Requirements Analysis & Specification Document
CodeKataBattles

Federica Maria Laudizi, Antonio Marusic, Sara Massarelli

A.A. 2023/2024

POLITECNICO
MILANO 1863

Contents

List of Tables

List of Figures

Abstract

This Requirements Analysis and Specification Document (RASD) is going to tackle objectives, requirements (functional and non functional) for the project CodeKataBattles.

It will be presented the scope, the functionalities and the domain of such project and we will give a comprehensive overview of the interaction with users and external components and the performance expected from the system.

This document is going to be a reliable point of reference for developers, since it will define clearly the functionalities of the system and it will give precise guidelines during the validation and verification process.

The profound goal of this document is to give a precise overview of the product to be, in order to give the stakeholders a general understanding of usage scenarios and avoiding changes of directions during the design and implementation part of the system development.

1 Introduction

1.1 Purpose

This system allows users to boost their learning by participating in code challenges called kata battles organised by teachers or independent tutors.

Challenges will be part of tournaments, so that it is possible for students to compete in multiple katas and seeing their position on a leaderboard with all participants. The system allows the creator of the tournament to define a set of rules regarding the size of the groups of students and give the possibility to delegate the creation of the katas and the managing of the tournament also to other teachers.

The system offers a platform integrated with GitHub, where teachers can publish the kata and define the test cases that the code provided by the students must pass and assess student's submissions. The students on the other hand, can submit their code through GitHub and see immediately its performances. The system assigns a score to each submission based on functional and non functional aspects of the code so that students can compare their work with other participants.

Moreover, the system promotes global participation in kata battles, since it is possible to join battles organised by educators from all around the world. It is worth noticing that, not only certified teachers can create tournaments, but also independent tutors, allowing them to engage their students in productive competitions.

1.1.1 Actors definitions

The actors interacting with the system were identified as follows:

- Guest: someone that accesses the website but is not yet registered to access its functionalities.
- User: a registered person in the system.
- Educator: an user with an educator profile.
- Student: an user with a student profile.
- Admin: educator that created a tournament or was invited by the creator as collaborator. Admin users have the privilege for that tournament to manage its settings and parameters and to generate new kata battles.
- Participant: student within a group.
- Group: team of students that participants in a tournament. Groups engage in kata battles.

For further details on user characteristic refer to *Section* 3.2.

1.1.2 Definitions

- Kata battle: a coding exercise where a task is given to the participants and they have to submit their solution through GitHub.
- Tournament: a competition made of a series of Kata battles , each contributing to a final score. The score of these battles are then used to establish a leaderboard that ranks the participants.

- Badge: A badge is a collectible award earned when specific predefined conditions are met. The admins define a boolean formula and the students who complete the challenge or meet the particular criterion get assigned the badge. We decided to use a simple logic language where the scope of the search is within a tournament and admins can use the provided variables to write logic statements. Each condition is evaluated for each participant of the tournament and the badge is given in case the condition is true.

  Example 1 In this example we award the badge to the student that performed the highest amount of commits. MAX is an aggregate operation that search the highest value amongst the commits made by the students of the tournament.

```
1                    RULE TopCommitterBadge
2                    CommitsMade = MAX(CommitsMade)
```

  Example 2

```
1                    RULE FirstCommit
2                    CommitsMade > 0
```

1.1.3 Goals

The system must satisfy the following goals:

G1: Student can join a tournament and thus, become a participant of such tournament.

G2: Participant can create a group when entering a kata battle

G3: Participant can join an existing group when entering a kata battle.

G4: Groups can submit their code through GitHub.

G5: User can view the current rank evolving through each battle and tournament.

G6: Admin can invite collaborators to the tournament that will be considered as admins once they accept the invite.

G7: The admin can create battles and badges within their specific tournaments

G8: Admin can adjust the automatic evaluation of a battle performed by the platform during the consolidation phase of the battle.

1.2 Scope

CodeKataBattle (CKB) is an innovative platform specifically designed for students to elevate their software development skills through collaborative training sessions focused on code katas. In the realm of education technology, CKB serves as a dynamic arena that empowers both students and educators in the field of programming. This platform is tailor-made for educators who seek to create engaging challenges for their students. These challenges manifest as code kata battles, in which teams of students engage in programming exercises using a programming language of choice (e.g. Java, Python). Each exercise comprises a succinct textual description and a well-structured software project with build automation scripts, like a Gradle project for Java sources. These projects are equipped with a comprehensive suite of test cases, requiring students to ensure their code meets the specified criteria and successfully passes them. Following the completion of each battle, student groups submit their finalized projects to the platform. The platform (or and educator) then meticulously assesses and assigns scores to each group, creating a competitive rank that reflects their performance and progress in mastering software development skills.

1.2.1 World phenomena

WP1: User has a GitHub account.

WP2: User has a personal computer with internet access.

WP3: User has an email account.

1.2.2 Shared phenomena controlled by the world and observed by the machine (SPWC)

SPWC1: Person registers in the system as student or educator.

SPWC2: User logs in the system.

SPWC3: Educator requests the system to create a new tournament.

SPWC4: Educator requests the system to create a new kata battle inside a tournament.

SPWC5: Educator requests the system to set configuration parameters of a tournament he/she is managing.

SPWC6: Educator requests the system to invite other educators to become admins in the tournament he/she is an admin in.

SPWC7: Educator modifies the score of a group in the consolidation phase of the kata battle.

SPWC8: Educator accepts or rejects an invitation to an admin group.

SPWC9: Student requests the system to join a tournament.

SPWC10: Student requests the system to join a battle.

SPWC11: Student invites other students to join his group for a battle.

SPWC12: Student pushes code on the group branch on the GitHub repository.

SPWC13: Student accepts or rejects an invitation to a group.

1.2.3 Shared phenomena Controlled by the machine and observed by the World (SPMC)

SPMC1: The system updates the group score.

SPMC2: At the expiry time of the registration phase of a kata battle, the system removes all the groups that don't match the requirements on the number of participants.

SPMC3: The system notifies every participant of the battle and sends a link to the GitHub repository. (This happens the expiry time of the registration phase of a kata battle, after the removal of incomplete groups).

SPMC4: When a tournament is created the system notifies every student on the platform of this event.

SPMC5: When a tournament closes the system notifies the participants of that tournament of this event.

SPMC6: The system notifies a student of the invitation in a group.

SPMC7: The system notifies an educator of the invitation to the admin group of a tournament.

SPMC8: The system updates battle's and tournament's rank.

SPMC9: The system assigns the badge to the student who won it.

1.2.4 Abbreviations

1. CKB: Code Kata Battle
2. Admin: Administrator

1.2.5 Acronyms

- UC: Use case
- G-i: i-th Goal
- DA-i: i-th Domain Assumption
- R-i: i-th requirement
- UDC: Use case diagram
- SPMC-i: i-th Shared Phenomena machine controlled
- SPWC-t: i-th Shared Phenomena world controlled
- WP-i: i-th World Phenomena
- SD: Sequence Diagram
- AD: Activity diagram

1.3 Document structure

- **Chapter 1**: Introduces and gives the purpose of the Code Kata Battles software including its goals. Here we include definitions, acronyms abbreviations, and revision history as well.
- **Chapter 2**: Here, an overall description of the software is given. The product perspective is illustrated and the software's functionalities are explained more in detail. It also contains class diagrams and state charts used to better describe the software.
- **Chapter 3**: Interface requirements such as user, software and hardware interfaces. Here we also include functional and non functional requirements. Functional requirements are associated with use cases and sequence diagrams. Non functional requirements such as performance and system attributes are also included.
- **Chapter 4**: Here we include the alloy code, and metamodels generated from it.
- **Chapter 5**: Effort spent from all the contributors.
- **Chapter 6**: Reference documents.

2 Overall characteristics

2.1 Product perspective

2.1.1 Scenarios

Creation of a tournament: Alice is an educator. She wants to create a new tournament for her students, so she opens the CodeKataBattle platform, logs in and select "Create a new tournament". At this point the system allows Alice to choose the name of the tournament, set the starting and the ending date and insert none or more badges. Alice adds as admin her assistant, Bob, to help her creating battles. After creating the tournament, the systems sends to all the students subscribed to the platform a notification. Alice can now proceed to create the first battle.

Creation of a battle: Carol, an educator on CodeKataBattle, is organizing a programming tournament for her students. After successfully creating the tournament, Carol decides to set up the first battle within the tournament. Logs into the CodeKataBattle platform. Navigates to the tournament management section. Selects the created tournament. Chooses the option to create a new battle within the tournament. Defines the battle parameters, including a title, description, programming language. Uploads the code kata, including a brief textual description and a software project with test cases. Sets the minimum and maximum number of students per group for the battle. Specifies a registration deadline and a final submission deadline. Configures additional scoring parameters using static analysis tools for evaluating aspects like security, reliability, and maintainability. Saves and completes the battle creation process.

Students join a tournament and create a group: Danielle is a computer science student. She receives the notification of a new tournament is online, so she opens the CodeKataBattle's platform. The first battle requires exactly three students, so she creates a new group and sends an invitation to the group to her friend Emanuel. Emanuel accepts the invite, so now he is a member of the team. Since they need a third member, Danielle decided to create a public group. This way, if someone is looking for a group, he/she can join them. When the registration deadline expires they still have not find a group so the system does not allow them to participate.

Evaluation and final ranking: Francis is participating to a battle by himself, as a group on one. He works hard and commit his code within the deadline. Right after Francis pushes his code on the fork of the repository created by the system. The system pulls the latest sources, analyzes them, and runs the tests on the corresponding executable to calculate and update the battle score. After reaching the deadline, the system ignores any further code submissions and shows the temporary leaderboard. Subsequently, there is the optional manual evaluation. When this is finished too, Francis get a notification that the final rank has been published and he won it.

Creation and assignment of a badge: Ian, an educator on CodeKataBattle, is organizing a programming tournament for his students. In an effort to recognize collaboration and productivity, he introduces a new badge called 'Top Collaborator.' This badge will be awarded to the student who contributes the highest number of commits, demonstrating exceptional commitment to the project and surpassing other participants. At the end of the tournament, the platform will utilize the GitHub API to assess commit activity and assign the 'Top Collaborator' badge to the most prolific student based on their commit contributions.

Closing of a tournament: After a series of intense battles, the "Programming Challenge 2023" tournament organized by Jacob comes to an end. The system, following the end date set by Jacob, automatically stops the acceptance of new battles and performs the evaluation of the badges. Students and educators receive notifications about the availability of the final tournament ranking and can view the results on CodeKataBattle.

Closing of a battle: After completing the development and testing phase, the "Java Coding Challenge" battle comes to an end. The system, after receiving the final commit before the deadline from the participants, ignores further submissions. The optional manual evaluation phase opens, where educators can review and assign additional scores to the resources. At the end of this phase, the rankling is updated and all participants are notified of the availability of the final battle ranking.

Submission of the code: Kevin and his team are participating in "Wordchecker", a competition where participants are tasked with developing a game similar to *Wordle*. After completing the coding and successfully passing all public tests, they decide to submit their code. Once uploaded to GitHub, the platform initiates private tests and subsequently provides the results. These results include both the execution time and the correctness of the output. Armed with the test outcomes, the platform can then proceed to update the rankings.

2.1.2 Domain class diagram

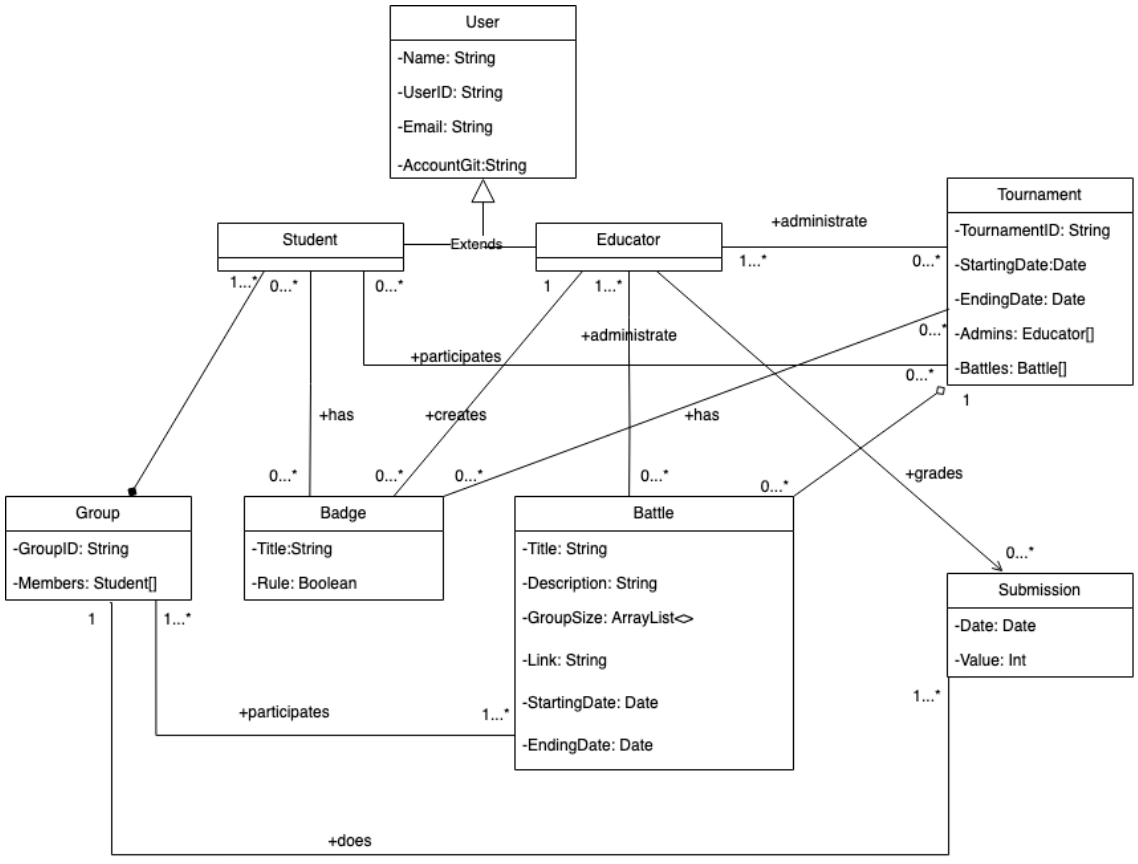


Figure 1: Domain class diagram

2.1.3 Activity diagrams

The charts below show the most significant flows of the platform.

