

**Code Kata Battle - Eusebio Alberto,
Martini Marcello**



POLITECNICO
MILANO 1863

Requirement Analysis and Specification Document

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	Eusebio Alberto, Martini Marcello
Version:	1.0
Date:	22-December-2023
Download page:	https://github.com/martinimarcello00/EusebioMartini
Copyright:	Copyright © 2023, Eusebio Alberto, Martini Marcello – All rights reserved

Contents

Table of Contents	3
List of Figures	5
List of Tables	5
1 Introduction	6
1.1 Purpose	6
1.1.1 Goals	6
1.2 Scope	6
1.2.1 World Phenomena	6
1.2.2 Shared Phenomena	7
1.3 Definitions, Acronyms, Abbreviations	7
1.3.1 Definitions	7
1.3.2 Acronyms	8
1.3.3 Abbreviations	8
1.4 Revision history	8
1.5 Reference Documents	8
1.6 Document Structure	8
2 Overall Description	10
2.1 Product perspective	10
2.1.1 Scenarios	10
2.1.2 Domain models	11
2.1.3 State Diagrams	12
2.2 Product functions	15
2.3 User characteristics	16
2.4 Assumptions, Dependencies, and Constraints	16
2.4.1 Privacy policy	16
2.4.2 Domain assumptions	16
2.4.3 Dependencies	17
3 Specific Requirements	18
3.1 External interface requirements	18
3.1.1 User interfaces	18
3.1.2 Hardware interfaces	21
3.1.3 Software interfaces	21
3.1.4 Communication interfaces	21
3.2 Functional requirements	21
3.2.1 Use Cases and Use Case Diagrams	23
3.2.2 Sequence diagrams	32
3.2.3 Requirements Mapping	42
3.3 Performance requirements	44
3.4 Design constraints	44
3.4.1 Standards compliance	44
3.4.2 Hardware limitations	44
3.5 Software systems attributes	44
3.5.1 Reliability	44
3.5.2 Availability	44
3.5.3 Security	44

3.5.4	Maintainability	45
3.5.5	Portability	45
4	Formal Analysis Using Alloy	46
4.0.1	Alloy model	46
4.0.2	Alloy relaxed version	50
5	Effort Spent	53

List of Figures

1	Domain class diagram	11
2	State diagram: Tournament status	12
3	State diagram: Enrollment in battles	13
4	State diagram: Students provide a solution	14
5	User interface: Student dashboard	19
6	User interface: Instructor dashboard	19
7	User interface: Instructor interface to create a battle	20
8	User case diagram: Student	23
9	User case diagram: Instructor	24
10	Sequence diagram UC 1: Login user	32
11	Sequence diagram UC 2: Sign-up Student	33
12	Sequence diagram UC 3: Sign-up Instructor	34
13	Sequence diagram UC 4: User confirms the email address	35
14	Sequence diagram UC 5: Tournament Creation	36
15	Sequence diagram UC 6: Tournament Close	37
16	Sequence diagram UC 7: Battle Creation	38
17	Sequence diagram UC 8: Battle Close	39
18	Sequence diagram UC 9: Joining a Battle	40
19	Sequence diagram UC 10: Students commit a battle solution to GitHub	41
20	Alloy model	52

List of Tables

15	Effort spent in the creation of the document	53
----	--	----

1 Introduction

1.1 Purpose

The purpose of this Requirements Analysis and Specification Document is to present the CodeKataBattle platform. CodeKataBattle (CKB) is a platform where students can improve their software development skills, by solving programming exercises, in this context called katas.

1.1.1 Goals

Here we identify the goals of the system:

- G1.** Students are able to practice on katas by participating in tournaments in groups of predefined minimum and maximum size.
- G2.** Students obtain an automatic evaluation of their proposed solutions.
- G3.** Instructors are able to create, manage and award tournaments.
- G4.** Instructors are able to provide a manual evaluation of the proposed solutions.

1.2 Scope

In recent years, the need for practical, hands-on software development training has become increasingly evident in educational environments. The CKB platform is an instrument to support instructors in the teaching process and students in improving their coding skills. The platform is held online and easily accessible by users through an application interface.

Instructors can create katas through the platform and such katas can be addressed by several groups of students in a programming language of choice.

The platform allows Educators to define deadlines on subscription and submission as long as other constraints such as the maximum and minimum number of students per group and other mechanisms for scoring.

The problems are addressed in battles, and organized in tournaments, where the groups of students can participate.

Participation in a tournament can improve a student's rank on the platform and grant them badges, based on the performance measured.

The thresholds for badges, their release, and the creation of other variables are features available only to educators, but their visualization is open to every other user.

The platform relies on GitHub actions for the creation of repositories containing code katas and the automatic testing of the proposed solutions. The students are required to fork the repositories and set up an automatic workflow through GitHub Actions to inform CKB that a commit has been made.

1.2.1 World Phenomena

Phenomena events that take place in the real world and that the machine cannot observe.

- W1.** Instructors define the most relevant variables and rules for badge assignment
- W2.** Instructors design the tests to be associated to each problem
- W3.** Instructors setup the software programs for the build automation
- W4.** Instructor designs a kata to be then uploaded to the platform.

- W5.** Students form teams outside the CKB.
- W6.** Students fork the GitHub repository provided by the CKB
- W7.** Students manually set up an automated workflow that informs CKB whenever a commit is made.
- W8.** Instructors manually review the student's codes to assign additional scores to the team.
- W9.** Students write code to solve the katas.
- W10.** Students follow a test-first approach to the resolution of katas

1.2.2 Shared Phenomena

World-controlled shared phenomena

- SP1.** Student inserts his/her personal institutional email address and a secure password to sign up to the platform
- SP2.** Instructor inserts his/her personal institutional email address and a secure password to sign up to the platform
- SP3.** All users insert login information to access a personalized page on the platform
- SP4.** Instructor interacts with the platform and creates a new tournament, by providing a unique name to it
- SP5.** Instructor interacts with the platform and creates a new battle, with a unique name and associated to a single tournament.

