



POLITECNICO
MILANO 1863

Department of Electronics, Information and Bioengineering
Doctoral Programme In Information Technology

Software Engineering 2 Requirements Analysis and Specification Document

Author(s): **Filippo Balzarini - 10719101**

Christian Biffi - 10787158

Michele Cavicchioli - 10706553

Contents

Contents	i
1 Introduction	1
1.1 Purpose	1
1.1.1 Goals	2
1.2 Scope	3
1.2.1 World phenomena	3
1.2.2 Shared phenomena	3
1.3 Definitions, Acronyms, Abbreviations	4
1.3.1 Definitions	4
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	7
2.1 Product perspective	7
2.1.1 Class Diagram	7
2.1.2 State Diagrams	8
2.1.3 Scenarios	9
2.2 Product functions	11
2.3 User characteristics	13
2.4 Assumptions, dependencies and constraints	13
3 Specific Requirements	15
3.1 External Interface Requirements	15
3.1.1 User Interfaces	15

3.1.2	Hardware Interfaces	15
3.1.3	Software Interfaces	15
3.1.4	Communication Interfaces	16
3.1.5	List of Requirements	16
3.1.6	Mapping on Goals	19
3.1.7	Use Cases	28
3.2	Performance Requirements	52
3.3	Design Constraints	52
3.3.1	Standard compliance	52
3.3.2	Hardware limitation	52
3.3.3	Any other constraint	53
3.4	Software System Attributes	53
3.4.1	Reliability	53
3.4.2	Availability	53
3.4.3	Security	53
3.4.4	Maintainability	53
3.4.5	Portability	53
4	Formal Analysis Using Alloy	55
4.1	Alloy Code	55
4.2	Assertion Results	65
4.3	World Generated	66
5	Effort Spent	67
6	References	69
6.1	Tool Used	69

1 | Introduction

1.1. Purpose

The CodeKata is a learning method that takes inspiration from the Kata techniques and is based on continuous practice which became very popular in those years.

CodeKataBattle delineates an innovative platform geared towards enhancing students' software development skills through collaborative learning using CodeKata's fundamentals. Facilitated by educators, CKB provides a dynamic environment where students engage in code kata battles, refining their programming proficiency and embracing best practices such as the test-driven development approach.

CKB empowers educators to orchestrate challenges within competition, stimulating healthy competition and cultivating an environment for skill enhancement. The platform enables educators to define battle parameters, set deadlines, and configure scoring criteria, fostering a tailored and effective learning experience.

At its core, a code kata battle presents students with programming challenges within specific language frameworks, coupled with exhaustive test cases. Teams collaboratively tackle these exercises, adhering to a test-first methodology and submitting solutions to the platform upon battle completion.

CKB's automated evaluation system ensures an impartial assessment of student submissions. Automated scrutiny covers mandatory factors, including functional aspects, timeliness, and source code quality, offering an unbiased representation of team performance. Educators can further enhance evaluations with optional manual assessments, providing nuanced insights into student work.

1.1.1. Goals

The platform will be used by two types of users: Educators (ED) and Students (ST). The ED will be able to create competitions and battles within competitions. The ST will be able to create teams and join battles as a team or individually. The platform will communicate with Github to pull the latest code submission and provides automated evaluation of the code submitted. The platform will also show a ranking of the competition and battles.

Below there is the table of goals that the platform will achieve:

#	Goal
G1	Enable ED to manage competitions
G2	Enable ED to manage code battles within competitions
G3	Enable ST to participate in a competition
G4	Enable ST to be part of a team within a battle
G5	Send Notifications to STs
G6	Automatically create GitHub repositories for every battle in a competition
G7	Synchronize the submission of each candidate with their GitHub repository
G8	CKB provides an evaluation of the code submitted
G9	Allow users to view rankings in both battles and competitions
G10	Allow to assign badges to the students
G11	Allow users to visualize ST profiles

Table 1.1: List of goals

1.2. Scope

1.2.1. World phenomena

ID	Definitions
WP1	ED wants to create a competitions
WP2	ED wants to create a battle
WP3	ST wants to participate in a competition
WP4	ST wants to participate in a battle
WP5	ST set up GitHub actions

Table 1.2: List of the world phenomena

1.2.2. Shared phenomena

ID	Definitions
SP1	ST creates an account on the platform
SP2	ED creates an account in the platform
SP3	ST logs in to the platform
SP4	ED logs in to the platform
SP5	ST registers for the competitions before the deadline
SP6	ED creates a badge with certain rules
SP7	ED manually evaluates the code submitted by students
SP8	ED creates a competition
SP9	ED creates a battle within a competition
SP10	ED closes a competition
SP11	ST pushes new commit(s) into their GitHub repository before the deadline

SP12	ST invites other STs to participate in a battle as a team
SP13	ST subscribes as a single/team for an incoming battle before the deadline
SP14	CKB sends a notification that a competition is available to ST
SP15	CKB sends a notification that a battle is created inside a competition to ST
SP16	CKB sends a notification that a competition has ended to ST
SP17	CKB sends a notification that a battle has ended to ST
SP18	CKB sends links to the GitHub repository to all the ST subscribed
SP19	CKB updates scores for each ST
SP20	CKB gives badge to ST
SP21	CKB updates the ranking of the competition
SP22	CKB updates the ranking of the battle

Table 1.3: List of the shared phenomena

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

User	Anyone interacting with the system, it can be both a Student or an Educator
Manage	Create, supervise and edit a certain element of the application.
Code Kata	A challenge intended to improve programming abilities, including description, test cases and

Table 1.4: List of definitions

1.3.2. Acronyms

ST	Student
ED	Educator
CKB	CodaKataBattle
RASD	Requirements Analysis and Specification Document
SAT	Static Analyzer Tool
T	Team

Table 1.5: List of Acronyms

1.3.3. Abbreviations

WPX	World Phenomena X
SPX	Shared Phenomena X
GX	Goal Number X
DX	Domain Assumption X
UCX	Use Case X

Table 1.6: List of abbreviations

1.4. Revision history

1.5. Reference Documents

- The specification document of the project: *Assignment RDD AY 2023-2024*

1.6. Document Structure

The document is divided in five main section described as below.

The first section is the introduction, that introduce the goals of the project, purpose and the analysis of world and shared phenomena. It also contains the definitions, acronyms and abbreviations used in the document.

The second section is the overall description, that contains the general factors that affect the product. Here there is also the analysis of the scenarios and functions of the platform and the domain assumptions.

Then as third section there is the specific requirements section, that contains the functional and non-functional requirements of the platform. Moreover, there is a more detailed analysis of the use cases and the mapping between goals and requirements. Then there is a description of the interfaces necessary for the platform to implement all the functionalities.

The fourth section is the formal analysis using Alloy. Here there is the description of the model and the world generated by the Alloy Analyzer. This section is very important to prove the correctness of the model described in the previous sections.

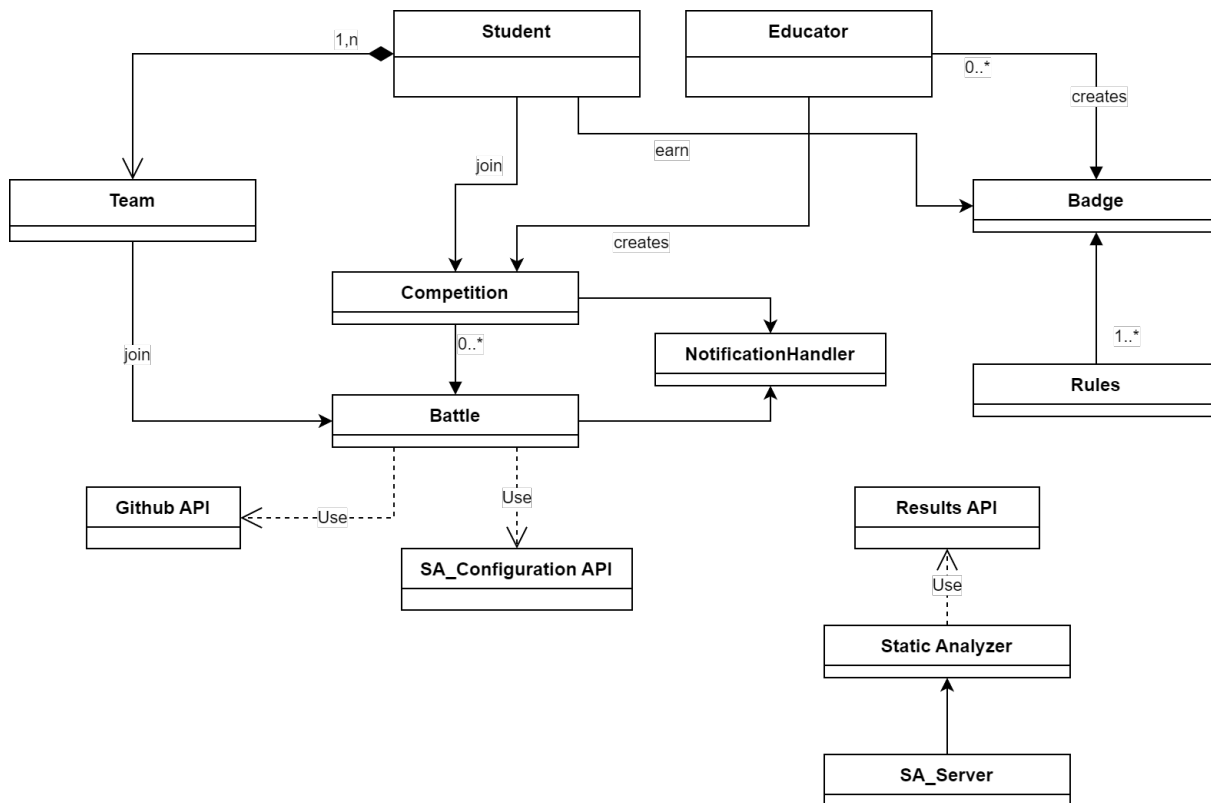
The last section is the effort spent by each member of the group to write this document.

2 | Overall Description

2.1. Product perspective

2.1.1. Class Diagram

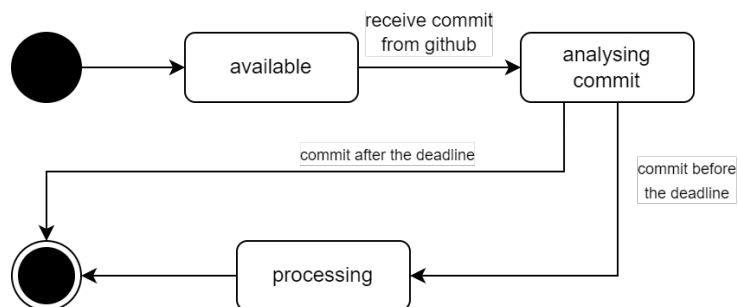
Here we provide the class diagram of our system and a brief description on some of the key aspects. The very first thing to explain is the logical split of the *Static Analyzer* from the rest of the system, the reason for this is that we wanted to have a system that could be more *flexible* than usual. The two logical parts can be implemented in the same tier or can also be split physically in terms of devices. What does not change is that when the STs set up github actions on their repository, the server that will be called when a commit is pushed is the ***SA_Server***, which will do the necessary checks and then sends the results to the main server (***Results API*** has this purpose), where the score will be computed. Note that the Static Analyzer can be configured from the main server by using the provided APIs in ***SA_Configuration API*** (e.g., upload the test cases...). As asked by the assignment we distinguished two types of users, ***Students*** and ***Educators***, in order to differentiate the features the platform offers to the two. To handle the participants of a battle we decided to represent every one of them as teams, therefore we can have a team that is composed by a single student, this will be very useful when analysing such teams to understand if they meet the criteria to join a battle. The ***NotificationHandler*** has the purpose of handling the notifications of the users and it provides the procedures to do just so.