Creation of a tournament:

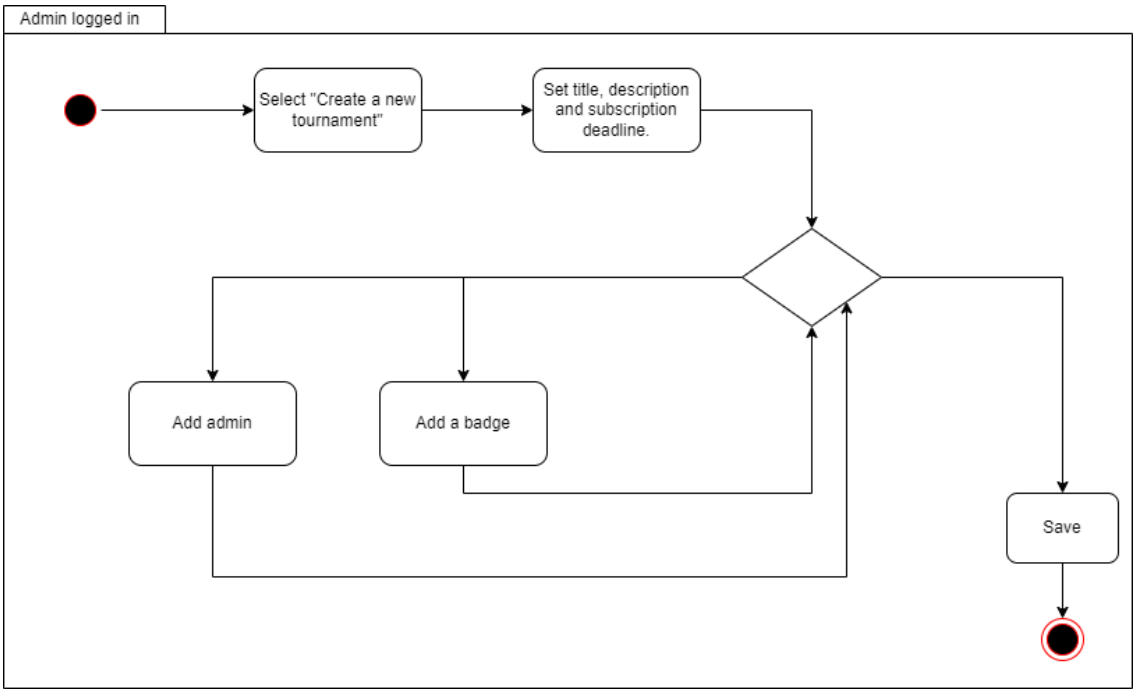


Figure 2: Creation of a tournament AD

Creation of a battle:

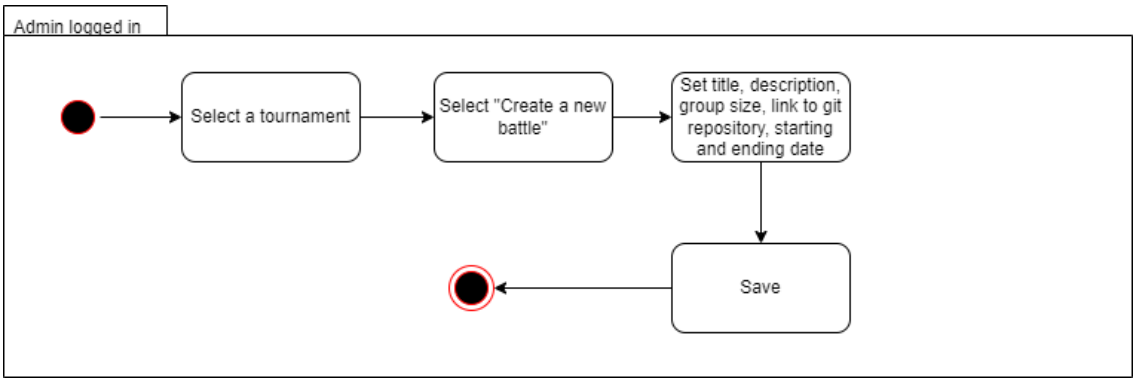


Figure 3: Creation of a battle AD

Creation of a group:

If a user wants to create a new group, after selecting the kata he/she can invite his/her friends using their ID. Note that the system will check whether the maximum number of participant has been reached or not.

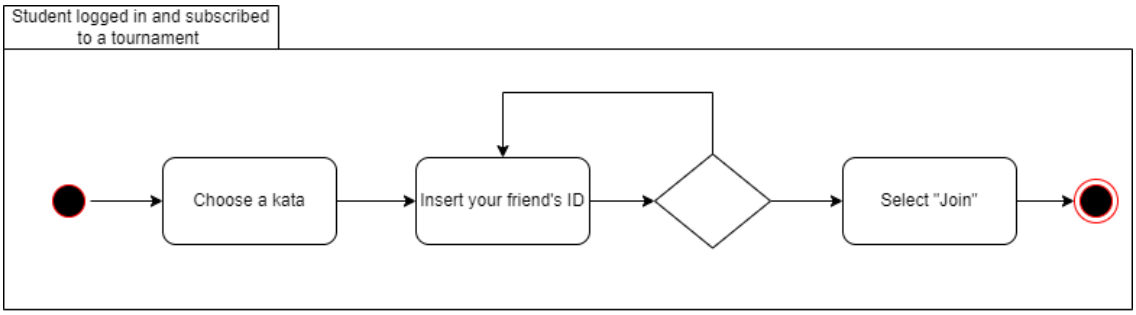


Figure 4: Creation of a group AD

Join a group:

There are two ways to join a group. The user can simply accept a received invitation or select a kata and a public group to join. At the end of this procedure the system saves the changes to the group.

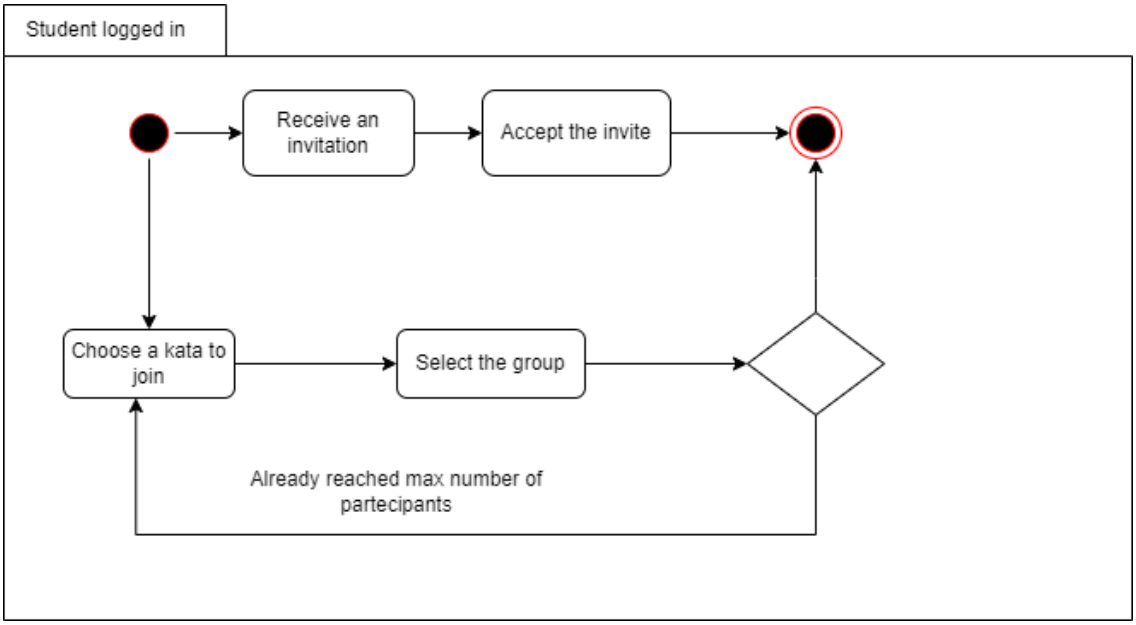


Figure 5: Joining a group AD

3 State diagram

Here we show how the state of the battle evolves during its lifetime. New battles are created in registration state. During this state new groups can join the battle and students can join the groups. Subsequently, when the registration deadline expires, the battle becomes in running mode, groups that don't suit the limitations on the number of participants are removed from the battle and the remaining group can start forking and pushing the battle's repository. When the battle ends it goes in evaluation mode, here educators can modify the mark automatically assigned by CKB. Finished this phase the battle becomes closed and remains so.

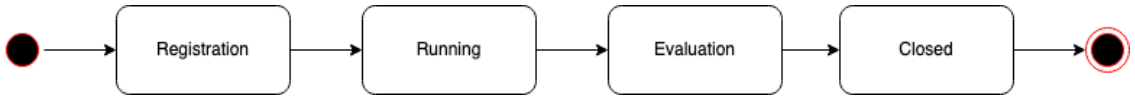


Figure 6: State diagram of the battle

3.1 Product functions

3.1.1 Sign-up and login

These functions will be accessible to all users. The sign-up feature enables users to register in the system. Specifically, each user will need to supply an email, a password and the GitHub account.

3.1.2 Create a tournament or a battle

Educators can create a tournament and battles within them. Educators have the privilege to invite collaborators and create badges for the tournament.

3.1.3 Join a battle

After a battle has been created, students utilize the platform to assemble teams for that specific battle. Each student has the option to participate individually in a battle, join an existing group or invite other students to join, adhering to the predetermined minimum and maximum number of students per group established for that battle. When the registration deadline expires, the platform will automatically generate a GitHub repository containing the code kata and subsequently send the repository link to all students who are members of the registered teams. At this stage, students are free to commence their work on the project.

3.1.4 Run tests

Each git push, performed on the challenge repository by a groups participating in the kata battle, triggers the platform that builds the project and runs the tests threw GitHub API. Once the tests are finished the platform calculates the score and updates the battle's and the tournament's ranking.

3.1.5 Assign score

After the submission has undergone testing, the system assigns a score to it. When the submission deadline expires, the kata battle enters in a consolidation stage where the admins can review the repositories and perform further evaluations manually. When this stage finishes all participants in the battle are notified when the final battle rank becomes available.

3.1.6 Create Badge

The system allows admins of a tournament to create badges. In order to create a new badge the admin has to write a formula that involves variables provided by the system. At the end of the tournaments, the system checks to see if students have met their goals, and if so, gives them the badges. Badges can be visualized by all users.

3.1.7 Public and private groups

The system allows for the creation of private and public groups. Private groups can be joined only by invitation, on the other hand, public groups can be joined both by invitation and searching for them when joining a battle.

3.1.8 Invitations

Admins of a tournament can invite collaborators to manage their tournament. Students that are part of a group can invite other students to join their group. They cannot invite students that are in other groups in the same battle and if they invite students that are not participant in the tournament the system registers the invite but when the invited student tries to accept it the system warns them that they have to participate the tournament first and doesn't put him in the group.

3.2 User characteristics

Each person, when registering in CodeKataBattle, can create a student or an educator profile. In particular:

- Educators use the platform to challenge participants by creating tournaments and code kata battles in which groups of participants can compete against each other, thus proving (and improving) their skills.
- Students are the ones who join groups to compete in the battles and use the system to improve their software development skills.

3.3 Assumptions, dependencies and constraints

The following are the assumptions made for the domain. Such assumptions are properties and/or conditions that the system takes for granted, mostly because they are out of the control of the system itself, and hence need to be verified to assure the correct behavior of CodeKataBattles.

DA1: User must have an internet connection

DA2: User must have a GitHub account

DA3: Each user must have one and only one account

DA4: Each user must have a device to use the system

DA5: Groups remain the same for the duration of the whole battle

DA6: A group in a battle is allowed to consist of just one person

DA7: During a battle student pushes correctly his code in the fork of the group inside the repository whose link is provided by the system.

DA8: The forked repository of each group can be modified only by the members of that group.

4 Specific requirements

Here we include more details on all aspects in Section 2 that can be useful for the development team.

4.1 External Interface Requirements

4.1.1 User interfaces

The following mockups are meant to communicate the look and feel of the app.