Machine-controlled shared phenomena

- SP6.** Platform notifies students whenever a battle of a specific tournament is created
- SP7.** Platform creates a repository for each group of student registered to a specific battle
- SP8.** Platform updates the rank of all students as soon as they perform a push action on GitHub repositories.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Code kata battle:** a programming exercise proposed by an instructor and tested by the machine. code kata battles are composed of the following elements:
 - A brief textual description of the problem
 - A software program with build automation scripts
 - A set of test cases that will be the base to evaluate the proposed solution
- **API:** stands for Application Program Interface and are a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.
- **Commit:** a name to indicate the action of saving the status of the code in versioning control systems such as Git
- **Push:** the action of uploading the local versioning history to a GitHub repository

- **Pull:** the action of updating the local versioning history with code coming from a GitHub repository
- **Badge:** a digital award that can be obtained through the matching of a set of rules
- **Rule:** a mathematical relation between variables
- **Variable:** a label associated to a measurable quantity
- **Rank:** an incremental value associated to each student registered on the application and updated through the solution of katas
- **Score:** a natural number comprised between 0 and 100

1.3.2 Acronyms

- **kata:** abbreviation for code kata battle
- **CKB:** used to identify the Code Kata Battle platform

1.3.3 Abbreviations

- **Gi:** goal number i
- **Wi:** world phenomena number i
- **SPi:** shared phenomena number i
- **Ri:** requirement number i
- **UCSi:** use case scenario number i
- **PFi:** product function number i
- **DAi:** domain assumption number i

1.4 Revision history

- Version 1.0 (22 December 2023)

1.5 Reference Documents

This document is strictly based on:

- 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;

1.6 Document Structure

Mainly the current document is divided in 4 chapters, which are:

1. **Introduction:** description of the platform and the environment in which it collocates in, identifies the main actors and stakeholders and defines goals, world phenomena and shared phenomena
2. **Overall Description:** expands the introduced shared phenomena and complements the description of the problem with scenarios, class and state diagrams. requirements and domain assumptions are introduced as long as the principal use cases categories.

3. **Specific Requirements:** it describes in very detail the requirements of the system and focuses on the details useful to the development team, such as external interface requirements, functional requirements, performance requirements, design constraints and software system attributes. (i.e information about HW and SW interfaces)
4. **Formal Analysis:** it contains the alloy model of the system as long as the purpose for which the formal model was created and what the model tries to accomplish.
5. **Effort Spent:** it shows the time spent to realize this document
6. **References:** it contains the references to any document used as reference and the software tools employed for the model creation

2 Overall Description

2.1 Product perspective

2.1.1 Scenarios

Here we include the list of scenarios that can be useful in understanding how the platform can be utilized in plain language, using fictitious names for the actors involved in the interactions.

- SC1. Creation of a tournament.** Giovanni is a professor at Politecnico di Milano teaching the course of Software Engineering I. Having noticed that his students struggle to master the concepts of object-oriented programming, he decides to organize a tournament composed of 10 battles, where the students can compete in groups against each other in solving the problems proposed. After having logged in to the CKB platform as an instructor, he is finally able to propose a new tournament, that he names "ExerciseSessions". After completing this step, all students enrolled in the CKB platform are notified about the tournament creation.
- SC2. Creation of battles.** Giovanni, after the creation of a tournament, needs to populate it with battles (katas). Since it will be very time-consuming to create all 10 battles, he decides to grant his teaching assistants (who are also registered as instructors on the platform) permission to create battles. They start creating the first battle, called "ExerciseSession01" and set the minimum and maximum number of students per group as well as the registration deadline and submission deadline. Finally, both Giovanni and his teaching assistants start designing the katas to be performed in the first battle: describe the problem to be solved, write the test cases for the evaluation of the answers, set up the automated build scripts, and upload them on the platform specifying as reference battle "ExerciseSession01". The students are notified about the creation of the first battle and allowed to enroll.
- SC3. Students participate in tournament,** Mario, Luca, and Carla are a group of students enrolled in the course Software Engineering I, taught by Giovanni. After the creation of a Battle ExerciseSession01, they decide to enroll as a group, so they all log into the platform, Carla (the group leader) registers the group with the name "LuMaCa" and Luca and Mario accept the invitation. As soon as the group is registered to the battle, all of them are notified about the creation of the GitHub repository "ExerciseSession01-kata" and are invited to fork it and set a workflow through GitHub Actions and proper API calls that informs CKB about each push action performed on the forked repositories.
- SC4. Students solve katas.** After the registration of group "LuMaCa" for the first battle, they decide to keep the team also for the following battles, so the group starts solving the battles present in the tournament. They decided to use Java as the language to solve the katas and Eclipse as the IDE. By implementing a test-first approach, they decide to perform a push action every time they pass a unit test. Automatically the system pulls the code from the repository, compiles it, runs the tests, and performs static analysis on the code they submit at each push action. It consequently updates their ranks, by evaluating both functional aspects, timeliness and security, maintainability and reliability of the proposed solutions. The system keeps performing this action until the final deadline for the battle. After that the pull action
- SC5. Instructor manually assigns scores.** After the submission of the final solution and the end of the battle, Giovanni is able to view the proposed solutions. Since he wants to evaluate also the readability of the code proposed and the quality of the comments documenting the solutions, he manually evaluates all the solutions and uses the CKB platform to assign a score to each one of them. Once done, using the platform labels the battle "ExerciseSession01" as completed and the system notifies all students about the update of their tournament-ranks.

SC6. Instructor closes a tournament. The course Software Engineering I is about to end, so Giovanni decides to close the tournament "ExerciseSessions". He logs into the CKB platform, searches for the specific tournament, and clicks on the option "close tournament". Once done, the students involved in the tournament are notified and they can see the final rank.

2.1.2 Domain models

Here we present the Domain Class diagram of the CKB platform. The previous description of the problem should be sufficient for the understanding of the following diagram. However, we present here some of the main features. In the following diagram, the system entity is not present, therefore we did not show the login and sign up relations that should be present for the students and instructors, also the automatic testing is associated with the GitHub repository.

- **Student** and **Instructor** are the two classes of users that we can find in our presentation of the problem. Since they are different entities with different authentication procedures and different roles, we present them as separate. Login and signup are not present, even though required.
- **GitHubRepository** is a class identifying the repository that is automatically created for each group of students and for which the students have set GitHub actions, to trigger the automatic testing
- **Test** is a class used both to identify the test cases and the tools used to perform static analysis on the code pushed into the repository

