SOFTWARE ENGINEERING II

CKB – CodeKataBattle
Requirements Analysis and Specification Document

Version 1.0

Feraboli Alessandro, Filippini Marco, Lucca Simone

December 22, 2023

CONTENTS

1. Introduction 4
1.1 Purpose 4
1.1.2 Goals 4
1.2 Scope 4
1.2.1 World phenomena 4
1.2.2 Shared phenomena 5
1.3 Definitions, Acronyms, Abbreviations 6
1.3.1 Definitions 6
1.3.2 Acronyms 6
1.3.3 Abbreviations 6
1.4 Revision history 6
1.5 Reference documents 6
1.6 Document structure 6
2. Overall Description 7
2.1 Product Perspective 7
2.1.1 Scenario 7
2.1.2 Domain class diagram 9
2.1.3 State diagrams 9
2.2 Product functions 11
2.2.1 Sing Up and Log In 11
2.2.2 Tournament 11
2.2.3 Battle 11
2.2.4 Group creation 12
2.2.5 Gamification Badges 12
2.3 User characteristics 12
2.3.1 Student 12
2.3.2 Educator 12
2.3.3 User 12
2.4 Assumptions, dependencies, and constraints 12
2.4.1 Regulatory policies 12
2.4.2 Domain assumptions 13
3. Specific requirements 14
3.1 External interface 14
3.2 Functional requirements 17
3.2.1 Use cases, Sequence Diagrams, Activity Diagrams 18

3.2.2 Requirements mapping 36
3.3 Performance requirements 39
3.4 Design constraints 39
3.4.1 Standard compliance 39
3.4.2 Hardware limitations 39
3.5 Software system attributes 39
3.5.1 Reliability 39
3.5.2 Availability 39
3.5.3 Security 39
3.5.4 Maintainability 39
3.5.5 Portability 40
4. Alloy 41
4.1 Signatures 41
4.2 Functions 43
4.3 Facts 44
4.4 Model 48
5. Time spent 50
6. References 50

1. Introduction

1.1 Purpose

The aim of the CodeKataBattle (CKB) platform is to facilitate an interactive and educational environment for students, enabling them to refine their software development skills through practical coding exercises. The platform also helps students to improve their collaboration skills, by allowing them to form teams and engage in competitive coding battles.

1.1.2 Goals

The following are the goals that the software aims to achieve:
[G1] Students learn to code in one or more programming languages.
[G2] Students learn to collaborate with peers.
[G3] Students are motivated and engaged in the activity of learning.
[G4] Educators share their programming skills and knowledge.

1.2 Scope

The CodeKataBattle (CKB) project requires the development of a comprehensive platform tailored for students.
Educators need tools to create diverse coding challenges. They will use the system to create battles, that are graded challenges, present inside a tournament, in which groups of students compete. The points of a battle for each group are assigned by the system through factors like correctness, timeliness, and quality of the code through static analysis.
Updates on battle progress, visible only once the code is submitted as a push on the GitHub repository, including scores and rankings, should be implemented.
Additionally, the implementation of gamification elements, such as badges based on achievements and performance, is required to enhance user engagement and motivation.

1.2.1 World phenomena

Phenomena that the machine cannot observe:

[WP1] Students want to code online to improve their skills.
[WP2] Leaders of a group send the link to the GitHub repository to other group members (if any).
[WP3] Students execute a “push” action of their code on GitHub.
[WP4] Leaders of a group perform a “fork” action of the GitHub repository created by an educator for a battle.
[WP5] Students write pieces of code.
[WP6] Students set up the GitHub Actions on the repository.
[WP7] Educators realize the code KATA for a battle.
[WP8] Students clone the GitHub repository.
[WP9] Students communicate with each other.

1.2.2 Shared phenomena

Phenomena controlled by the world and observed by the machine:

[SP1] Students register on the platform.
[SP2] Students log in to the platform.
[SP3] Educators register on the platform.
[SP4] Educators log in to the platform.
[SP5] The system gets a notification through GitHub Actions when a student has pushed his/her code on GitHub.
[SP6] Students invite other students to join the battle.
[SP7] Educator creates a tournament.
[SP8] Students join a tournament.
[SP9] Students join a battle.
[SP10] Educator creates a battle.
[SP11] Tournament leader educator grants other educators the permission to create battles in his/her tournaments.
[SP12] Educators upload to the platform the code KATA.
[SP13] Educators set a minimum and maximum number of students per group for a single battle.
[SP14] Educators set a registration deadline for a tournament/battle.
[SP15] Educators set a final submission deadline for a battle.
[SP16] Educators assign an extra personal score to each group/student after the final submission of the battle.
[SP17] Educators close a tournament.
[SP18] Educators select specific aspects for automated code evaluation.
[SP19] A student joins a group with other students.
[SP20] Educators create one or more gamification badges.
[SP21] Educators choose one or more gamification badges for the tournament.

Phenomena controlled by the machine and observed by the world:

[SP22] CKB sends notification to the students signed up to the tournament about the creation of a new battle in the tournament itself.
[SP23] CKB shows the score of the battle for each group every time they push the code on GitHub.
[SP24] CKB shows a final score to each student after the final submission.
[SP25] CKB shows the personal tournament score of each student, that is, the sum of all battle scores received in that tournament.
[SP26] CKB sends gamification badges to the students.
[SP27] CKB sends the link of the repository to all the students (who are leader of a group) subscribed at the tournament.
[SP28] CKB sends notifications to the students signed up to the platform about the creation of a new tournament.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Code KATA: A kata, or code kata, is defined as an exercise in programming which helps hone your skills through practice and repetition

1.3.2 Acronyms

- CKB: Code Kata Battle
- RASD: Requirement Analysis and Specification Document

1.3.3 Abbreviations

- [Gn] → the n-th goal
- [WPn] → the n-th world phenomenon
- [SPn] → the n-th shared phenomenon
- [Dn] → the n-th domain assumption
- [Rn] → the n-th requirement
- [UCn] → the n-th use case

1.4 Revision history

- Version 1.0 (12/22/2023)

1.5 Reference documents

- Specification of the RASD assignment of the Software Engineering II course, held by professors Matteo Camilli, Elisabetta Di Nitto and Matteo Rossi at Politecnico di Milano, A.Y. 2023/2024.
- Slides of the Software Engineering II course, held by professors Matteo Camilli, Elisabetta Di Nitto and Matteo Rossi at Politecnico di Milano, A.Y. 2023/2024.

1.6 Document structure

Here it is the document structure:

1. **Introduction** → Description of the reasons and the goals that are going to be achieved with the project development.
2. **Overall Description** → High-level description of the system focusing on shared phenomena and the domain level.
3. **Specific Requirement** → Detailed description of the requirements and domain assumptions needed to fulfill the goals.
4. **Formal Analysis** → Formal description of world phenomena main aspects using Alloy.
5. **Effort Spent** → Time spent on this document (per person and per section).
6. **References** → References to any document or software used to realize this document.

2. Overall Description

2.1 Product Perspective

2.1.1 Scenario

1. **Educator creates a tournament:**
   Tom is a teacher that wants to test his student's knowledge about programming in the C programming language. To solve this problem, he opens CKB software, on which he is logged in with an educator account.
   He begins the creation processes by performing these actions:
   - Add Ben (his collaborator) as an educator of that tournament.
   - Select 24h as the deadline for joining the tournament.
   - Add three pre-created badges.
   - Add the description "Amatorial tournament for C programming."
   - Confirm his choices.

2. **Student joins a tournament:**
   Harry, a novice in the programming field, wants to improve his capabilities in programming. Luckily, he just received a notification from the CKB system that indicates that a new tournament has just been created. He opens the application, on which he is already logged in with a student account, sees the tournament in the dedicated section and decides to join it.

3. **Educator creates a battle:**
   Charles, an Educator who is already in a tournament, now wants to create a battle. He opens CKB, where he is already logged in and enters the tournament page he previously created. To create a battle, he needs to do these actions:
   - Name the battle as "Divisors of a Number".
   - Insert "given an integer, write a function that returns an array containing all the divisors of that number" as the description of the project.
   - Insert test cases and build automation scripts.
   - Set the as [1, 5] the range of students per group.
   - Set 24h as the final registration deadline, and 24h as final submission deadline, counted starting from the beginning of the battle.
   - Select security and reliability as aspects for automated evaluation.
   - Confirm the choices.

4. **Educator creates a badge:**
   Mary, an educator, has a dedicated section for creating new badges for the tournaments she will create. She wants to create a badge with the title "Verbose". This badge is assigned to the student that will have pushed the most lines of code in a tournament and will have pushed at least 1000 lines.
   She writes in the title section "Verbose" and selects in the menu the Boolean attributes that must be verified, by adding "1000" to the "number of lines" predicate. After creation, she will see the created badge in her library and will be able to add it to the tournament she will create in the future.

5. **Student joins a battle:**
   John, a student who is subscribed to a tournament called “C programming language”, receives a notification that a new battle for that tournament has been created.
   He opens the notification to join the battle. He enters a “room” in which he is the leader. John read that the battle requires groups composed of at least two students. John must search for a teammate. He sees a list of all the players that are connected for that battle and decides to invite Marco and Anna. John sees that Anna joined the group.
   After the registration deadline expires, John receives the link to the GitHub repository that has been created by the platform and which includes the code kata and proceeds to send it to Anna. At this point, John and his group start to work on the project.

6. **Educator manually evaluates a student:**
   After the submission deadline of a tournament has expired, Charles, an educator, decides to manually evaluate the performance of Emily, one of the students who has participated in the tournament. Thanks to the CKB platform, where he is logged in, he accesses all the code produced by Emily within the context of the tournament. After reviewing it, he enters in the system the additional score he wants to give her and confirms his evaluation.

7. **Tournament is closed by an educator:**
   Victoria, an educator who has previously created a tournament decides to close it. There are still two battles to be concluded, so before the final scores are ready, she must wait until all battles are concluded. At that moment, Victoria closes the tournament and all the students involved in the concluded tournament receive a notification by the CKB platform with their own score.

2.1.2 Domain class diagram

The following diagram shows the relationship between elements without getting into the specific detail of each one. It was made to give a generic overview of the system and the world that interacts with it.