2.1.2. State Diagrams

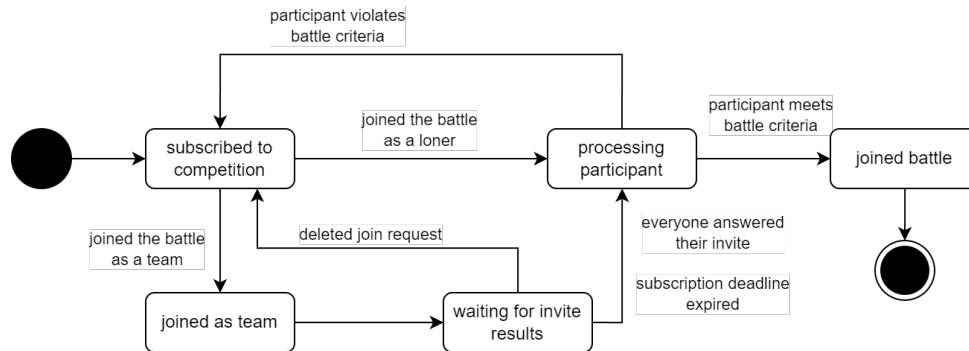
Commit in github repository

Here there is the state diagram related to the system that processes the push of a commit in a ST repository.



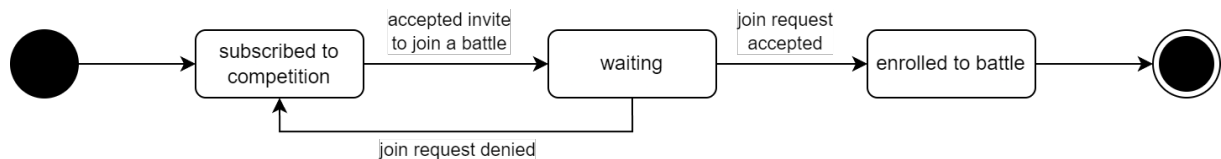
Join battle

Here you can find the state diagram representing the process of joining a battle from the system perspective.



Accept invite to join battle

Below you can find the state diagram representing the process of a ST accepting an invite to join a battle.



2.1.3. Scenarios

Scenario 1

Professor Harry is a professor teaching at Politecnico di Milano together with professor Donald. Harry would like to encourage his students to study during the course, instead of having to study everything a few days before the exam. To do so he came up with the idea to create a challenge where the students can test their preparation and earn some extra points in the exam. While talking with other colleagues, Prof. Harry discovered CKB and he thought it was the perfect fit to implement his idea. The first thing he does is to go to the webpage of CKB and create an account by clicking the **Sign up** button and providing some information about himself. Afterwards he is redirected to the home page of the platform where he can click the button **Create Competition**, and finally he inserts the name of the competition and the subscription deadline. At this point he wants to invite his colleague Donald to manage the competition with him; since he is an ED in the competition he can click on **Invite Educator** in the competition page, then provides the email of Donald's account, who will be part of the competition once he accepts the

invite.

Scenario 2

Professor Harry is an ED of a competition, within which he wants to create a battle. To do so he enters the dashboard of the competition, clicks on the button **Create Battle** and provides everything the platform needs: description, test cases, build automation scripts, deadlines, accepted sizes of groups. Marco, a ST who subscribed to the competition, received the notification about the newly created battle via email. Outside of the platform Marco agreed with a couple of friends to participate in the battle together. Marco then goes on the competition page and finds the newly created battle, here he finds two buttons, **Participate as: 1. Loner 2. Team**; he clicks the second button to participate as a team and invites his friends by providing the platform his friends' account email. Once Marco's friends accepted the invite the subscription to the battle will be automatically finalized by the platform.

Scenario 3

Professor Harry wants to give credit to the hardest working student, so while creating the competition he decided to create a new badge. The hardest working ST is the one that has written the highest amount of code lines among all the battles in the same competition. To implement this badge Prof. Harry must create a new variable **hardest_worker** and provide the code that defines how to compute the value of such variable. Some time after the specification of this new badge, a ST, participating in the competition and in the current battle, called Marco, pushes a commit to his repository. Since all students are supposed to setup *GitHub Actions*, CKB is notified about Marco's commit, so it proceeds to run the required processes to calculate the new score, but also checks if Marco acquired new badges by checking their rules. Assuming that with the last commit Marco has now the most written lines of code, CKB assigns to him the **hardest_worker** badge.

Scenario 4

Marco and his team participated in a battle provided by Prof. Harry, one of the ED of the competition. Since the battle ends the next day, Marco wants to look at the partial rankings of the battle, so he goes on the page related to the battle and clicks on the **Results** section, and sees his team at the bottom of the chart. Understandably, Marco's team resumes to work on the problem and they are able to commit a new version of their solution, which increased their placement in the partial rankings of the battle. The

submission deadline now expired and the EDs now want to manually check the work of their STs to assign manually a score to each team; to do this Prof. Harry goes on the battle page and clicks on ***Perform Manual check***, which will redirect the ED to another page where he can inspect the source code of each team and give a score to each final work. Once this consolidation phase has been declared finished by an ED, CKB sends to all the STs subscribed to the competition a notification that the battle's results are available and the global scores of the competition have been updated.

2.2. Product functions

The ED can create a new competition

The CKB allows the ED to create a new competition by clicking on the ***Create Competition*** button on the home page, then providing the name of the competition and the subscription deadline.

The ED inside of the competition have the possibility to create a new battle by clicking on the ***Create Battle*** button, where he can then insert the description, test cases and the solution. He can also set which aspects of the code he wants that CKB evaluate, such as the quality of the code, the number of lines of code, the security, the readability and the maintainability. Moreover he can select the minimum and maximum number of team components and the deadline for the submission.

Inside of each battle the ED can add badges by clicking on the ***Add Badge*** button, where he can then insert the name of the badge, the description and the rules to assign the badge.

The ED can also invite other EDs to manage the competition with him by clicking on the ***Invite Educator*** button and providing the email of the account of the ED he wants to invite. This will allow the colleague to change the settings of the competition and create new battles.

The ST can participate in a battle

The ST, assuming it is enrolled in a competition, can see the list of the battles available, and can subscribe to them by clicking on the ***Subscribe*** button. When the student clicks on the button he can choose to participate as a loner or as a team, in the latter case he has two possibilities:

- Create a new team by clicking on the ***Create Team*** button, where he can then

insert the name of the team and the email of the other STs he wants to invite (CKB will send an invite via email to those addresses). Once the other STs accepted the invite the subscription to the competition will be automatically finalized by the platform.

- To join an existing team a ST can click on the ***Join Team*** button, where he can then insert the name of the team he wants to join. Another option is to accept an invite that a team leader sent to another ST.

Inside the competition ST can also see the general ranking of the competition, which is updated after each battle. It is also possible to see the partial ranking of each battle he has participated in, which is updated after each submission.

Another feature available to the ST is the possibility to see the list of the badges he earned, and the list of the badges he can earn with the corresponding rules.

The CKB can evaluate the submissions

The CKB is able to automatically evaluate the submissions of the STs by running the test cases provided by the EDs. Each new code submission made on the Github repository of the STs is notified to the CKB, which then runs the test cases on the code.

The test cases are run in a sandbox environment to prevent malicious code from damaging the system. The CKB is also able to run the build automation scripts provided by the EDs to check if the code compiles and if it satisfies the requirements.

The code is evaluated also considering the quality of the sources by running static analysis tools on the code that considers the complexity of the code, the readability and the maintainability.

The CKB after performing all the checks assigns a score to the submission and updates the ranking of the battle and the competition. It also checks if the STs earned new badges by checking their rules and, in case, assigns them to the STs.

It is also possible for the ED managing the battle to manually evaluate the submissions by clicking on the ***Perform Manual Check*** button on the battle page. This will redirect the ED to another page where he can inspect the source code of each team and give a score to each final work. Once this consolidation phase has been declared finished by an ED, CKB sends to all the STs subscribed to the competition a notification that the battle's results are available and the global scores of the competition have been updated.

2.3. User characteristics

The actors that are going to use the CKB system are:

- **Educator (ED)**: an educator is a user that can create competitions and battles within competitions. He can set the parameters of the battles and the deadlines. He can also invite other educators to manage the competition with him.
- **Student (ST)**: a student is a user that can create teams and join battles as a team or individually. He can earn points and badges by participating in the competitions and battles.

2.4. Assumptions, dependencies and constraints

ID	Description
DA1	ST owns a device able to connect to the internet
DA2	ST owns a GitHub account
DA3	ST has installed Git on his computer
DA4	ST knows how to use Git
DA5	ED knows how to use Git
DA6	ED owns a device able to connect to the internet
DA7	ED writes correct tests
DA8	ED correctly evaluates the final source code of a T
DA9	GitHub permits automatic push to a repository
DA10	GitHub permits automatic pull from a repository
DA11	ST knows the usernames of other STs they want to invite to a T
DA12	ST has an internet connection
DA13	ED has an internet connection
DA14	ST writes code only with languages that are treatable by the platform

DA15	ED knows the email of the other EDs he wants to invite to manage a competition
DA16	ED writes the correct badge' rules
DA17	User knows the username of the STs' profile they want to visualize

Table 2.1: List of the domain assumption

3 | Specific Requirements

3.1. External Interface Requirements

3.1.1. User Interfaces

The CKB user interface will be a web page that will be accessed through a web browser. The web page will be designed to be simple and easy to use with the support for multiple screen sizes and devices.

3.1.2. Hardware Interfaces

The platform requires a computer with a web browser and an internet connection to access the CKB web page.

3.1.3. Software Interfaces

CKB will be using some software interfaces both from external systems and internal implemented ones in order to provide the services. The interfaces used are listed below:

- **Github API:** CKB will use Github as source control system for the projects. The Github API will be used to create the repositories for each team in the battle and share it with the team members.
- **Static Analyzer:** CKB will use a static analysis tool to analyze the code of each team after every new commit and will use the result of the analysis to assign points to each team. The purpose of this interface is to give the possibility to the system to configure the analyzer as the educator needs, in terms of test cases, languages supported and other parameters.
- **Results API:** The *SA_Server* will use this API to send the results of the analysis to the main server.
- **Github Actions:** Github Actions will notify the system when a participant has

pushed a new commit to its repository. The *SA_Server* will be listening to this notifications and will trigger the analysis of the source code.

3.1.4. Communication Interfaces

All the communication between CKB, the external interfaces and the user will be done using HTTPS protocol.