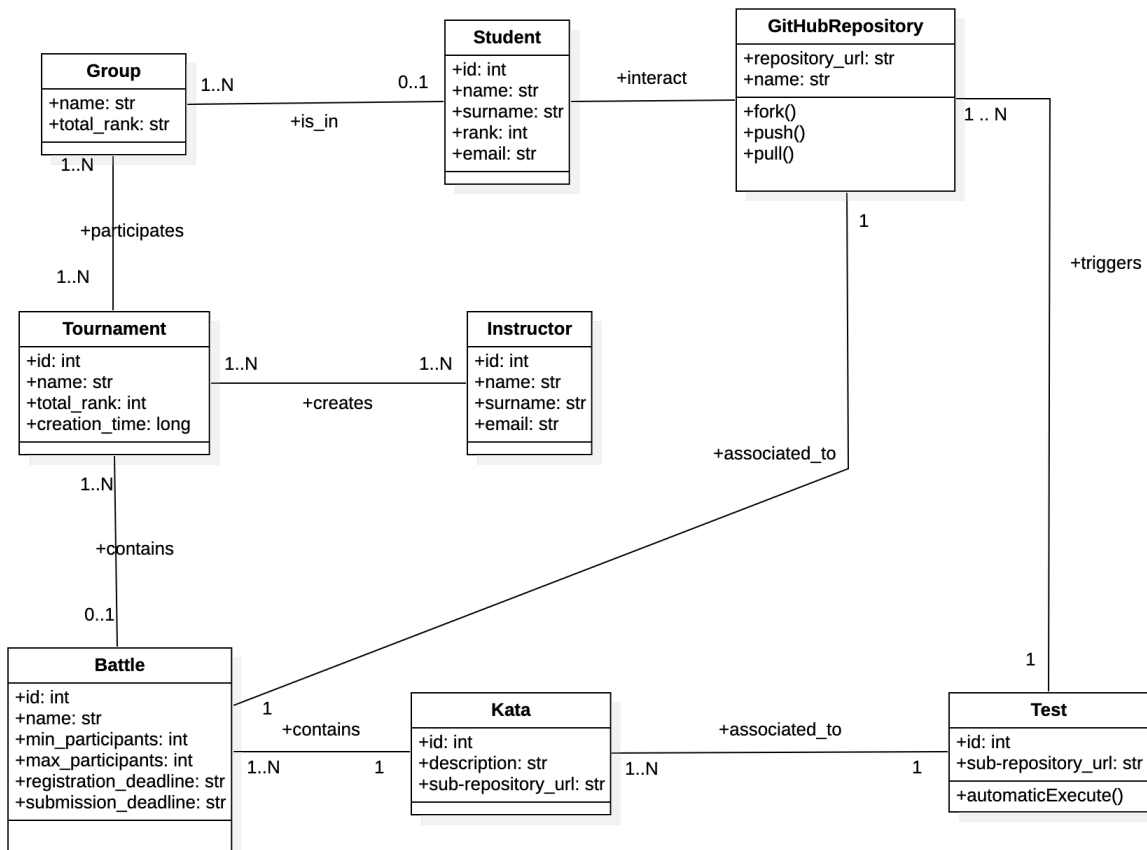


Figure 1: Domain class diagram

2.1.3 State Diagrams

- **Tournament status:** After the tournament is created, applications are open for students to join. Once the tournament application deadline has passed, the battles of the tournament are carried out. After all battles are completed and no battles are remaining, the tournament is closed. If the instructor needs to set manual scores, the system will then await these evaluations.

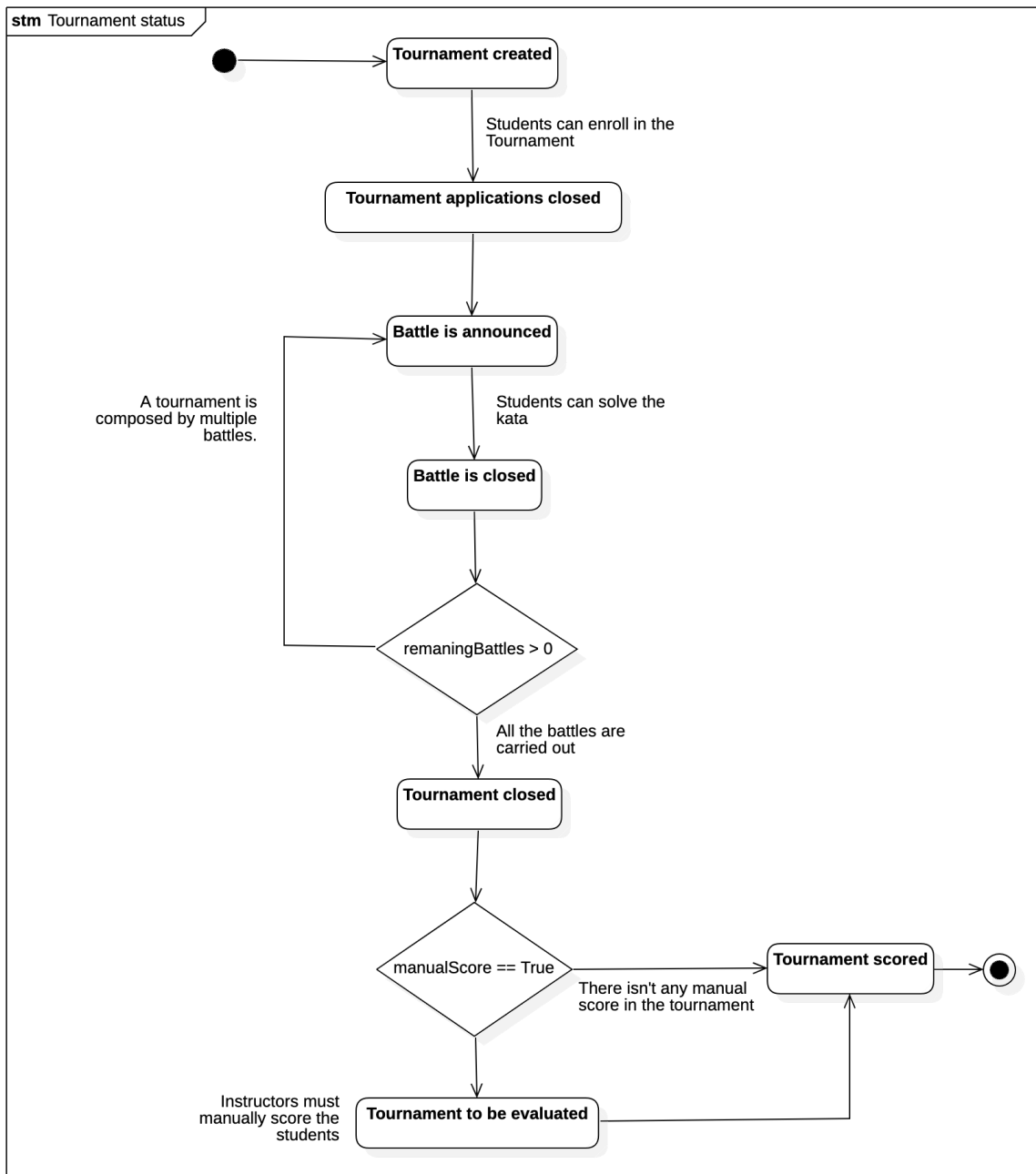


Figure 2: State diagram: Tournament status

- **Enrollment in battles:** the students have time until a predefined deadline to present a team for the battle and set up a workflow on the generated repository. They must insert a unique name for the team in the tournament and must add each other using their institutional email addresses. The rank of the team is calculated as the sum of the individual ranks and the group is accepted if it can set up the correct workflow before the enrollment deadline.

