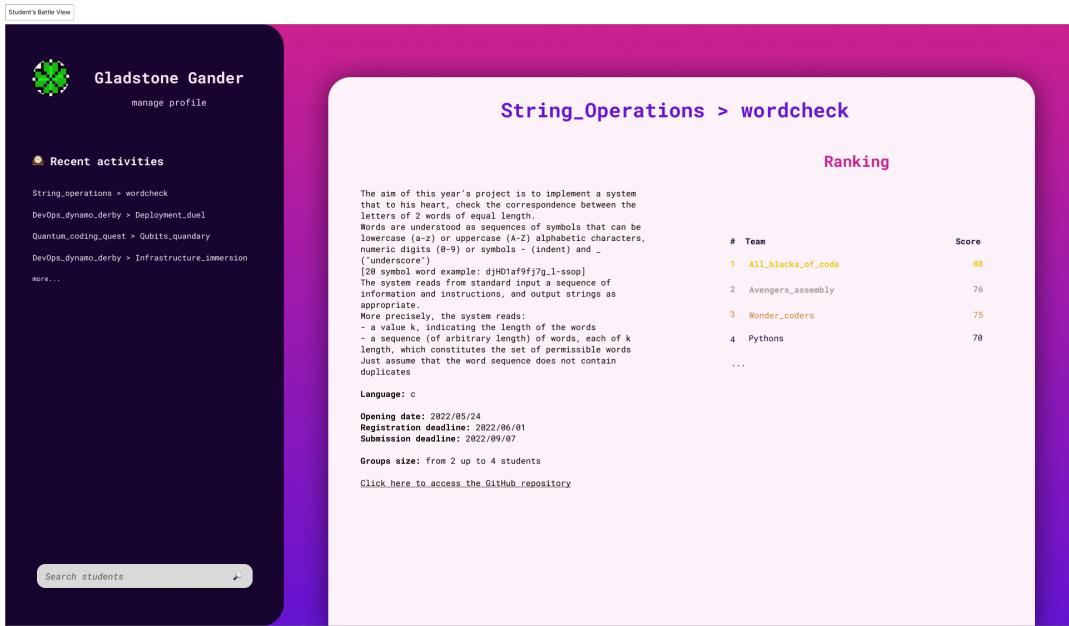


Figure 3.10: View of a battle

### 3.1.2 Hardware Interfaces

To use the CKB platform, both the Educators and the Students need an electronic device connected to the Internet, like a computer, a tablet or a smartphone. As the platform's primary functionality is closely tied to coding activities, it is expected that users will predominantly employ personal computers to access an Integrated Development Environment (IDE). Consequently, the platform's interfaces have been optimized for use on computer screens.

### 3.1.3 Software Interfaces

Since the platform is web-based, it is compatible with all the major operating systems, as long as they have a modern browser installed.

### 3.1.4 Communication Interfaces

The system requires a stable internet connection to work properly. The backend of the system will expose a unified RESTful API to communicate with all clients. Furthermore, the system relies on various external interfaces accessible via uniform web API. These services are:

- **GitHub API:** to create and manage repositories and to retrieve the code of the students and to authenticate users.
- **Mail API:** to send emails to the users to notify them about events.

## 3.2 Functional Requirements

In order to work properly, the software must fulfill the following functional requirements, which are written in hierarchical order, starting from the requirements of EDUs agents and then STUs agents...

### 3.2.1 Use cases Diagrams

### 3.2.2 Use cases Description

Name	Register
ID	UC1
Actors	EDU or STU
Entry conditions	The actor is not registered and wants to create an account
Event flow	<ol style="list-style-type: none"> <li>1. The actor enters the registration page</li> <li>2. The system shows the registration form</li> <li>3. The actor decides whether he is an educator or a student</li> <li>4. The actor fills the form with truthful informations</li> <li>5. The actor confirm the account creation</li> <li>6. The system creates an account with respect to the informations inserted by the actor</li> </ol>
Exit conditions	The system creates successfully the account
Exceptions	The actor inserts informations (i.e. email) about an account which already exists

Name	Log in
ID	UC2
Actors	EDU or STU
Entry conditions	The actor is already subscribed to the CKB platform
Event flow	<ol style="list-style-type: none"> <li>1. The actor enters the login page</li> <li>2. The system shows the login form</li> <li>3. The actor fills the form with its credentials and submits it</li> <li>4. The system checks the credentials and logs the actor in</li> </ol>
Exit conditions	The actor is logged in and it has access to the home page
Exceptions	The actor inserts a wrong combination of username - password: a human-readable message is displayed