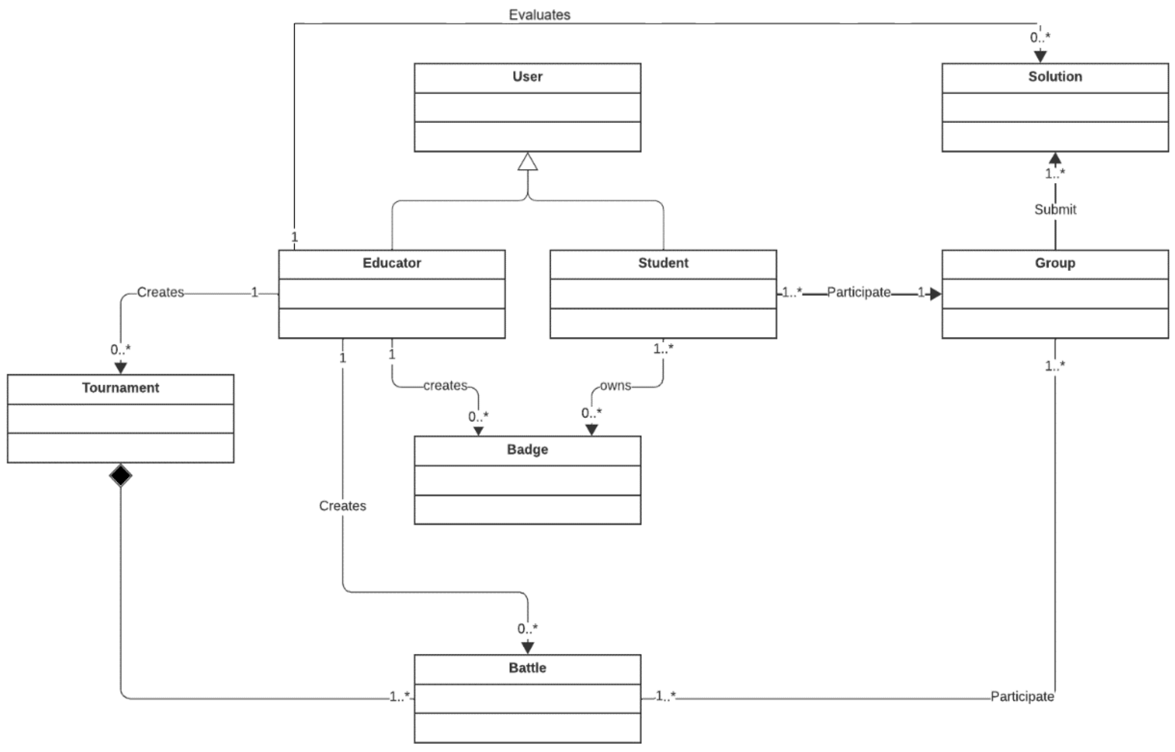


Figure 2.1

2.1.3 State diagrams

Tournament

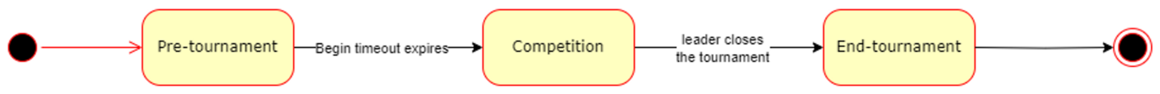


Figure 2.2

Battle

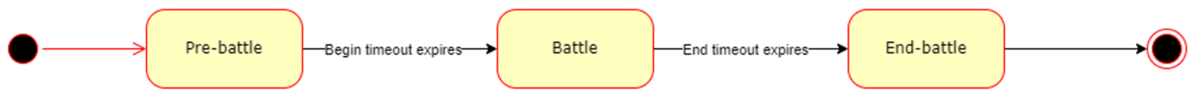


Figure 2.3

Student

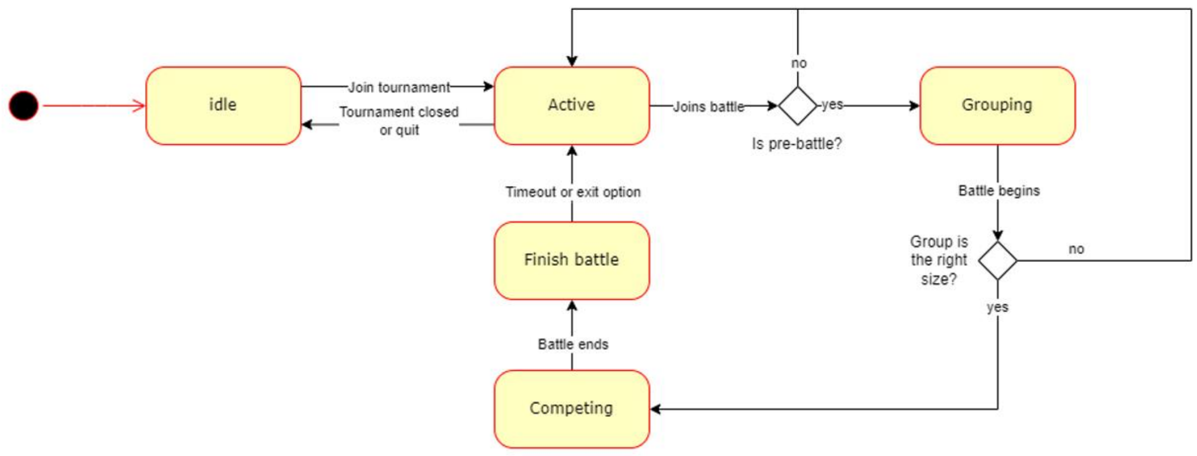


Figure 2.4

Educator

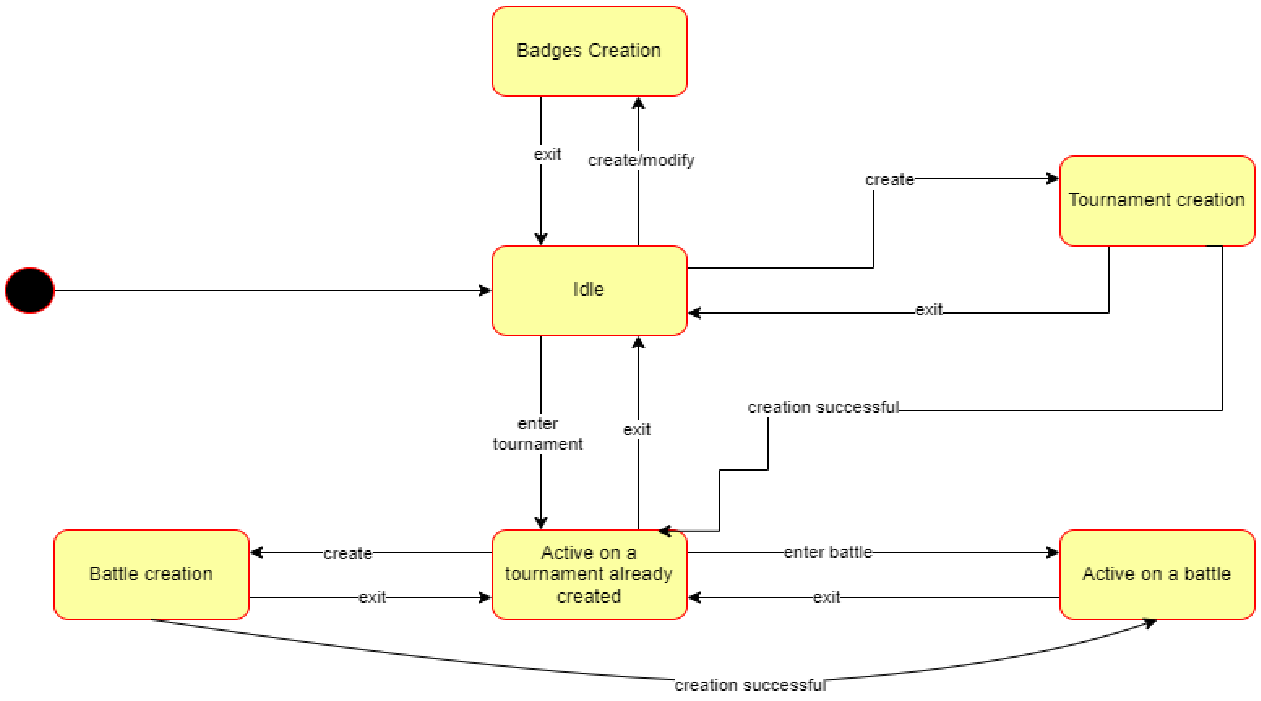


Figure 2.5

2.2 Product functions

2.2.1 Sing Up and Log In

The registration process enables users to create an account within the system. Subsequently, a confirmation email will be sent to the user for verification purposes. After the user confirms their email, the system will allow him/her to enter the application.
The login process uses the data given in the registration process to identify the user in a secure manner.

2.2.2 Tournament

An educator that creates a tournament becomes the tournament leader.
The tournament ranking and badges must be visible to every user that accesses the platform, even if the user is not enrolled in the specified tournament.
The points inside a tournament of each student are the sum of the points gained in every battle enrolled by the student. The winner of the tournament is the student with the highest score, in case of a draw the system chooses as the winner the student that got the highest score in a single battle. If everything is equal, the winner is chosen randomly.
A tournament is closed when the tournament leader decides to. So, the system will wait until all the ongoing battles (if any) will be concluded, then updates the tournament score of each student and declares the winners of the battle.

2.2.3 Battle

Battles cannot overlap with each other within the context of the same tournament. The battle is created by a single educator that specifies the minimum and maximum number of students allowed per group.
The system will send a notification to every user that is enrolled in that tournament.
A student that is participating in a tournament can freely enter a battle only if it is not yet started and if he/she is not participating in another battle.
The battle starts when the timer, set upon creation, expires.
The system creates a GitHub repository for the specific battle (the profile hosting the repository has been previously created for this purpose and can be used only by the application itself). This repo must be cloned by each group leader, that will set up a GitHub actions workflow to notify the CKB application if any “push” is done on the group repository. To ease this process, the system will send the user the instructions for the matter. The leaders will then share the forked repository link with their group.
Each group will develop their solution locally, outside the CKB application, and push their solution on the group repository. The score is calculated every time a “push” on the main branch is detected, but it will only be visible by the members of the group. A battle terminates when the timeout, set upon creation, expires.
The rankings are shown to each student. If the points are equal, the winner will be the group with less students. If everything is equal, the winner is chosen randomly.

2.2.4 Group creation

A group is created only for a single battle, and it exists only in the context of that battle. Students must be able to invite other students that joined a specific battle, but only if it has not started yet. When a student joins a battle, a group of one person is created and the single component is the leader.
When a student is invited in a group, he/she can decide to accept or decline the invitation. The student that accepts the invitation can enter the group only if it is not full, losing the privileges of leader. In case the battle begins before a decision is made, the system supposes that the invitation was declined.
A student can participate in a battle only if he/she is in a group and if it does not satisfy the number of students needed, all its members are expelled from the battle.

2.2.5 Gamification Badges

All gamification badges of a tournament need to be generated before the tournament has been created. To do so, an educator needs to access the dedicated section of CKB platform, choose a name and a predicate to be satisfied to achieve it. Once a badge has been collected from a student, every other student who looks at his profile page will be able to see that badge, even if the tournament in which it was achieved is concluded.

2.3 User characteristics

2.3.1 Student

A person who wants to participate actively in code battles and tournaments as code maker.

2.3.2 Educator

User that has the aim to organize and handle tournaments and battle.

2.3.3 User

Student or educator.

2.4 Assumptions, dependencies, and constraints

2.4.1 Regulatory policies

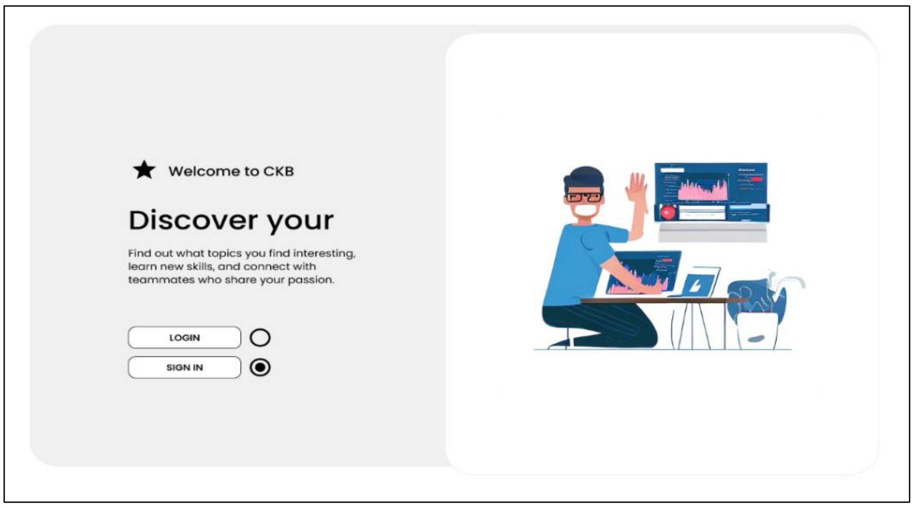
The CKB application will ask for user personal information like email address and a nickname. None of this will be used for commercial purposes. Personal information will be processed in compliance with the GDPR (General Data Protection Regulation).
Moreover, the application needs access to the repository forked by each group of students for every battle in order to pull every new release and evaluate any piece of code.

2.4.2 Domain assumptions

[D1] Students know the basics of git and have it installed on their machine.
[D2] Leader of a group, when requested, will follow the instructions to fork the repository, set up the GitHub Actions and send the link to the repository to the other group members (if any).
[D3] Students who participate in a battle have installed on their machine an IDE or a text editor.
[D4] Students and educators have an internet connection.
[D5] Students communicate with each other through external channels.

3. Specific requirements

3.1 External interface



Welcome to CKB
Discover your
Find out what topics you find interesting, learn new skills, and connect with teammates who share your passion.
LOGIN
SIGN IN

Figure 3.1

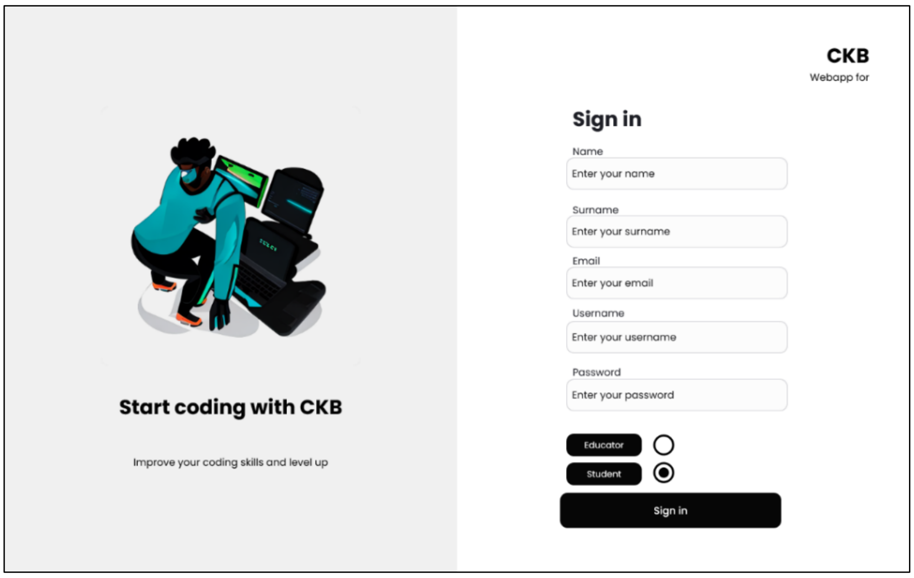


Figure 3.2

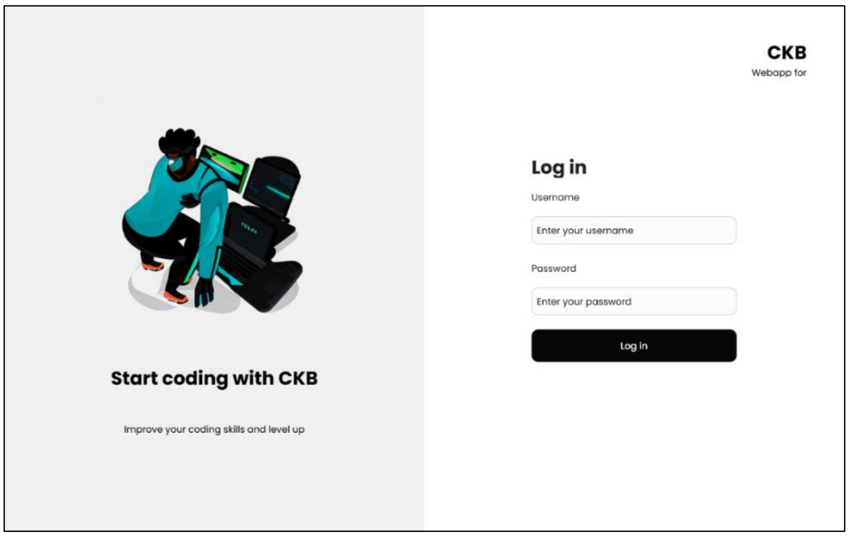


Figure 3.3

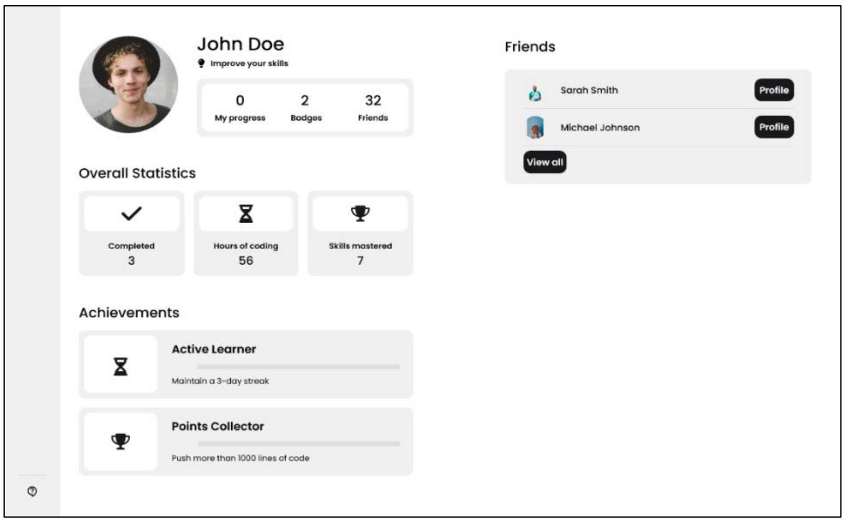


Figure 3.4

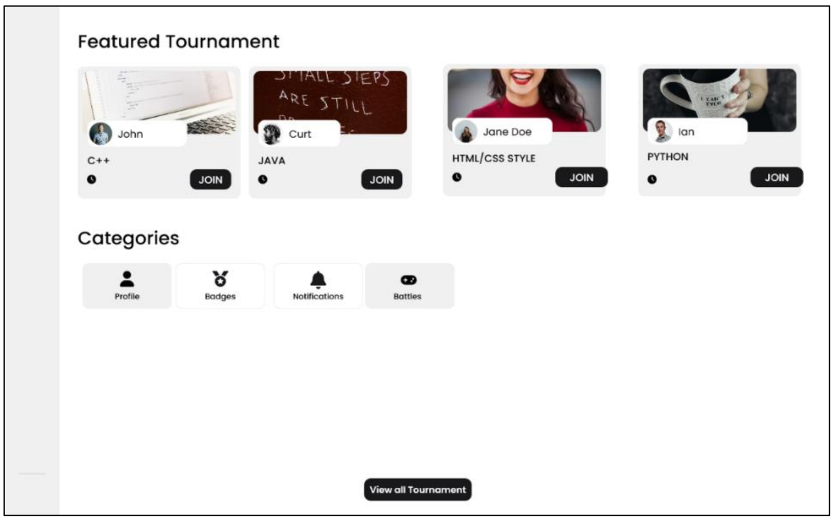


Figure 3.5

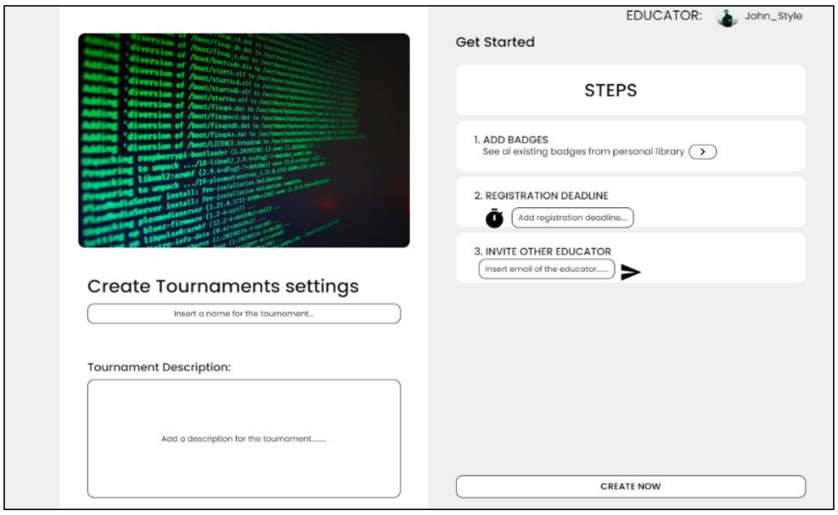

EDUCATOR: John_Style
Get Started
STEPS
1. ADD BADGES
See al existing badges from personal library
2. REGISTRATION DEADLINE
Add registration deadline...
3. INVITE OTHER EDUCATOR
Insert email of the educator......
Create Tournaments settings
Insert a name for the tournament...
Tournament Description:
Add a description for the tournament.......
CREATE NOW