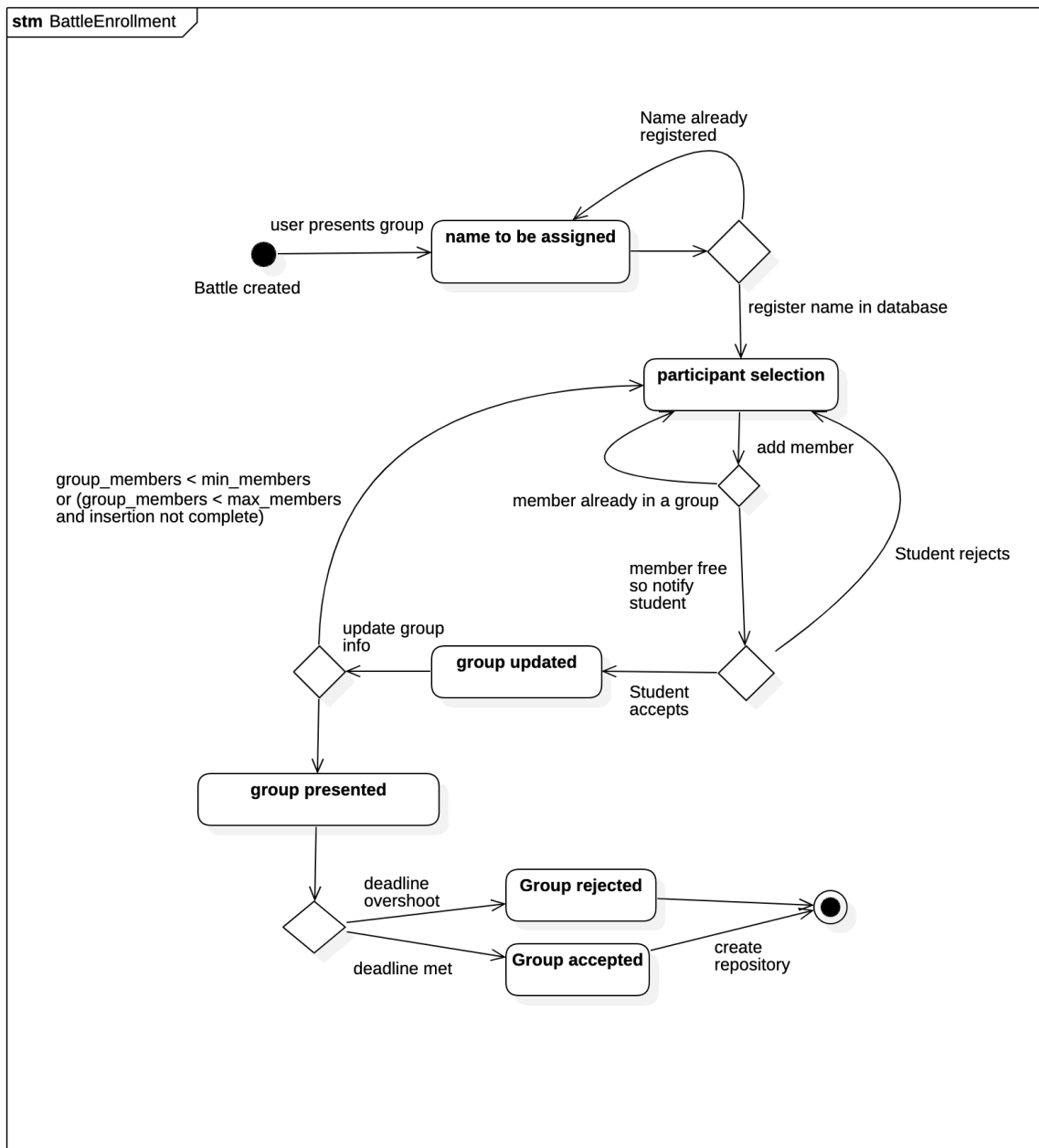


Figure 3: State diagram: Enrollment in battles

- **Students provide a solution:** The state "Kata to be resolved" represents the initial state, in which the challenge is published and is awaiting resolution from the students. When a student group pushes the code to GitHub, if the solution is provided within the battle deadline, the system automatically scores the solution by performing some automatic tests. Otherwise, the solution will be rejected because the time to solve the kata has elapsed. Additionally, if the instructor assigns manual scores to the battle, the solution will undergo manual evaluation by the instructor once the battle concludes.