3.1.5. List of Requirements

Requirement	Description
R1	CKB shall allow an unregistered user to create an account
R2	CKB shall allow users to log in
R3	CKB shall allow ED to create competition
R4	CKB shall allow ED to create battle within a competition that he/she manages
R5	CKB shall allow ED to invite other EDs to manage battles in a competition
R6	CKB shall allow ED to upload the code kata
R7	CKB shall allow ED to set a registration deadline to the battle
R8	CKB shall allow ED to set a minimum number of STs per group in a battle
R9	CKB shall allow ED to set the maximum number of STs per group in a battle
R10	CKB shall allow ED to set a final submission deadline
R11	CKB shall allow ED to set how to perform static analysis
R12	CKB shall allow ST to subscribe to a competition
R13	CKB shall send notifications about a new competition to ST

R14	CKB shall send notification about battle created within a competition ST are subscribed in
R15	CKB shall allow ST to join a battle on his own
R16	CKB shall allow ST to invite other ST in a T for a battle
R17	CKB shall create a GitHub repository containing the description, software project and the build automation scripts
R18	CKB shall send the Github repository link to ST member of a T competing in the battle
R19	CKB shall supply API to call with Github actions
R20	CKB shall be able to pull sources from GitHub
R21	CKB shall be able to send the ST source code to the correct SAT
R22	CKB shall be able to receive the evaluation given by SAT on a source code
R23	CKB shall be able to run tests on code
R24	CKB shall evaluate the code in terms of test cases passed
R25	CKB shall evaluate the code in terms of timeliness
R26	CKB shall allow ED to assign a score to codes
R27	CKB shall update the score of a T (as soon as new push actions are performed)
R28	CKB shall allow ED to go through sources produced by Ts
R29	CKB shall notify ST when final battle ranks are available
R30	CKB shall update the personal competition score of a ST at the end of each battle
R31	CKB shall create a rank with students' performances in a competition
R32	CKB shall allow ST to see all ST's rank in battle where is enrolled

R33	CKB shall allow ED to see all ST's ranks in the battle that he/she manages
R34	CKB shall allow EDs and STs to see all ST's rank in competitions
R35	CKB shall allow ST to see the list of ongoing competitions
R36	CKB shall allow ED to close a competition
R37	CKB shall allow ED to define badges in the context of a competition
R38	CKB shall assign badges to students at the end of the competition
R39	CKB shall allow ED to define new rules for badges
R40	CKB shall allow ED to define new variables for badges
R41	CKB shall allow users to visualize badges obtained by a ST
R42	CKB shall allow users to visualize a ST profile
R43	CKB shall allow ST to join a T for which is invited
R44	CKB shall allow ST to join a public T
R45	CKB shall allow ST to create a T
R46	CKB shall allow ST to set a T to public or private
R47	CKB can distinguish between an ED user and a ST user
R48	CKB shall not allow ST/ED to see the rankings of battles they are not involved in
R49	CKB shall have the environments for all the programming language it supports
R50	CKB shall allow ED to close a battle they manage

3.1.6. Mapping on Goals

Goal	Requirements	Domain Assumptions
G1	R1 R2 R3 R5 R36 R37 R39 R40 R47	DA6 DA13 DA15 DA16
G2	R1 R2 R4 R5 R6 R7 R8 R9 R10 R11 R33 R47 R48 R50	DA6 DA7 DA13
G3	R1 R2 R12 R13 R30 R34 R35 R47	DA1 DA12
G4	R1 R2 R14 R15 R16 R43 R44 R45 R46 R47	DA1 DA11 DA12
G5	R13 R14 R16 R29 R47	DA1 DA11 DA12
G6	R17 R18 R19	DA9
G7	R2 R19 R20	DA3
G8	R20 R21 R22 R23 R24 R25 R26 R27 R28 R30 R31 R49	DA5 DA6 DA7 DA8 DA10 DA13 DA14
G9	R1 R2 R29 R30 R31 R33 R32 R34 R48	DA1 DA6 DA12 DA13
G10	R37 R38 R39 R40 R41 R47	DA16
G11	R1 R2 R41 R42	DA1 DA6 DA12 DA13 DA17

Table 3.2: Mapping between goals, requirements, and domain assumptions

G1	Enable ED to manage competitions
R1	CKB shall allow an unregistered user to create an account
R2	CKB shall allow users to log in
R3	CKB shall allow ED to create competition
R5	CKB shall allow ED to invite other EDs to manage battles in a competition
R36	CKB shall allow ED to close a competition
R37	CKB shall allow ED to define badges in the context of a competition
R39	CKB shall allow ED to define new rules for badges
R40	CKB shall allow ED to define new variables for badges
R47	CKB can distinguish between an ED user and a ST user
DA6	ED owns a device able to connect to the internet
DA13	ED has an internet connection
DA15	ED knows the email of the other EDs he wants to invite to manage a competition
DA16	ED writes the correct badge' rules

Table 3.3: Specific mapping on G1

G2	Enable ED to manage Code Battles within Competitions
R1	CKB shall allow an unregistered user to create an account
R2	CKB shall allow users to log in
R4	CKB shall allow ED to create battle within a competition that he/she manages
R5	CKB shall allow ED to invite other EDs to manage battles in a competition

R6	CKB shall allow ED to upload the code kata
R7	CKB shall allow ED to set a registration deadline to the battle
R8	CKB shall allow ED to set a minimum number of STs per group in a battle
R9	CKB shall allow ED to set the maximum number of STs per group in a battle
R10	CKB shall allow ED to set a final submission deadline
R11	CKB shall allow ED to set how to perform static analysis
R33	CKB shall allow ED to see all ST's ranks in the battle that he/she manages
R47	CKB can distinguish between an ED user and a ST user
R48	CKB shall not allow ST/ED to see the rankings of battles they are not involved in
R50	CKB shall allow ED to close a battle they manage
DA6	ED owns a device able to connect to the internet
DA7	ED writes correct tests
DA13	ED has an internet connection

Table 3.4: Specific mapping on G2

G3	Enable ST to participate in a Competition
R1	CKB shall allow an unregistered user to create an account
R2	CKB shall allow users to log in
R12	CKB shall allow ST to subscribe to a competition
R13	CKB shall send notifications about a new competition to ST
R30	CKB shall update the personal competition score of a ST at the end of each battle
R34	CKB shall allow EDs and STs to see all ST's rank in competitions
R35	CKB shall allow ST to see the list of ongoing competitions
R47	CKB can distinguish between an ED user and a ST user
DA1	ST owns a device able to connect to the internet
DA12	ST has an internet connection

Table 3.5: Specific mapping on G3

G4	Enable ST to be part of a team within a battle
R1	CKB shall allow an unregistered user to create an account
R2	CKB shall allow users to log in
R14	CKB shall send notification about battle created within a competition ST is subscribed in
R15	CKB shall allow ST to join a battle on his own
R16	CKB shall allow ST to invite other ST in a T for a battle
R43	CKB shall allow ST to join a T for which is invited
R44	CKB shall allow ST to join a public T
R45	CKB shall allow ST to create a T

R46	CKB shall allow ST to set a T to public or private
R47	CKB can distinguish between an ED user and a ST user
DA1	ST owns a device able to connect to the internet
DA11	ST knows the usernames of other STs they want to invite to a T
DA12	ST has an internet connection

Table 3.6: Specific mapping on G4

G5	Send Notifications to STs
R13	CKB shall send notifications about a new competition to ST
R14	CKB shall send notification about battle created within a competition ST is subscribed in
R16	CKB shall allow ST to invite other ST in a T for a battle
R29	CKB shall notify ST when final battle ranks are available
R47	CKB can distinguish between an ED user and a ST user
DA1	ST owns a device able to connect to the internet
DA11	ST knows the usernames of other STs they want to invite to a T
DA12	ST has an internet connection

Table 3.7: Specific mapping on G5

G6	Automatically Create GitHub Repositories for Every Battle in a Competition
R17	CKB shall create a GitHub repository containing the description, software project and the build automation scripts

R18	CKB shall send the Github repository link to ST member of a T competing in the battle
R19	CKB shall supply API to call with Github actions
DA9	GitHub permits automatic push to a repository

Table 3.8: Specific mapping on G6

G7	Synchronize the Submission of Each Candidate with Their GitHub Repository
R2	CKB shall allow users to log in
R19	CKB shall supply API to call with Github actions
R20	CKB shall be able to pull sources from GitHub
DA2	ST owns a GitHub account

Table 3.9: Specific mapping on G7

G8	CKB Provides an evaluation of the code submitted
R20	CKB shall be able to pull sources from GitHub
R21	CKB shall be able to send the ST source code to the correct SAT
R22	CKB shall be able to receive the evaluation given by SAT on a source code
R23	CKB shall be able to run tests on code
R24	CKB shall evaluate the code in terms of test cases passed
R25	CKB shall evaluate the code in terms of timeliness
R26	CKB shall allow ED to assign a score to codes

R27	CKB shall update the score of a T (as soon as new push actions are performed)
R28	CKB shall allow ED to go through sources produced by Ts
R30	CKB shall update the personal competition score of a ST at the end of each battle
R31	CKB shall create a rank with students' performances in a competition
R49	CKB shall have the environments for all the programming language it supports
DA5	ED knows how to use Git
DA6	ED owns a device able to connect to the internet
DA7	ED writes correct tests
DA8	ED correctly evaluates the final source code of a T
DA10	Github permits automatical pull from a repository
DA13	ED has an internet connection
DA14	ST writes code only with languages that are treatable by the platform

Table 3.10: Specific mapping on G8

G9	Allow users to View Rankings in both battles and competitions
R1	CKB shall allow an unregistered user to create an account
R2	CKB shall allow users to log in
R29	CKB shall notify ST when final battle ranks are available
R30	CKB shall update the personal competition score of a ST at the end of each battle

R31	CKB shall create a rank with students' performances in a competition
R32	CKB shall allow ST to see all ST's rank in a battle where is enrolled
R33	CKB shall allow ED to see all ST's ranks in the battle that he/she manages
R34	CKB shall allow EDs and STs to see all ST's rank in competitions
R48	CKB shall not allow ST/ED to see the rankings of battles they are not involved in
DA1	ST owns a device able to connect to the internet
DA6	ED owns a device able to connect to the internet
DA12	ST has an internet connection
DA13	ED has an internet connection

Table 3.11: Specific mapping on G9

G10	Allow to assign badges to the students
R37	CKB shall allow ED to define badges in the context of a competition
R38	CKB shall assign badges to students at the end of the competition
R39	CKB shall allow ED to define new rules for badges
R40	CKB shall allow ED to define new variables for badges
R41	CKB shall allow users to visualize badges obtained by a ST
R47	CKB can distinguish between an ED user and a ST user
DA16	ED writes the correct badge' rules

Table 3.12: Specific mapping on G10

G11	Allow users to visualize ST profiles
R1	CKB shall allow an unregistered user to create an account
R2	CKB shall allow users to log in
R41	CKB shall allow users to visualize badges obtained by a ST
R42	CKB shall allow users to visualize a ST profile
DA1	ST owns a device able to connect to the internet
DA6	ED owns a device able to connect to the internet
DA12	ST has an internet connection
DA13	ED has an internet connection
DA17	User knows the username of the STs' profile they want to visualize

Table 3.13: Specific mapping on G11

UC1: Unregistered ST creates an account

Actor	Unregistered ST
Entry conditions	The ST isn't already registered in the CKB platform and he clicks on the sign-up button
Event Flow	<ol style="list-style-type: none"> 1. CKB asks the ST to insert the personal information (i.e. name, surname, nickname, e-mail and password) 2. The ST fills out the form with the requested information and accepts the "Terms & Conditions" and "Privacy Policy" 3. CKB validate the inserted information 4. CKB sends an account activation link to the email inserted by the ST 5. ST clicks on the link received in the email 6. CKB confirms the account creation
Exit condition	The ST account is created
Exceptions	<ol style="list-style-type: none"> 3.1 CKB isn't able to validate the information inserted (i.e. duplicate email, duplicate username and wrong email address) 5.1 The link is expired <p>In all the cases the ST is notified with an error message</p>

Table 3.14: ST signs up case.