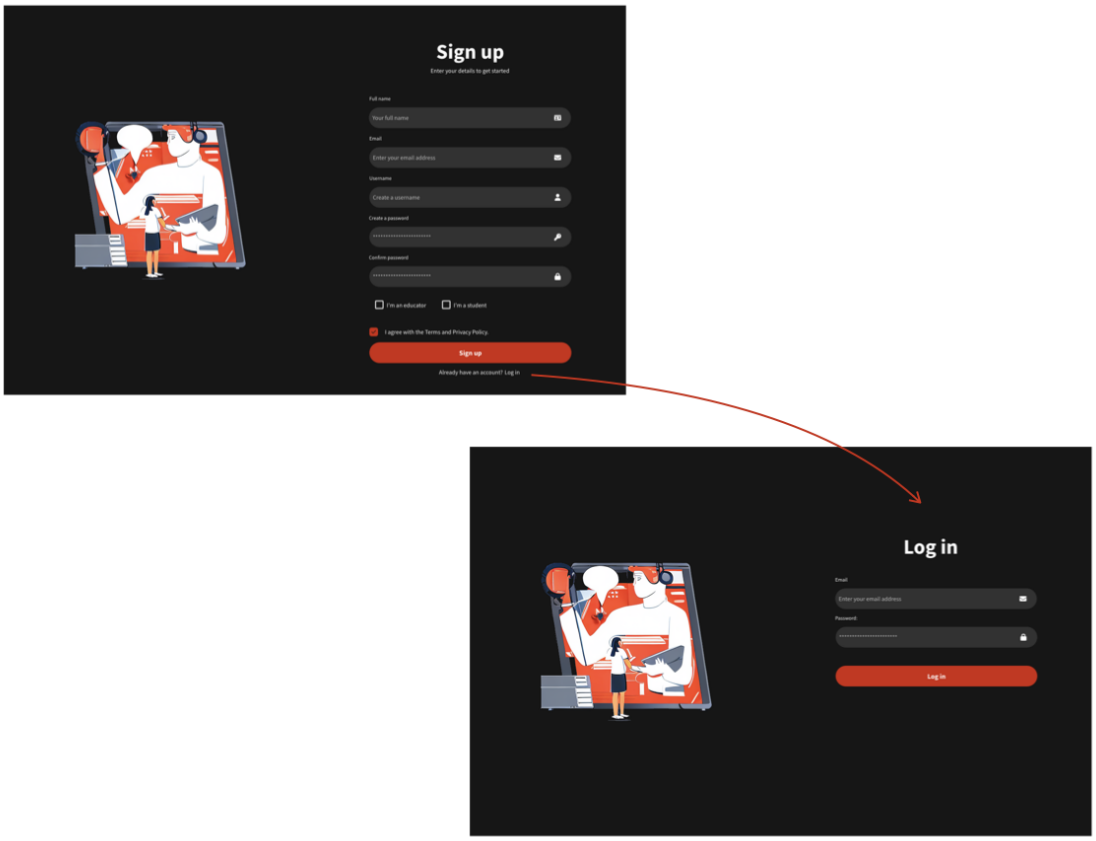


Figure 7: Signing up or logging in the system MOCKUP

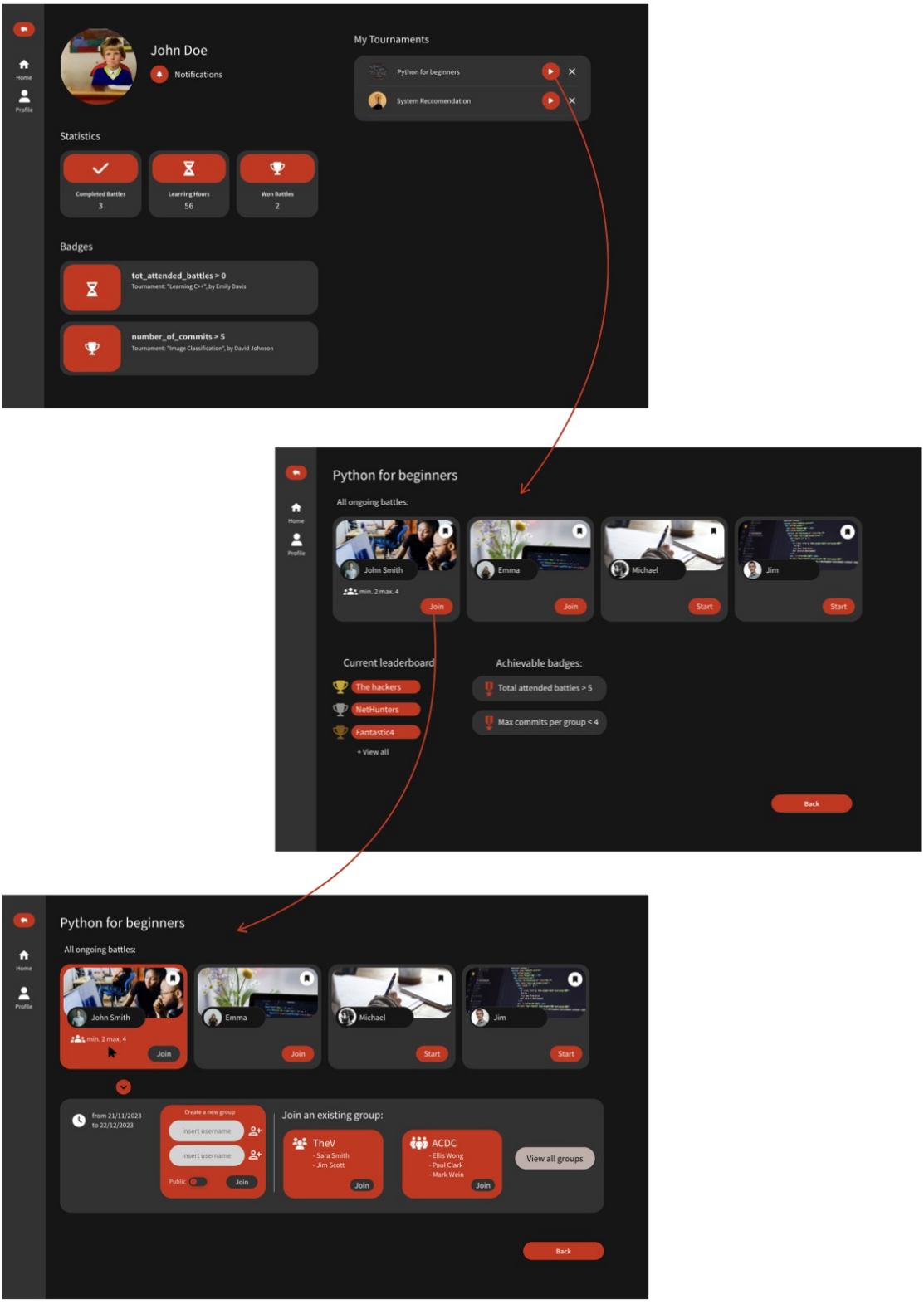


Figure 8: Student joining a battle MOCKUP

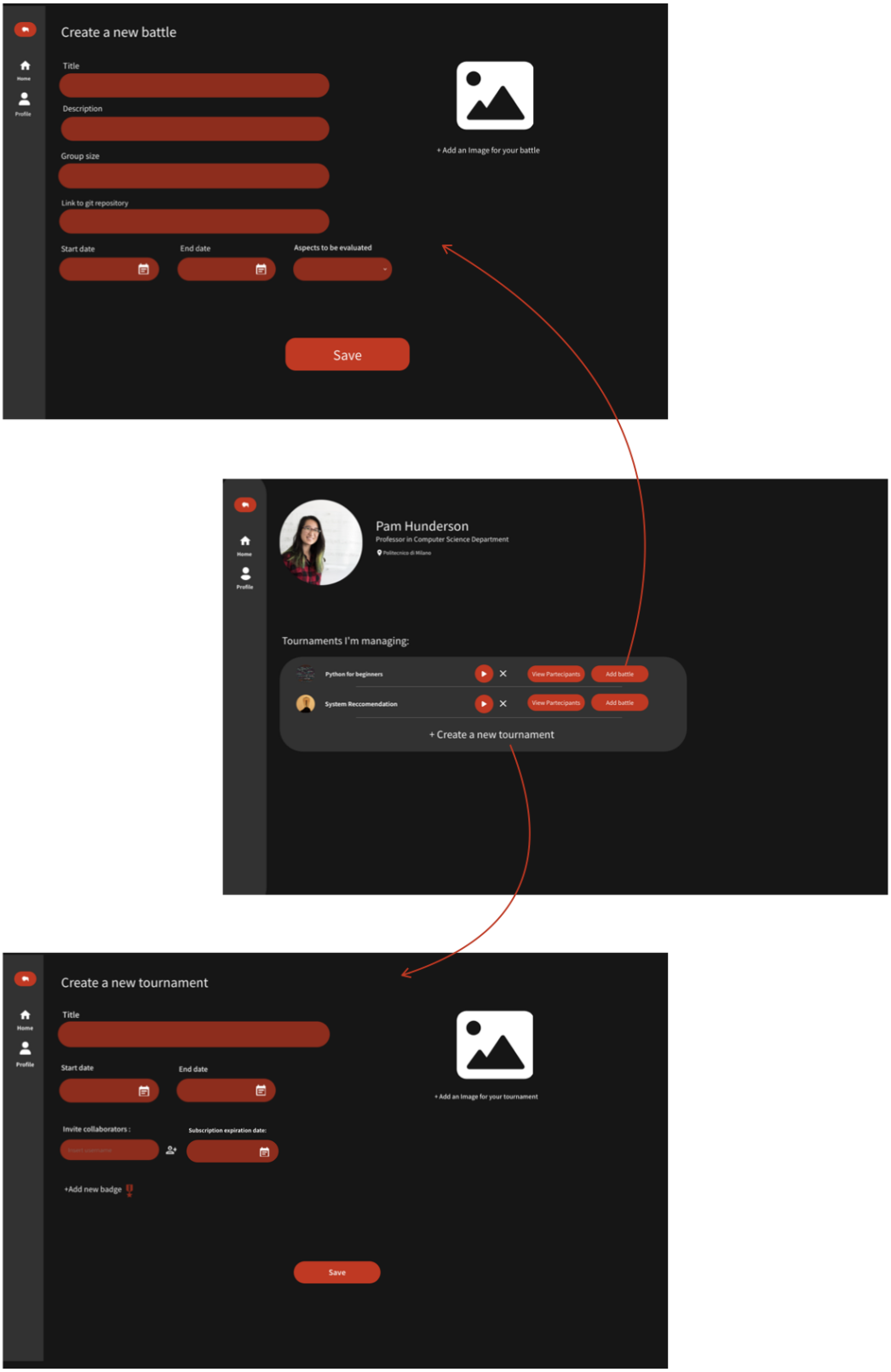


Figure 9: Educator creating tournament or battle MOCKUP

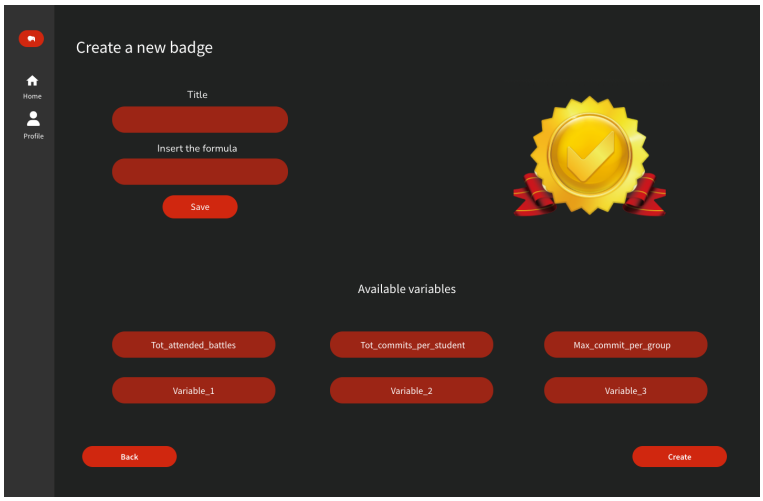


Figure 10: Creating a badge MOCKUP

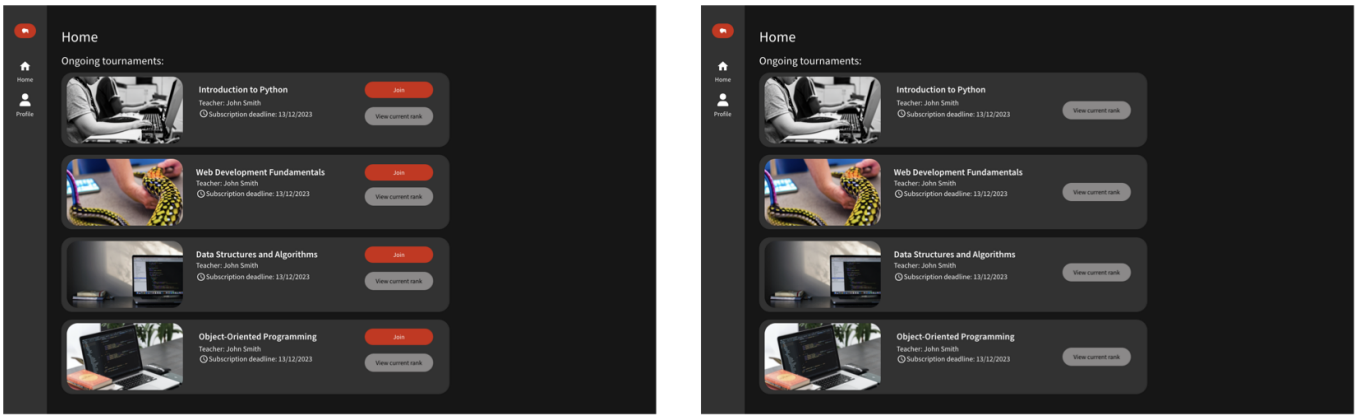


Figure 11: Students and Educator home page MOCKUP

4.1.2 Hardware interfaces

Both students and educator need an electronic device connected to internet to use the platform. Since the platform's primary functionality is closely tied to coding activities it is expected it will be used with a computer.

4.1.3 Software interfaces

The system has its own GitHub account. When a tournament is created by an admin and the deadline for creating a group for a battle has expired, the platform creates a repository through GitHub tokens and sends a notification with the link of the new repository to all the students subscribed to the battle. Students are then asked to fork it and to set an automated workflow through GitHub Actions that will trigger CKB whenever they push on the main branch of the fork.

4.1.4 Communication interfaces

The system uses the HTTPS protocol to transmit data over the internet.

4.2 Functional requirements

In the following are specified all the requirements that the system has to fulfill.

R1: The system allows user to sign up

R2: The system allows registered user to log in

R3: The system allows educator to create a tournament

R4: The system allows admin to invite collaborators to the tournament

R5: The system allows admin to create battles within the tournament they are managing

R6: The system allows admin to create badges when they create a tournament

R7: The system allows admin to evaluate manually participant's work

R8: The system allows admins to close tournaments

R9: The system allows user to visualize the rank

R10: The system allows user to visualize other user's details

R11: The system allows user to see the currently active tournaments

R12: The system allows user to see the battles created in a tournament

R13: The system allows student to join a tournament

R14: The system allows tournament's participant to join a battle

R15: The system allows tournament's participants to create a group

R16: The system allows tournament's participants to join an existing group

R17: The system allows tournament's participants to submit their code

R18: Notify the users of the newly created tournaments

R19: Notify the tournament's participants of the newly created battle

R20: The system creates a GitHub repository containing the code kata.

R21: The systems sends the link of the created GitHub repository to all students who are members of subscribed teams when the registration deadline expires.

R22: The system after each push before the deadline pulls the code, analyzes it and runs tests

R23: The system after each push updates the battle's and tournament's rank

R24: The system notifies all the participants of a tournament when consolidation stage has finished and the final rank is available

R25: The system notifies the tournament's participants when an admin closes a tournament they are subscribed to or when the expiration date elapses .

R26: The system awards the badge to the deserving student at the end of the tournament

R27: The system allows participant of a group to invite new group members unless the invited is in another group of the same battle.