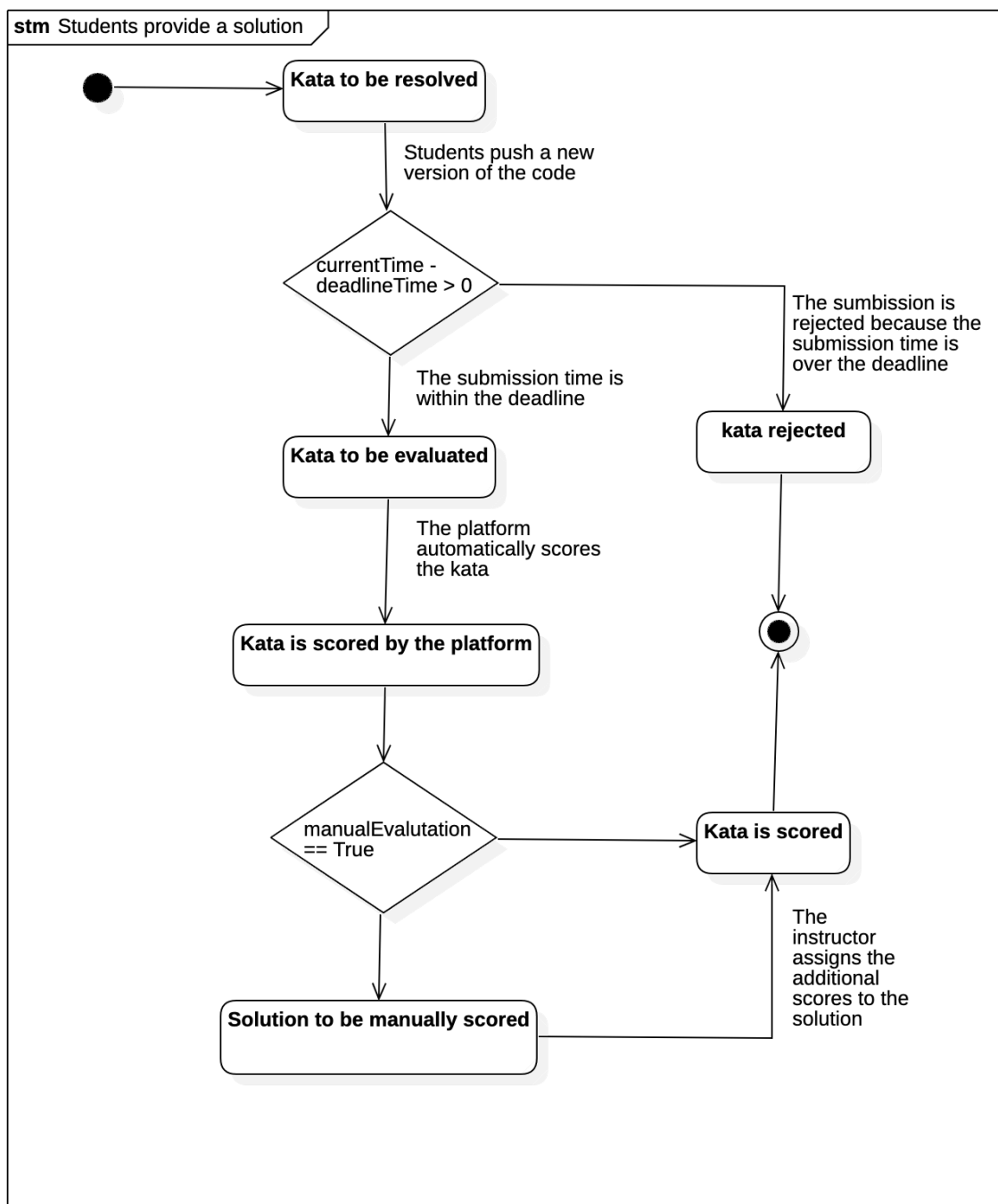


Figure 4: State diagram: Students provide a solution

2.2 Product functions

PF1. Signup: the function will be available to both the instructors and the students, allowing them the possibility to register to the platform. The system will ask the user to clarify whether he/she wants to register as a student or as an instructor. The user is required to provide his/her institutional email, and after that the system will perform a check on the domain of such email, to verify that the type of user that wants to register (instructors must have a different type of domain: name.surname@polimi.it; with respect to the students: name.surname@mail.polimi.it). After, the system will send an email to the user containing a link to an internet page, where it will be possible to verify the email and will be prompted to insert a password to complete the process.

PF2. Login: The system allows the already registered users to login into the platform using the credentials(email and password) provided during the signup phase. The instructor and students will access from the same page and will be then the platform to provide different views and authorizations based on the type of user. In particular, the student can only enroll in a tournament and participate in battles by joining a team, while the instructor can also create and manage tournaments and battles.

PF3. Manage a Tournament: the system allows the instructor to create a tournament by inserting a unique name for the tournament and specifying the deadline for the tournament enrollment. After the creation, the students subscribed to the platform will automatically receive a notification about the tournament creation. After the creation of the tournament, the instructor is able to grant the authorization for managing it to other instructors registered to the platform and the system allows the instructors the possibility of inserting battles after the deadline for registration to the tournament and the possibility to close it.

PF4. Manage a Battle: the system allows the instructor to create a new battle by inserting the following data: To create a new battle, an educator uses the CKB platform to perform the following steps:

- Upload the code kata (description and software project, including test cases and build automation scripts),
- Set minimum and maximum number of students per group,
- Set a registration deadline,
- Set a final submission deadline,
- Set some additional scoring criteria.

After creating the battle, the platform will create a GitHub repository through the GitHub API containing all the documents related to the battle and a configuration file that students can use to set up the required GitHub action. When students join a battle, the KCB platform will invite students to fork the previously created repository. Upon setting these initial parameters, the educator can further manage the battle through the following functionalities: send notifications to students regarding the registration opening, monitor the groups' progress during the battle, and send announcements or updates related to the battle.

PF5. Kata evaluation: the platform, through GitHub actions, is notified when a push action occurs on the student's group repository and each time the platform will perform the following sequence of actions:

- (a) pull the latest version of the code from the repository
- (b) execute the test cases associated to the kata(s) being solved
- (c) performs static analysis on the code

- (d) update the battle rank of the group
- (e) allows students and instructors to view the updated rank

the following actions are run continuously during the battle and push actions are allowed and valid for the rank creation until the specified deadline. After the deadline the system allows instructors to manually assign some additional scores to the single groups. Once such evaluation is terminated, the system allows the instructors to close the battle and the tournament score is updated.

2.3 User characteristics

- **Student:** A student is a user who belongs to the institute that hosts the CKB platform. He/She has an institutional e-mail address provided by his/her institute. A student can register to the platform, join teams, and enroll in tournaments. He/She can also monitor his/her scores on the platform. A student can invite other students to join the team.
- **Instructor:** An instructor is a user who works as an instructor in the institute that hosts the CKB platform. He/She has an institutional e-mail address provided by his/her institute which has a different domain from the students' emails. An instructor can create and manage tournaments and battles and can also delegate other colleagues allowing them to manage tournaments and assign manual scores to teams or add new battles.

2.4 Assumptions, Dependencies, and Constraints

2.4.1 Privacy policy

The platform complies with the GDPR regulation for data collection and implements a low-data collection policy by storing just the institutional email of the students and instructors and an associated encrypted password for the account.

2.4.2 Domain assumptions

- DA1.** Users have an internet connection.
- DA2.** The notifications sent to the students arrive at a reasonable time before the deadline for subscriptions.
- DA3.** The students write their solutions in a programming language for which test cases are provided by the instructor.
- DA4.** The students follow a test-first approach
- DA5.** The students are able to form groups outside the platform
- DA6.** Users have an institutional email address complying with the whitelist of domains specified by the platform.
- DA7.** Instructors' and students' emails have a different domain.
- DA8.** CKB platform is an open-source project that can be cloned and hosted by each university. So that they can specify the whitelist of email domains that can subscribe to the platform.
- DA9.** During the entire duration of the battle, the GitHub services are fully operative.
- DA10.** Students have a valid GitHub account.
- DA11.** Students are able to fork the repository and set up the GitHub action that notifies the CKB platform when a new PUSH is made.