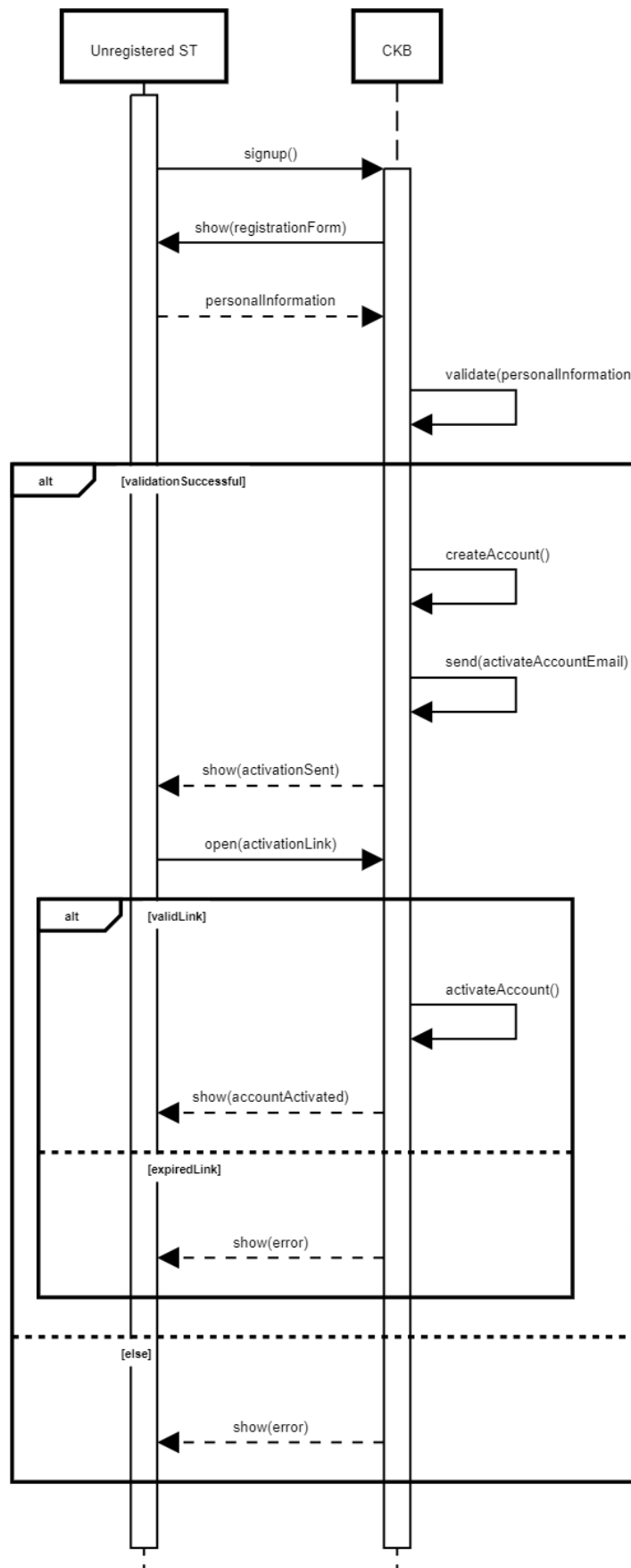


Figure 3.2: UC1 sequence diagram.

UC2: Unregistered ED creates an account

Actor	Unregistered ED
Entry conditions	The ED isn't already registered in the CKB platform and he clicks on the sign-up button
Event Flow	<ol style="list-style-type: none"> 1. CKB asks the ED to insert the personal information (i.e. name, surname, nickname, e-mail and password) 2. The ED fills out the form with the requested information and accepts the "Terms & Conditions" and "Privacy Policy" 3. CKB validate the inserted information 4. CKB asks the ED to insert the information about the institution (i.e. name, address, city, country, website) 5. The ED fills out the form with the requested information 6. CKB validate the information about the institute 7. CKB sends an account activation link to the email inserted by the ED 8. ED clicks on the link received in the email 9. CKB confirms the account creation
Exit condition	The ED account is created
Exceptions	<ol style="list-style-type: none"> 3.1 CKB isn't able to validate the information inserted (i.e. duplicate email, duplicate username and wrong email address) 6.1 CKB isn't able to validate the information about the institute 8.1 The link is expired <p>In all the cases the ST is notified with an error message</p>

Table 3.15: ED signs up case.

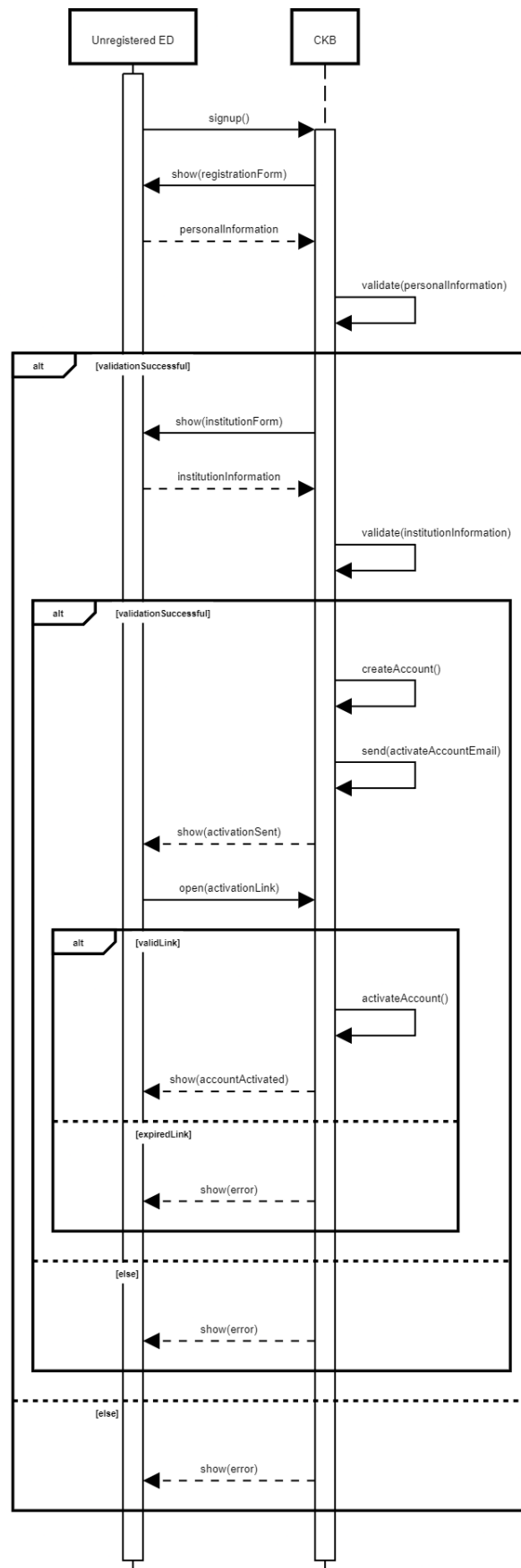


Figure 3.3: UC2 sequence diagram.

UC3: ST or ED logs in

Actor	Registered ST Registered ED
Entry conditions	The ED isn already registered in the CKB platform
Event Flow	<ol style="list-style-type: none"> 1. The user clicks on the login button 2. CKB asks for the username and the password 3. The user inserts the username and the password 4. CKB validates the information 5. CKB redirects the user to the home page
Exit condition	The user is logged in
Exceptions	<p>4.1 The username or the password are wrong</p> <p>In this case the user is notified with an error message</p>

Table 3.16: ST or ED logs in.

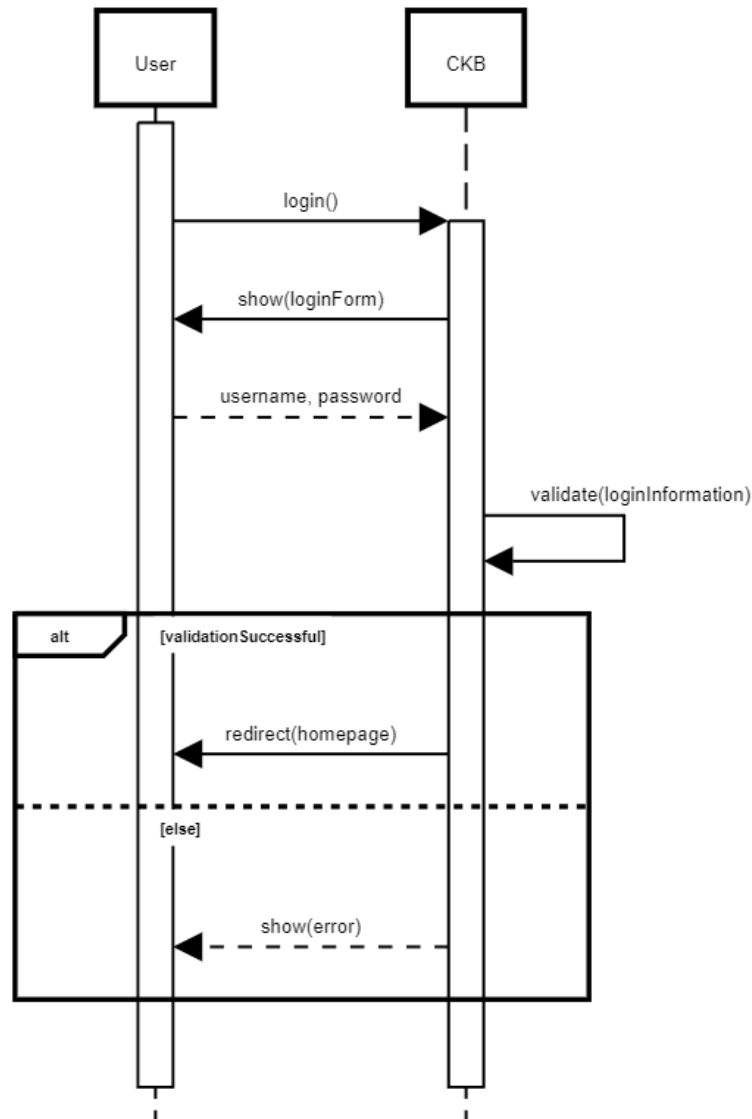


Figure 3.4: UC3 sequence diagram.

UC4: ED creates a new competition

Actor	Registered ED
Entry conditions	The ED is already registered and logged in The ED clicks on the “Create Competition” button
Event Flow	1. CKB asks for a name for the competition 2. The ED inserts the name

	<div>3. CKB checks if the name is available</div> <div>4. CKB asks for the information of the competition (i.e. description, start date, end date, programming languages allowed)</div> <div>5. CKB validates the information</div> <div>6. CKB creates the competition</div>
Exit condition	The competition is correctly created
Exceptions	<div>3.1 The name is already used by another competition</div> <div>5.1 CKB isn't able to validate the information</div> <div>In the first case the ED is notified with an error message and the flow restarts from the step 1</div> <div>In the other case the ED is notified with an error message</div>

Table 3.17: ED creates a competiton.

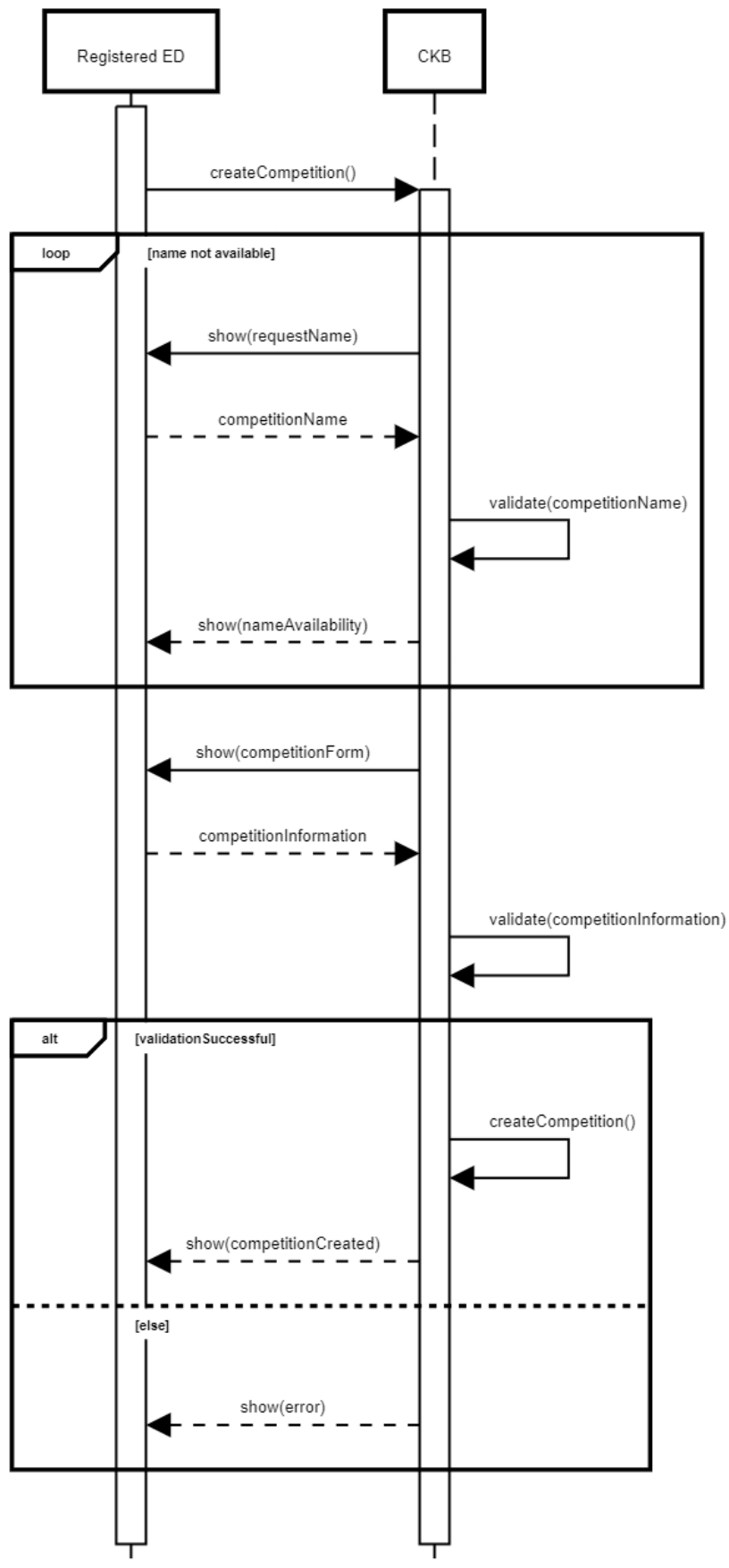


Figure 3.5: UC4 sequence diagram.