Name	Battle creation
ID	UC4
Actors	EDU
Entry conditions	The actor is already subscribed to the CKB platform and he is an educator already logged in the platform
Event flow	<ol style="list-style-type: none"> <li>1. The actor opens the page correlated to the tournament in which he wants to create a new battle</li> <li>2. The system shows the tournament page</li> <li>3. If the actor has enough permissions, the system shows the "create a new battle" button</li> <li>4. The actor opens the battle creation page</li> <li>5. The system shows the battle creation form</li> <li>6. The actor fills the form with the details about the battle (i.e. code language, deadlines)</li> <li>7. The actor uploads the code kata</li> <li>8. The system checks if all fields are correctly filled</li> <li>9. The system creates the battle</li> </ol>
Exit conditions	The system creates successfully the new battle
Exceptions	<ol style="list-style-type: none"> <li>1. The actor has not enough permissions for the selected tournament.</li> <li>2. The actor uploads files not allowed</li> </ol>

Name	Sharing permission
ID	UC5
Actors	EDU
Entry conditions	A tournament is in progress <b>and</b> the actor is logged in <b>and</b> it is on the home page
Event flow	<ol style="list-style-type: none"> <li>1. The EDU selects a tournament he has created</li> <li>2. The system shows the tournament page</li> <li>3. The EDU clicks on the "Share permission" button</li> <li>4. The system shows the list of all the EDUs subscribed to the CKB platform</li> <li>5. The EDU selects the EDU(s) to which he wants to share the permission</li> </ol>
Exit conditions	The other EDU(s) can now start battles within the selected tournament
Exceptions	<i>None</i>
Note	In order to make the research of a specific EDU easier, the system provides a search bar

Name	Join a Battle
ID	UC7
Actors	STU
Entry conditions	The actor is an authenticated STU <b>and</b> a tournament is in progress <b>and</b> the STU is subscribed to the tournament
Event flow	<ol style="list-style-type: none"> <li>1. The actor opens the tournament page in which he is subscribed</li> <li>2. The system shows the battles of the tournament</li> <li>3. The actor opens the kata battle page</li> <li>4. If the subscribing deadline is not expired yet, the system shows the kata battle page with the function "join and create a team"</li> <li>5. The actor clicks on "join and create a team"</li> <li>6. The system subscribes the actor to the battle</li> </ol>
Exit conditions	The actor can now invite other STUs to his team or he can wait the start of the battle and participating alone
Exceptions	<i>None</i>
Note	The system provides a function to send via mail the link in order to let other STUs join the team created by the actor

Name	Manual evaluation
ID	UC9
Actors	EDU
Entry conditions	The deadline for submission a certain battle has expired <b>and</b> the battle has the "manual evaluation" enabled <b>and</b> the EDU is logged in <b>and</b> it is on the tournament page referred to that battle
Event flow	<ol style="list-style-type: none"> <li>1. The EDU selects a battle he has created</li> <li>2. The system shows the battle page, that has a list of all the teams that have submitted their work</li> <li>3. One by one, the EDU selects a team and the system shows the team code</li> <li>4. The EDU evaluates the code and assigns a score to the team (in the range [0, 100])</li> <li>5. The EDU clicks on the "Submit" button</li> <li>6. The system updates the battle score of the team and shows the next team's code to the EDU</li> </ol>
Exit conditions	For every team the final score is available, the battle has a final rank and all the participants to the battle get notified
Exceptions	<ul style="list-style-type: none"> <li>• The EDU doesn't evaluate the code of a team and clicks on the "Submit" button: a human-readable message is displayed</li> <li>• The EDU assigns a score that is not in the range [0, 100]: a human-readable message is displayed</li> <li>• The EDU stops the evaluation process: the system saves the scores assigned to the teams and the evaluation process can be resumed later from the first team that hasn't been evaluated yet</li> <li>• The EDU wants to change the score assigned to a team: the system lets the EDU change the score and then it updates the battle score of the team</li> </ul>

Name	Closing a tournament
ID	UC11
Actors	EDU
Entry conditions	The actor has the authorization to create battles within the tournament <b>and</b> it is on the tournament page <b>and</b> the actor is logged in
Event flow	<ol style="list-style-type: none"> <li>1. The EDU clicks on the "Close tournament" button</li> <li>2. The system shows a confirmation message</li> <li>3. The EDU clicks on the "Confirm" button</li> <li>4. The system closes the tournament and elaborates the final tournament rank</li> </ol>
Exit conditions	All the STUs enrolled to the tournament gets notified and can access to the tournament final ranking
Exceptions	The EDU clicks on the "Cancel" button: the system closes the message and shows normally the tournament page