2.4.3 Dependencies

The system relies on external factors for the correct functioning. The main external actor is GitHub, where repositories are created, forked, and through the GitHub Action service, the platform is notified about the push actions performed on the repositories.

3 Specific Requirements

3.1 External interface requirements

3.1.1 User interfaces

This section of the document presents the Web platform UI: the views both the students and the instructors will see when they login to the platform. The student dashboard (fig 5) gives an overview of the activities related to the CKB: the student can look at which Battles is enrolled, and can enroll in a battle. Furthermore, he/she can have an overview of the Katas he/she should submit and view the scores of the previously submitted ones. The instructor dashboard (fig 6) gives the instructors an overview of the battles and tournaments created inside the CKB platform. The instructor can create a Battle/Tournament using the "create battle" form (fig 7), in which he/she can specify all the parameters of the battle such as the minimum number of students per team or the tournament in which the battle is part. The instructor, for each battle created, should upload at least one kata.

Dashboard Carla Lombardi Student

2 Katas to submit

2 Enrolled battles

1 Enrolled tournaments

Katas

Kata Name	Status	Submission deadline	Team	Battle	Toolbar
Mastering the Fibonacci s...	Open to submission	30/12/2023	ByteBrawlers	Fibonacci	Kata description
Enigma machine emulator	Submitted	18/12/2023	CodeNinjas	Enigma	View score

Battles

Kata Name	Status	Enrollment deadline	Katas to be resolved	Team	Toolbar
Fibonacci	Enrollment open	22/12/2023	7	Create/Join a Team	Enroll
Enigma	Closed	08/08/2023	3	CodeNinjas	View final rank

My teams View ranks

Figure 5: User interface: Student dashboard

Dashboard Giovanni Ferrari Instructor

4 Active battles

2 Active tournaments

26 Solutions to be evaluated

Battles + Create a new battle

Kata Name	Status	Enrollment deadline	Enrolled teams	Submitted solutions	Toolbar
Fibonacci	Open	22/12/2023	5	3	View details
Enigma	Closed	18/12/2023	7	5	Score solutions

Tournaments + Create a new tournament


Kata Name	Status	Enrollment deadline	Enrolled teams	Battles	Toolbar
School competition	Open	01/12/2023	7	5	View details
Olimpiadi dell'informatica	Closed	08/08/2023	3	4	View final rank

Manage teams Manage educators View ranks

Figure 6: User interface: Instructor dashboard


← → ↻ <https://ckb.pollimi.it/create-battle> ✕

Create a battle

Giovanni Ferrari
Instructor 

Battle name

Select a Tournament

 Enrollment deadline

Minimum team members

Max team members

Katas

<input checked="" type="checkbox"/> Kata Name	Submission deadline	Toolbar
<input checked="" type="checkbox"/> Dynamic Deciphering Deligh	12/01/2024	<input type="button" value="Edit Kata"/>
<input checked="" type="checkbox"/> Graph Guru Gauntlet	14/02/2024	<input type="button" value="Edit Kata"/>

Figure 7: User interface: Instructor interface to create a battle

3.1.2 Hardware interfaces

The users approaching the platform need a desktop or laptop computer with a working and stable internet connection.

3.1.3 Software interfaces

The system requires some software interfaces to provide its services. Below are reported the most significant:

- **Web Browser:** having a Web Browser(Chrome, DuckDuckGo, Mozilla Firefox) is required in order to access the platform.
- **GitHub API:** the CKB platform uses the GitHub API to provide services related to creating and managing the battles' repositories.
- **GitHub Action service:** the service provides a trigger that the platform can use to be notified every time a user pushes on a GitHub repository. The CKB platform exposes an API that GitHub action can call to communicate with the system.
- **Notification service API:** the system uses an external API to send notifications to users. The notification can be an e-mail (the system needs a transactional email API like Mailchimp) or a push notification on a smartphone or web browser (the system needs a notification service like OneSignal)

3.1.4 Communication interfaces

We require a stable internet connection for the communication happening between the server and the devices.

3.2 Functional requirements

- R1.** The system allows students to sign-up.
- R2.** The system allows instructors to sign-up.
- R3.** The system allows students to log-in.
- R4.** The system allows instructors to log in.
- R5.** The system allows instructors to create a tournament.
- R6.** The system allows the instructors to insert the deadline for the enrollment in the tournament.
- R7.** The system allows the instructors to insert the name for the tournament.
- R8.** The system allows instructors to close a tournament.
- R9.** The system allows instructors to create a battle in a tournament.
- R10.** The system allows instructors to manage a battle in a tournament.
- R11.** The system allows students to form teams.
- R12.** The system allows instructors to give access to peers to the tournament management.
- R13.** The system provides visibility of tournament ranks to all platform users.
- R14.** The system creates a new repository for each created battle.

- R15.** The system automatically evaluates submissions based on functional aspects, timeliness, and code quality.
- R16.** The system allows students to FORK the provided repository.
- R17.** The system allows students to provide solutions to katas within the deadline through COMMIT actions.
- R18.** The system automatically updates students' rank based on the number of correct solutions provided and the manual evaluation.
- R19.** The system allows instructors to provide test cases in different programming languages to katas.
- R20.** The system allows instructors to score students' solutions manually.
- R21.** The system allows students to join battles.
- R22.** The system is informed every time a PUSH action is performed on a repository within the deadline of the battle
- R23.** The system performs a PULL action every time a PUSH has occurred