Figure 3.6

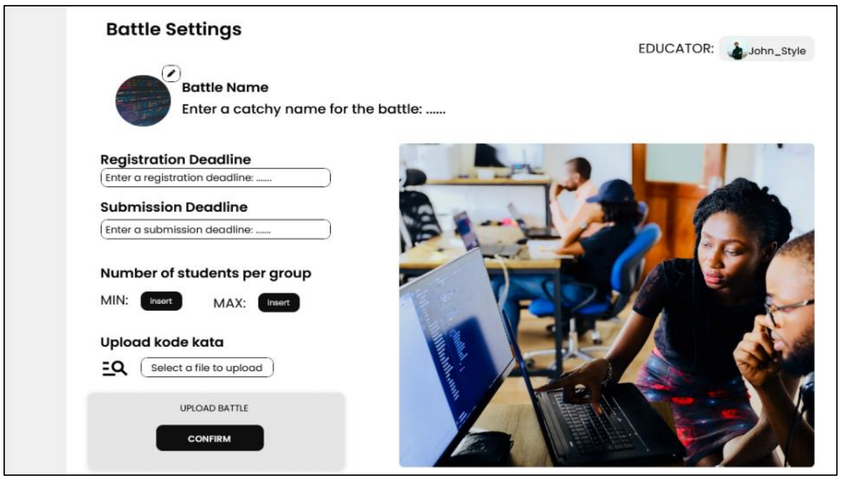
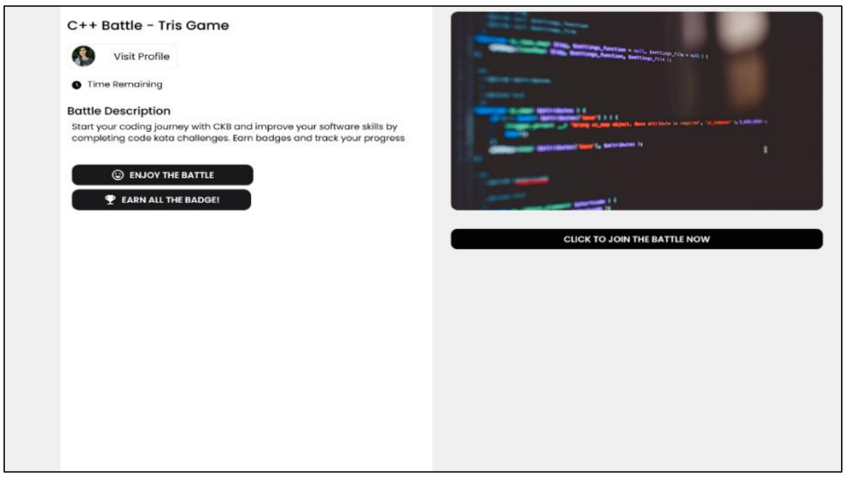


Figure 3.7



Start your coding journey with CKB and improve your software skills by completing code kata challenges. Earn badges and track your progress

Figure 3.8

3.2 Functional requirements

[R1] The system allows users to sign-up and log-in.
[R2] The system sends to users an email containing the link to confirm the registration of the account.
[R3] The system allows each educator to create a tournament at any moment.
[R4] The system updates, upon the end of a battle, the tournament score of the students who have participated in it.
[R5] The system allows the tournament leader to add pre-existing gamification badges upon creation of the tournament.
[R6] The system allows the tournament leader to add other educators to the tournament at any point in time.
[R7] The system allows the tournament leader to close a hosted tournament.
[R8] The system allows an educator to create a battle within the context of a tournament.
[R9] The system allows students to join a battle only if it has not started yet.
[R10] The system requires students to enter a group of an acceptable size before the battle starts, otherwise they will be expelled when the battle begins.
[R11] The battle starts when the timer set upon creation expires.
[R12] The system creates a GitHub repository for the battle.
[R13] The system pushes in the appropriate GitHub repository the code KATA previously uploaded by the educator.
[R14] The system sends set up instructions to group leaders.
[R15] The system is notified about new pushes by the GitHub Action set up by the group leader.
[R16] The system evaluates the pushed solution giving a temporary score after each push.
[R17] The ranking is shown at the end of the battle.
[R18] The system allows group leaders to invite peers in the group if the battle has not started yet. Students can be invited even if they are participating in another group, but they need to choose only one.
[R19] The system stops students from entering groups that have reached the maximum number of participants.
[R20] The system allows educators to create gamification badges which can be used only in possible future tournaments.
[R21] Tournament leaders can add zero or more preexisting gamification badges to the tournament only during creation.
[R22] The system disjoins a group if its leader quits the battle in the registration phase.
[R23] The system allows educators to select aspects for the automated evaluation.
[R24] The system registers as the score obtained in a battle the last score calculated (which corresponds to the score of the last push on the repo).

3.2.1 Use cases, Sequence Diagrams, Activity Diagrams

User:

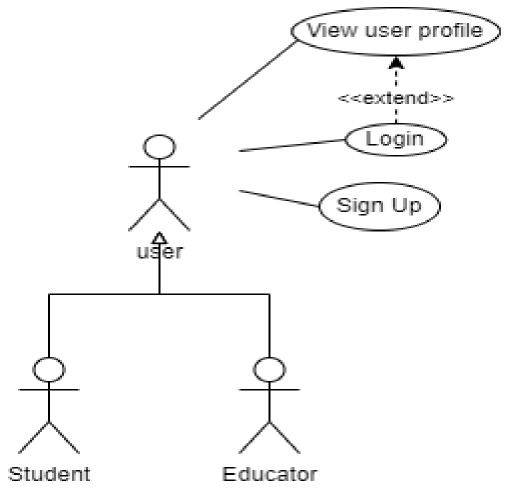


Figure 3.9

[UC1] - Login

Name	Login
Actor	User
Entry condition	True
Event flow	1. User goes to login page 2. User inserts personal email 3. User inserts password 4. User clicks on "confirm" button 5. The system checks the credentials 6. The user is allowed to enter the system
Exit condition	The user can access to the functionalities provided by the type of the possessed account
Exceptions	Credentials do not correct, try again

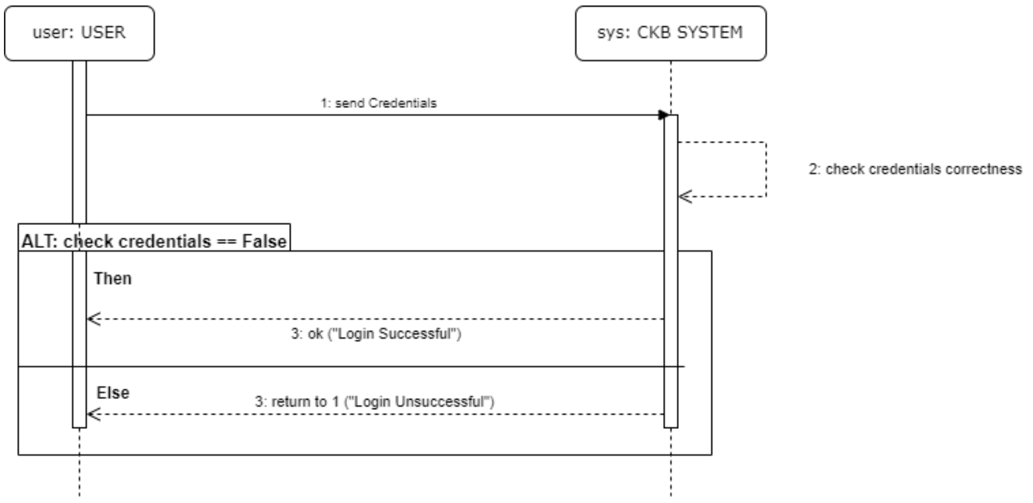

user: USER
sys: CKB SYSTEM
1: send Credentials
2: check credentials correctness
ALT: check credentials == False
Then
3: ok ("Login Successful")
Else
3: return to 1 ("Login Unsuccessful")

Figure 3.10

[UC2] – Sign up

Name	Sign Up
Actor	User
Entry condition	True
Event flow	1. User goes to sign up page 2. User inserts personal email 3. User inserts password 4. User inserts type of account (student or educator) 5. User clicks on "confirm" button 6. The system checks if the account already exists 7. The system stores the credentials 8. The system sends an email to the user to verify the account 9. The user clicks on the link provided in the email 10. The user is allowed to enter the system
Exit condition	The user can access to the functionalities provided by the type of the possessed account
Exceptions	The account already exists, return to the entry condition

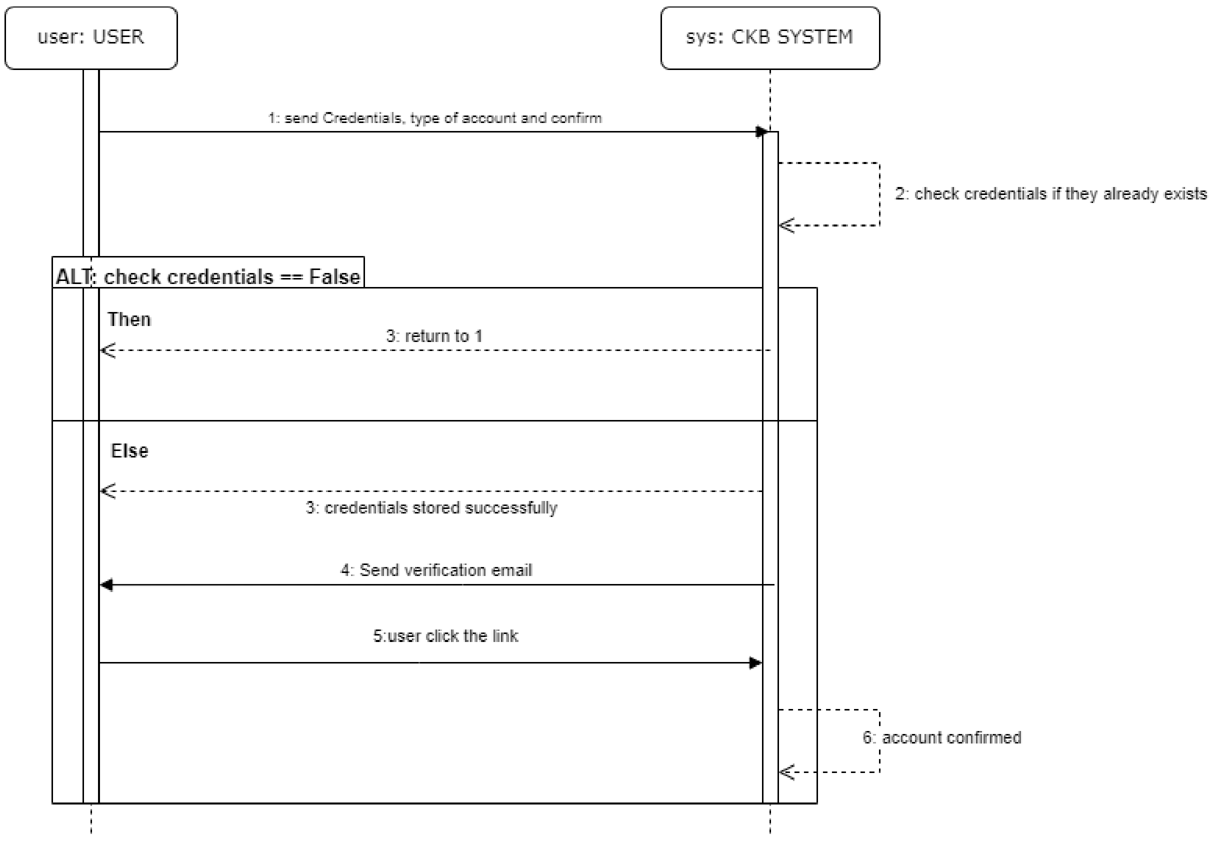

user: USER
sys: CKB SYSTEM
1: send Credentials, type of account and confirm
2: check credentials if they already exists
ALT: check credentials == False
Then
3: return to 1
Else
3: credentials stored successfully
4: Send verification email
5:user click the link
6: account confirmed

Figure 3.11

[UC3] – View user profile

Name	View user profile
Actor	User
Entry condition	User is logged in
Event flow	1. User goes to the profile searching section 2. User enters the name to be searched 3. The system checks the existence of a profile with the given name 4. The system displays the profile information
Exit condition	User can see all the tournaments, battles, and badges in which the searched profile has been involved
Exceptions	Entered name does not exist, try again

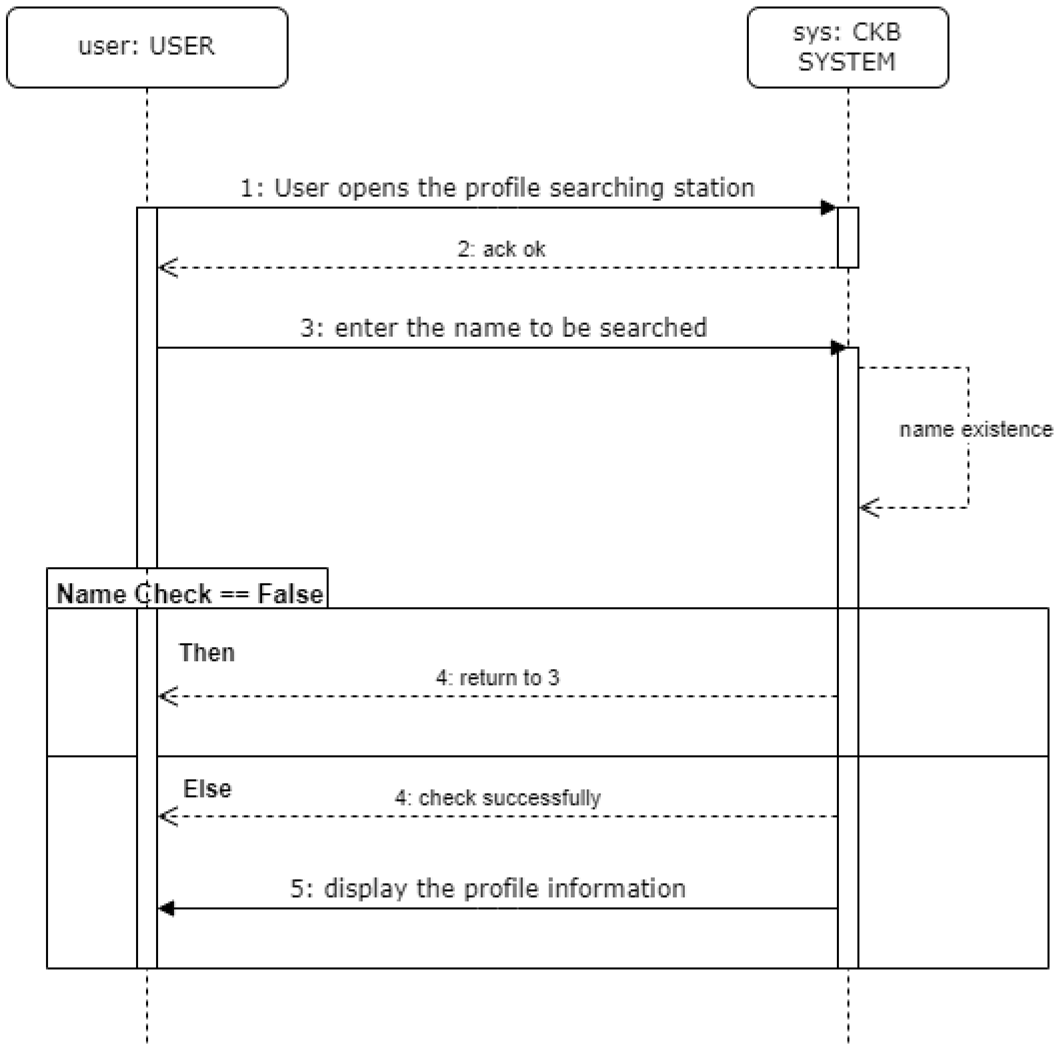


Figure 3.12

Educator:

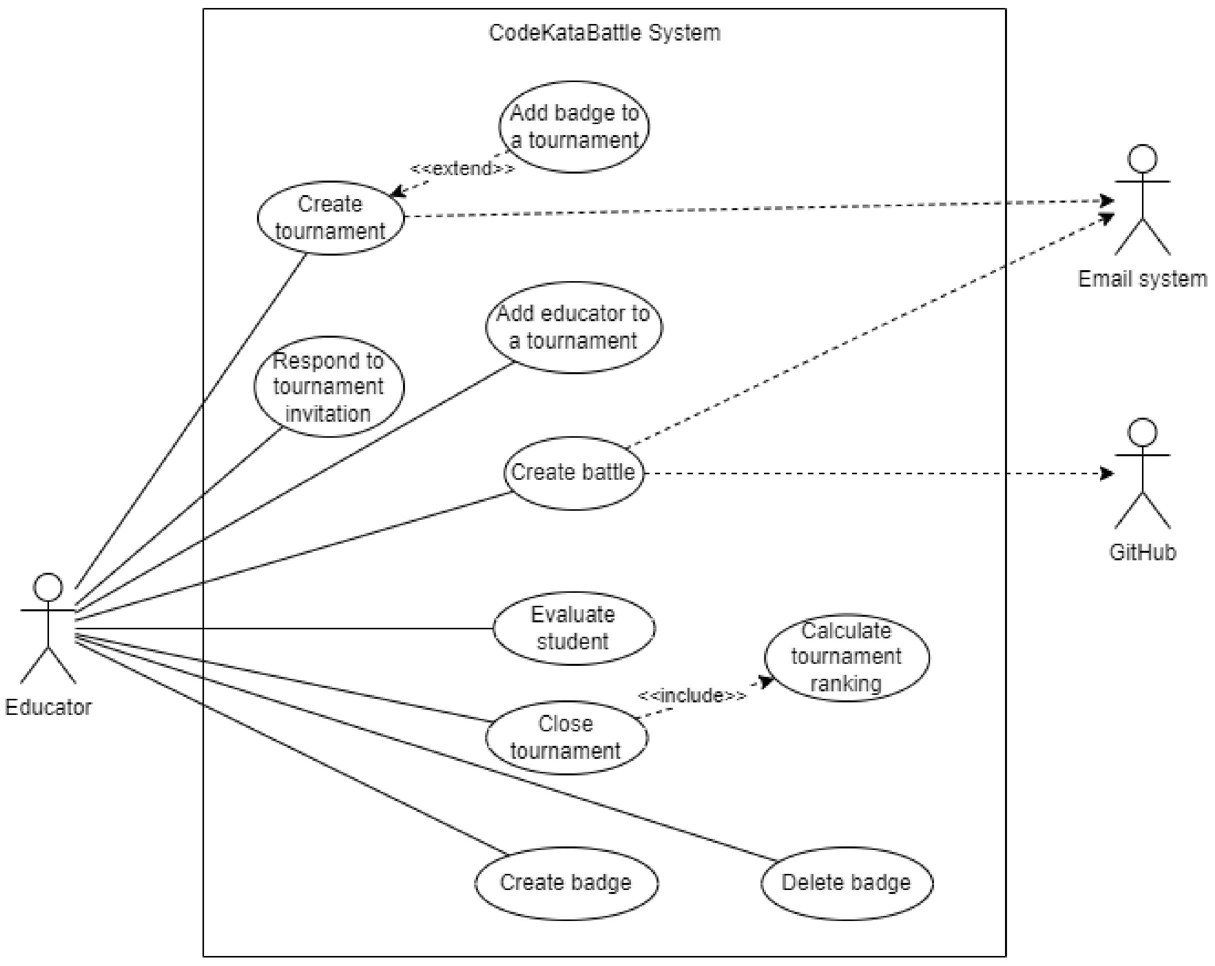


Figure 3.13

[UC4] – Create tournament

Name	Create tournament
Actor	Educator
Entry condition	Educator is logged in the system
Event flow	1. Educator clicks on "new tournament" button 2. Educator inserts a tournament name 3. Educator inserts a description (optional) 4. Educator chooses badges from the personal library 5. Educator chooses registration deadline 6. Educator clicks on "confirm" button 7. The system notifies every registered student of the new tournament 8. The system identifies the creator as the tournament leader
Exit condition	The tournament has been successfully created

Exceptions	The tournament name is already in use by a running tournament, go back to the entry condition

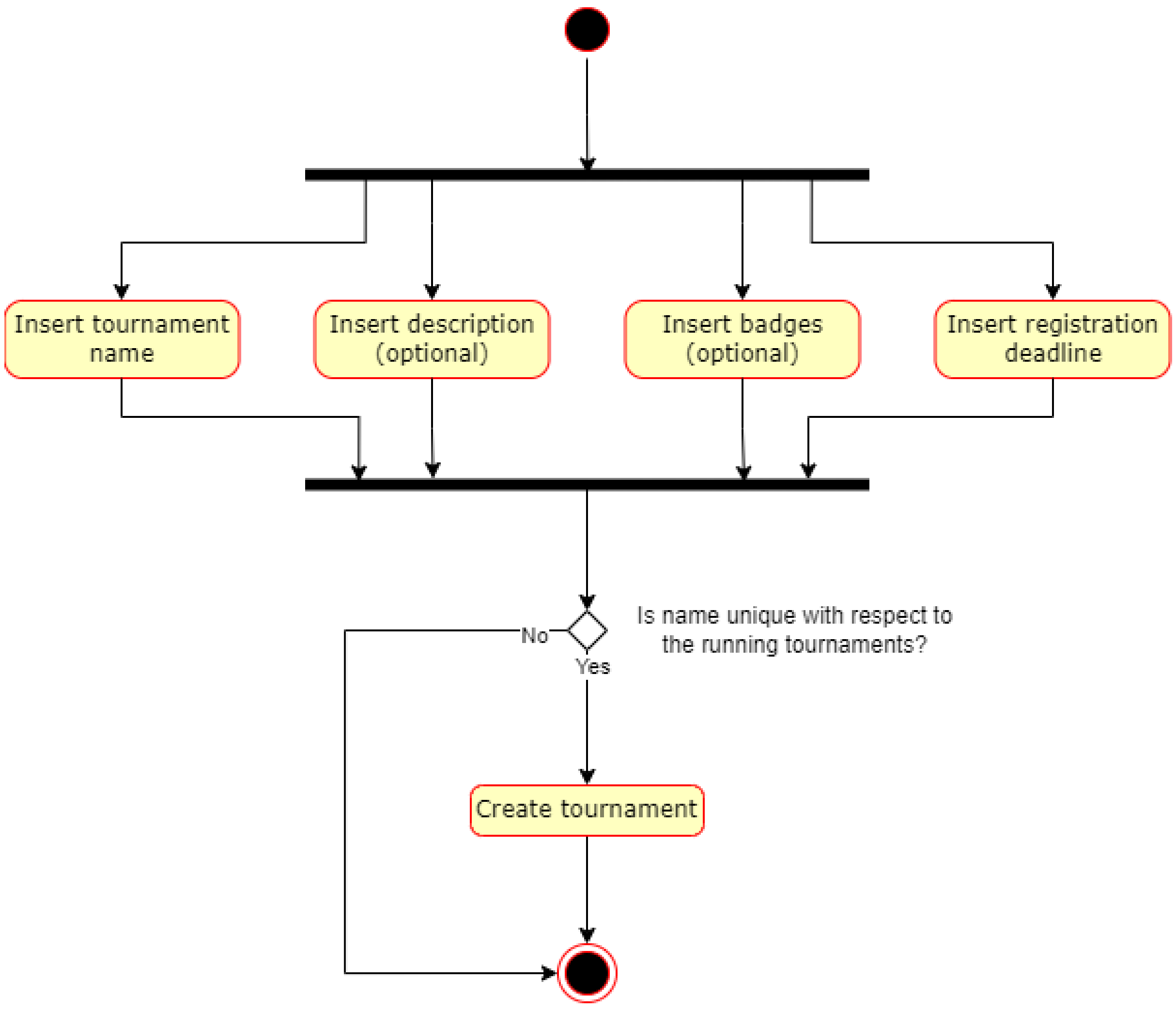
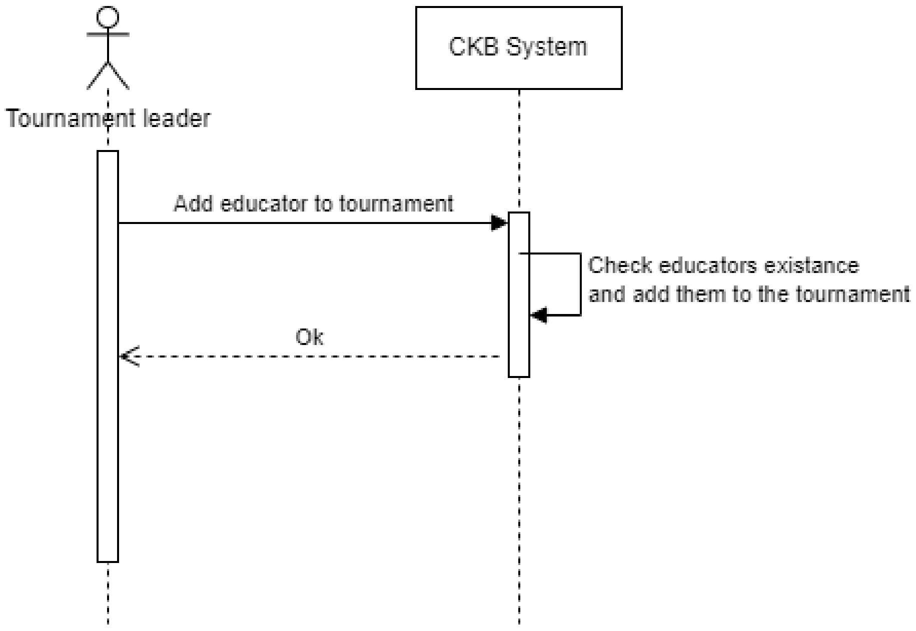

Insert tournament name
Insert description (optional)
Insert badges (optional)
Insert registration deadline
Is name unique with respect to the running tournaments?
No
Yes
Create tournament

Figure 3.14

[UC5] – Add educator to a tournament

Name	Add educator to a tournament
Actor	Educator
Entry condition	The educator is logged in and is the tournament leader
Event flow	1. Educator goes to the tournament section 2. Educator clicks on "add educator" button 3. Educator enters the name of the educator(s) he/she wants to add 4. The system adds the educators in the tournament
Exit condition	Other educators have been added to the tournament

Exceptions	There is no educator account with the given name, go back to entry condition



Tournament leader
CKB System
Add educator to tournament
Check educators existance and add them to the tournament
Ok

Figure 3.15

[UC6] – Create battle

Name	Create battle
Actor	Educator
Entry condition	The educator is logged in and has access to the tournament
Event flow	1. Educator goes to the tournament section 2. Educator clicks on “new battle” button 3. Educator inserts the battle name, description (optional) 4. Educator chooses the registration deadline 5. Educator chooses the submission deadline 6. Educator uploads the code KATA 7. Educator enters minimum and maximum number of students allowed per group 8. Educator selects the features that have to be relevant in the evaluation through code static analysis 9. Educator clicks on “confirm” button 10. The system creates a GitHub repository 11. The systems notify students that are subscribed to the tournament about the new battle
Exit condition	The battle has been successfully created

Exceptions	1. The battle name is unique between the ones previously disputed in the same tournament 2. The battle overlaps with other ones already scheduled within the tournament, go back to entry condition

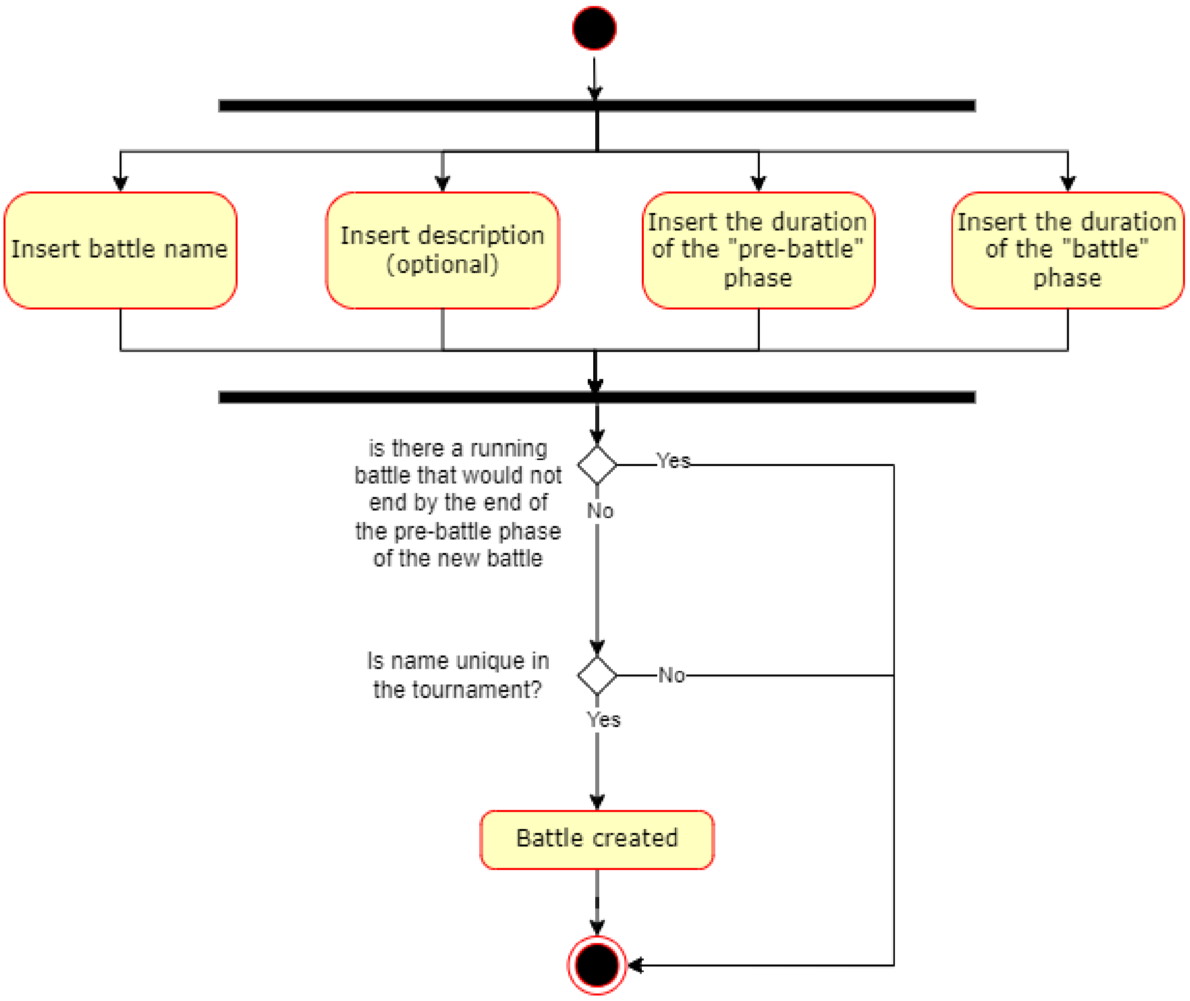

Insert battle name
Insert description (optional)
Insert the duration of the "pre-battle" phase
Insert the duration of the "battle" phase
is there a running battle that would not end by the end of the pre-battle phase of the new battle
Yes
No
Is name unique in the tournament?
No
Yes
Battle created