UC5: ST joins a competition

Actor	Registered ST
Entry conditions	The ST is already registered and logged in
Event Flow	<div>1. CKB shows the list of the available competitions</div> <div>2. The ST selects the competition</div> <div>3. CKB shows the information about the competition</div> <div>4. The ST clicks on the “Join” button</div> <div>5. CKB adds the ST to the competition</div> <div>6. ST can now see the competition in the “My Competitions” section</div>
Exit condition	The ST has joined the competition
Exceptions	<div>2.1 There are no competitions available</div> <div>In this case the ST visualizes a message that there are no competi- tion available</div>

Table 3.18: ST joins a competition.

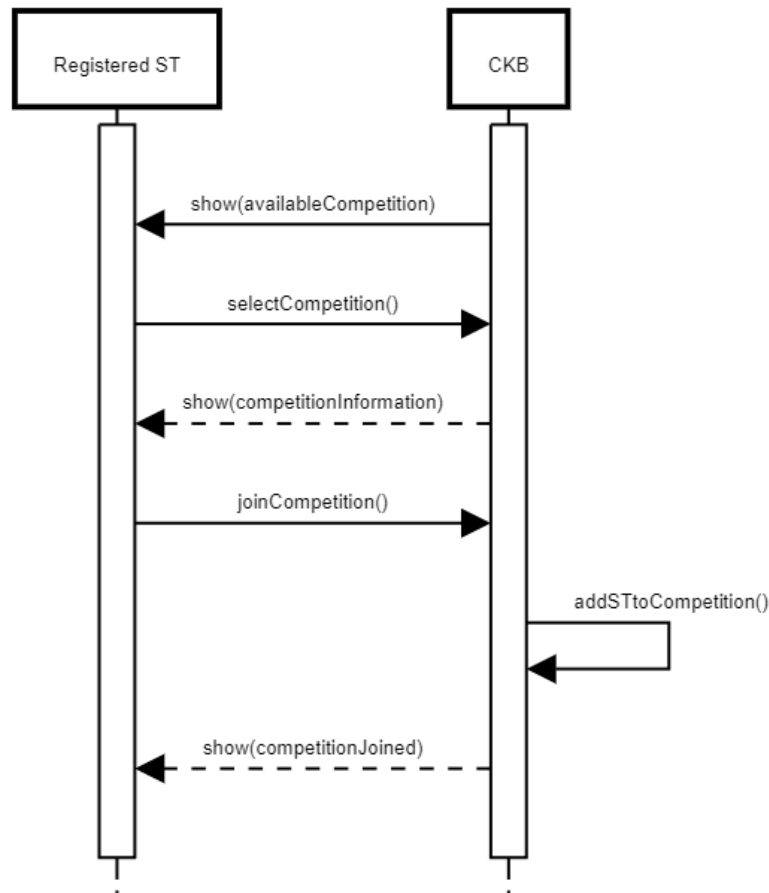


Figure 3.6: UC5 sequence diagram.

UC6: ED creates a new battle inside a competition

Actor	Registered ED
Entry conditions	<p>The ED is already registered and logged in</p> <p>The ED is the creator or a collaborator of the competition</p> <p>The ED is in the competition page</p> <p>The ED clicks on the “Create Battle” button</p>
Event Flow	<ol style="list-style-type: none"> 1. CKB asks for a name for the battle 2. CKB checks if the name is not already used for another battle inside the competition

	<div>3. CKB asks for the information of the battle (i.e. description, start date, end date, programming languages allowed, number of teams, number of members per team)</div> <div>4. ED inserts the information</div> <div>5. CKB asks for the configuration of the static analyzer</div> <div>6. CKB asks to upload the test cases and the solution</div> <div>7. ED uploads the files and insert all the information requested</div> <div>8. CKB validates the information</div> <div>9. CKB creates the battle</div>
Exit condition	The battle is created correctly
Exceptions	<div>6.1 The upload of the files fails</div> <div>7.1 CKB isn't able to validate the information</div> <div>In all the cases the ED is notified with an error message</div>

Table 3.19: ED creates a battle inside a competition.

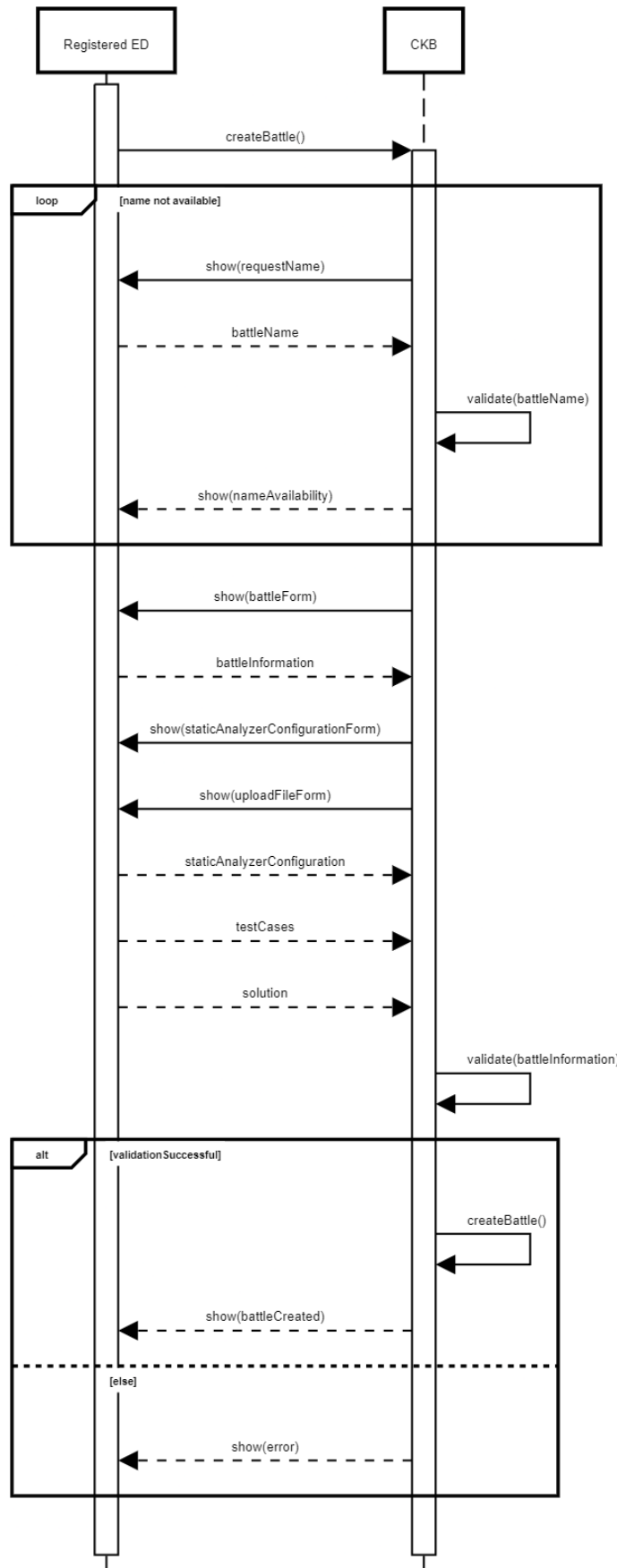


Figure 3.7: UC6 sequence diagram.

UC7: ED invites other EDs to a competition

Actor	Registered ED
Entry conditions	<p>The ED is already registered and logged in</p> <p>The ED is the creator or a collaborator of the competition</p> <p>The ED is in the competition page</p>
Event Flow	<ol style="list-style-type: none"> 1. ED clicks on the "Settings" button inside the competition 2. ED scrolls down to the "Collaborators" section 3. ED clicks on the "Invite" button 4. CKB asks for the email or the username of the ED to invite 5. ED inserts the email or the username 6. CKB validates the information 7. CKB sends an invitation email to the ED 8. The invited ED clicks on the link received in the email 9. CKB confirms the invitation
Exit condition	The invited ED is a collaborator of the competition
Exceptions	<ol style="list-style-type: none"> 6.1 The email or the username is not valid 8.1 The link is expired <p>In all the cases the ED is notified with an error message</p>

Table 3.20: ED invites other EDs to a competition.

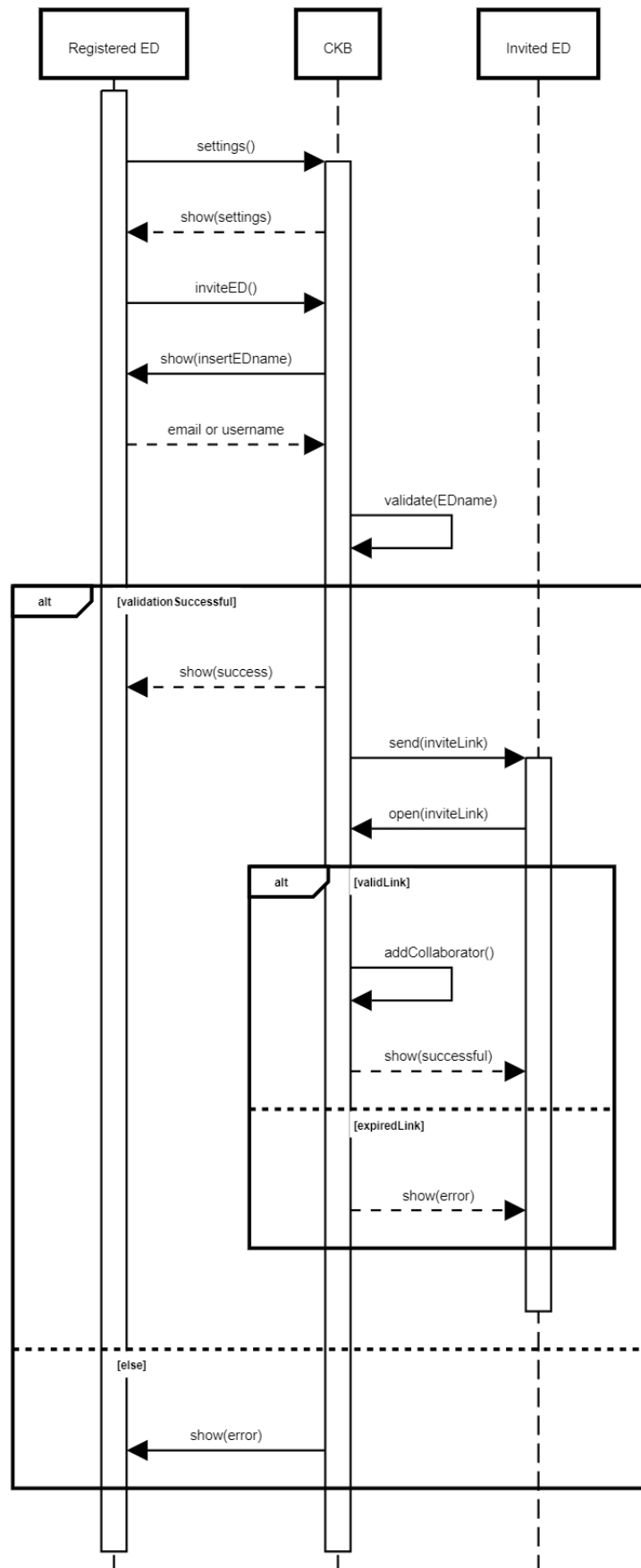


Figure 3.8: UC7 sequence diagram.