Name	Visualizing tournament ranking
ID	UC13
Actors	EDU or STU
Entry conditions	The actor is logged <b>and</b> it is on the home page
Event flow	<ol style="list-style-type: none"> <li>1. The actor clicks on the "Tournaments" button</li> <li>2. The system shows the list of all the tournaments</li> <li>3. The actor selects a tournament, either ongoing or closed</li> <li>4. The system shows the tournament page</li> </ol>
Exit conditions	The actor can see the tournament rank
Exceptions	<i>None</i>
Notes	The tournament page shows both the rank and the list of battles within it, ongoing or ended

Name	Visualizing tournaments
ID	UCNON SO.1
Actors	EDU
Entry conditions	The actor is logged <b>and</b> he is an EDU <b>and</b> he is in his dashboard
Event flow	<ol style="list-style-type: none"><li>1. The actor clicks on the "My Tournaments" button</li><li>2. The system shows the list of all the tournaments the actor has created</li><li>3. The actor selects a tournament, either ongoing or closed</li><li>4. The system shows the tournament page</li></ol>
Exit conditions	The actor can see the tournament page
Exceptions	<i>None</i>

Name	Visualizing tournaments
ID	UCNON SO.2
Actors	STU
Entry conditions	The actor is logged <b>and</b> he is an STU <b>and</b> he is in his dashboard
Event flow	<ol style="list-style-type: none"> <li>1. The actor clicks on the "My Tournaments" button</li> <li>2. The system shows the list of all the tournaments to which the actor has subscribed</li> <li>3. The actor selects a tournament, either ongoing or closed</li> <li>4. The system shows the tournament page</li> <li>5. The actor returns to his dashboard</li> <li>6. The actor clicks the search bar and insert the name of a tournament he is looking for</li> <li>7. The system shows the tournament page corresponding to the search</li> </ol>
Exit conditions	The actor can see the tournament he has selected
Exceptions	If the tournament does not exists, the system suggests other tournaments showing their names and the subscribing deadlines
Notes	In the dashboard are shown both the suggested tournaments and the tournaments to which the actor is subscribed

Name	Visualizing battles in a tournament
ID	UC16
Actors	EDU or STU
Entry conditions	The actor is logged <b>and</b> it is on the home page
Event flow	<ol style="list-style-type: none"><li>1. The actor clicks on the "Tournaments" button</li><li>2. The system shows the list of all the tournaments</li><li>3. The actor selects a tournament, either ongoing or closed</li><li>4. The system shows the tournament page</li></ol>
Exit conditions	The actor can see the list of all battles within the tournament
Exceptions	<i>None</i>
Notes	The tournament page shows both the rank and the list of battles within it, ongoing or ended

Name	Push commit and score updating
ID	UC18
Actors	CKB platform, GitHub, STU
Entry conditions	The registration deadline for a battle has expired <b>and</b> STU is subscribed to the battle <b>and</b> the battle is not ended yet
Event flow	<ol style="list-style-type: none"> <li>1. STU pushes his changes to the GitHub repository to Github</li> <li>2. GitHub actions use the CKB platform 's API to inform of the new changes</li> <li>3. CKB platform pulls the latest sources and analyzes them</li> <li>4. CKB platform runs the tests on the corresponding executables</li> <li>5. CKB platform calculate and update the battle score of the team corresponding to STU</li> </ol>
Exit conditions	The STU's team can see the updated score
Exceptions	<i>None</i>
Notes	We have assumed that the STU setted properly the Github repository and the GitHub actions in the domain assumptions, so, by the side of the platform, nothing could go wrong

Name	Repository creation
ID	UC19
Actors	CKB platform, GitHub, STU
Entry conditions	The registration deadline for a battle has expired <b>and</b> STU is subscribed to the battle
Event flow	<ol style="list-style-type: none"> <li>1. The CKB platform creates a GitHub repository for the battle through the GitHub API</li> <li>2. The CKB platform sends an email to all STUs subscribed to the battle with the link to the GitHub repository</li> <li>3. The STU clicks on the link and is redirected to the GitHub repository</li> <li>4. The STU forks the repository</li> </ol>
Exit conditions	The STU and its team can start working on the coding battle
Exceptions	<i>None</i>
Notes	The correct functioning are assumed, so, by the side of the platform, nothing could go wrong