Figure 3.16

Some examples are provided to clarify the meaning of the condition:

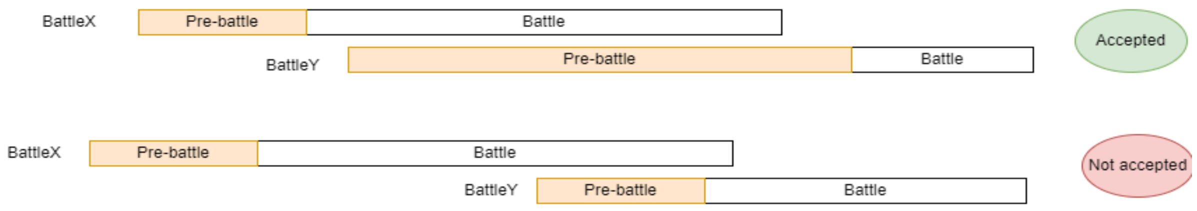


Figure 3.17

[UC7] – Evaluate group

Name	Evaluate group
Actor	Educator
Entry condition	• The battle has ended • Creator of that battle has decided to manually evaluate groups
Event flow	1. Educator goes to the tournament section 2. Educator goes to the battle section 3. Educator selects the group to be evaluated 4. Educator accesses all the material realized by the group during the battle 5. Educator assigns a score from 0 to 100 which will be averaged with the previous grade
Exit condition	The group has been manually evaluated
Exceptions	

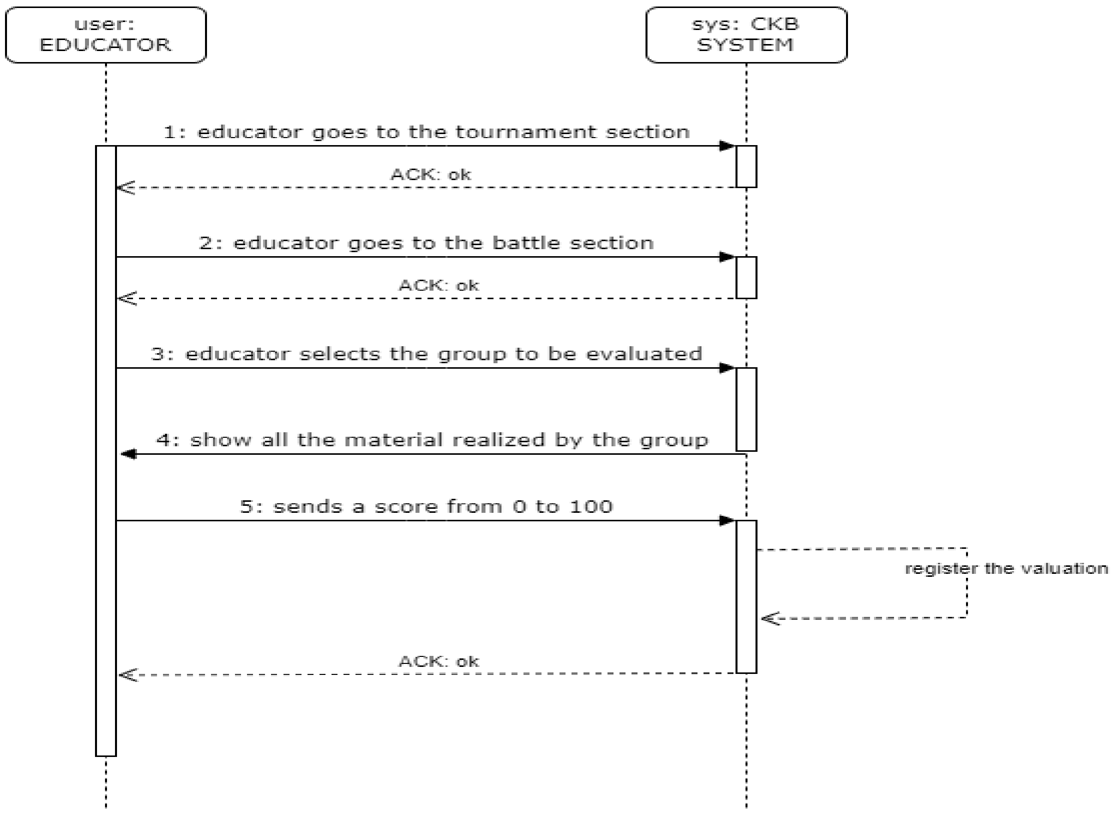

user: EDUCATOR
sys: CKB SYSTEM
1: educator goes to the tournament section
ACK: ok
2: educator goes to the battle section
ACK: ok
3: educator selects the group to be evaluated
4: show all the material realized by the group
5: sends a score from 0 to 100
register the valuation
ACK: ok

Figure 3.18

[UC8] – Close a tournament

Name	Close a tournament
Actor	Educator
Entry condition	Educator is logged in and he/she is the creator of the tournament
Event flow	1. Educator goes to the tournament section 2. Educator decides if there is the need of manually evaluate the solutions of every group 3. Educator clicks on “close tournament” battle 4. System calculates the rankings and present them to the students
Exit condition	No educator involved in the tournament can create other battle within it. Any ongoing or already scheduled battle will be held until its end
Exceptions	

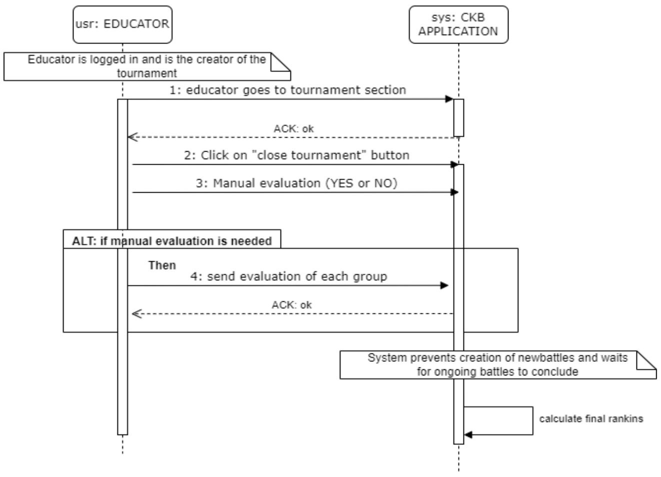

usr: EDUCATOR
sys: CKB APPLICATION
Educator is logged in and is the creator of the tournament
1: educator goes to tournament section
ACK: ok
2: Click on "close tournament" button
3: Manual evaluation (YES or NO)
ALT: if manual evaluation is needed
Then
4: send evaluation of each group
ACK: ok
System prevents creation of newbattles and waits for ongoing battles to conclude
calculate final rankins

Figure 3.19

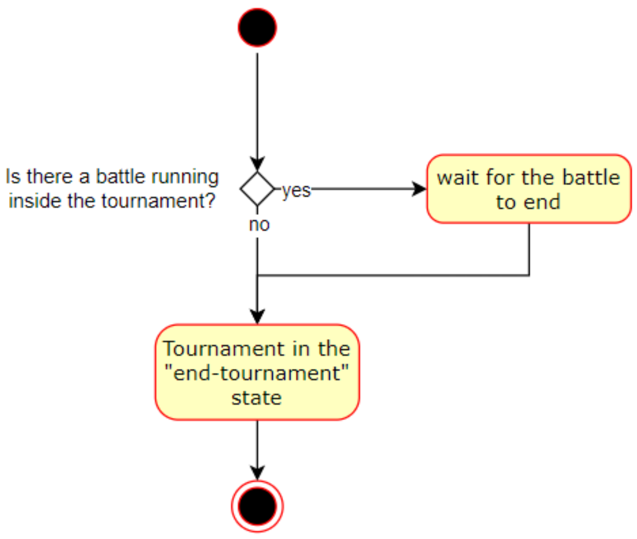

Is there a battle running inside the tournament?
yes
no
wait for the battle to end
Tournament in the "end-tournament" state

Figure 3.20

[UC9] – Calculate tournament ranking

Name	Calculate tournament ranking
Actor	CKB system
Entry condition	• Tournament closed • Points have been assigned to each group
Event flow	1. The system elaborates the group points and builds a ranking 2. Show the ranking to every user 3. Assign badges to the students
Exit condition	
Exceptions	

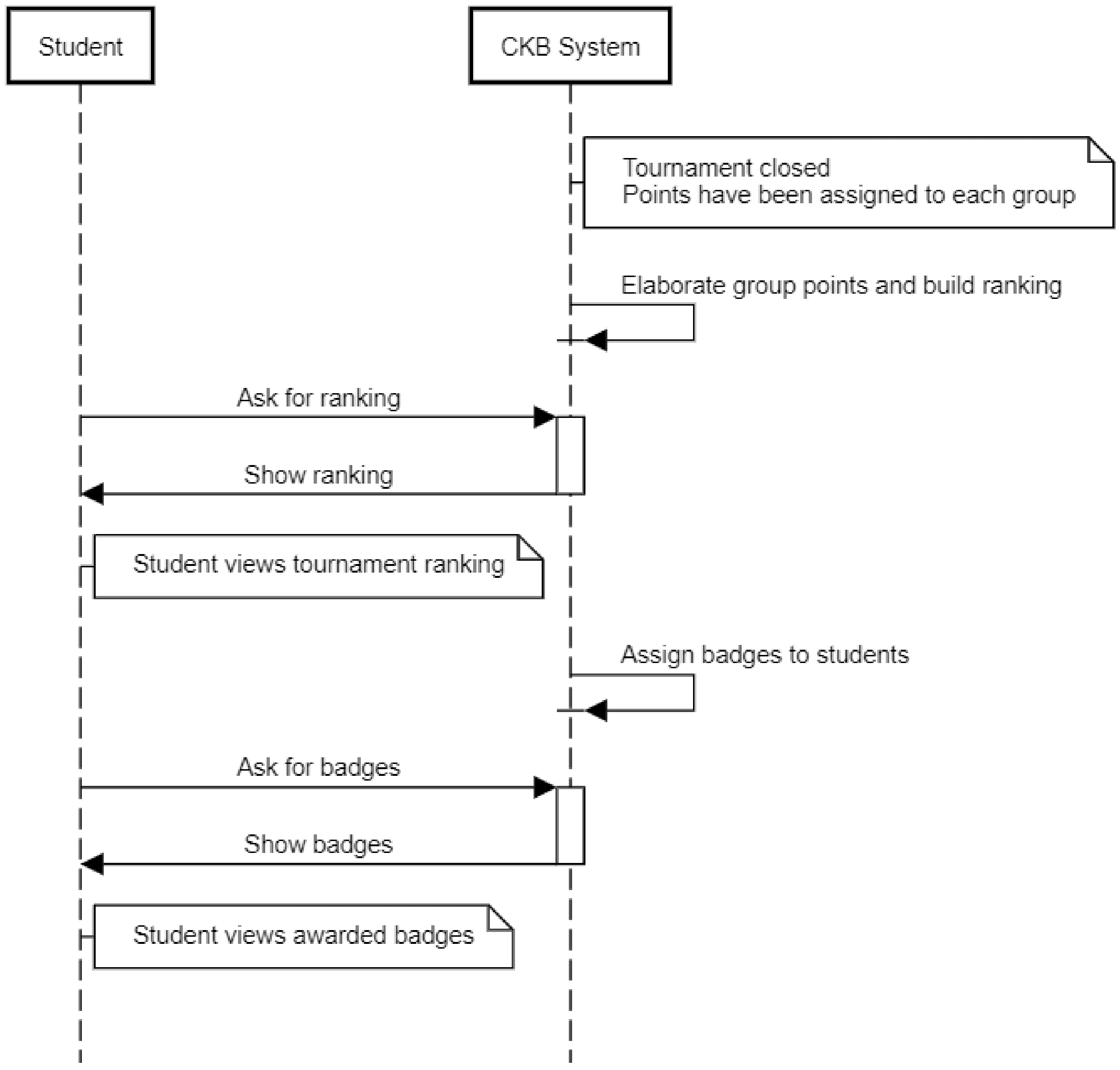

Student
CKB System
Tournament closed
Points have been assigned to each group
Elaborate group points and build ranking
Ask for ranking
Show ranking
Student views tournament ranking
Assign badges to students
Ask for badges
Show badges
Student views awarded badges

Figure 3.21

[UC10] – Create badge

Name	Create badge
Actor	Educator
Entry condition	Educator is logged in
Event flow	1. Educator goes to badges section 2. Educator clicks on “new badge” button 3. Educator enters a badge title 4. Educator selects one or more Boolean attributes that have to be verified in order to achieve the badge 5. Educator clicks on “confirm” button
Exit condition	The badge has been successfully created
Exceptions	The name is already in use by one of the badges created by the same educator, go back to entry condition

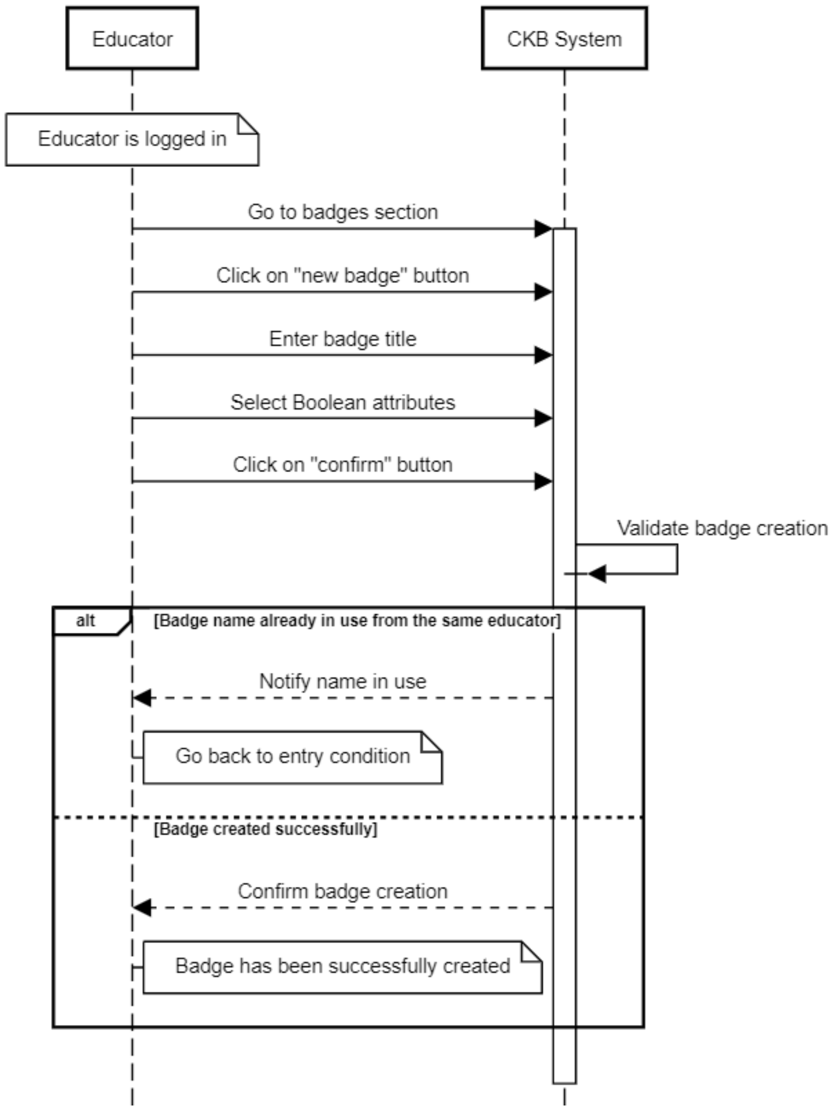

Educator
CKB System
Educator is logged in
Go to badges section
Click on "new badge" button
Enter badge title
Select Boolean attributes
Click on "confirm" button
Validate badge creation
alt
[Badge name already in use from the same educator]
Notify name in use
Go back to entry condition
[Badge created successfully]
Confirm badge creation
Badge has been successfully created

Figure 3.22

[UC11] – Delete badge

Name	Delete badge
Actor	Educator
Entry condition	Educator is logged in
Event flow	1. Educator goes to badges section 2. Educator clicks on the badge he/she wants to delete 3. Educator clicks on “delete” button
Exit condition	The badge is no longer available in the educator library
Exceptions	

Note: a badge is only deleted from the shelf of the educator, all the students that previously won that badge or will win it in a current tournament will not experience any difference.