4.2.1 Use case diagram

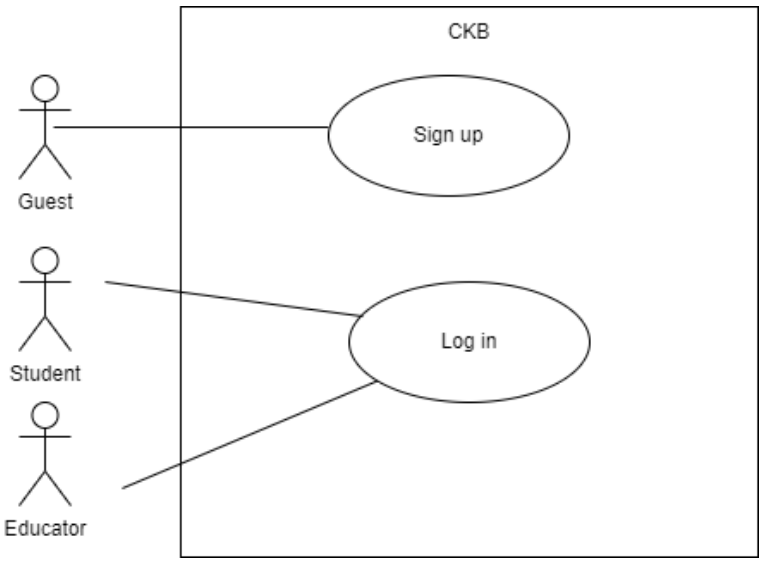


Figure 12: User login and guest sign up

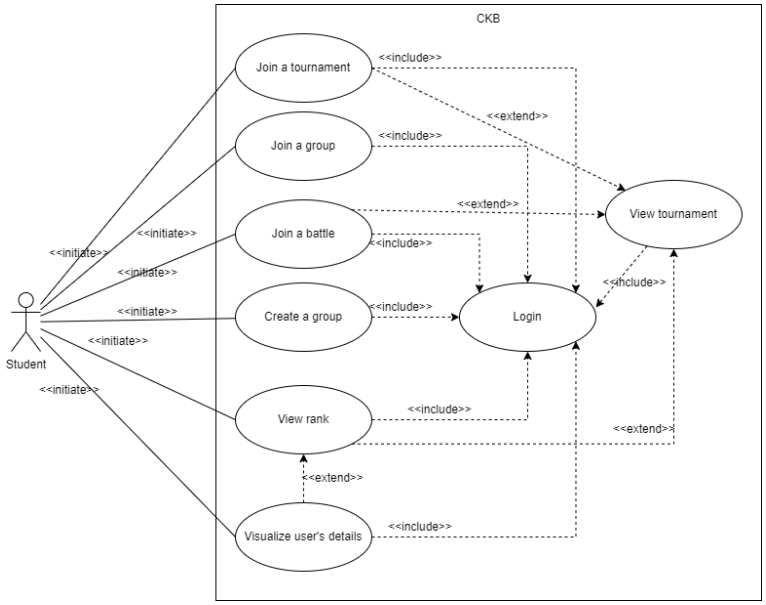


Figure 13: Student's use case diagram

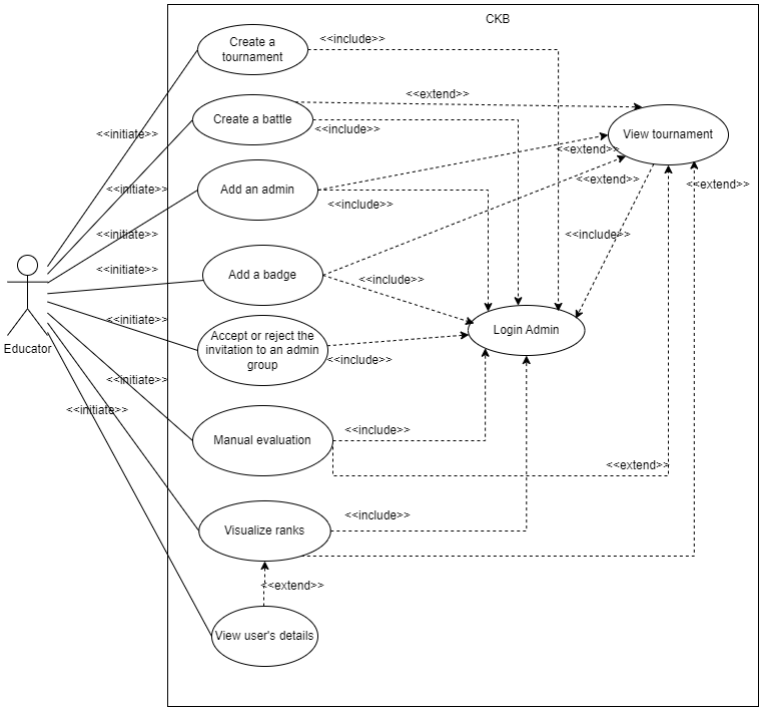


Figure 14: Educator's use case diagram

UC1: Login

Name	Login
Actors	User
Entry condition	The user has opened the CodeKataBattle website
Event flow	**1**: The user inserts his username and password in the form **2**: The user clicks on the "Login" button **3**: The system checks the credentials **4**: The application shows the proper dashboard
Exit condition	The user has access to the services for the right interface provided by CodeKataBattle
Exception	**(3)** The data inserted are not valid. The system returns to the entry condition.

Table 1: Login UC

UC2: Sign-up

Name	Sign-up
Actors	Guest
Entry condition	A person has downloaded the website and has a working internet connection
Event flow	**1**: Clicks on "Sign Up" **2**: Inserts his data and fills mandatory fields **3**: Account is registered by the system
Exit condition	The guest is now registered into the system and becomes a User
Exception	**(2)** Email is already associated with another account, User was already registered in the system

Table 2: Sign-up UC

UC3: Create a tournament

Name	Create a tournament
Actors	Educator
Entry condition	The admin is logged in and has a working internet connection
Event flow	**1**: Visits the "home" pages **2**: Clicks on "Create new tournament" **3**: Inserts tournament name, duration, badges and invites other colleagues as collaborators in the tournament. **4**: Tournament is registered by the system
Exit condition	The tournament is now registered into the system, a notification is sent to all students which can now subscribe
Exception	**(3)** Colleague is not registered

Table 3: Create Tournament UC

UC4: Create a badge

Name	Create a badge
Actors	Educator
Entry condition	The educator is logged in and has a working internet connection. He/she is an admin in a tournament and is creating it.
Event flow	**1**: educator clicks on "Add a badge". **2**: educator writes a formula in the formula section using the variables available, visible on the screen. **3**: educator saves changes.
Exit condition	The system creates the badge.
Exception	The formula is always false or always true, so the system reject the changes and prompts on the screen an informative message.

UC5: Create a battle

Name	Create a battle
Actors	Admin
Entry condition	The admin is logged in and has a working internet connection
Event flow	**1**: Admin clicks on the tournament for which a battle needs to be added from the home page **2**: Clicks on "Add a new battle" **3**: Inserts the description and software project (including test cases and build automation scripts),sets the minimum and maximum number of students per group, the registration deadline and the final submission deadline and finally clicks on "Create" **4**: Battle is registered by the system
Exit condition	The battle is now registered into the system, and and a notification is sent to all students in the tournament

Table 4: Create a battle UC

UC6: Join a tournament

Name	Join a tournament
Actors	Groups
Entry condition	The student is logged in and has a working internet connection
Event flow	**1**: Student receives notification of the newly created tournament **2**: Clicks on "Join tournament" from his notification page
Exit condition	Student is successfully subscribed to the tournament
Exception	**(2)** The registration deadline is expired

Table 5: Join a tournament via notification UC

Name	Join a tournament
Actors	Groups
Entry condition	The student is logged in and has a working internet connection
Event flow	**1**: Student goes to homepage **2**: Clicks on "Join tournament" from his homepage
Exit condition	Student is successfully subscribed to the tournament
Exception	**(2)** The registration deadline is expired

Table 6: Join a tournament via homepage UC

UC7: Join a battle

Name	Join a battle
Actors	Groups and Student
Entry condition	Each one in the group has an CodeKataBattle account and is subscribed to a tournament
Event flow	**1**: Student receives notification of the newly created battle. **2**: Reaches the tournament page either by clicking on the notification or selecting the tournament from his home page. **3**: Clicks on the "Join battle" button. **4**: Either creates a new group and invites his friends or joins an existing group.
Exit condition	The system has recorded the group's registration, and it is pending approval from all the invited members of the group
Exception	**(2)** The registration deadline is expired **(3)** The invited student doesn't exist (is not registered in the platform) **(4)** The registration deadline expires and the minimum number of group member has not been reached

Table 7: Join a battle UC

Name	Join a battle
Actors	Groups and Student
Entry condition	Each one in the group has an CodeKataBattle account and is subscribed to a tournament
Event flow	**1**: Student receives an invite in a group for a battle. **2**: Student accept the invitation and joins the group and consequently the battle
Exit condition	The system has recorded the group's registration, and it is pending approval from all the other invited members of the group
Exception	**(2)** The registration deadline is expired and the minimum number of group member has not been reached

Table 8: Join a battle when invited UC

UC8: Create a group

Name	Create a group
Actors	Student, Group
Entry condition	The student is logged in and has a working internet connection. He is a participant in a tournament, he clicked on the join button of a battle.
Event flow	**1**: The Student clicks on the create group button. **2**: The Student can either invite other students or not. **3**: The Student choose to make the group public or private.
Exit condition	Student is successfully in the group and eventual invitations to other students are sent.

Table 9: Create a group UC

UC9: Submission of the code

Name	Submission of the code
Actors	Student, GitHub repository
Entry condition	The student is logged in and has a working internet connection. He/she is a participant to a tournament, is currently in a battle, received the link to the repository and forked the repository.
Event flow	**1**: Student pushes on the main branch of his group fork. **2**: The GitHub action provided in the repository notifies the system of the push. **3**: The system runs test on the code. **4**: A score is provided based on the system performances. **5**: If any push has been performed before and the new score surpasses the old one, the points that were given are revoked and the new points are assigned to each one of the group members.
Exit condition	The system updates the points and consequently the leaderboard.
Exception	The battle expired so the code is not evaluated and no points are assigned to the group.

Table 10: Submission of the code UC

UC10: Visualize active tournaments

Name	Visualize tournaments where you are neither a participant neither an admin
Actors	User
Entry condition	The user is correctly logged in and has working internet connection
Event flow	**1**: Visits the "home" page
Exit condition	The user can correctly visualize on the screen all the disposable tournaments

Table 11: Visualize active tournaments UC

UC11: Visualize the participants of a tournament

Name	Visualize the participants of a tournament
Actors	User
Entry condition	The user is correctly logged in and has working internet connection
Event flow	**1**: Visits the "home" pages **2**: Select a tournament **3**: Open the ranking
Exit condition	The participants of the selected tournament are visible now on the screen

Table 12: Visualize participants of a tournaments UC

UC12: Visualize an user details

Name	Visualize an user details
Actors	User
Entry condition	The user is correctly logged in and has working internet connection
Event flow	**1**: Visits the "home" pages **2**: Select a tournament **3**: Select a user from the rank
Exit condition	The system shows on the screen the user's details including name, user ID and badges

Table 13: Visualize an users detail UC

UC13: Add an admin

Name	Add an admin
Actors	Admin, Educator
Entry condition	The admin is correctly logged in and has a working internet connection
Event flow	**1**: Visits the personal area **2**: Select one of the tournament in "Managing" area. **3**: Select the icon to see tournament's details. **4**: Select "Add admin". **5**: Insert educator's ID or name. **6**: Commit the changes.
Exit condition	The system correctly added the user to the list of admin and gave him/her all the admins privileges
Exception	**(5)** User is not registered, user is already an admin