UC8: ED creates a new badge inside a competition

Actor	Registered ED
Entry conditions	<p>The ED is already registered and logged in</p> <p>The ED is the creator or a collaborator of the competition</p> <p>The ED is in the competition page</p>
Event Flow	<ol style="list-style-type: none"> 1. ED clicks on the "Settings" button inside the competition 2. ED scrolls down to the "Badges" section 3. ED clicks on the "Create new Badge" button 4. CKB asks for the information (i.e. name, description) of the badge 5. ED inserts the information 6. CKB asks for the rules of the badge 7. ED inserts the rules 8. CKB validates the information 9. CKB creates the badge
Exit condition	The badge is added to the competition
Exceptions	<p>8.1 CKB isn't able to validate the information</p> <p>The ED is notified with an error message</p>

Table 3.21: ED creates a badge inside a competition.

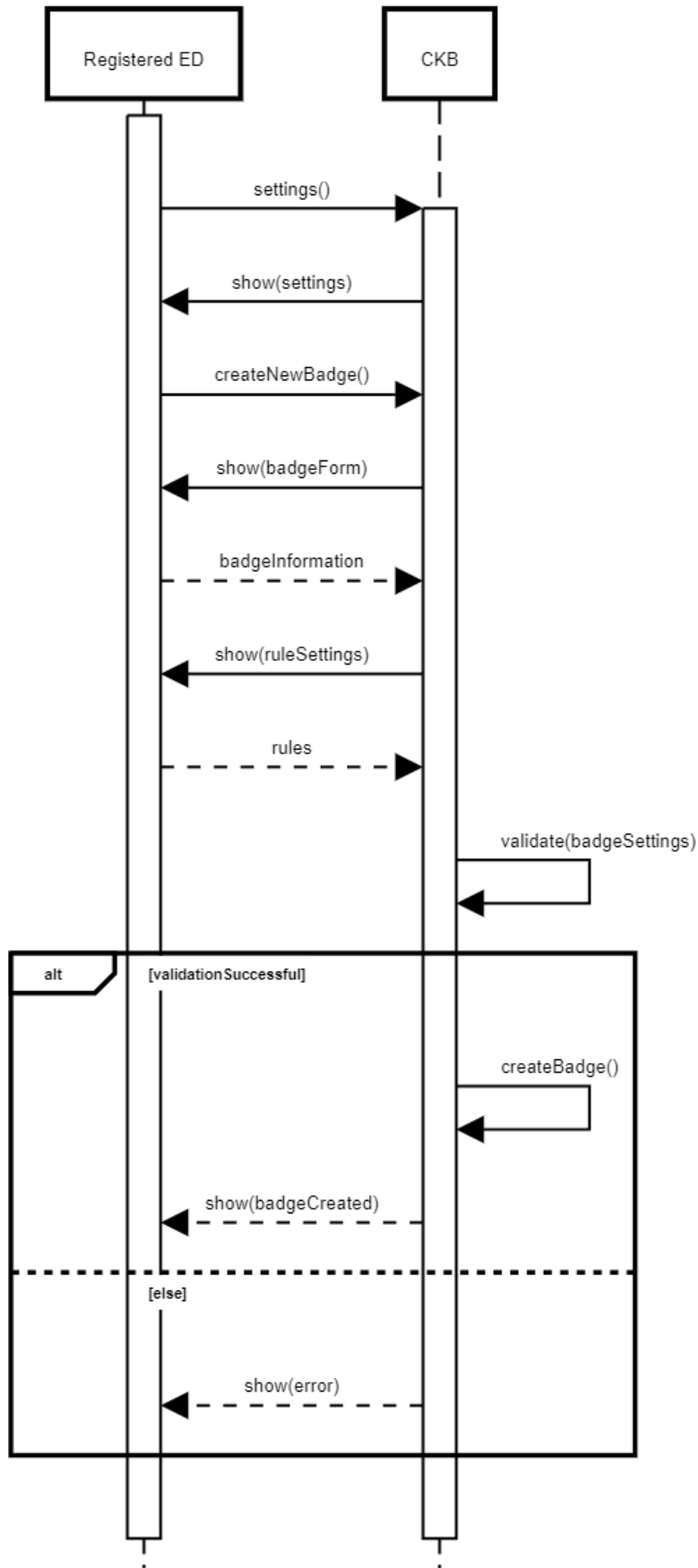


Figure 3.9: UC8 sequence diagram.

UC9: ST joins a battle

Actor	Registered ST
Entry conditions	<p>The ST is already registered</p> <p>The ST is enrolled in a competition</p> <p>The ST is in the competition page</p>
Event Flow	<ol style="list-style-type: none"> 1. CKB show the list of the battles inside of the competition 2. The ST select the battle that wants to join 3. CKB shows the information about the battle 4. The ST clicks on the “Join” button 5. CKB asks to create a new T or to join an existing one <p>Here there are two possibilities:</p> <ol style="list-style-type: none"> 6.1 The ST selects to create a new T <ol style="list-style-type: none"> 6.1.1 CKB asks for the information about the team (i.e. name, number of members) 6.1.2 The ST inserts the information 6.1.3 CKB validate the information 6.1.4 CKB asks for the name of the other STs to invite 6.1.5 The ST inserts the name of the other STs 6.1.6 CKB validate the information 6.1.7 CKB sends the invites to the other STs 6.1.8 CKB creates the T 6.2 The ST selects to join an existing public T <ol style="list-style-type: none"> 6.2.1 CKB shows the list of the public available Ts 6.2.2 The ST selects the T

	6.2.3 CKB adds the ST to the T
Exit condition	ST has joined the battle
Exceptions	6.1.3 CKB isn't able to validate the information 6.1.6 CKB isn't able to validate the information In all the cases the ST is notified with an error message

Table 3.22: ST joins a battle.

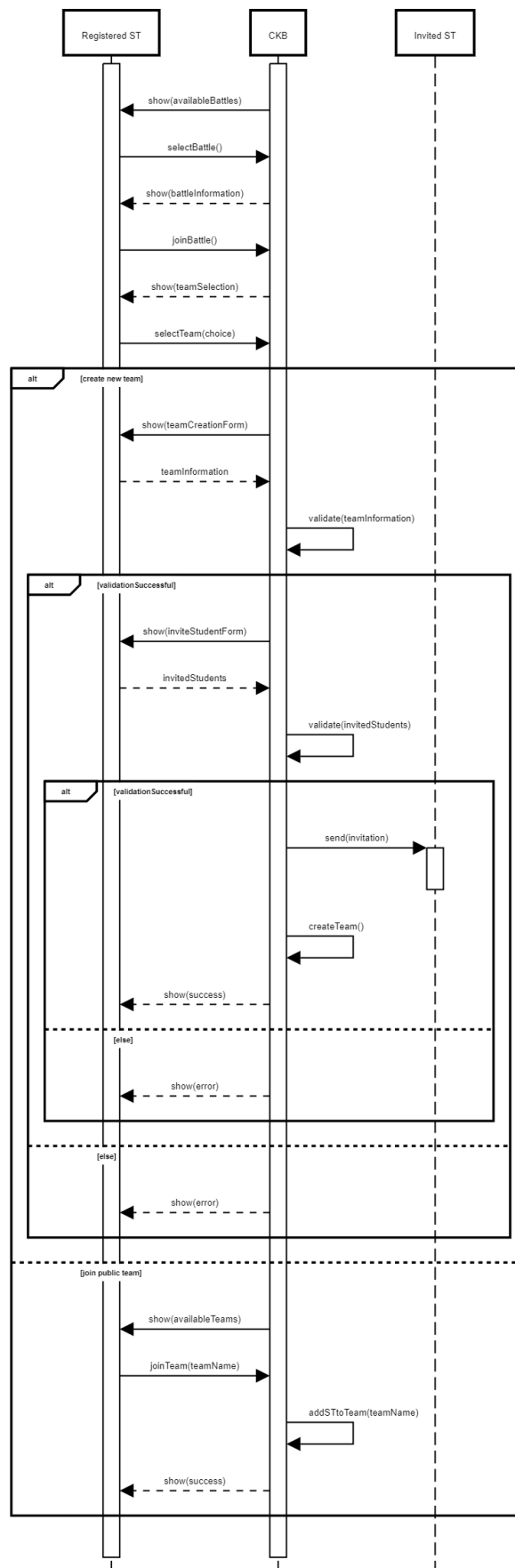


Figure 3.10: UC9 sequence diagram.

UC10: ST visualizes the ranking of their T

Actor	Registered ST
Entry conditions	<p>The ST is already registered and logged in</p> <p>The ST is enrolled in a competition</p> <p>The ST is enrolled in a battle</p> <p>The ST is in the competition page</p>
Event Flow	<ol style="list-style-type: none"> 1. CKB shows the list of the battles in which the ST is enrolled 2. The ST selects the battle 3. The ST goes to the “Ranking” section 4. CKB shows the ranking of the battle
Exit condition	The ST visualize the ranking of their team
Exceptions	<p>2.1 There are no battles available</p> <p>In this case the ST visualizes a message that there are no battles available</p>

Table 3.23: ST visualizes the ranking of their T.

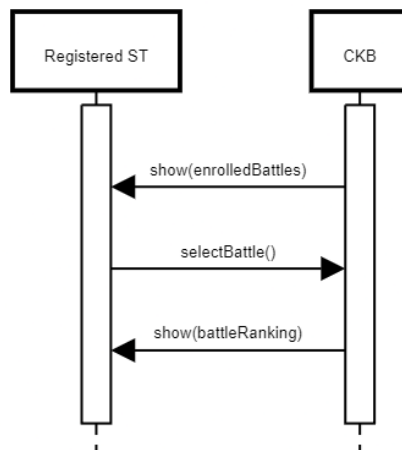


Figure 3.11: UC10 sequence diagram.

UC11: ED manually evaluates the code

Actor	Registered ED
Entry conditions	<div>The ED is already registered</div> <div>The ED is the creator or a collaborator of the competition</div> <div>The ED is in the battle page</div>
Event Flow	<div>1. ED clicks on a T name</div> <div>2. CKB shows the information about the T</div> <div>3. ED clicks on the “Evaluate” button</div> <div>4. CKB shows the information about the last code submission</div> <div>5. ED navigates through the code</div> <div>6. ED assigns a score to the code</div> <div>7. CKB saves the score</div>
Exit condition	The score is saved
Exceptions	<div>4.1 The T didn’t submit any code</div> <div>7.1 CKB isn’t able to save the score</div> <div>In the first case the ED visualize a message that there are no code submissions</div> <div>In the other case the ED is notified with an error message</div>

Table 3.24: ED manually evaluates the code.