Student:

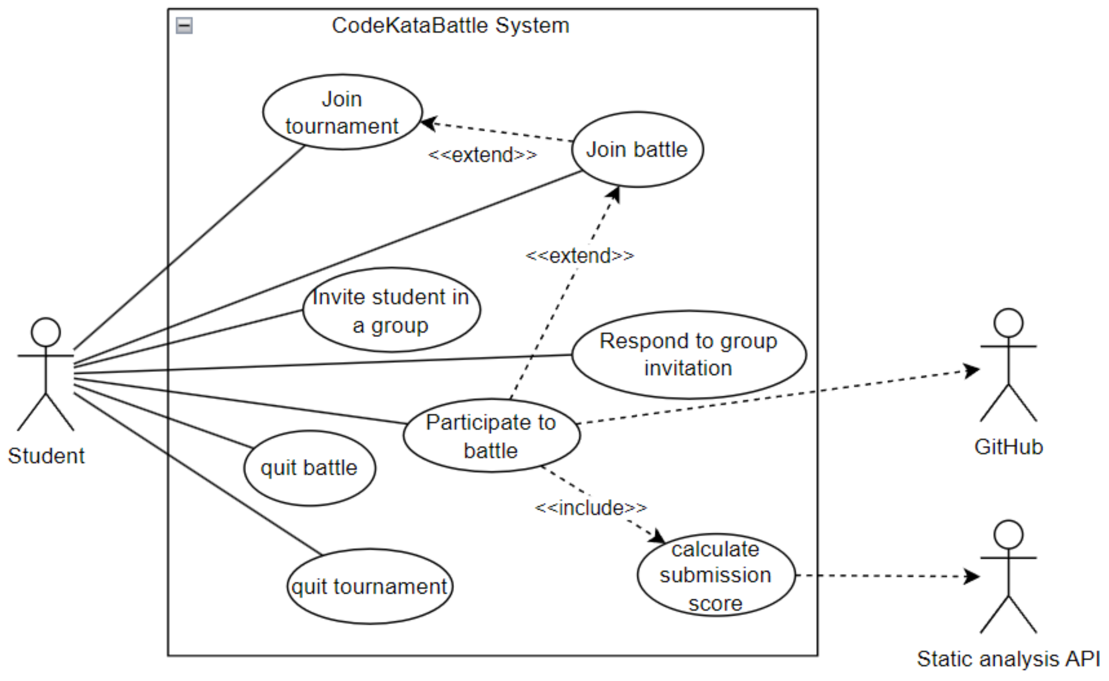


Figure 3.23

[UC12] – Join tournament

Name	Join tournament
Actor	Student
Entry condition	Student is logged in and the registration deadline of the tournament has not expired yet
Event flow	1. Student goes to tournaments section 2. Student clicks on the tournament he/she wants to join 3. Student clicks on “join” button
Exit condition	The student is registered to the tournament
Exceptions	

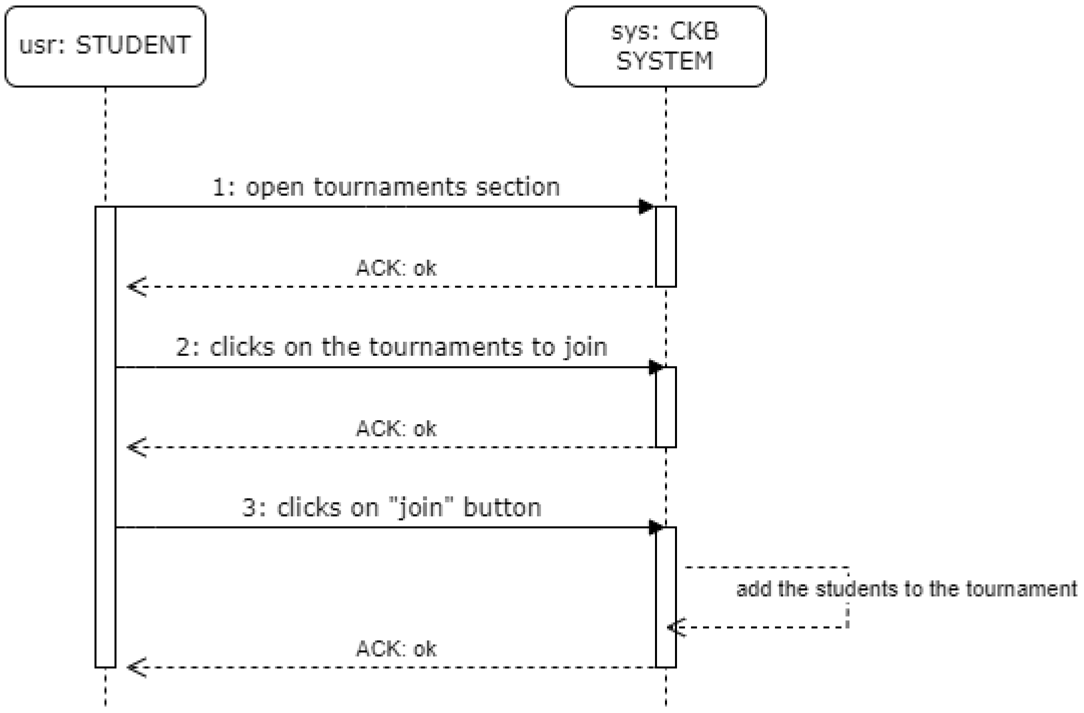

usr: STUDENT
sys: CKB SYSTEM
1: open tournaments section
ACK: ok
2: clicks on the tournaments to join
ACK: ok
3: clicks on "join" button
add the students to the tournament
ACK: ok

Figure 3.24

[UC13] – Join battle

Name	Join battle
Actor	Student
Entry condition	• The registration deadline of the battle has not expired yet • The student is logged in • The student is in the tournament of the battle
Event flow	1. Student accesses the battle section 2. Student clicks on “join” button
Exit condition	The student is registered to the battle as a singleton
Exceptions	

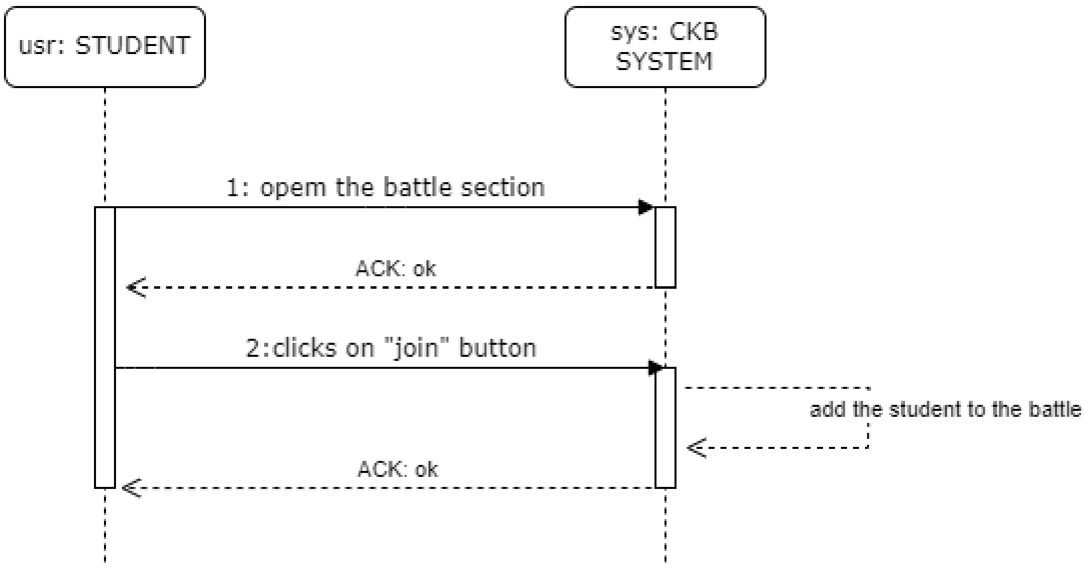

usr: STUDENT
sys: CKB SYSTEM
1: opem the battle section
ACK: ok
2:clicks on "join" button
add the student to the battle
ACK: ok

Figure 3.25

[UC14] – Invite student in the group

Name	Invite student in the group
Actor	Student
Entry condition	• Student is logged in and he/she is the group leader • The student that invites and the invited student are in the same battle • The battle is in the pre-battle state
Event flow	1. Student goes to the battle section 2. Student chooses a single peer from the list of the participants of the battle 3. The system sends the invitation to that student
Exit condition	The invited student receives a notification
Exceptions	

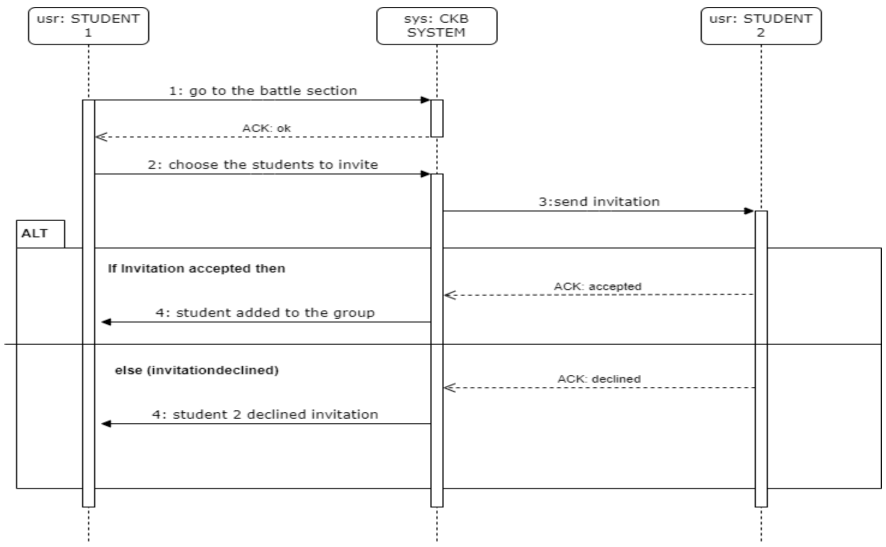

usr: STUDENT 1
sys: CKB SYSTEM
usr: STUDENT 2
1: go to the battle section
ACK: ok
2: choose the students to invite
3:send invitation
ALT
If invitation accepted then
ACK: accepted
4: student added to the group
else (invitationdeclined)
ACK: declined
4: student 2 declined invitation

Figure 3.26

[UC15] – Respond to group invitation

Name	Respond to group invitation
Actor	Student
Entry condition	• Student is logged in • Student has received an invitation • Student is in a battle that is in the pre-battle state
Event flow	1. Student opens the invitation 2. Student chooses to accept or refuse it
Exit condition	The student joins the group if the respond was "accept" otherwise the invitation is discarded
Exceptions	

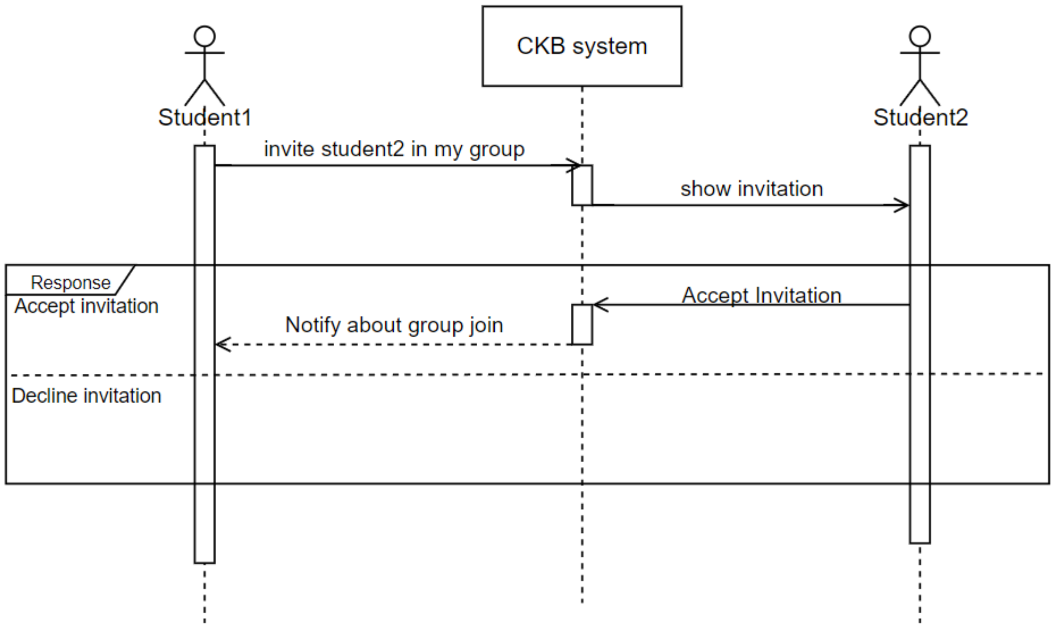


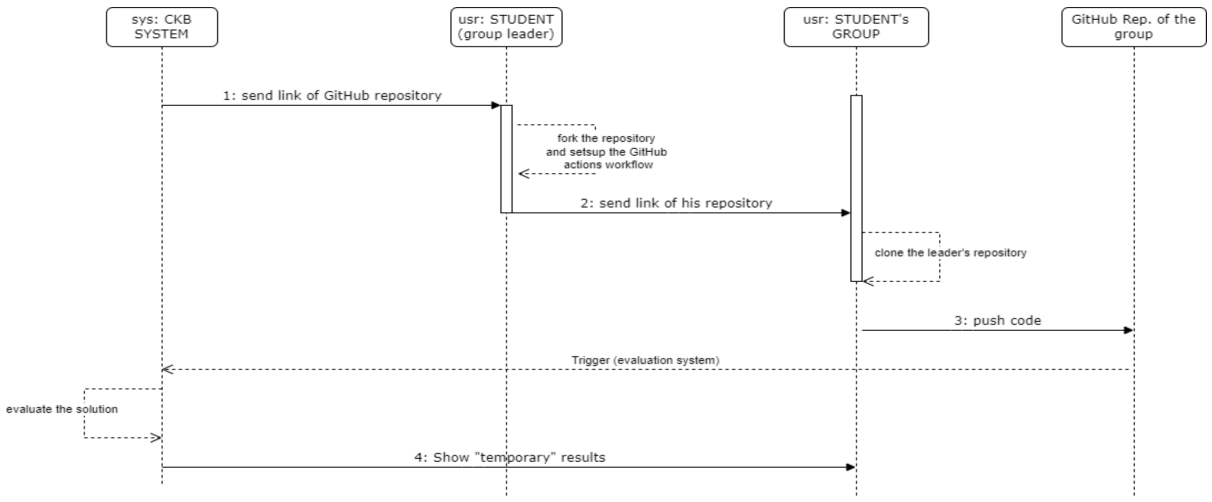
Figure 3.27

[UC16] – Participate to battle

Name	Participate to battle
Actor	Student, GitHub
Entry condition	• The student has joined to the battle • The student is in a group that has a valid number of participants • The battle is in the phase "battle"
Event flow	1. The system shares the link of the GitHub repository to the group leader 2. The leader fork the repository 3. The leader sets up a GitHub actions workflow that notifies the application of a push on the team repository 4. The leader sends the link of his repository to the group via external communication