Table 14: Add an admin UC

4.2.2 Sequence diagrams

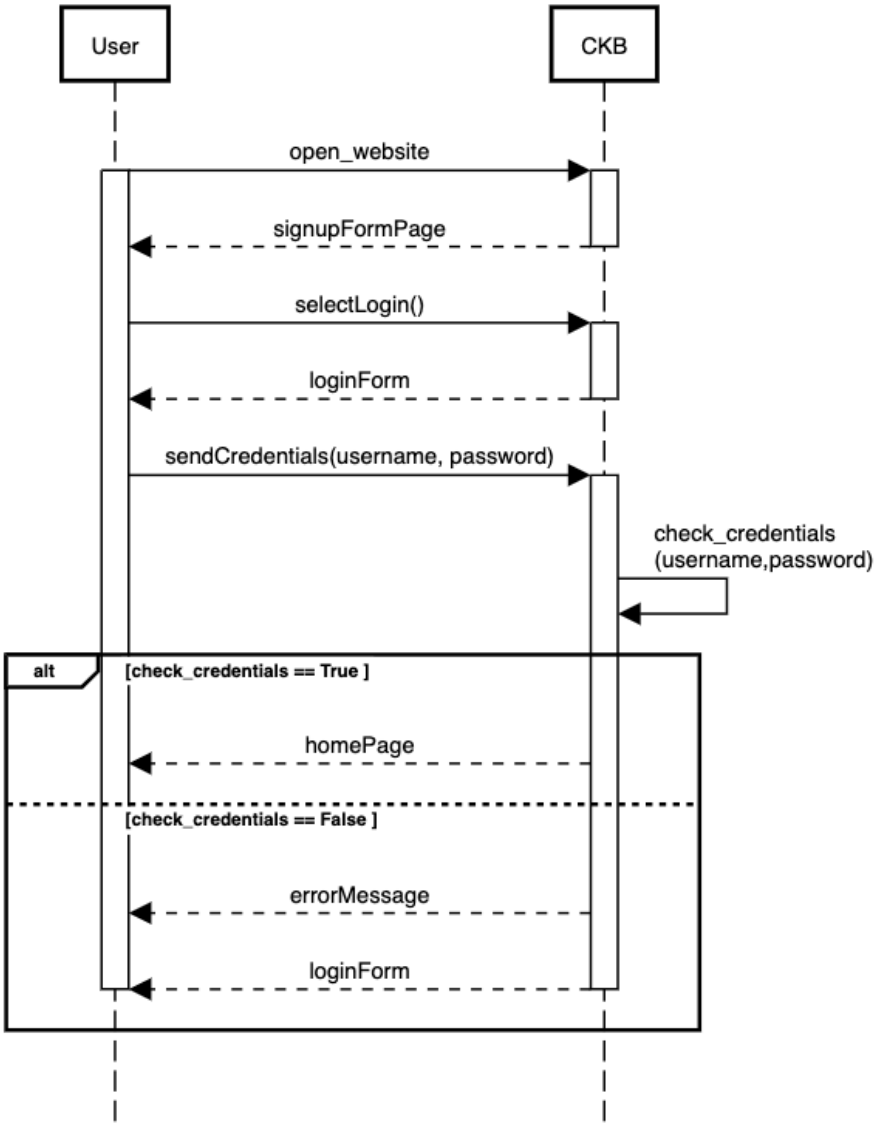


Figure 15: User Login SD

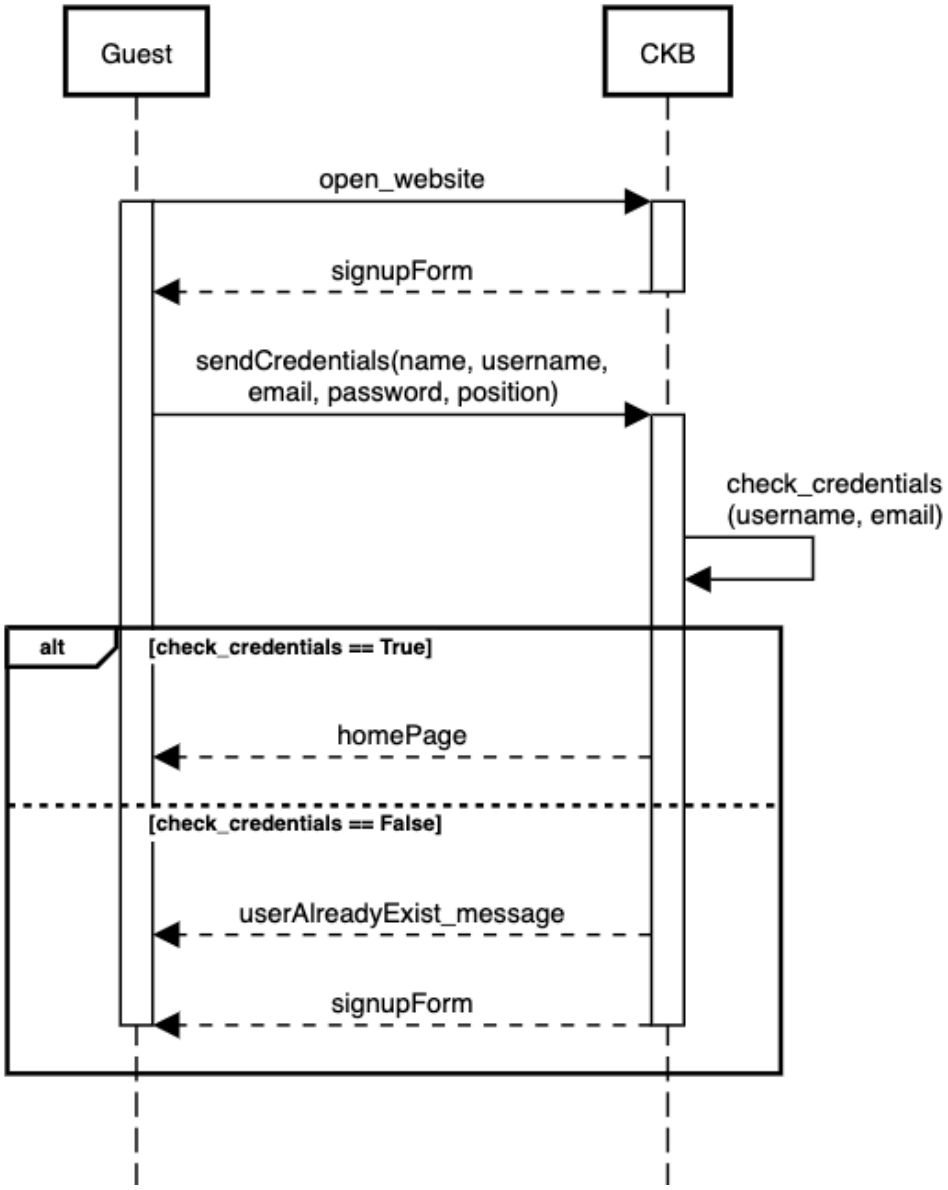


Figure 16: Sign-up SD

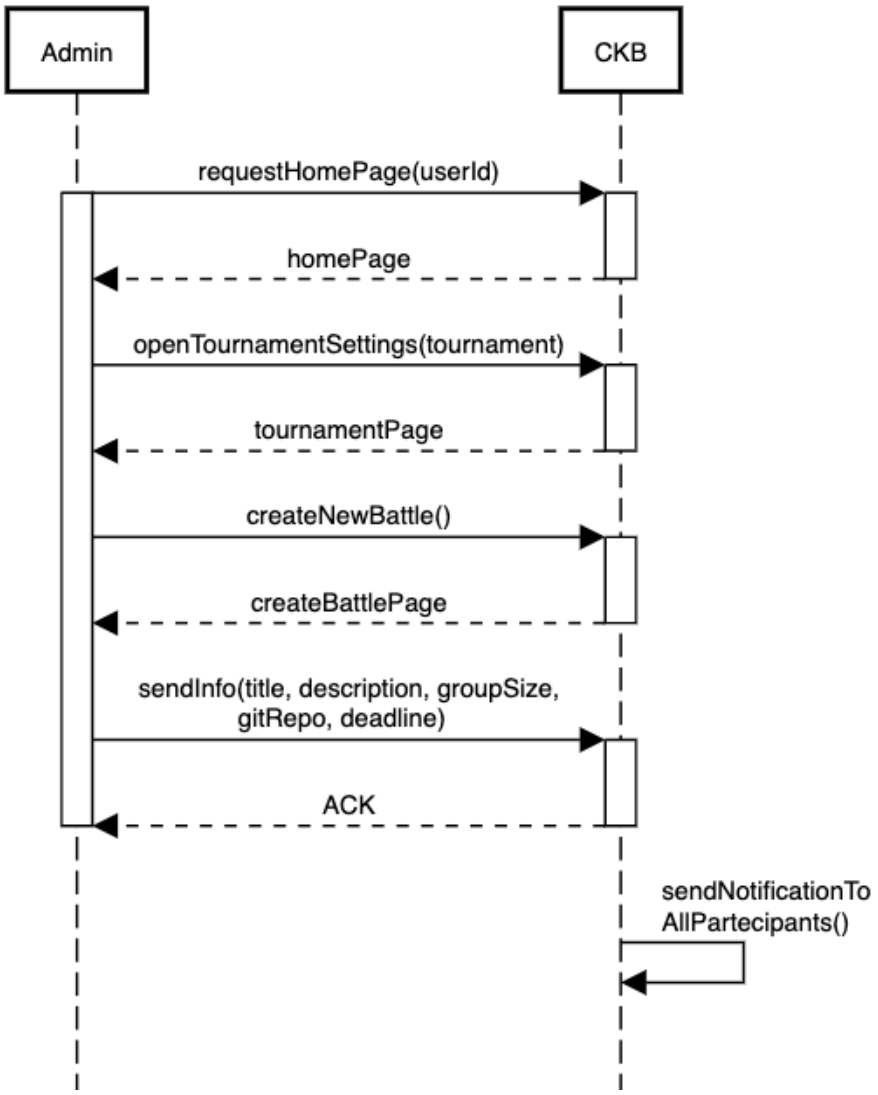


Figure 17: Create new battle SD

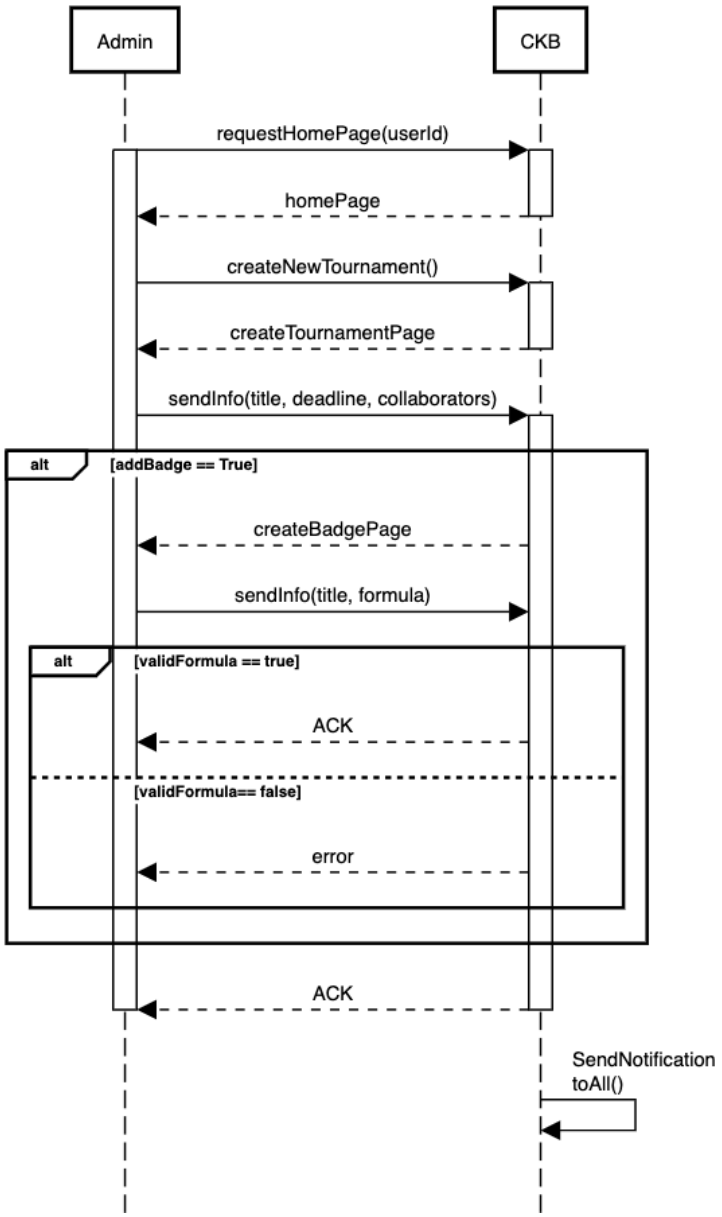


Figure 18: Create new tournament and create a badge SD

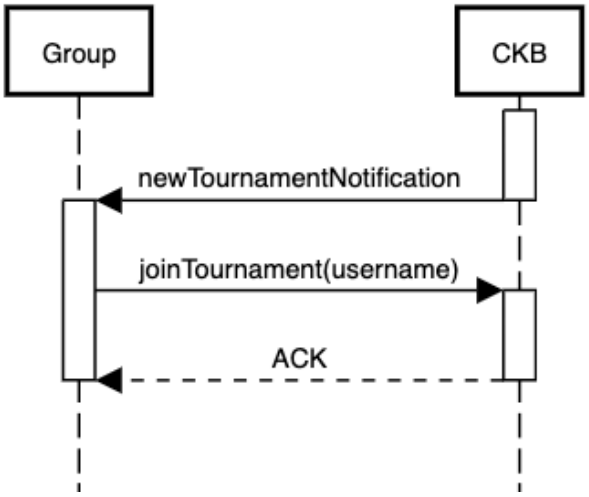


Figure 19: Join tournament via notification SD

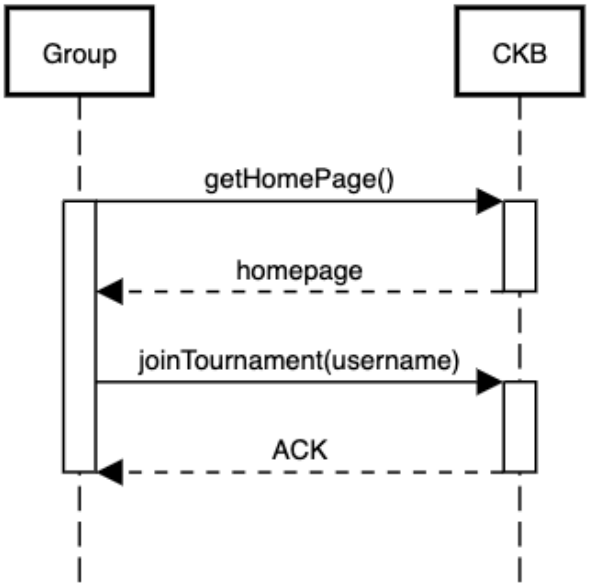
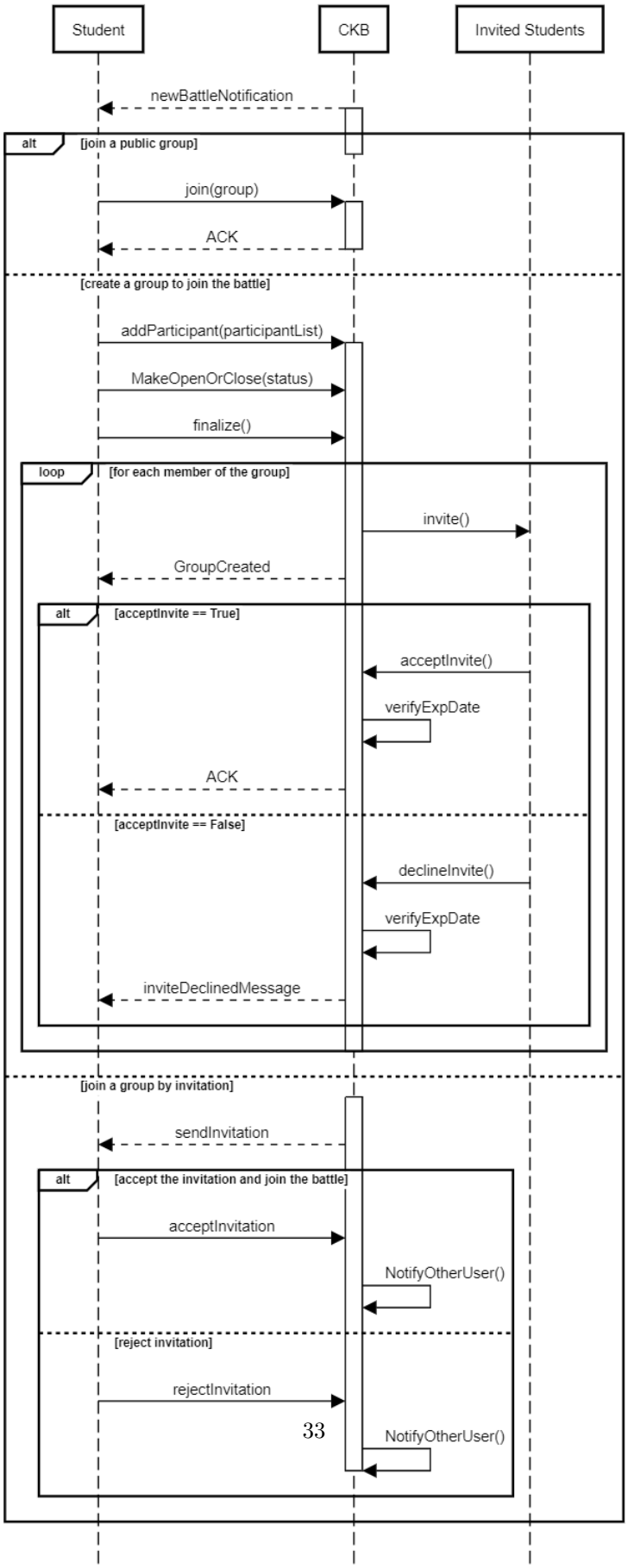


Figure 20: Join tournament via homepage SD


Student
CKB
Invited Students
newBattleNotification
alt
[join a public group]
join(group)
ACK
[create a group to join the battle]
addParticipant(participantList)
MakeOpenOrClose(status)
finalize()
loop
[for each member of the group]
invite()
GroupCreated
alt
[acceptInvite == True]
acceptInvite()
verifyExpDate
ACK
[acceptInvite == False]
declineInvite()
verifyExpDate
inviteDeclinedMessage
[join a group by invitation]
sendInvitation
alt
[accept the invitation and join the battle]
acceptInvitation
NotifyOtherUser()
[reject invitation]
rejectInvitation
NotifyOtherUser()