### 3.2.3 Use cases Sequence Diagrams

### 3.2.4 List of functional requirements

#### EDU functional requirements

- R1.** The software shall allow the unregistered EDUs to create an account
- R2.** The software shall allow the registered EDUs to login
- R3.** The software shall allow the authenticated EDUs to create new tournaments
- R4.** The software shall allow an authenticated EDU to grant to other authenticated colleagues the permission to create battles in his own tournament
- R5.** The software shall allow the authenticated EDUs to create coding kata battles in their own tournaments by letting them uploading the code kata, setting minimum and maximum number of STUs per group, setting a registration and a final submission deadlines, setting score configurations

- R6.** The software shall allow the authenticated EDUs to manually evaluate the work done by STUs subscribed to its own kata battle
- R7.** The software shall allow the authenticated EDUs to see the sources produced by each team partecipating to their own torunaments
- R8.** The software shall allow the authenticated EDUs to see the personal tournament score of each STU (which is the sum of all battle scores received in that tournament)
- R9.** The software shall allow the authenticated EDUs to see a rank that measures how a STU's performance compares to other STUs in the context of that tournament
- R10.** The software shall allow the authenticated EDUs to see the list of ongoing and finished tournaments as well as the corresponding tournament rank
- R11.** The software shall allow the authenticated EDUs to see the list of ongoing and finished battles as well as the corresponding battle rank
- R12.** The software shall allow an authenticated EDU to close a tournament if and only if it is one of the owners of that tournament
- R13.** When the authenticated EDU creates a tournament, the software shall allow it to define gamification badges concerning that specific tournament
- R14.** The software shall allow the authenticated EDU to create new badges and define new rules as well as new variables associated with them
- R15.** The software shall allow to all authenticated EDUs to visualize the badges: in particular, EDUs that visualize the profile of a STU can see its collected badges

### **STU functional requirements**

- R16.** The software shall allow the unregistered STUs to create an account
- R17.** The software shall allow the registered STUs to login
- R18.** The software shall allow the authenticated STUs to form teams by inviting other STUs respecting the minimum and maximum number of STUs per group set for that battle
- R19.** The software shall allow the authenticated STUs to join a team by an invite
- R20.** The software shall allow the authenticated STUs to subscribe to a tournament until a certain deadline
- R21.** The software shall allow the authenticated STUs subscribed to a kata battle to upload their work until the final submission deadline of that kata battle

- R22.** When the registration deadline of a kata battle expires, the software shall create a GitHub repository containing the code kata
- R23.** The software shall send to all authenticated STUs who are members of subscribed teams to a kata battle the link to a GitHub repository containing the code kata
- R24.** The software shall run the tests on executables pushed by a team and it shall also calculate and update the battle score of the corresponding team
- R25.** At the end of the consolidation stage of a specific battle  $b$ , the software shall send a notification to all authenticated STUs participating to  $b$  when the final battle rank becomes available
- R26.** The software shall allow the authenticated STUs to see the list of ongoing and finished battles as well as the corresponding battle rank
- R27.** The software shall allow to all authenticated STUs to see the personal tournament score of each STU (which is the sum of all battle scores received in that tournament)
- R28.** The software shall allow to all authenticated STUs to see a rank that measures how a STU's performance compares to other STUs in the context of that tournament.
- R29.** The software shall allow to all authenticated STUs to see the list of ongoing and finished tournaments as well as the corresponding tournament rank
- R30.** The software shall notify all authenticated STUs involved in a closed tournament when the final tournament rank becomes available
- R31.** The software shall allow to all authenticated STUs to visualize the badges: in particular, STUs can see collected badges when they visualize the profile of a STU

### 3.2.5 Traeciability matrices

#### Mapping of functional requirements on use cases

Use case	Functional requirements
UC1	
UC2	R2, R17
UC3	
UC4	
UC5	R4
UC6	

UC7	
UC8	
UC9	R6, R7
UC10	
UC11	R12
UC12	
UC13	R8, R9, R10, R27, R28, R29
UC14	
UC15	
UC16	R11, R26
UC17	
UC18	
UC19	R21, R23

**Mapping  $D \wedge R \models G$** 


---

G1: Educators can create tournaments that involve coding battles to challenge students.

D2, D5, D6	R1, R2, R3, R4, R5, R6, R12, R13, R16, R17, R20, R21, R22, R23
------------	---

---

G2: Provides educators with the ability to track student software development knowledge.

D2, D3, D4, D5, D6, D8	R1, R2, R6, R7, R8, R9, R10, R11, R13, R14, R15, R16, R17, R20, R21, R23, R24
------------------------	--

---

G3: Students can improve software development skills by taking part in coding tournaments and battles where they must write programs.