3.2.1 Use Cases and Use Case Diagrams

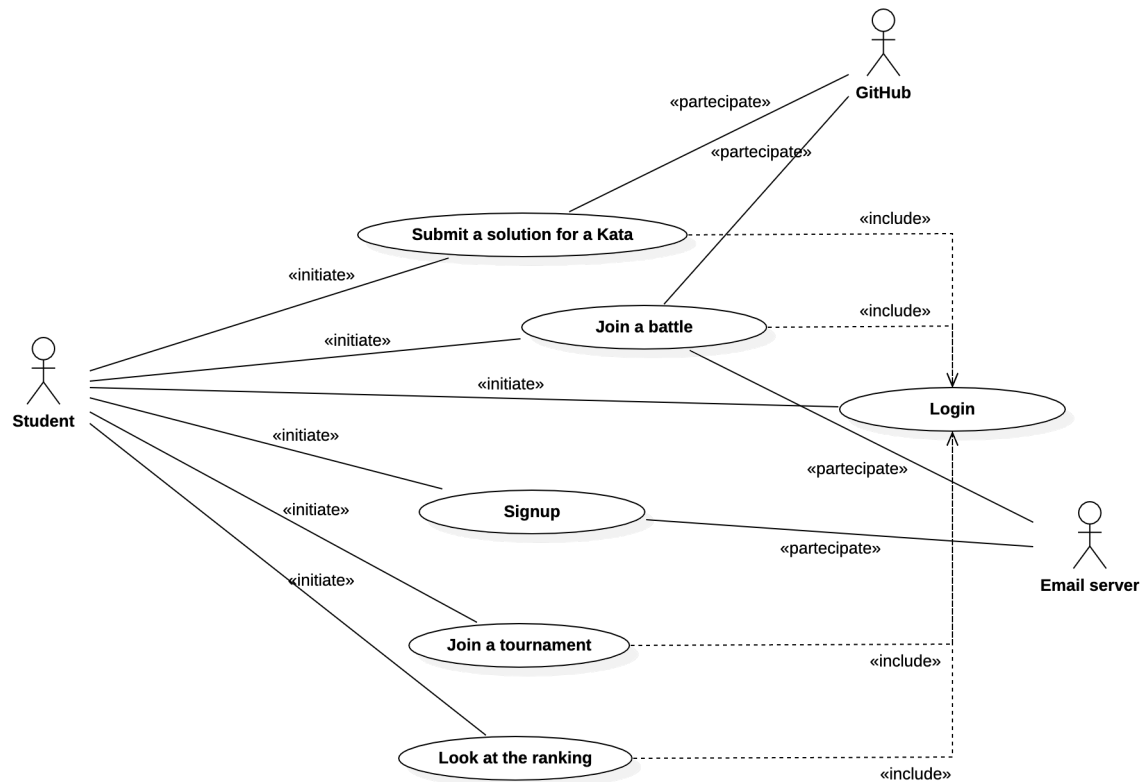


Figure 8: User case diagram: Student

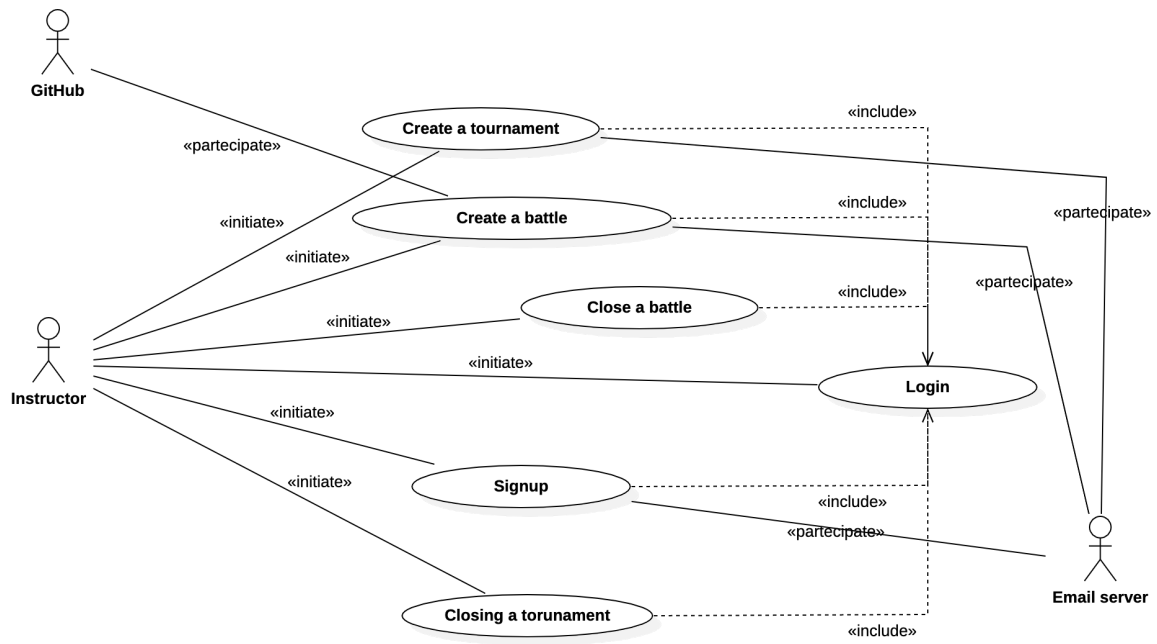


Figure 9: User case diagram: Instructor

UC1. Login user

Name	login user
Actors	Student or Instructor
Entry condition	Student already registered
Events	<ul style="list-style-type: none"> a) User inserts credential in the proper input fields(email, password) b) The system checks the validity of such information c) The system redirects the user to his/her personal page
Exit condition	User is logged in
Exceptions	Invalid credentials

UC2. Sign-up Student

Name	Sign-up Student
Actors	Student, Email server (external)
Entry condition	Student is not registered
Events	<ul style="list-style-type: none"> a) The student inserts his/her email provided by the institute and his/her data in the proper input fields. b) The system checks if the email belongs to the whitelist of email domains and if the email has never been used. c) The system sends a confirmation email to the student with a link to set the account password. d) The system redirects the student to a confirmation page.
Exit condition	The student is registered and is waiting to confirm his e-mail address
Exceptions	The student's email is already registered or the domain of the email provided is not included in the whitelist of the students' email domains

UC3. Sign-up Instructor

Name	Sign-up Instructor
Actors	Instructor, Email server (external)
Entry condition	Instructor is not registered
Events	<ul style="list-style-type: none"> a) The instructor inserts his/her email provided by the institute and his/her data in the proper input fields. b) The system checks if the email belongs to the whitelist of email domains and if the email has never been used. c) The system sends a confirmation email to the instructor with a link to set the account password. d) The system redirects the instructor to a confirmation page.
Exit condition	The instructor is registered and is waiting to confirm his e-mail address
Exceptions	The instructor's email is already registered or the domain of the email provided is not included in the whitelist of the instructors' email domains

UC4. User confirms the email address

Name	User confirm the email address
Actors	Instructor and student
Entry condition	User is already registered and the registered email is correct
Events	<ul style="list-style-type: none"> a) The user opens the link in the sign-up confirmation email. b) The system checks if the link is not expired and redirects the user to the "set password" page. c) The user inserts a secure password in the proper field. d) The system checks if the password meets the security standard and stores the encrypted password on the database. e) The user is redirected to the homepage of the platform as a logged-in user.
Exit condition	The user is registered and logged in to the platform.
Exceptions	The confirmation link is invalid or expired, the password doesn't meet the security requirements.