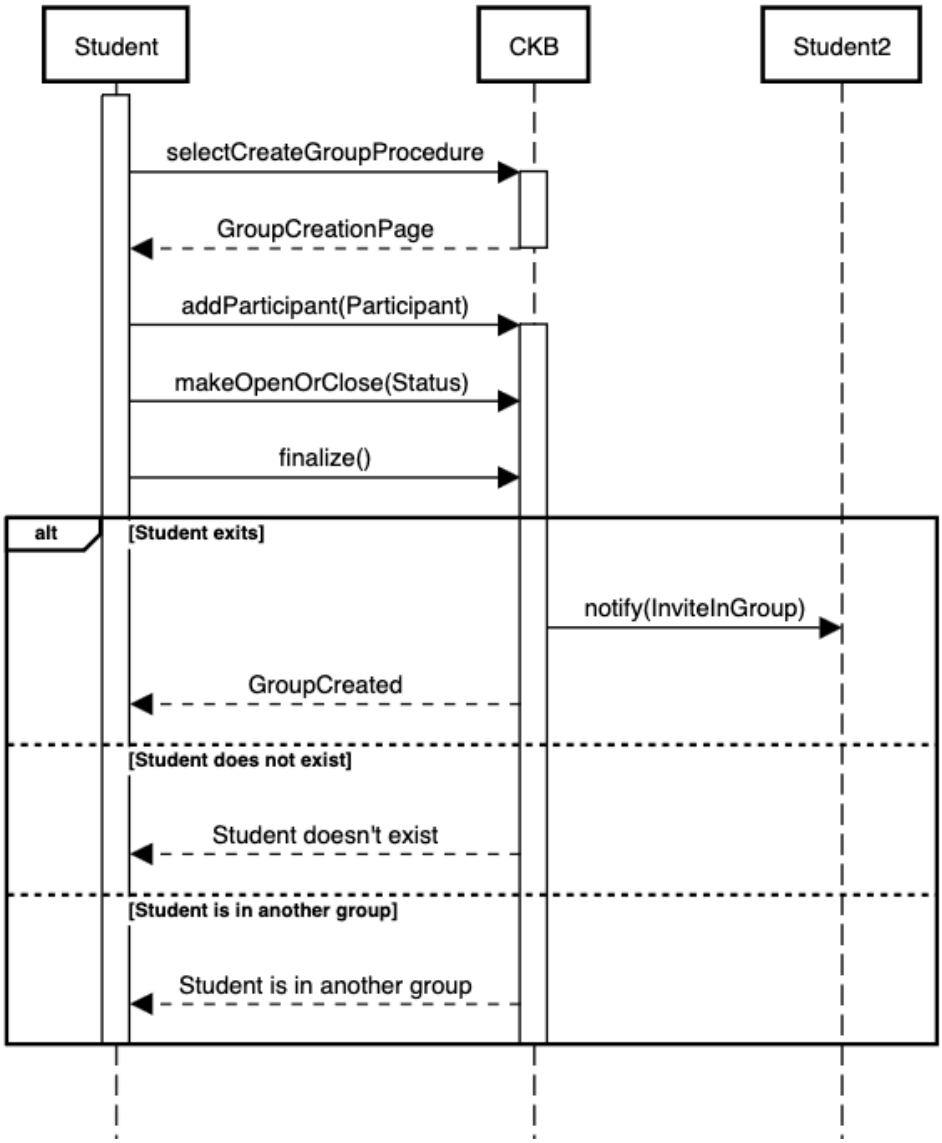


Figure 22: Create a group SD

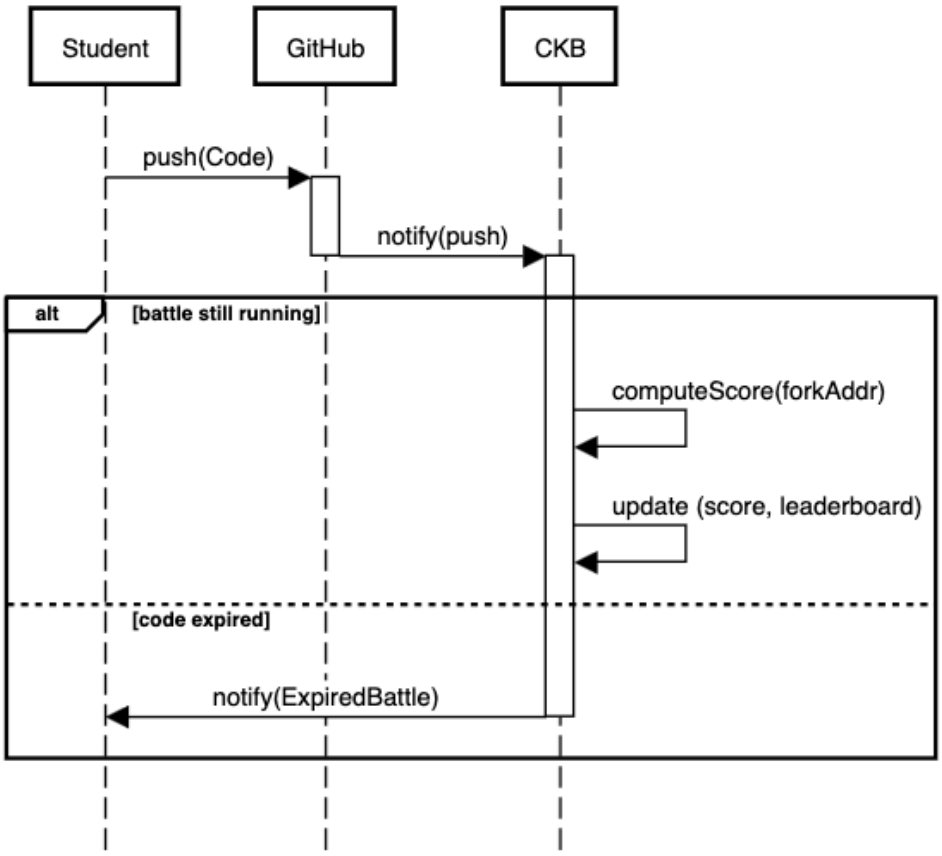


Figure 23: Submission of the code on GitHub SD

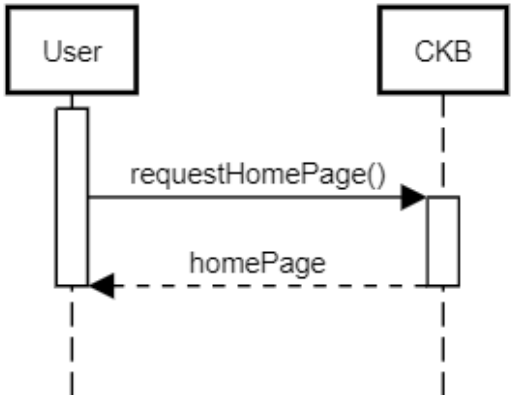


Figure 24: Visualize active tournaments SD

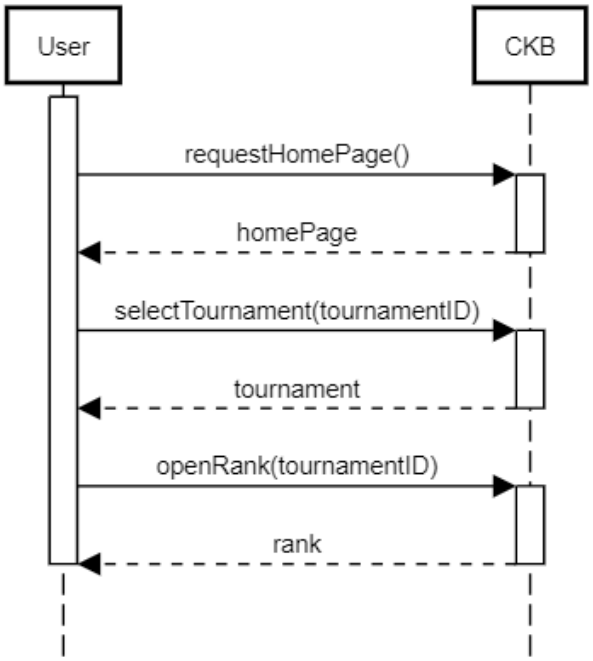


Figure 25: Visualize the participants of a tournament SD

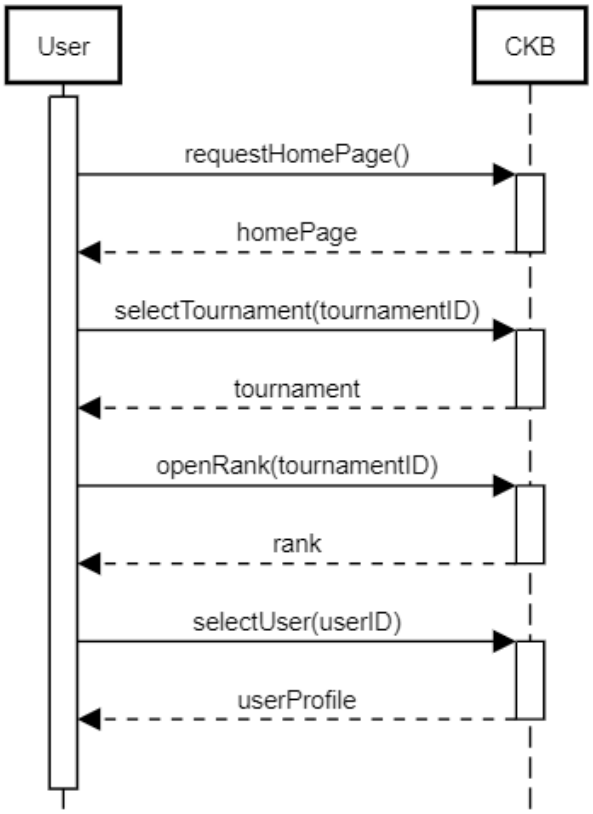


Figure 26: Visualize a user's details SD

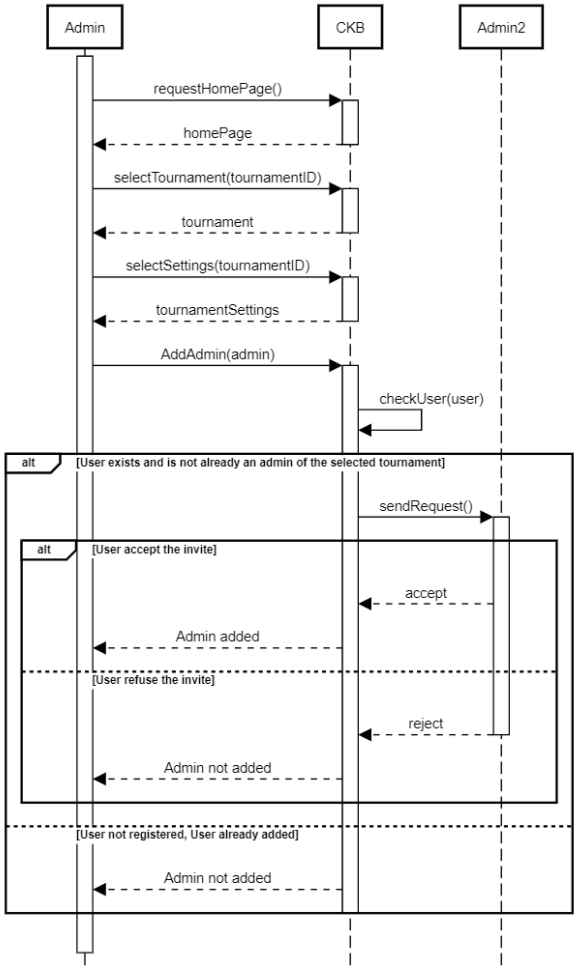


Figure 27: Add an educator SD

4.2.3 Requirements mapping

[G1] Student can join a tournament and thus, become a participant of such tournament.	
R1: The system allows user to sign up	DA1: User must have internet connection
R2: The system allows registered user to log in	DA2: User must have a GitHub account
R11: The system allows user to see the currently active tournaments	DA3: Each user must have one and only one account
R13: The system allows student to join a tournament	DA4: Each user must have a device to use the system
R18: Notify the users of the newly created tournaments	

Table 15: Requirements and assumptions mapping of G1

[G2] Participant can create a group when entering a kata battle	
R12: The system allows user to see the battles created in a tournament	DA1: User must have an internet connection
R14: The system allows tournament's participant to join a battle	DA2: User must have a GitHub account
R15: The system allows tournament's participant to create a group	DA3: Each user must have one and only one account
R27: The system allows participant of a group to invite new group members unless the invited is in another group of the same battle	DA4: Each user must have a device to use the system
	DA5: Groups remain the same for the duration of whole battle
	DA7: A group in a battle is allowed to consist of just one person

Table 16: Requirements and assumptions mapping of G2

[G3] Participant can join an existing group when entering a kata battle	
R12: The system allows user to see the battles created in a tournament	DA1: User must have an internet connection
R14: The system allows tournament's participant to join a battle	DA2: User must have a GitHub account
R16: The system allows tournament's participant to join an existing group	DA3: Each user must have one and only one account
R27: The system allows participant of a group to invite new group members unless the invited is in another group of the same battle	DA4: Each user must have a device to use the system
	DA7: A group in a battle is allowed to consist of just one person

Table 17: Requirements and assumptions mapping of G3

[G4] Groups can submit their code through GitHub	
R17: The system allows tournament's participants to submit their code	DA1: User must have an internet connection
R20: The system creates a GitHub repository containing the code kata	DA2: User must have a GitHub account
R21: The system sends the link of the created GitHub repository to all the students who are members of subscribed teams when the registration deadline expires	DA3: Each user must have one and only one account
	DA4: Each user must have a device to use the system
	DA5: Groups remain the same for the duration of whole battle
	DA8: During a battle student pushes correctly his code in the branch of group inside the repository whose link is provided by the system

Table 18: Requirements and assumptions mapping of G4

[G5]User can view the current rank evolving through each battle and tournament	
R9: The system allows user to visualize the rank	DA1: User must have an internet connection
R24: The system notifies all the participants of a tournament when consolidation stage has finished and the final rank is available	DA4: Each user must have a device to use the system

Table 19: Requirements and assumptions mapping of G5

[G6] Admin can invite collaborators to the tournament that will be considered as admins once they accept the invite	
R4: The system allows admin to invite collaborators to the tournament	DA1: User must have an internet connection DA4: Each user must have a device to use the system DA3: Each user must have one and only one account

Table 20: Requirements and assumptions mapping of G6

[G7] The admin can create tournaments, battles, and badges within the specific tournaments.	
R1: The system allows user to sign up R2: The system allows registered user to log in R3: The system allows educator to create a tournament R5: The system allows admin to create battles within the tournament they are managing R6: The system allows admin to create badges when they create a tournament	DA1: User must have an internet connection DA4: Each user must have a device to use the system

Table 21: Requirements and assumptions mapping of G7

[G8] Admin can adjust the automatic evaluation of a battle performed by the platform during the consolidation phase	
R7: The system allows admin to evaluate manually participant's work	DA1: User must have an internet connection DA4: Each user must have a device to use the system

Table 22: Requirements and assumptions mapping of G8

4.3 Performance requirements

The system does not provide a critical service, however it needs to be usable and reasonably updated with all the GitPushes. We compiled a list of requirements and measurements that concur to reach this goal:

- Responsiveness:
  - Requirement: The system should provide a responsive interface, ensuring that users experience minimal delay when interacting with the application.
  - Measurement: The application should respond to user actions within 1-2 seconds under normal conditions.
- GitHub Push Processing:
  - Requirement: The system should update the status of the leaderboard within a reasonable time after a GitHub push, with a maximum processing time of 10 seconds.
  - Measurement: The leaderboard should reflect changes within 10 seconds of a GitHub push.
- Scalability:

  - Requirement: The system should be designed to handle a simultaneous traffic load of around 100,000 users.
  - Measurement: The application should maintain acceptable performance, responsiveness, and availability with 100,000 simultaneous users.

- Global Accessibility:
  - Requirement: The application should be accessible and performant for users around the world. However, we expect the participants in the same tournament to be located in the same area, so development can take this into account.
  - Measurement: Users from different geographical locations should experience consistent performance and responsiveness.
    * Sub-Measurement: While considering that participants from the same tournament may be in the same area, optimize server distribution or content delivery to enhance performance for geographically clustered users.