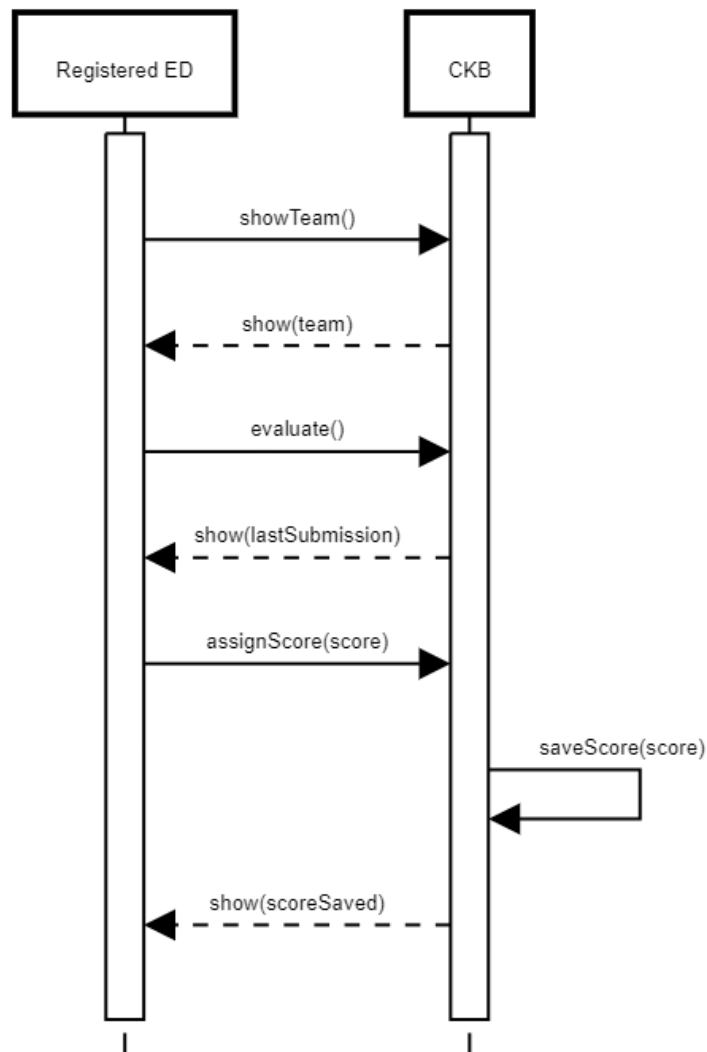


Figure 3.12: UC11 sequence diagram.

UC12: ST accepts an invitation to join a T

Actor	Registered ST
Entry conditions	The ST is already registered
Event Flow	<ol style="list-style-type: none"> 1. CKB sends an invitation email to join a team to the ST 2. The ST clicks on the link received in the email 3. CKB adds the ST to the T
Exit condition	The ST joined the team

Exceptions	<div>2.1 The link is expired</div> <div>In this case the ST is notified with an error message</div>
------------	---

Table 3.25: ST accepts an invitation to join a T.

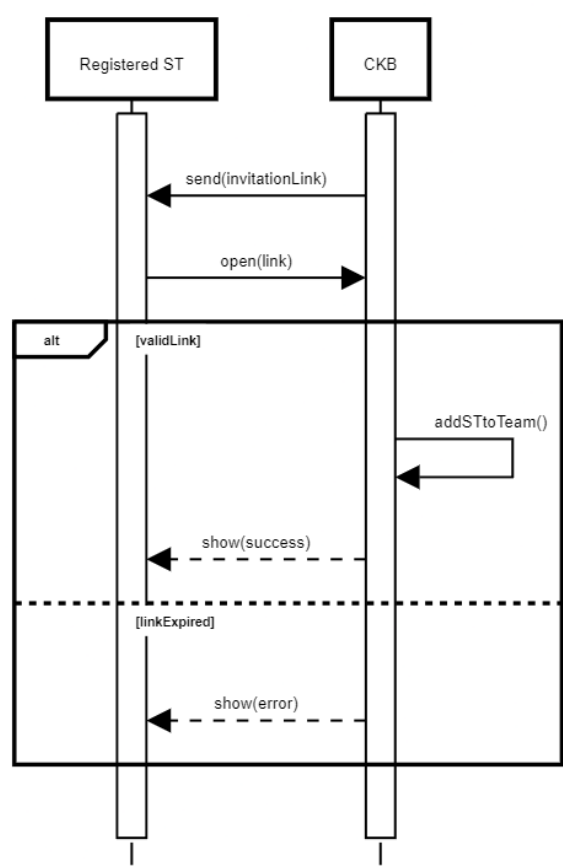


Figure 3.13: UC12 sequence diagram.

3.2. Performance Requirements

Concurrent connections

Since the connections during the phases of the competitions will for sure be very high we need to ensure that the system can handle a big number of concurrent connections. Considering an average number of participant in a competition of 300 and an average number of battles per competition of 5, we can estimate that the number of concurrent connections will be around 2000. To ensure that multiple competition can be run at the same time we need to ensure that the system can handle at least 10000 concurrent connections.

Response time

The system should be able to respond to a request in less than 1 second. In particular the system should be able to respond to a request in less than 0.5 seconds for the most common requests (i.e. login, signup, create competition, create battle, join battle, join team, evaluate code, visualize ranking).

Scalability

The system should be able to scale to support more competitions and battles. In particular the system should be able to support at least 1000 competitions and 10000 battles.

3.3. Design Constraints

3.3.1. Standard compliance

The system should respect the laws about privacy and data protection of the country in which it is used. In particular for the use in Europe it should respect the GDPR. Moreover the system should respect the data treatment for exchange with third party systems (i.e. GitHub).

3.3.2. Hardware limitation

The system should be able to run on any device with a browser and an internet connection. The system should be able to run on any operating system.

3.3.3. Any other constraint

There are no other constraints.

3.4. Software System Attributes

3.4.1. Reliability

Since the system is not critical as the code from the student is uploaded on Github, it is not necessary to have a high reliability. It is reasonable to have a failure rate between 0.1% and 1% to ensure an high quality service. In any case the system should be able to recover the data and state in case of failure.

3.4.2. Availability

To ensure a good user experience and permit the ED to make changes to the competition and battles, the system should guarantee an availability of 99%.

3.4.3. Security

The system guarantees the security of the communication between the users and the server using the HTTPS protocol. The system should guarantee the security of the data stored in the database using encryption and access control.

3.4.4. Maintainability

The system should be easy to maintain and update. The code should be well documented and the architecture should be well designed to allow the addition of new features and the correction of bugs.

3.4.5. Portability

The system does not have restriction on the operating system and the browser. The system should be developed as a web application with a responsive design to allow the use on different devices.

4 | Formal Analysis Using Alloy

In our formal analysis, we deliberately focused on highlighting various snapshots that capture the intricate representation of the system, opting against the utilization of Alloy 6's time features. This strategic decision stems from our dedication to modeling the inherent complexity of the problem in a more comprehensive manner. The incorporation of Alloy 6 features would have presented substantial challenges, given the heightened complexity they introduce. Moreover, it would have necessitated a compromise in the completeness of our representation, as it would require reducing the intricacy to accommodate the depiction of time evolution.

4.1. Alloy Code

```
open util/boolean
-----
-- ENUMS --
-----
-- state of a team
enum TeamState{ WAITING, READY}

enum BattleState{CREATED,STARTED,ENDED}
-- type of possible condition for badges
-- EQUAL: ==, GEQ: >=, LEQ: <= , LT: <, GT: >
enum ConditionType { EQUAL, GEQ, LEQ, LT, GT}

-----
-- SIG --
-----
-- value that can be both Integer, Double or String used for badge conditions
sig GenericValue{}
-- Generalites such as name, surname
sig Generalities{}
sig Name{}
sig Username{}
sig Email{}
sig GitHubLink{}
-- a rule can be also created without a badge assignation?
sig Rule{
    variables: Name,
    conditionType: one ConditionType,
    value: one GenericValue
```

```

}

sig Badge{
    conditions: some Rule,
}

abstract sig User{
    generalities: Generalities,
    username: Username,
    email: Email
}

sig Student extends User{
    competitions: set Competition,
    badges: set Badge,
    team: set Team
}

sig Educator extends User{
    created_badge: set Badge,
    manage_competition: set Competition,
    manage_battle: set Battle,
    manual_evaluations: set ManualEvaluation
}

sig Team{
    teamStudents: some Student,
    team_name: Name,
    joined_battle: one Battle,
    teamState: one TeamState,
    link: lone GitHubLink,
    public: Bool,
    invitedStudents: set Student,
    points: set Point
}

abstract sig TimeEvent{
    startTime: Int,
    endTime: Int
}{
    startTime < endTime
}

sig Competition extends TimeEvent{
    battles: set Battle,
    name: Name,
    managers: some Educator,
    students: set Student,
    badges: set Badge,
}

sig Battle extends TimeEvent{
    name: Name,

```

```

    -- educators that manages the battle
    link: one GitHubLink,
    manager: set Educator,
    participant: set Team,
    evaluations: set Point,
    battleState: one BattleState,
    maxNstudentPerTeam: Int,
    minNstudentPerTeam: Int,
  }{
    --the min number of student is always less than equal than the max
    minNstudentPerTeam > 0 and
    minNstudentPerTeam < (maxNstudentPerTeam + 1)
  }

sig PointValue{}

abstract sig Point{
    value: one PointValue
}

sig ManualEvaluation extends Point{}

sig AutomaticEvaluation extends Point{
    sateValue: one PointValue
}

-----
-- PREDICATES --
-----

pred UserSameMail[u1:User,u2:User]{
    u1.email = u2.email
}

pred UserSameUsername[u1:User,u2:User]{
    u1.username = u2.username
}

pred CompetitionSameName[c1:Competition,c2:Competition]{
    c1.name = c2.name
}

pred TeamSameName[t1:Team,t2:Team]{
    t1.team_name = t2.team_name
}

-----
-- FACTS --
-----
-- usernames, emails and generalities always linked to an user
fact GenNeverAlone{
    no g: Generalities |
        all u: User |
            u.generalities ≠ g

```

```

}

fact usernameNeverAlone{
    no un: Username |
        all u: User |
            u.username ≠ un
}

fact mailNeverAlone{
    no m: Email |
        all u: User |
            u.email ≠ m
}

-- two different users cannot have same mail
fact UserCannotHaveSameMail{
    all disj u1,u2 : User |
        !UserSameMail[u1,u2]
}

-- two different users cannot have the same username
fact UserCannotHaveSameUsername{
    all disj u1,u2 : User |
        !UserSameUsername[u1,u2]
}

-- two different competitions cannot have the same name
fact CompetitionCannotHaveSameName{
    all disj c1,c2 : Competition |
        !CompetitionSameName[c1,c2]
}

-- two different teams cannot have the same name
fact TeamCannotHaveSameName{
    all disj t1,t2 : Team |
        !TeamSameName[t1,t2]
}

-- there is no Competition which is not created by an educator
fact aCompetitionIsAlwaysCreatedByAnEducator{
    all c: Competition |
        one e: Educator |
            c in e.manage_competition
}

-- if a student is in a competition, then the competition contains that student
fact studentInsideCompetitionIffCompetitionContainsit{
    all c: Competition, s: Student |
        c in s.competitions iff s in c.students
}

-- there is no badges which is not created by an educator
fact aBadgeIsAlwaysCreatedByAnEducator{