	5. The members of the group clone the leader repository 6. A student pushes the code on the repository, triggering the evaluation system 7. The system evaluates the solution, that is not definitive, and shows the score to all the students in the battle
Exit condition	
Exceptions	

Note: the students can push the solution how many times they want.



sys: CKB SYSTEM
usr: STUDENT (group leader)
usr: STUDENT's GROUP
GitHub Rep. of the group
1: send link of GitHub repository
fork the repository and setsup the GitHub actions workflow
2: send link of his repository
clone the leader's repository
3: push code
Trigger (evaluation system)
evaluate the solution
4: Show "temporary" results

Figure 3.28

[UC17] – Quit battle

Name	Quit battle
Actor	Student
Entry condition	Student is logged in and registered in a battle
Event flow	1. Student accesses the battle section 2. Student clicks on "quit" button
Exit condition	The student quits the battle. If he/she was the group leader, the group is disjointed, and any other member is part of a group of his/her own
Exceptions	

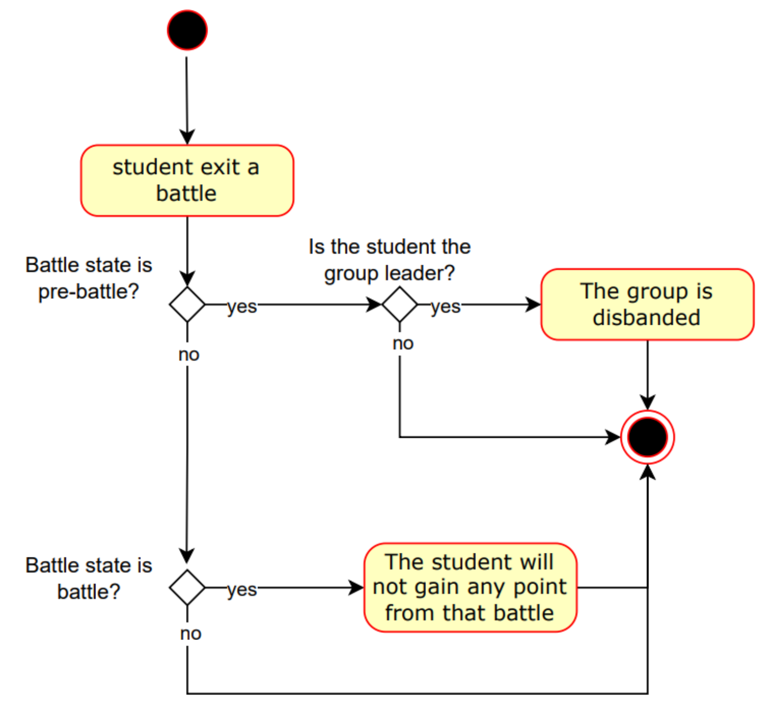

student exit a battle
Battle state is pre-battle?
yes
no
Is the student the group leader?
yes
no
The group is disbanded
Battle state is battle?
yes
no
The student will not gain any point from that battle

Figure 3.29

[UC18] – Quit tournament

Name	Quit tournament
Actor	Student
Entry condition	Student is logged in and he/she is registered to the tournament
Event flow	1. Student goes to the tournament section 2. Student clicks on "quit tournament" button
Exit condition	The student has successfully quit the tournament and any battle within it.
Exceptions	

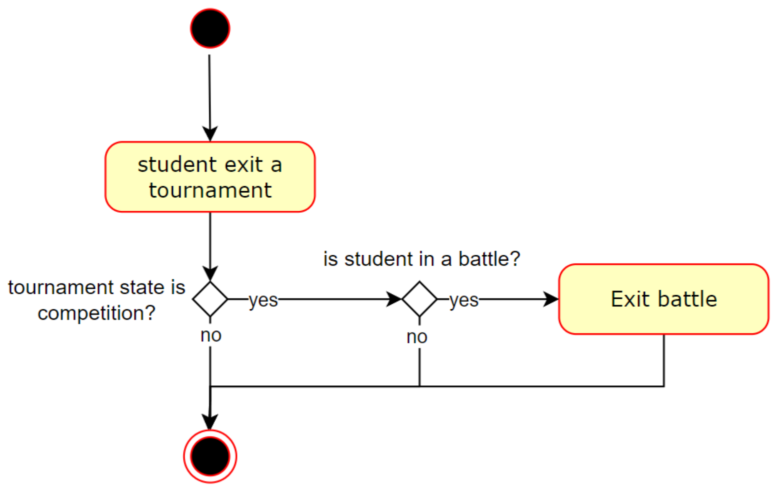


Figure 3.30

Note: student may be able to win a tournament even if he/she quits the tournament. The badges will be shown in his achievements after the tournament ends.

[UC19] – Calculate submission score

Name	Calculate submission score
Actor	CodeKataBattle System, Static Analysis API, GitHub
Entry condition	• Students push a solution to their GitHub repository • The group has set up a GitHub action that triggers the system
Event flow	1. The system pulls the latest source from the team's repository 2. Automated analysis tools run to evaluate functional aspects, timeliness, 3. Static analysis is performed to test source code quality based on the inserted upon battle creation 4. The system updates the battle scores in real-time based on the automated evaluations 5. If manual evaluation is required, and the end of the battle is reached, the system allows the educator to evaluate the group submission.
Exit condition	Every student inside the battle can see the updated score reached by the group
Exceptions	

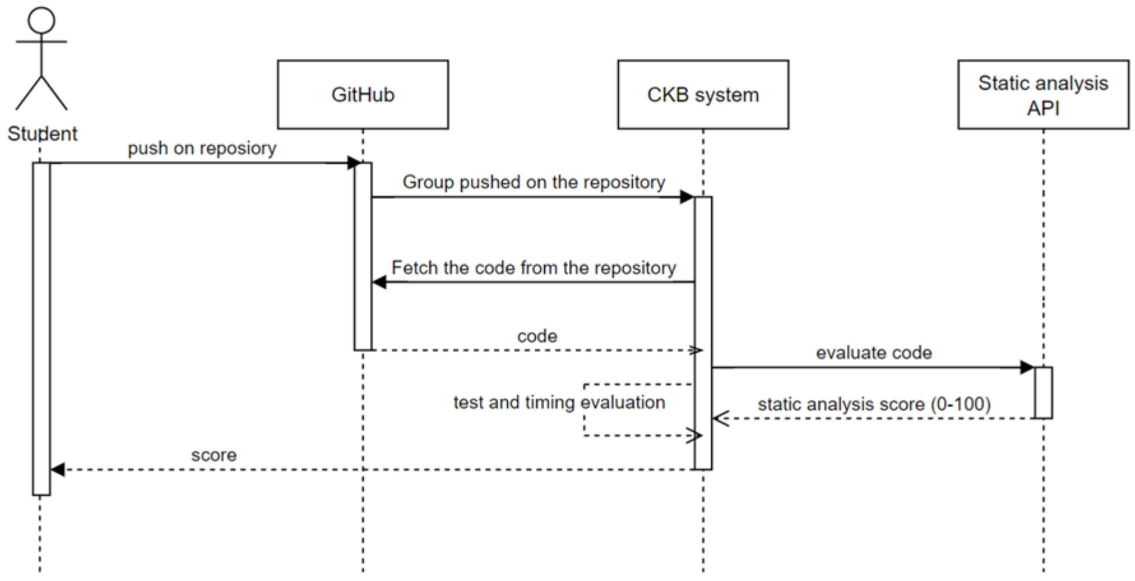

Student
GitHub
CKB system
Static analysis API
push on reposiory
Group pushed on the repository
Fetch the code from the repository
code
evaluate code
test and timing evaluation
static analysis score (0-100)
score

Figure 3.31

3.2.2 Requirements mapping

[G1] Students learn to code in one or more programming languages.	
[R1] The system allows users to sign-up and log-in. [R2] The system sends to users an email containing the link to confirm the registration of the account. [R3] The system allows each educator to create a tournament at any moment. [R8] The system allows an educator to create a battle within the context of a tournament. [R9] The system allows students to join a battle only if it has not started yet. [R11] The battle starts when the timer set upon creation expires. [R12] The system creates a GitHub repository for the battle. [R13] The system pushes in the appropriate GitHub repository the code KATA previously uploaded by the educator. [R14] The system sends setup instructions to group leaders. [R15] The system is notified about new pushes by the GitHub Action set up by the group leader. [R16] The system evaluates the pushed solution giving a temporary score after each push. [R17] The ranking is shown at the end of the battle. [R19] The system stops students from entering groups that have reached the maximum number of participants. [R22] The system disjoins a group if its leader quits the battle in the registration phase. [R23] The system allows educators to select aspects for the automated evaluation.	[D1] Students know the basics of git and have it installed on their machine. [D3] Students who participate in a battle have installed on their machine an IDE or a text editor. [D4] Students and educators have an internet connection. [D5] Students communicate with each other through external channels.

[G2] Students learn to collaborate with peers.	
[R1] The system allows users to sign-up and log-in. [R2] The system sends to users an email containing the link to confirm the registration of the account. [R3] The system allows each educator to create a tournament at any moment. [R8] The system allows an educator to create a battle within the context of a tournament. [R9] The system allows students to join a battle only if it has not started yet. [R10] The system requires students to enter a group of an acceptable size before the battle	[D1] Students know the basics of git and have it installed on their machine. [D2] Leader of a group, when requested, will follow the instructions to fork the repository, setup the GitHub Actions and send the link to the repository to the other group members (if any). [D4] Students and educators have an internet connection. [D5] Students communicate with each other through external channels.

starts, otherwise they will be expelled when the battle begins. [R11] The battle starts when the timer set upon creation expires. [R14] The system sends setup instructions to group leaders. [R15] The system is notified about new pushes by the GitHub Action set up by the group leader. [R18] The system allows group leaders to invite peers in the group if the battle has not started yet. Students can be invited even if they are participating in another group, but they need to choose only one. [R19] The system stops students from entering groups that have reached the maximum number of participants. [R22] The system disjoins a group if its leader quits the battle in the registration phase.	

[G3] Students are motivated and engaged in the activity of learning.	
[R1] The system allows users to sign-up and log-in. [R2] The system sends to users an email containing the link to confirm the registration of the account. [R3] The system allows each educator to create a tournament at any moment. [R4] The system updates, upon the end of a battle, the tournament score of the students who have participated in it. [R5] The system allows the tournament leader to add pre-existing gamification badges upon creation of the tournament. [R8] The system allows an educator to create a battle within the context of a tournament. [R9] The system allows students to join a battle only if it has not started yet. [R11] The battle starts when the timer set upon creation expires. [R14] The system sends setup instructions to group leaders. [R15] The system is notified about new pushes by the GitHub Action set up by the group leader. [R16] The system evaluates the pushed solution giving a temporary score after each push. [R17] The ranking is shown at the end of the battle. [R18] The system allows group leaders to invite peers in the group if the battle has not started yet. Students can be invited even if they are	[D4] Students and educators have an internet connection.

participating in another group, but they need to choose only one. [R20] The system allows educators to create gamification badges which can be used only in possible future tournaments. [R21] Tournament leaders can add zero or more preexisting gamification badges to the tournament only during creation. [R23] The system allows educators to select aspects for the automated evaluation. [R24] The system registers as the score obtained in a battle the last score calculated (which corresponds to the score of the last push on the repo).	

[G4] Educators share their programming skills and knowledge.	
[R1] The system allows users to sign-up and log-in. [R2] The system sends to users an email containing the link to confirm the registration of the account. [R3] The system allows each educator to create a tournament at any moment. [R6] The system allows the tournament leader to add other educators in the tournament at any point in time. [R7] The system allows the tournament leader to close a hosted tournament. [R8] The system allows an educator to create a battle within the context of a tournament. [R11] The battle starts when the timer set upon creation expires. [R12] The system creates a GitHub repository for the battle. [R13] The system pushes in the appropriate GitHub repository the code KATA previously uploaded by the educator. [R16] The system evaluates the pushed solution giving a temporary score after each push. [R20] The system allows educators to create gamification badges which can be used only in possible future tournaments. [R21] Tournament leaders can add zero or more preexisting gamification badges to the tournament only during creation. [R23] The system allows educators to select aspects for the automated evaluation.	[D4] Students and educators have an internet connection.

3.3 Performance requirements

The system must support a daily workload of 100k users.
It is assumed that there will be 10 active students for each active educator.
Students must be able to see the creation of a battle or tournament in at most 10 seconds.
The latency of in-battle events must be less than a second.
The system must be able to handle traffic spike during peak hours.

Note: the numbers proposed in this section refers only to the system, excluding the user internet connectivity capacity.

3.4 Design constraints

3.4.1 Standard compliance

The CodeKataBattle software must comply with regulations such as GDPR for data protection, copyright laws, electronic commerce and privacy directives, consumer rights, accessibility regulations, and cybersecurity laws. Ethical considerations, including fairness and inclusivity, should also be prioritized.

3.4.2 Hardware limitations

The client is meant to run on regular personal computers, that have access to the internet.

3.5 Software system attributes

3.5.1 Reliability

The system is expected to fail, on average, once per year.

3.5.2 Availability

The availability must be at least 99.9%.

3.5.3 Security

The software needs strong authentication, data encryption, secure APIs and GDPR compliance.

3.5.4 Maintainability

1. Modularity: Code organized into modular components.
2. Documentation: Comprehensive and updated documentation.
3. Readability: Clear and consistent coding standards.
4. Version Control: Use of version control systems (e.g., Git).
5. Automated Testing: Implementation of automated testing.
6. Dependency Management: Effective management of dependencies.
7. Code Comments: Meaningful comments for complex logic.
8. Error Handling: Robust error handling for quick issue resolution.
9. Scalability: Design for scalability in user base and data volume.

10. Compatibility: Ensure compatibility with future technologies.

3.5.5 Portability

1. Cross-Browser Compatibility: Ensure compatibility with major web browsers (e.g., Chrome, Firefox, Safari).
2. Cross-Platform Compatibility: Support multiple operating systems (e.g., Windows, macOS, Linux).
3. Mobile Responsiveness: Optimize for seamless functionality on various screen sizes and devices.
4. Database Independence: Design the system to be compatible with different database systems.
5. Deployment Flexibility: Allow for easy deployment in different environments (e.g., on-premises, cloud).