- Load Testing:
  - Requirement: The system should undergo load testing to ensure that it meets performance expectations under simulated heavy traffic conditions.
  - Measurement: The application should demonstrate stability and responsiveness during load tests with 100,000 simultaneous users.

- Scalability Planning:
  - Requirement: The system architecture should allow for easy scalability to accommodate potential increases in user traffic in the future.
  - Measurement: The system should be capable of scaling horizontally or vertically based on changing user demands.

- Monitoring and Optimization:
  - Requirement: The system should include monitoring tools to identify performance bottlenecks and optimize system components.
  - Measurement: Regular performance monitoring should be conducted, and optimizations should be implemented as needed to maintain responsiveness.

- User Feedback:
  - Requirement: Collect user feedback on application performance and responsiveness.
  - Measurement: Regularly gather and analyze user feedback to identify areas for improvement in performance and responsiveness.

4.4 Design Constraints

4.4.1 Standards compliance

- **Code Quality and documentation**

  The codebase of CodeKataBattle should adhere to the requirements contained in this document. To enhance code readability and maintainability comprehensive comments should be provided for both methods and classes.

- **Privacy of the data**

  The CodeKataBattle project is subject to the General Data Protection Regulation (GDPR) (a regulation in EU law on data protection and privacy), so it is obligated to guarantee privacy and protection for user data. The software must implement secure transmission

- **International Standards for Date and Time**

  The system must adopt international standards for the representation of date, like the ISO 8601, to ensure consistency and interoperability between users of different time zones.

4.4.2 Hardware limitations

All users must have an internet connection (2G/3G/4G/5G/Wi-Fi) to enable the transmission and reception of data. Additionally, users need a personal computer to access the software's features through a web browser.

The web-based nature of the application ensures cross-platform compatibility, allowing users to engage with the software using various operating systems (Windows, macOS, Linux) and browsers (Chrome, Firefox, Safari).

Compatibility testing should be conducted regularly to address any potential issues arising from browser updates.

4.5 Software System Attributes

4.5.1 Reliability

To maintain the reliability of the CodeKataBattle (CKB) platform, we will establish regular data backup mechanisms and a robust error-handling system to minimize the impact of unforeseen issues. While not the primary focus, reliability measures will be in place to ensure a stable user experience.

4.5.2 Availability

High availability is a primary focus for the CKB platform to guarantee great usability also in case of periods of high pressure and demand on the system. Implementation of a redundancy system will be a key strategy to achieve this goal.

4.5.3 Security

While the CKB platform does not store highly sensitive personal information, a baseline level of security is essential. User passwords and data will be encrypted to prevent unauthorized access. Secure connections (HTTPS) will be enforced for all interactions, prioritizing a secure environment.

4.5.4 Maintainability

Maintaining the CKB platform's codebase is vital for long-term success. Code will be thoroughly commented and documented. Each addition to the codebase will be accompanied by unit and integration tests, covering at least 80% of the code.

4.5.5 Portability

Portability is a key design consideration for the CKB platform. The goal is to ensure the software runs seamlessly on all operating systems that support a web browser.

5 Formal analysis using alloy

5.1 Signatures, functions and show predicate

In this section we documented the signatures and the utility functions that we wrote in our alloy model. We tried to keep the model as simple as possible, nevertheless we included all the relevant attributes for each class.

```
abstract sig User{
}

sig Educator extends User {
       manages :some Tournament
}

sig Student extends User {
       , participates :some Tournament
       , badges :some Badge
}

sig Group {
       , creator :one Student
       , composed :set Student
       , participates :one Battle
}

sig Tournament {
       , creator :one Educator
       , admins: set Educator
       , contains :set Battle
       , badges :set Badge
}

sig Battle{
       , groupSize: Int,
       , publicGroups :some Group,
       , privateGroups :some Group
       , partOf :one Tournament
       , status :BattleStatus
}

enum BattleStatus {Registration, Running, Closed}

sig InvitationStudent {
       , between :Student -> Student
       , status :one InvitationStatus
       , toGroup :one Group
} {#between =1}

sig InvitationEducator {
       , between :Educator -> Educator
       , status :one InvitationStatus
       , to :one Tournament
} {#between =1}

enum InvitationStatus {
       Accepted, Rejected, Pending
}

sig Badge{}

sig Submission {
       , mark :one Int
       , fromTo :Group -> Battle
} {mark >0 #fromTo =1}

/*-----------------------------------------------------------------------------------------*/

pred show {
       #Educator =3
       #Student =6
       #InvitationEducator >2
```

```
        #Tournament =2
        #Battle =2
        #InvitationStudent >6
        #Group =4
        all g :Group |#g.composed >1
        all b :Battle |b.groupSize <4
        //some g : Group | #g.composed != g.participates.groupSize
        #Badge =3
        all s :Student |#s.badges >1
        #Submission =3
}

run show for 24

/*-------------------------------------------------------------------------------------------*/

//UTILITY

fun invited [ie :InvitationEducator] :Educator {
        Educator.(ie.between)
}

fun inviter [ie :InvitationEducator] :Educator {
        ie.between.Educator
}

fun invited [is :InvitationStudent] :Student {
        Student.(is.between)
}

fun inviter [is :InvitationStudent] :Student {
        is.between.Student
}
```

5.2 Facts

In this section we wrote all the facts that we thought relevant in the development of the alloy model.

```
/*-------------------------------------------------------------------------------------------*/

//TOURNAMENT
//someone is an admin in a tournament iff he/she manages the tournament
fact adminManagesAndViceversa {
        no t :Tournament, e :Educator |(e in t.admins and t not in e.manages) or (t in e.manages and e not in t.admins)
}

//the creator is part of the admins
fact creatorIsAdmin {
        all t :Tournament |t.creator in t.admins
}


//the educators or are not admins of a tournament, or are admins and creators, or they accepted
//an invitation to the tournament
fact authorizedAdmins {
        all t :Tournament, e :Educator |(e not in t.admins)
        or (e in t.admins and e =t.creator)
        or (some ie :InvitationEducator |ie.to =t and ie.invited =e and ie.status =Accepted)
}

/*-------------------------------------------------------------------------------------------*/

//ADMIN_INVITATIONS
//an admin cannot invite himself
fact noSelfInviteAdmin {
        no iA :InvitationEducator |some e: Educator |#(iA.between :>e & e <:iA.between) >0
}


//if the number of admins is greater than one there is at least an invitation to that tournament where
//the inviter is the creator
fact initialInvitation {
```

```
        no t :Tournament |#(t.admins) >1 and no ie :InvitationEducator |
        ie.to =t and ie.inviter =t.creator
}


//the inviter has to be an admin in that tournament
fact authorizedInviter {
        no ie :InvitationEducator |ie.inviter not in ie.to.admins
}


//an admin cannot invite another admin
fact noInvitationBetweenAdmins {
        no ie :InvitationEducator |some ie2 :InvitationEducator |ie.status =Accepted and ie.inviter =ie2.invited
}

//if there is an accepted invitation to an admin group the invited has to be part of the group
fact acceptedInvitationsAreMeaningful {
        all ie :InvitationEducator |ie.status =Accepted implies ie.invited in ie.to.admins
}

/*-----------------------------------------------------------------------------------------*/
//GROUP
//the creator is part of the group
fact creatorInComposed {
        all g :Group |g.creator in g.composed
}

//in the same battle there are no overlapping groups
fact noOverlappingGroups {
        all b :Battle |not some disj g1, g2 :Group |
        ((g1 in b.publicGroups) or (g1 in b.privateGroups)) and ((g2 in b.publicGroups) or (g2 in b.privateGroups)) and
        #(g1.composed & (g2.composed)) >0
}

//a student cannot be part of a group that participates a tournament he is not a participant of
fact validGroupMembers {
        no g :Group |some s :Student |s in g.composed and
        g.participates.partOf not in s.participates
}

//if a student is part of a group he is either the creator or the group is public or
// if the group is private he accepted an invitation
fact authorizedParticipants {
        all g :Group, s :Student |s in g.composed implies (
                (g.creator =s) or
                (g in g.participates.privateGroups and
                some is :InvitationStudent |is.status =Accepted and is.invited =s
                and is.inviter in g.composed and is.toGroup =g) or
                (g not in g.participates.privateGroups)
        )
}


//in the same battle the intersection between the set of public groups and the set of private
//groups is empty
fact disjointedPrivatePublicGroups {
        all b :Battle |b.privateGroups & b.publicGroups =none
}


/*-----------------------------------------------------------------------------------------*/

//STUDENT_INVITATIONS

//a student cannot invite himself
fact noSelfInviteStudent {
        no is :InvitationStudent |is.inviter =is.invited
}

//if the number of students in a group is greater than one and the group is closed there is at least an
//invitation to that tournament where the inviter is the creator
fact InitialInvitation {
        no g :Group |g in g.participates.privateGroups and #g.composed >0 and no is :InvitationStudent |
```

```
        is.toGroup =g and is.inviter =g.creator
}

//if there is an accepted invitation to a group the invited has to be part of the group
fact usefulInvitation {
        all is :InvitationStudent |is.status =Accepted implies is.invited in is.toGroup.composed
}

//the inviter must be a participant in the group
fact authorizedInviter {
        all is :InvitationStudent |is.inviter in is.toGroup.composed
}

//a student cannot invite another participant in the group
fact noInvitationsOfOtherParticipants {
        no is :InvitationStudent |some is2 :InvitationStudent |is.toGroup =is2.toGroup and
        is.status =Accepted and is.inviter =is2.invited
}

//there can be only one accepted invite in the group
fact noMultipleAccepted {
        no disj is1, is2 :InvitationStudent |is1.toGroup =is2.toGroup and is1.inviter =is2.inviter
        and is1.invited =is2.invited and is1.status =Accepted and is2.status =Accepted
}
/*-------------------------------------------------------------------------------------------*/

//BATTLE

//a tournament contains a battle iff the battle has the tournament as partOf
fact coherentContainsAndPartOf {
        all t :Tournament, b :Battle |b in t.contains iff t =b.partOf
}

//the battle where the group participates contains the group either in the private or in the public groups
fact coherentParticipatesContains {
        all g :Group |g in (g.participates.privateGroups + g.participates.publicGroups) and (not some
        b :Battle |b ≠g.participates and g in (b.privateGroups + b.publicGroups))
}

//there are no groups in battle where status is closed or running and contains groups with number of participants different
//than the one specified by the battle
fact validGroupSize {
        no g :Group, b :Battle |g in (b.privateGroups + b.publicGroups) and #g.composed ≠b.groupSize
        and (b.status =Closed or b.status =Running)
}

/*-------------------------------------------------------------------------------------------*/

//BADGE
//a student to have a badge must have been a participant to the tournament
fact legallyObtainedBadge {
        all s :Student, b :Badge |b in s.badges implies (some t :Tournament |b in t.badges and t in s.participates)
}

//there are no badges that are not part of tournaments
fact noDisconnectedBadges {
        no b :Badge |not some t :Tournament |b in t.badges
}

//a badge belongs only to one tournament
fact badgeBelongsToOnlyOneTournament {
        all b :Badge, t1 :Tournament |b in t1.badges implies (not some t2 :Tournament |t2 ≠t1 and
        b in t2.badges)
}


/*-------------------------------------------------------------------------------------------*/

//SUBMISSION
//the group has to have been part of a battle of that tournament to have a submission for that tournament
fact legalSubmission {
        no s :Submission |s.fromTo.Battle not in (Group.(s.fromTo).privateGroups + Group.(s.fromTo).publicGroups)
}
```

5.3 Assertions

In this section we report the assertions that we wrote to verify some of the properties that the model has to satify.

```
/*-------------------------------------------------------------------------------------*/
//ASSERTIONS

//a group participates only one battle
assert eachGroupIsInOnlyOneBattle {
       all disj b1, b2 :Battle |no g :Group |(g in b1.publicGroups or g in b1.privateGroups) and
       (g in b2.publicGroups or g in b2.privateGroups)
}

//if the battle is running all the groups are of the size specified in groupSize
assert smallerGroups {
       all g :Group |g.participates.status =Running implies
       #g.composed =g.participates.groupSize
}

//the status of the invitation cannot be accepted if the invited is not part of the tournament
assert noInvalidAccept {
       no is :InvitationStudent |is.status =Accepted and is.toGroup.participates.partOf not in is.invited.participates
}

//if an educator is an admin he manages that tournament
assert coherentManages {
       all t :Tournament, e :Educator |e in t.admins iff t in e.manages
}

//a student cannot accept invitations to different groups
assert noMultipleAccept {
       no disj is1, is2 :InvitationStudent |is1.invited =is2.invited and is1.toGroup ≠is2.toGroup
       and is1.toGroup.participates =is2.toGroup.participates and
       is1.status =Accepted and is2.status =Accepted
}

check eachGroupIsInOnlyOneBattle for 15
check smallerGroups for 15
check noInvalidAccept for 10
check coherentManages for 15
check noMultipleAccept for 10
```

5.4 Alloy Results

We report in Figure 28 the results of the analysis made by the Alloy analyzer. The picture exhibits that the analyzer found an instance where predicate show is true and, subsequently, it states that couldn't find a counterexample for every one of the assertions, meaning that the model is coherent with the specifications, at least within the scope verified.

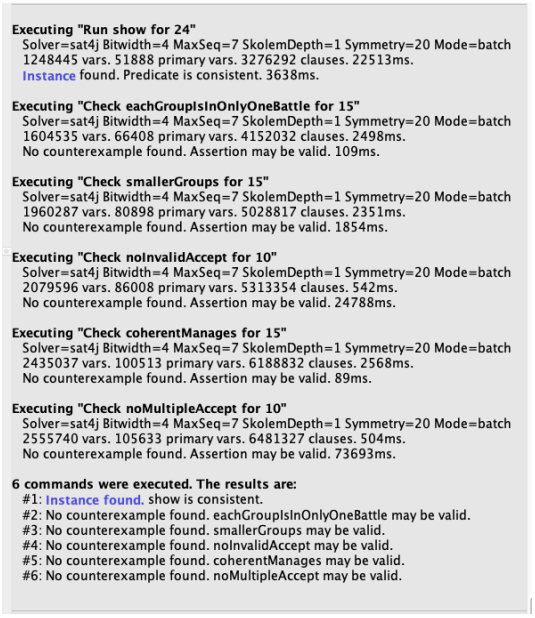


Figure 28: Alloy results

5.5 Brief analysis of the invitations

Here we analyze one instance of the show predicate, in particular we focus on the student invitations in groups. We used the tabular visualization option for clarity.