```

```

    all b: Badge |
        one e: Educator |
            b in e.created_badge
}

-- a battle is part of only a competition
fact battleOnlyInOneCompetition{
    all b: Battle, c1: Competition |
        b in c1.battles implies
            no c2: Competition |
                b in c2.battles and c2 ≠ c1
}

-- a battle is always linked to a competition
fact battleAlwaysInCompetition{
    all b: Battle |
        one c: Competition |
            b in c.battles
}

-- a battle starts and ends always inside a competition
fact battleTimeInsideItsCompetition{
    all c: Competition |
        all bc: c.battles |
            bc.startTime > (c.startTime - 1) and bc.endTime < (c.endTime + 1)
}

-- if a badge is created by an Educator, is unique and is its creator
fact badgeCreatedByOnlyOneEducator{
    all b: Badge, e: Educator |
        b in e.created_badge
            implies
                no e2: Educator |
                    b in e2.created_badge and e ≠ e2
}

-- a badge is always part of one competition
fact badgeAlwaysAssignedToCompetition{
    all b: Badge |
        one c: Competition |
            b in c.badges
}

-- a badge is always linked

-- a badge is assigned to a student only if it participated to the competition where the
    ↪ badge
-- has been assigned
fact studentsEarnsBadgesInsideRightCompetition{
    all b: Badge, s: Student |
        b in s.badges
            implies
                one cs : s.competitions |
                    b in cs.badges
}

```

```

}

-- a badge created by an Educator is always part of a competition MANAGED by the same
  ↪ educator
fact badgeCreatedByEdAreInsideCompetitionManagedByHim{
    all b: Badge, e: Educator |
        b in e.created_badge
            implies
        b in e.manage_competition.badges
}

-- same as above for battles
-- a badge created by an Educator is always part of a competition MANAGED by the same
  ↪ educator
fact battlesCreatedByEdAreInsideCompetitionManagedByHim{
    all b: Battle, e: Educator |
        b in e.manage_battle
            iff
        b in e.manage_competition.battles
}

-- if a team is in a battle it has joined it
fact teamInBattleParticipantIffTeamJoinedBattle{
    all t: Team, b: Battle |
        t in b.participant iff t.joined_battle = b
}

--@toFix could be redundant if above true
-- a team is part on only one Battle
fact teamOnlyInOneBattle{
    all t: Team, b: Battle |
        t in b.participant
            implies
        no b2: Battle |
            b2 ≠ b and t in b2.participant
}

-- a student is part of battles only if is also part of the competition
fact stdInsideBattleInConsistentCompetition{
    all s: Student, c: Competition |
        s.team.joined_battle in c.battles
            iff c in s.competitions
}

-- a team part of a battle respects its number constraints
fact teamCapacityRespectBattleConstraints{
    all t: Team, b: Battle |
        t in b.participant and t.teamState = READY
            implies
        (
            #t.teamStudents > b.minNstudentPerTeam - 1
            and
            #t.teamStudents < b.maxNstudentPerTeam + 1
        )
}

```



```

-- a team which is not ready yet cannot have more students than the maximum allowed by
  ↪ the battle
fact teamCapacityRespectBattleConstraints{
    all t: Team, b: Battle |
        t in b.participant and t.teamState = WAITING
        implies
            (
                #t.teamStudents < b.maxNstudentPerTeam + 1
            )
}

-- battles have all different github links
fact noSameGitHubLinksBattles{
    all disj b1,b2: Battle|
        b1.link ≠ b2.link
}

-- a battle does not share the same github link with a team
fact noSameGitHubLinkBetweenBattleAndTeam{
    all b: Battle, t: Team|
        b.link ≠ t.link
}

-- all github links are linked to a battle or a team
fact allGitHubLinkPartOfSomething{
    no g: GitHubLink |
        all t: Team, b: Battle |
            t.link ≠ g and b.link ≠ g
}

-- teams have all different github links
fact noSameGitHubLinksTeams{
    all disj t1,t2: Team|
        t1.link ≠ t2.link
}

-- a student is part of a team only if the team has the student in it
fact StudentPartaTeamiffTeamHasStudent{
    all t: Team, s: Student |
        t in s.team iff s in t.teamStudents
}

-- a student cannot be part of two teams in the same battle
fact StudentPartOnlyOfATeamInsideABattle{
    all t: Team, s: Student |
        t in s.team
        implies
            #(t.joined_battle.participant & s.team) = 1
}

-- student are part of a team that battle in a competition they are part of
fact StudentPartOfTeamInsideTheSameCompetition{
    all s: Student, t: Team |
        t = s.team implies

```

```

        t in s.competitions.battles.participant
    }

    -- no invited students are part of the team
    fact StudentInvitedNotInsideTeam{
        all s:Student, t:Team |
            t = s.team implies not (s in t.invitedStudents)
    }

    -- a team is in waiting on a battle only if the battle is not started
    fact teamInWaitingOnlyInBattleNotStarted{
        all b:Battle |
            b.battleState ≠ CREATED implies
            (no t:Team |
                t.joined_battle = b and t.teamState = WAITING)
    }

    -- POINTS ---

    -- every team which has been part of a ended competition
    -- has an automatic point and can have a manual
    -- evaluation
    fact teamHasConsistentPoints{
        all b:Battle, t:Team |
            ((t in b.participant and b.battleState = ENDED)
            implies
                (
                    some ae: AutomaticEvaluation | ae in t.points
                    and
                    lone me : ManualEvaluation | me in t.points
                )
            ) or
            ((t in b.participant and b.battleState = STARTED)
            implies
                (
                    -- if commit happened, it could be already a
                    -- an Automatic one
                    no me : ManualEvaluation | me in t.points
                )
            )
        or
            ((t in b.participant and b.battleState = CREATED)
            implies
                (
                    no ae: AutomaticEvaluation | ae in t.points
                    and
                    no me : ManualEvaluation | me in t.points
                )
            )
    }
}

```

```

--no one shares the same evaluation
fact noSameEvaluation{
    all disj t1,t2 : Team |
        #(t1.points & t2.points) = 0
}

-- educators can evaluate only inside a battle they manage
fact educatorsCanEvaluateOnlyInsideABattleTheyManage{
    all e: Educator, b: Battle |
        b.evaluations in e.manual_evaluations implies
            b in e.manage_battle
}

-- only educators that manage a battle can assign manual evaluation to a student
fact teamManualEvaluationsAreGivenByConsistentsEducators{
    all e: Educator, t: Team |
        all i : t.points & e.manual_evaluations |
            i in t.joined_battle.evaluations
}

-- all points are assigned inside a battle
fact allPointsAssigendInBattle{
    all p: Point |
        one b: Battle |
            p in b.evaluations
}

-- a pointvalue is always assigned to points
fact allPointValueAssignedtoPoints{
    all pv: PointValue |
        some p: Point |
            p.value = pv
}

-- all points are assigned to a team and are assigend inside the battle
-- they are part of
fact allPointsAssigendToTeam{
    all p: Point |
        one t: Team |
            p in t.points and p in t.joined_battle.evaluations
}

--there is no manual evaluation not assigned by an educator
fact manualEvaluationIsAlwaysMadeByanEducator{
    all me: ManualEvaluation |
        one e: Educator |
            me in e.manual_evaluations and
            me in e.manage_battle.evaluations
}

-- No student is part of a team and an invited student simultaneously
fact noStudentBothInvitedAndJoinedInTheSameTeam{

```

```

    all t: Team |
        all ts: t.teamStudents, ti: t.invitedStudents |
            #(ts & ti) = 0
}

-- Battle can't start if there are not enough teams / at least one team
fact noBattleStartsWithoutTeams{
    all b: Battle |
        b.battleState = STARTED
        implies
            #b.participant > 0
}

-----
-- ASSERTIONS --
-----

-- there is no student in a battle inside competitions which
-- are not joined by the student
assert noStudentInABattleInCompetitionNotJoined{
    all s: Student |
        no c: Competition |
            #(s.team.joined_battle) > 0 and
            s.team.joined_battle in c.battles
            and s not in c.students
}

-- no battle started with a team not ready inside
assert noStartedBattleWithWaitingTeams{
    all b: Battle, t: Team |
        (t in b.participant and b.battleState = STARTED)
        implies
            t.teamState = READY
}

-- there is no student inside two teams in the same
-- battle
assert noStudentInsideABattleWith2Teams{
    all s: Student, t1: Team, t2: Team |
        t1 in s.team and
        t2 in s.team and
        t2 ≠ t1
        implies
            t2.joined_battle ≠ t1.joined_battle
}

assert allFinishedBattleGavePointsToTeams{
    all b: Battle |
        b.battleState = ENDED
        implies
            no t: b.participant |
                #(b.evaluations & t.points) = 0
}

```

```

}

-- no badge assigned in student not joined a competition
assert noBadgeAssignedToStudentOutsideTheCompetition{
    all s: Student |
        s.badges in s.competitions.badges
}

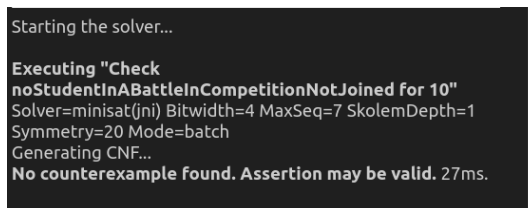
--check noStudentInABattleInCompetitionNotJoined for 5
--check noStartedBattleWithWaitingTeams for 5
--check noStudentInsideABattleWith2Teams for 5
--check allFinishedBattleGavePointsToTeams for 5
--check noBadgeAssignedToStudentOutsideTheCompetition for 5
-----
--      RUN      --
-----

pred show{
    #Competition = 1
    #Battle = 2
    #Team = 4
    #Student > 3
    #Badge > 0
    #Educator = 2
    #Student.team > 2
    --some t: Team / t.teamState = WAITING
}

run show for 10

```

4.2. Assertion Results



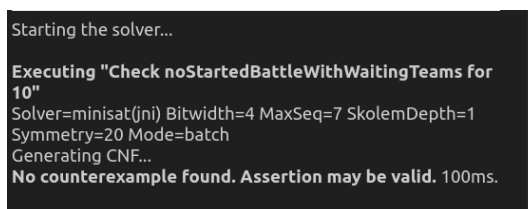
```

Starting the solver...

Executing "Check
noStudentInABattleInCompetitionNotJoined for 10"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1
Symmetry=20 Mode=batch
Generating CNF...
No counterexample found. Assertion may be valid. 27ms.

```

Figure 4.1: Assert there is no student part of a battle for a competition it did not join



```

Starting the solver...

Executing "Check noStartedBattleWithWaitingTeams for
10"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1
Symmetry=20 Mode=batch
Generating CNF...
No counterexample found. Assertion may be valid. 100ms.

```

Figure 4.2: Assert there is no started battle with waiting teams inside

```
Starting the solver...  
  
Executing "Check noStudentInsideABattleWith2Teams  
for 10"  
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1  
Symmetry=20 Mode=batch  
Generating CNF...  
No counterexample found. Assertion may be valid. 21ms.
```

Figure 4.3: Assert there is no student inside a battle with 2 teams

```
Starting the solver...  
  
Executing "Check allFinishedBattleGavePointsToTeams  
for 10"  
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1  
Symmetry=20 Mode=batch  
Generating CNF...  
No counterexample found. Assertion may be valid. 54ms.
```

Figure 4.4: Assert all finish battle gave points to the teams

4.3. World Generated

5 | Effort Spent

Members of group	Effort spent (hours)	
Filippo Balzarini	Introduction	2 <i>h</i>
	Overall description	1 <i>h</i>
	Specific requirements	5 <i>h</i>
	Formal analysis	13 <i>h</i>
	Reasoning	7 <i>h</i>
Christian Biffi	Introduction	3 <i>h</i>
	Overall description	3 <i>h</i>
	Specific requirements	8 <i>h</i>
	Formal analysis	1 <i>h</i>
	Reasoning	8 <i>h</i>
Michele Cavicchioli	Introduction	1 <i>h</i>
	Overall description	4 <i>h</i>
	Specific requirements	6 <i>h</i>
	Formal analysis	0 <i>h</i>
	Reasoning	6 <i>h</i>

Table 5.1: Effort spent by each member of the group

6 | References

6.1. Tool Used

- **TeXstudio** to compile and format this document.
- **Alloy** to verify the consistency of our model.
- **draw.io** to draw the diagrams.
- **sequencediagram.org** to draw the sequence diagrams.
- **GitHub** to share and collaborate on the project.
- **Visual Studio Code** to write this document and the alloy code.
- **Trello** to organize the work.