4. Alloy

4.1 Signatures

In the following are stated all the signatures defined for the alloy modelization.

```
//Definitions of useful signatures
sig Nickname{}
sig Title{}
sig ProgrammingLanguage{}

//Definitions of deadline statuses
abstract sig DeadlineStatus{}
one sig Expired extends DeadlineStatus{}
one sig Unexpired extends DeadlineStatus{}

//Used to represent a tuple <date, time>
sig DateTime{
    date: one Int,
    time: one Int
}{
    time >= 0
    date >= 1
}

//Definitions of tournament statuses
abstract sig TournamentStatus{}
one sig Ongoing extends TournamentStatus{}
one sig Close extends TournamentStatus{}

//Definition of battles and tournaments that have involved a user
abstract sig History{}
one sig StudentHistory{
    var battlesAsStudent: set Student -> Battle -> BattleScore,
    var tournamentsAsStudent: set Student -> Tournament -> TournamentScore,
    var achievedBadges: set Student -> GamificationBadge
}
one sig EducatorHistory{
    var battlesAsEducator: set Educator -> Battle,
    var tournamentsAsEducator: set Educator -> Tournament,
    var createdBadges: set Educator -> GamificationBadge
}

//Type of users (Student, Educator)
abstract sig User{}
sig Student extends User{
    41sername: one Nickname
}
```

```
sig Educator extends User{
    42sername: one Nickname
}

//Definition of tournament
sig Tournament{
    name: one Title,
    creator: one Educator,
    registrationDeadline: one DateTime,
    var registrationDeadlineStatus: one DeadlineStatus,
    chosenBadge: set GamificationBadge,
    var status: one TournamentStatus
}

//Definition of battle
sig Battle{
    name: one Title,
    creator: one Educator,
    gitRepoLink: one Repository,
    registrationDeadline: one DateTime,
    submissionDeadline: one DateTime,
    var registrationDeadlineStatus: one DeadlineStatus,
    var submissionDeadlineStatus: one DeadlineStatus,
    language: one ProgrammingLanguage
}{
    registrationDeadline.date < submissionDeadline.date or
    (registrationDeadline.date = submissionDeadline.date and
   registrationDeadline.time < submissionDeadline.time)
}

//Definition of Boolean predicate for gamification badges
sig BooleanPredicate{}

//Definition of gamification badge
sig GamificationBadge{
    name: one Title,
    predicates: some BooleanPredicate,
    creator: one Educator
}

//Definition of GitHub repository
sig Repository{}

//Definition of battle’s score
sig BattleScore{
    score: one Int
}{
    score >= 0
}

//Definition of tournament’s score
sig TournamentScore{
    score: one Int
}{
    score >= 0
}

//Assosiaction between tournaments and their battles
sig BattlesTournamentsAssociation{
    link: Tournament -> set Battle
}
```

4.2 Functions

In the following are stated all the functions defined in alloy to ease the creation of facts.

```
//Get tournaments done by the given student
fun getTournamentsByStudent[s: Student]: set Tournament{
    s.(StudentHistory.tournamentsAsStudent.TournamentScore)
}

//Get battles done by the given student
fun getBattlesByStudent[s: Student]: set Battle{
    s.(StudentHistory.battlesAsStudent.BattleScore)
}

//Get tournaments that involved the given educator
fun getTournamentsByEducator[e: Educator]: set Tournament{
    e.(EducatorHistory.tournamentsAsEducator)
}

//Get badges achieved by the given student
fun getAchievedBadgesByStudent[s: set Student]: set GamificationBadge{
    s.(StudentHistory.achievedBadges)
}

//Get badges created by the given educator
fun getCreatedBadgesByEducator[e: Educator]: set GamificationBadge{
    e.(EducatorHistory.createdBadges)
}

//Get battles in the given tournament
fun getBattlesByTournament[t: Tournament]: set Battle{
    t.(BattlesTournamentsAssociation.link)
}
```

```
//Get tournament of the given battle
fun getTournamentByBattle[b: Battle]: one Tournament{
    BattlesTournamentsAssociation.link.b
}

//Get the score of the given student in the given tournament
fun getScoreByStudentAndTournament[s: Student, t: Tournament]: set
TournamentScore{
    t.(s.(StudentHistory.tournamentsAsStudent))
}

//Get scores by student and set of battles
fun getScoreByBattleAndStudent[s: Student, b: set Battle]: set BattleScore{
    b.(s.(StudentHistory.battlesAsStudent))
}

//Get scores by the given battle
fun getScoresByBattle[b: Battle]: set BattleScore{
    b.(Student.(StudentHistory.battlesAsStudent))
}
```

4.3 Facts

In the following are stated in Alloy the facts that must hold for the domain model in order to be consistence w.r.t. the real world.

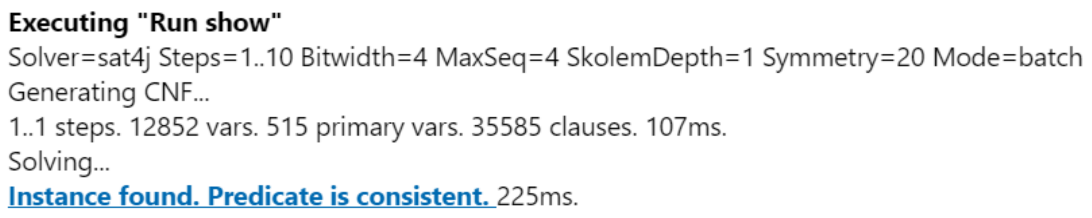
```
//Every battle has its own repository
fact differentReposForDifferentBattles {
    all disj b1, b2: Battle |
             b1.gitRepoLink != b2.gitRepoLink
}

//Students registered to a battle are registered also to the tournament in
which the battle is held
fact studentsInBattleAreInItsTournament {
    all s: Student |
        all b: Battle | b in s.getBattlesByStudent implies
            b.getTournamentByBattle in s.getTournamentsByStudent
}

//Every user has a unique nickname
fact noStudentAndEducatorWithSameUsername {
    all s: Student |
        all e: Educator |
            s.userName != e.userName
}
fact noStudentsWithSameUsername {
    all disj s1, s2: Student |
             s1.userName != s2.userName
}
fact noEducatorsWithSameUsername {
    all disj e1, e2: Educator |
             e1.userName != e2.userName
}

//Tournament name is unique with respect to those tournaments whose status is
Open
fact noOpenTournamentWithTheSameName {
    all disj t1, t2: Tournament |
             t1.name = t2.name implies not (t1.status = Ongoing and t2.status
                  = Ongoing)
}

//Every battle within a tournament has a unique name
fact noBattleWithTheSameNameInTheSameTournament{
    all disj b1, b2: Battle |
             b1.getTournamentByBattle = b2.getTournamentByBattle implies
                 b1.name != b2.name
}

//The educator who creates a battle is the creator of its tournament or has
been given permission by its creator
fact battleCreatorIsInvolvedInTheTournament{
    all b: Battle |
        b.getTournamentByBattle in b.creator.getTournamentsByEducator
}

//No student can register to a battle if its registration deadline has expired
fact noRegistrationToBattleIfDeadlineExpired{
    all b: Battle |
        all s: Student |
            always b.registrationDeadlineStatus = Expired and b not in
                s.getBattlesByStudent implies
                    always b not in s.getBattlesByStudent
}

//No student can register to a tournament if its registration deadline has
expired
fact noRegistrationToTournamentIfDeadlineHexpired{
    all t: Tournament |
        all s: Student |
            always t.registrationDeadlineStatus = Expired and t not in
s.getTournamentsByStudent implies
                always t not in s.getTournamentsByStudent
}
```

```
//When the deadline has expired, it remains expired forever
fact onceTournamentDeadlineExpiredItRemainsExpiredForever{
    all t: Tournament |
        always t.registrationDeadlineStatus = Expired implies
            after always t.registrationDeadlineStatus = Expired
}
fact onceBattleRegistrationDeadlineExpiredItRemainExpiredForever{
    all b: Battle |
        always b.registrationDeadlineStatus = Expired implies
            after always b.registrationDeadlineStatus = Expired
}
fact onceBattleSubmissionDeadlineExpiredItRemainsExpiredForever{
    all b: Battle |
        always b.submissionDeadlineStatus = Expired implies
            after always b.submissionDeadlineStatus = Expired
}

//Gamification badges in a tournament has been generated by the tournament
creator
fact tournamentBadgesGeneratedByTournamentCreator{
    all t: Tournament |
        all gb: GamificationBadge |
            gb  in t.chosenBadge implies gb in
                 t.creator.getCreatedBadgesByEducator
}

//No overlapping battles in the same tournament
fact noOverlappingBattlesInTheSameTournament{
    all disj b1, b2: Battle |
        b1.getTournamentByBattle = b2.getTournamentByBattle implies
            ((b1.submissionDeadline.date < b2.registrationDeadline.date) or
                (b1.submissionDeadline.date = b2.registrationDeadline.date and
                    b1.submissionDeadline.time <=
                        b2.registrationDeadline.time))
}

//Once a badge is achieved, it cannot be removed from the student's history
fact achievedBadgeesRemainForever{
    all s: Student |
        all gb: GamificationBadge |
            always gb in s.getAchievedBadgesByStudent implies
                after always gb in s.getAchievedBadgesByStudent
}

//Once a tournament has been closed, its states remains close forever
fact onceTournamentClosedItRemainsClosed{
    all t: Tournament |
        always t.status = Close implies
            after always t.status = Close
}

//The score gotten at the end of a tournament is the sum of all scores gotten
in the battles within the tournament itself
fact sumBattlesScoresEqualToTournamentScore{
    all s: Student |
        all t: Tournament |
            getScoreByStudentAndTournament[s, t].score =
sum[getScoreByBattleAndStudent[s, getBattlesByTournament[t]].score]
}

//All battles must be within a tournament
fact allBattlesAreWithinATournament{
    all b: Battle |
        b.getTournamentByBattle != none
}

//If a tournament is closed, its registration deadline has expired
fact registrationDeadlineExpiredIfTournamentClose{
    all t: Tournament |
        t.status = Close implies t.registrationDeadlineStatus = Expired
}

//Succession of TournamentStatus
fact successionOfTournamentStatus{
    all t: Tournament |
        t.status = Close implies
            before t.status = Ongoing
}

//If the registration deadline of a tournament has not expired yet, no buttle
can be created within it
fact noBattleIfTournamentRegistrationDeadlineHasNotExpired{
    all t: Tournament |
        t.registrationDeadlineStatus = Unexpired implies no
            t.getBattlesByTournament
}

//If a battle is not concluded yet, no scores can be definitively assign w.r.t
it
fact noScoreIfBattleNotConluded{
    all b: Battle |
        (b.registrationDeadlineStatus = Unexpired or
             b.submissionDeadlineStatus = Unexpired) implies
                 no b.getScoresByBattle
}
```

4.4 Model

In the following are stated the predicate used to show the model and the model itself.

```
Pred show {
    #Battle = 3
    #StudentHistory.battlesAsStudent = 1
    #Student = 2
    #GamificationBadge = 2
    #Tournament = 2
}

run show
```



```
Executing "Run show"
Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
Generating CNF...
1..1 steps. 12852 vars. 515 primary vars. 35585 clauses. 107ms.
Solving...
Instance found. Predicate is consistent. 225ms.
```

Figure 4.1

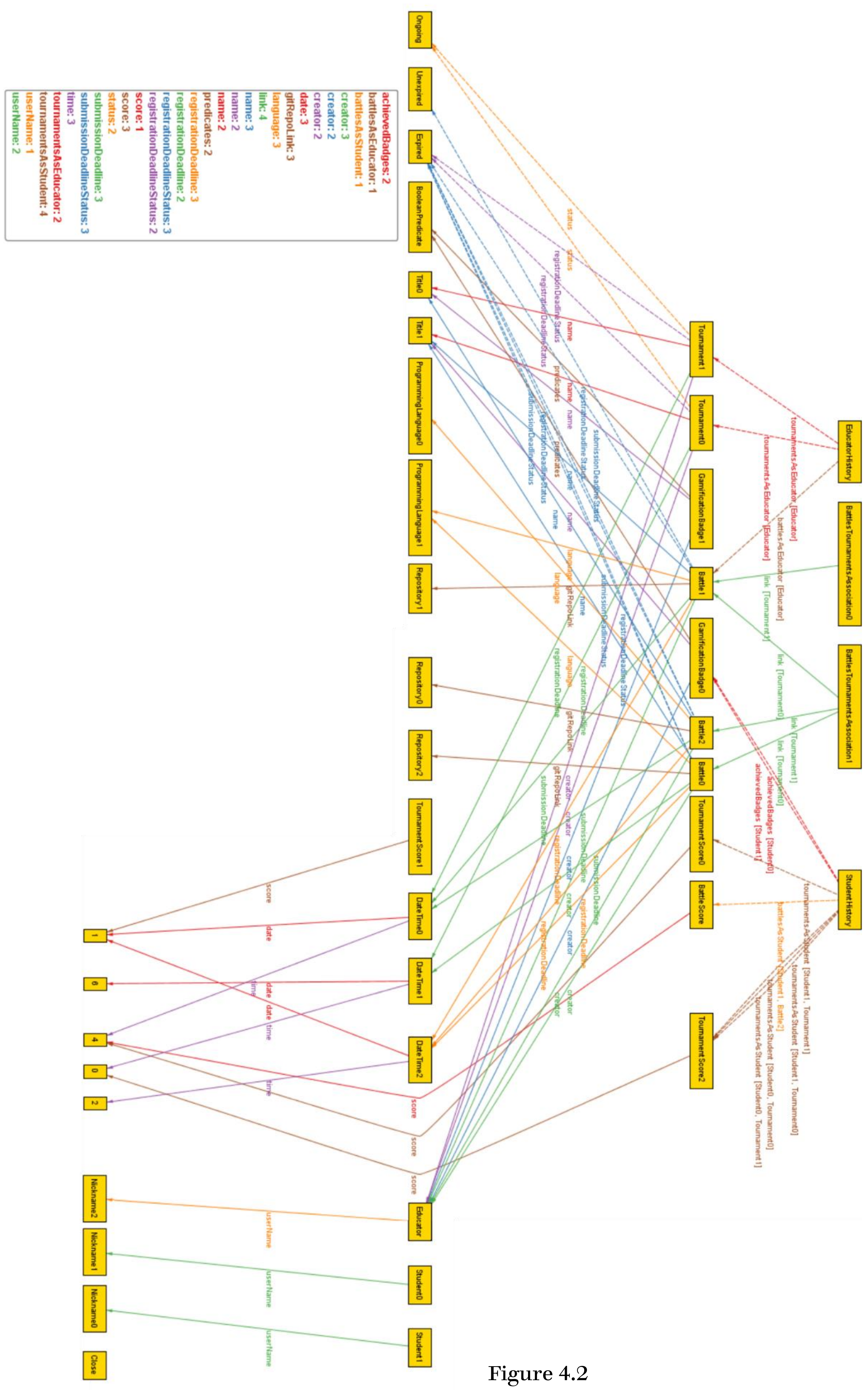

achievedBadges: 2
battlesAsEducator: 1
battlesAsStudent: 1
creator: 3
creator: 2
creator: 2
creator: 2
date: 3
gitRepoLink: 3
language: 3
link: 4
name: 3
name: 2
name: 2
name: 2
predicates: 2
registrationDeadline: 3
registrationDeadline: 2
registrationDeadlineStatus: 3
registrationDeadlineStatus: 2
score: 1
score: 3
status: 2
submissionDeadline: 3
submissionDeadlineStatus: 3
time: 3
tournamentsAsEducator: 2
tournamentsAsStudent: 4
userName: 1
userName: 2
Ongoing
Unexpired
Expired
BooleanPredicate
Title0
Title1
ProgrammingLanguage0
ProgrammingLanguage1
Repository1
Repository0
Repository2
TournamentScore1
DateTime0
DateTime1
DateTime2
Educator
Student0
Student1
1
6
4
0
2
Nickname2
Nickname1
Nickname0
Close
Tournament1
Tournament0
GamificationBadge1
Battle1
GamificationBadge0
Battle2
Battle0
TournamentScore0
BattleScore
TournamentScore2
EducatorHistory
BattlesTournamentsAssociation0
BattlesTournamentsAssociation1
StudentHistory

Figure 4.2

5. Time spent

Section	Feraboli	Filippini	Lucca
1	4	4.5	4
2	8	7	8.5
3	18	9	17.5
4	2	12	2

6. References

- Alloy models made with: Visual Studio Code (used extensions: Alloy, Alloy VSCode)
- Diagrams made with StarUML and Draw.io
- Graphical User Interface mockups made with uizard.io