5.5.1 Invitations in groups

In Figure 29 are reported the battles created by the model: There are two battles (Battle0 and Battle1), both of them have two groups registered and in particular Battle0 has group0 and group3, while Battle 1 group1 and group2. The groups in the publicGroup relation are considered public, meaning that there are no particular restrictions in the way participants can join. In this case all the participants of group0 joined without invitations. However, we gave the possibility to students participating public groups to invite other students. The groups in privateGroup relation are private, meaning that the only possible way to join it is via invitation. In particular, student1, the creator of the group, invited both student2 and student3 and they both accepted. It is worth noticing that not only the creator of the group can invite new participants, in fact in group3 student0 was invited student3.

this/Battle	groupSize	publicGroups	privateGroups	partOf	status
Battle⁰	2	Group⁰	Group³	Tournament¹	Closed⁰
Battle¹	1	Group¹	Group²	Tournament⁰	Registration⁰

Figure 29: Alloy battles

this/Group	creator	composed	participates
Group⁰	Student¹	Student¹	Battle⁰
		Student²	
Group¹	Student⁵	Student⁰	Battle¹
		Student⁵	
Group²	Student¹	Student¹	Battle¹
		Student²	
		Student³	
Group³	Student⁴	Student⁰	Battle⁰
		Student⁴	

Figure 30: Alloy groups

this/InvitationStudent	between		status	toGroup
InvitationStudent⁰	Student⁴	Student³	Pending⁰	Group³
InvitationStudent¹	Student⁰	Student³	Pending⁰	Group³
InvitationStudent²	Student⁴	Student⁵	Pending⁰	Group³
InvitationStudent³	Student⁴	Student⁵	Pending⁰	Group³
InvitationStudent⁴	Student⁴	Student⁰	Accepted⁰	Group³
InvitationStudent⁵	Student¹	Student²	Accepted⁰	Group²
InvitationStudent⁶	Student¹	Student³	Accepted⁰	Group²

Figure 31: Alloy student invitations

5.6 Tabular view of the complete model

Here we report a complete model generated by the visualizer using the show predicate in the code.

this/Battle	groupSize	publicGroups	privateGroups	partOf	status
Battle⁰	2	Group⁰	Group³	Tournament¹	Closed⁰
Battle¹	1	Group¹	Group²	Tournament⁰	Registration⁰

this/User	Educator⁰	Student⁰	Educator¹	Student¹	Educator²	Student²	Student³	Student⁴	Student⁵

⁻¹

this/InvitationEducator	between		status	to
InvitationEducator⁰	Educator⁰	Educator²	Rejected⁰	Tournament¹
InvitationEducator¹	Educator¹	Educator⁰	Rejected⁰	Tournament¹
InvitationEducator²	Educator¹	Educator⁰	Accepted⁰	Tournament¹

this/Educator	manages
Educator⁰	Tournament¹
Educator¹	Tournament¹
Educator²	Tournament⁰

Figure 32: Battles, users, invitations educator, educators

this/Student	participates	badges
Student⁰	Tournament⁰	Badge⁰
	Tournament¹	Badge¹
Student¹	Tournament⁰	Badge⁰
	Tournament¹	Badge²
Student²	Tournament⁰	Badge⁰
	Tournament¹	Badge²
Student³	Tournament⁰	Badge⁰
	Tournament¹	Badge¹
Student⁴	Tournament¹	Badge⁰
		Badge²
Student⁵	Tournament⁰	Badge⁰
	Tournament¹	Badge²

this/Tournament	creator	admins	contains	badges
Tournament⁰	Educator²	Educator²	Battle¹	
Tournament¹	Educator¹	Educator⁰	Battle⁰	Badge⁰
		Educator¹		Badge¹
				Badge²

this/Badge	Badge⁰	Badge¹	Badge²

⁻¹

Figure 33: Students, Tournaments, Badges

this/Group	creator	composed	participates
Group⁰	Student¹	Student¹	Battle⁰
		Student²	
Group¹	Student⁵	Student⁰	Battle¹
		Student⁵	
Group²	Student¹	Student¹	Battle¹
		Student²	
		Student³	
Group³	Student⁴	Student⁰	Battle⁰
		Student⁴	

this/Submission	mark	fromTo	
Submission⁰	7	Group³	Battle⁰
Submission¹	6	Group²	Battle¹
Submission²	5	Group¹	Battle¹

this/InvitationStatus	Accepted⁰	Pending⁰	Rejected⁰

-1

this/BattleStatus	Closed⁰	Registration⁰	Running⁰

-1

this/InvitationStudent	between		status	toGroup
InvitationStudent⁰	Student⁴	Student³	Pending⁰	Group³
InvitationStudent¹	Student⁰	Student³	Pending⁰	Group³
InvitationStudent²	Student⁴	Student⁵	Pending⁰	Group³
InvitationStudent³	Student⁴	Student⁵	Pending⁰	Group³
InvitationStudent⁴	Student⁴	Student⁰	Accepted⁰	Group³
InvitationStudent⁵	Student¹	Student²	Accepted⁰	Group²
InvitationStudent⁶	Student¹	Student³	Accepted⁰	Group²

Figure 34: Groups, submissions, invitation Status, Battle status, student invitations

5.7 Graphical view of the complete model

In *Figure* 35 we document the visual interface of the model.

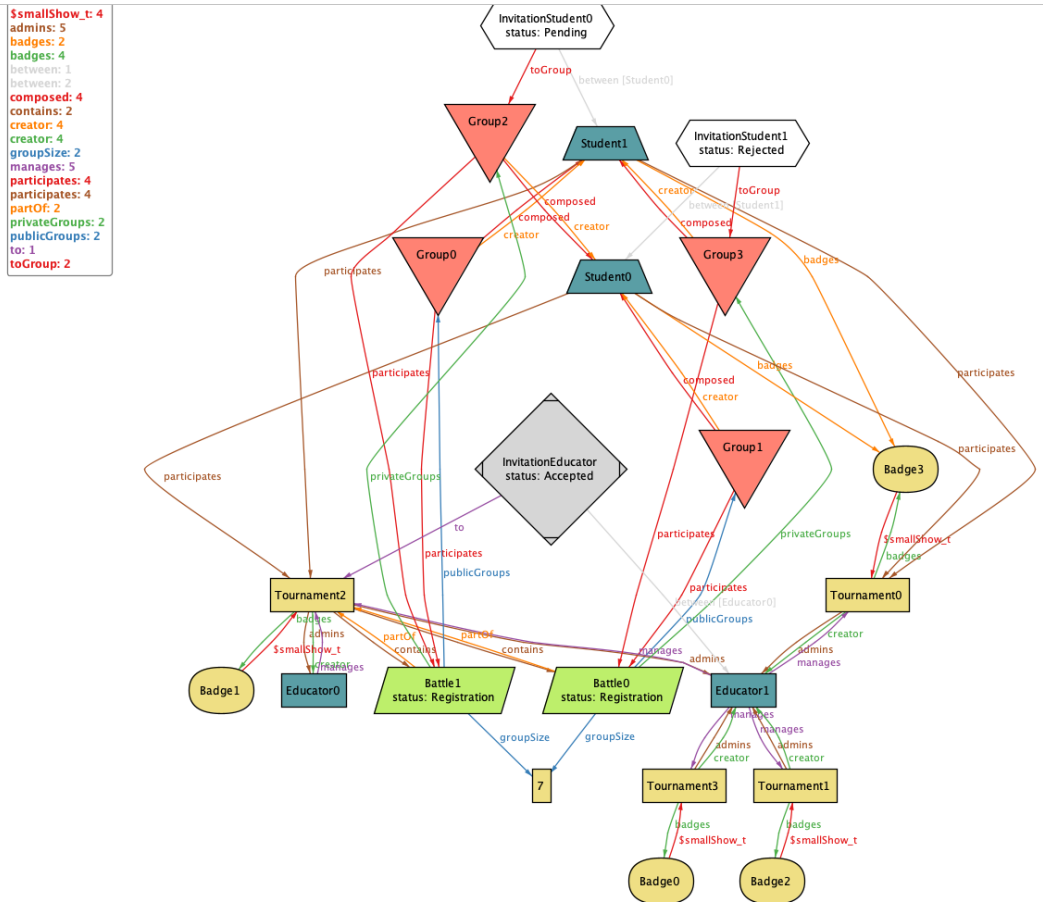


Figure 35: Graphical alloy view

6 Second model with alloy 6 temporal constructs

We decided to implement two alloy models since introducing alloy 6 facts in the first model was difficult to test since the evaluation time was growing exponentially with the number of facts added. In this second model we focused on some aspects of the system that could be modeled with the temporal logic of alloy 6. This included the dynamic growth of the system, the various statuses of the battle, and the transitions between them. Notably, we addressed the restriction that only authorized students can make submissions, and submissions are allowed only during the correct phase.

```
var abstract sig User{}

var sig Educator extends User {}

var sig Student extends User {}

var sig Tournament {
       , var creator :one Educator
       , var contains :set Battle
       , var participants :set Student
}

var sig Battle{
       , var partOf :one Tournament
       , var status :one BattleStatus
```

```
        , var admins :one Educator
        , var participants :set Student
}

enum BattleStatus {Registration, Running, Closed}

var sig Submission {
        , var mark :one Int
        , var student :one Student
        , var battle :one Battle
} {mark >0}


/*-------------------------------------------------------------------------------------*/

assert allBattlesEventuallyEnd {
        all b :Battle |always (eventually b.status =Closed)
}

assert growingUsers {
        always eventually (#User' >#User)
}

assert growingTournaments {
        always eventually (#Tournament' >#Tournament)
}

assert growingBattles {
        always eventually (#Battle' >#Battle)
}

assert allClosedBattlesWereInRegistration {
        all b :Battle |always (b.status =Closed implies once b.status =Registration)
}

assert growingParticipants {
        all b :Battle, t :Tournament |always (eventually #b.participants' >#b.participants
        and #t.participants' >#t.participants)
}



check allBattlesEventuallyEnd for 10 but 10 steps
check growingUsers for 10 but 10 steps
check growingTournaments for 10 but 10 steps
check growingBattles for 10 but 10 steps
check allClosedBattlesWereInRegistration for 10 but 10 steps
check growingParticipants for 10 but 10 steps


/*-------------------------------------------------------------------------------------*/

pred createUser {
        #User' =#User + 1
        User in User'
}

pred createTournament [e :Educator]{
        #Tournament' =#Tournament + 1
        Tournament in Tournament'
        some t :Tournament |t in Tournament' and not t in Tournament and
        t.creator =e
}

pred createBattle [t :Tournament, e :Educator]{
        #Battle' =#Battle + 1
        Battle in Battle'
        some b :Battle |b in Battle' and not b in Battle and
        b.partOf =t and b.admins =e
}

pred closeBattle [b :Battle] {
        b.status =Running
        b.status' =Closed
```

```
}

pred runBattle [b :Battle] {
       b.status =Registration
       b.status' =Running
}

pred joinTournament [s :Student, t :Tournament] {
       s not in t.participants
       s in t.participants'
}

pred joinBattle [s :Student , b :Battle] {
       b.status =Running
       s not in b.participants
       s in b.participants'
}

pred addSubmission [b :Battle, s :Student] {
       s in b.participants
       b.status =Running
       #Submission' =#Submission + 1
       Submission in Submission'
       some sub :Submission |sub in Submission' and not sub in Submission and
       sub.student =s and sub.battle =b
}

/*-----------------------------------------------------------------------------------------*/

//all battles will be closed
fact closeBattle {
       all b :Battle |always (eventually b.closeBattle)
}

//battles are created in Registration status, then are in running status, and then in closed status
fact battleState {
       all b :Battle |
       always (
              (b.status =Closed implies once b.status =Running) and
              (b.status =Running implies once b.status =Registration) and
              (b.status =Closed implies after b.status =Closed)
       )
}

//all battles are close if and only if the predicate closeBattle is true
fact ifClosedWasClosed {
       all b :Battle |always(b.status =Closed iff once closeBattle[b])
}

//all battles are runned if and only if the predicate runBattle is true
fact ifRunningWasRunned {
       all b :Battle |always(b.status =Running iff once runBattle[b])
}

//every moment one of this things can happen
fact systemBehaviour {
       no Tournament
       no Battle
       no User
       no Submission
       always (
              createUser
              or
              (some b :Battle |runBattle[b] or closeBattle[b])
              or
              (some e :Educator |createTournament[e])
              or
              (some t :Tournament |createBattle[t, t.creator])
              or
              (some s :Student, t :Tournament |joinTournament [s, t])
              or
              (some s :Student, b :Battle |joinBattle [s, b])
              or
              (some s :Student, b :Battle |addSubmission [b, s])
```

```
        )
}
```

6.1 Result of the assertions on the model

Figure 36 shows that the alloy evaluator couldn't find counterexamples for the model.

```
Executing "Check allBattlesEventuallyEnd for 10 but 10 steps"
  Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20 Mode=batch
  1..10 steps. 981453 vars. 57610 primary vars. 1356482 clauses. 11642ms.
  No counterexample found. Assertion may be valid. 17ms.

Executing "Check growingUsers for 10 but 10 steps"
  Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20 Mode=batch
  1..10 steps. 1963311 vars. 115120 primary vars. 2714374 clauses. 7642ms.
  No counterexample found. Assertion may be valid. 7ms.

Executing "Check growingTournaments for 10 but 10 steps"
  Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20 Mode=batch
  1..10 steps. 2945169 vars. 172630 primary vars. 4106256 clauses. 7485ms.
  No counterexample found. Assertion may be valid. 18ms.

Executing "Check growingBattles for 10 but 10 steps"
  Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20 Mode=batch
  1..10 steps. 3927027 vars. 230140 primary vars. 5497376 clauses. 7493ms.
  No counterexample found. Assertion may be valid. 7ms.

Executing "Check allClosedBattlesWereInRegistration for 10 but 10 steps"
  Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20 Mode=batch
  1..10 steps. 4910601 vars. 287750 primary vars. 6845129 clauses. 8169ms.
  No counterexample found. Assertion may be valid. 5ms.

Executing "Check growingParticipants for 10 but 10 steps"
  Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20 Mode=batch
  1..10 steps. 5929607 vars. 345460 primary vars. 8367914 clauses. 8198ms.
  No counterexample found. Assertion may be valid. 6ms.

Executing "Run run$7 for 10"
  Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20 Mode=batch
  1..10 steps. 6907583 vars. 402860 primary vars. 9705685 clauses. 7726ms.
  No instance found. Predicate may be inconsistent. 7ms.

7 commands were executed. The results are:
  #1: No counterexample found. allBattlesEventuallyEnd may be valid.
  #2: No counterexample found. growingUsers may be valid.
  #3: No counterexample found. growingTournaments may be valid.
  #4: No counterexample found. growingBattles may be valid.
  #5: No counterexample found. allClosedBattlesWereInRegistration may be valid.
  #6: No counterexample found. growingParticipants may be valid.
```

Figure 36: Results for the second model

7 Effort spent

	Section 1	Section 2	Section 3	Section 4
Federica Maria Laudizi	6h	6h	6h	6:30h
Antonio Marusic	6h	6h	5h	8h
Sara Massarelli	6h	6h	6h	6:30h

Table 23: Work Hours Schedule

8 References

- GitHub REST API documentation
- The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2023/2024;
- Slides of Software Engineering 2 course on WeBeep;
- RASD Sample from A.Y. 2022-2023