D1, D2, D3, D4,D5, D6, D7, D8	R1, R2, R3, R4, R5, R6, R7, R13, R14, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31
-------------------------------	--

---

G4: Coding battles enable students to enhance their soft skills, such as communication, collaboration, and time management, by creating a team and collaborating with the members.

D3, D5, D7, D8

R5, R16, R17, R18, R19

---

### **3.3 Performance Requirements**

In order to guarantee a good user experience, the system must:

- Make sure the backend can grow as needed, respond quickly to changes, and balance the workload effectively.
- Be protected against DDoS attacks to keep the system safe and stable.
- Create a user-friendly, responsive front-end. It should handle well even when the internet isn't great, so users have a smooth experience.
- Send email notifications really quickly, so users don't even notice the delay.

### **3.4 Design Constraints**

#### **Standards Compliance**

Since there are a lot of interactions between the various components of the system, it is important to follow some communication standards: in particular, the system will use the REST architectural style to communicate between the frontend and the backend, and the data will be exchanged in JSON format.

Furthermore, Source code of the application must be commented on and documented adequately.

#### **Hardware Limitations**

The system is designed to be used on any device with a modern browser installed, so the only hardware limitation is the presence of a stable internet connection.

### **3.5 Software System Attributes**

#### **3.5.1 Reliability**

Since some functionality of the system relies on external APIs, though the system should not completely fail because of failure in one of those.

It's also important to avoid data loss through redundant storage methods.

### **3.5.2 Availability**

In the event of an unplanned system downtime, all features should be restored as quickly as possible to minimize any inconvenience. To prevent such occurrences, it is crucial for the CKB platform to have a reliable infrastructure, including redundant servers, to ensure continuous operation. The aimed availability of the system is 3-nines availability (99.9%), which means that the system can be down for a less than 9 hours per year. The system should also be able to handle a large number of concurrent users.

### **3.5.3 Security**

Users of the system have distinct privileges according to their roles (student / educator), determined during the login process.

All data and information transferred and stored within the system are secured through robust encryption methods, such as HTTPS, ensuring data privacy and security.

### **3.5.4 Maintainability**

The source code and associated documentation must include clear comments and should be consistently maintained. During the design and development phases, emphasis should be placed on achieving modularity, minimizing coupling, and ensuring high cohesion between components. This is especially crucial for both the front-end and back-end, allowing developers to make updates to the back-end seamlessly without causing any disruptions or noticeable changes for users.

To avoid inconvenience in solving any type of problem (e.g. server downtime), maintenance services are notified to all users with an advance notice of at least 36 hours.

### **3.5.5 Portability**

Due to the fact that the CKB platform is a distributed system, and it doesn't rely on a specific hardware or software, it can be used / accessed in multiples way.

## 4. Formal Analysis Using Alloy

```
// Obviously, this is a mockup of the Alloy code that
// Michelangelo will write
sig Person {
    name: one String,
    age: one Int,
    address: one Address
}

sig Address {
    street: one String,
    city: one String,
    state: one String,
    zip: one Int
}

sig Car {
    model: one String,
    owner: lone Person
}

sig Company {
    name: one String,
    employees: set Person
}

sig Department {
    name: one String,
    company: one Company,
    employees: set Person
}

fact {
    all c: Car | c.owner.age > 18
```

```
    all p: Person | p.age > 0 and p.age < 120
    all a: Address | a.zip > 0 and a.zip < 100000
    all c: Company | #c.employees > 0
    all d: Department | #d.employees > 0
}

run {
    some p: Person | #p.age > 30
    some c: Company | #c.employees > 100
} for 5
```

## 5. Effort Spent

### Team

Topic	Time
Division of work	2h
Revision of chapters 1 and 2	2h
Definition and division of use cases	2h30m

Table 5.1: Effort Spent during team meetings

### Tommaso Pasini

Topic	Time
Organizaionion document	1.5 h
Completion and correction chapter 1	4h

Table 5.2: Effort Spent by Tommaso Pasini

*Effort Spent*

---

## Elia Pontiggia

Topic	Time
Scenarios	1h
State diagrams	2h
Specific requirements	2h30m
Use cases	2h
User interface mockups	5h
LAT <sub>E</sub> X document setup and configuration	2h

Table 5.3: Effort Spent by Elia Pontiggia

## Michelangelo Stasi

Topic	Time
Functional Requirements	2h
Domain Assumptions	1h30m
Use Cases	2h
Traeciability Matrix	30m
Class Diagram	1h

Table 5.4: Effort Spent by Michelangelo Stasi