UC5. Tournament Creation

Name	Tournament Creation
Actors	Instructor, Email Server (external)
Entry condition	instructor registered to platform
Events	<ul style="list-style-type: none"> a) Instructor logins b) Instructor begins creation operation c) Instructor inserts the tournament name, an enrollment deadline d) System notifies all students registered to CKB about the tournament creation
Exit condition	Tournament created
Exceptions	The name for such tournament is already present or the deadline is before the date of creation

UC6. Tournament Close

Name	Tournament Close
Actors	Instructor
Entry condition	instructor registered to platform; tournament created
Events	<ul style="list-style-type: none"> a) Instructor logins b) Instructor finds specific tournament c) Instructor clicks on 'close tournament' d) System impedes the instructor from creating new battles within that tournament e) Instructor can manually assign scores to single groups through the platform f) Instructor clicks on 'finalize scores' g) System finalizes the ranks of the students h) System allows students to see their score
Exit condition	Tournament closed; student ranks updated
Exceptions	The name for such tournament is not present or the tournament has already been closed

UC7. Battle Creation

Name	Battle Creation
Actors	Instructor, Students, GitHub(External), Email Server(External)
Entry condition	instructor registered to platform and tournament created
Events	<ul style="list-style-type: none"> a) Instructor logs in b) Instructor begins creation operation c) Instructor inserts the battle name, an enrollment and termination deadline, specifies the minimum and maximum number of students d) Instructor creates katas to be inserted into the battle, specifying the text and providing test cases for different languages e) System creates a repository on GitHub and sets it private f) System notifies all students registered to CKB about the battle creation
Exit condition	Battle created
Exceptions	The name for such battle is already present; the deadlines are not met;

UC8. Battle Close

Name	Battle Close
Actors	Instructor, Students
Entry condition	instructor registered to the platform, the deadline expired
Events	<ul style="list-style-type: none"> a) Instructor logs in b) Instructor views the battle she/he wants to close c) Instructor clicks on 'close battle' for the specified battle d) System interrupts connection to forked repositories e) System notifies students about battle closing f) Instructor can manually assign scores to single groups through the platform g) Instructor clicks on 'finalize scores' h) System finalizes the ranks of the students and updates the tournament rank i) System allows students to see their tournament score
Exit condition	Battle closed; rank updated
Exceptions	The deadline has not yet expired; the specified battle is already closed or not present

UC9. Joining a battle

Name	Joining a battle
Actors	Students, GitHub(external), Email Server(external)
Entry condition	students registered to the platform; students notified about battle creation
Events	<ul style="list-style-type: none"> a) Students login. b) Students form a group outside the platform c) Student team leader opens CKB d) Student presents group to platform e) System sends a confirmation email to invited students f) Student team members accept the invitation g) System registers the team if deadline not yet expired h) System detects deadline expiration i) System sets battle repository to public j) System sends the link of the battle repository to students k) The group forks the main repository and sets up a workflow through GitHub Actions and API calls
Exit condition	The groups are registered and workflow is setup
Exceptions	The deadline of enrollment is over

UC10. Students commit a battle solution to GitHub

Name	Students commit a battle solution to GitHub
Actors	Students, GitHub (external)
Entry condition	The student is enrolled in a tournament, he/she has forked the GitHub repository and properly set up the GitHub action
Events	<ul style="list-style-type: none"> a) Students commit the solution of the kata battle on GitHub. b) Github runs the GitHub action that notifies the CKB platform that a new solution has been published. c) The system pulls the solution and runs automatic tests on the code to score the solution. d) The system computes the scores and stores them.
Exit condition	The solution is scored and users can look at the scores.
Exceptions	The commit time is over the kata submission deadline, the submitted code doesn't run due to compiling errors.

3.2.2 Sequence diagrams

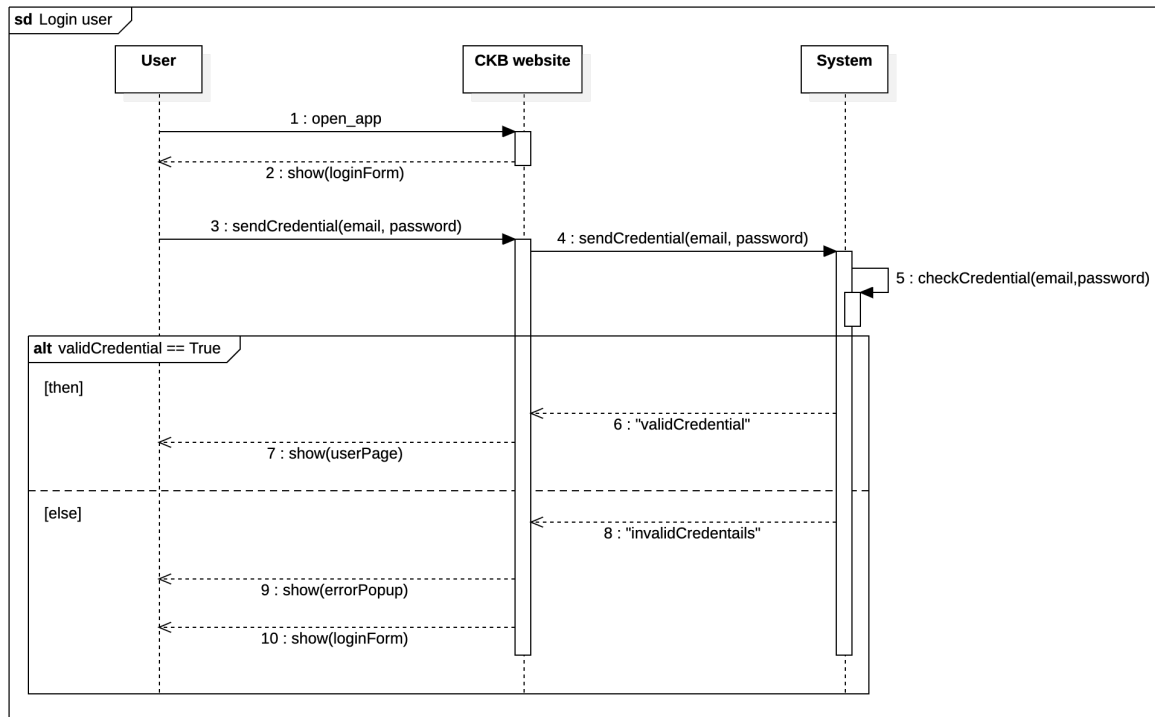


Figure 10: Sequence diagram UC 1: Login user

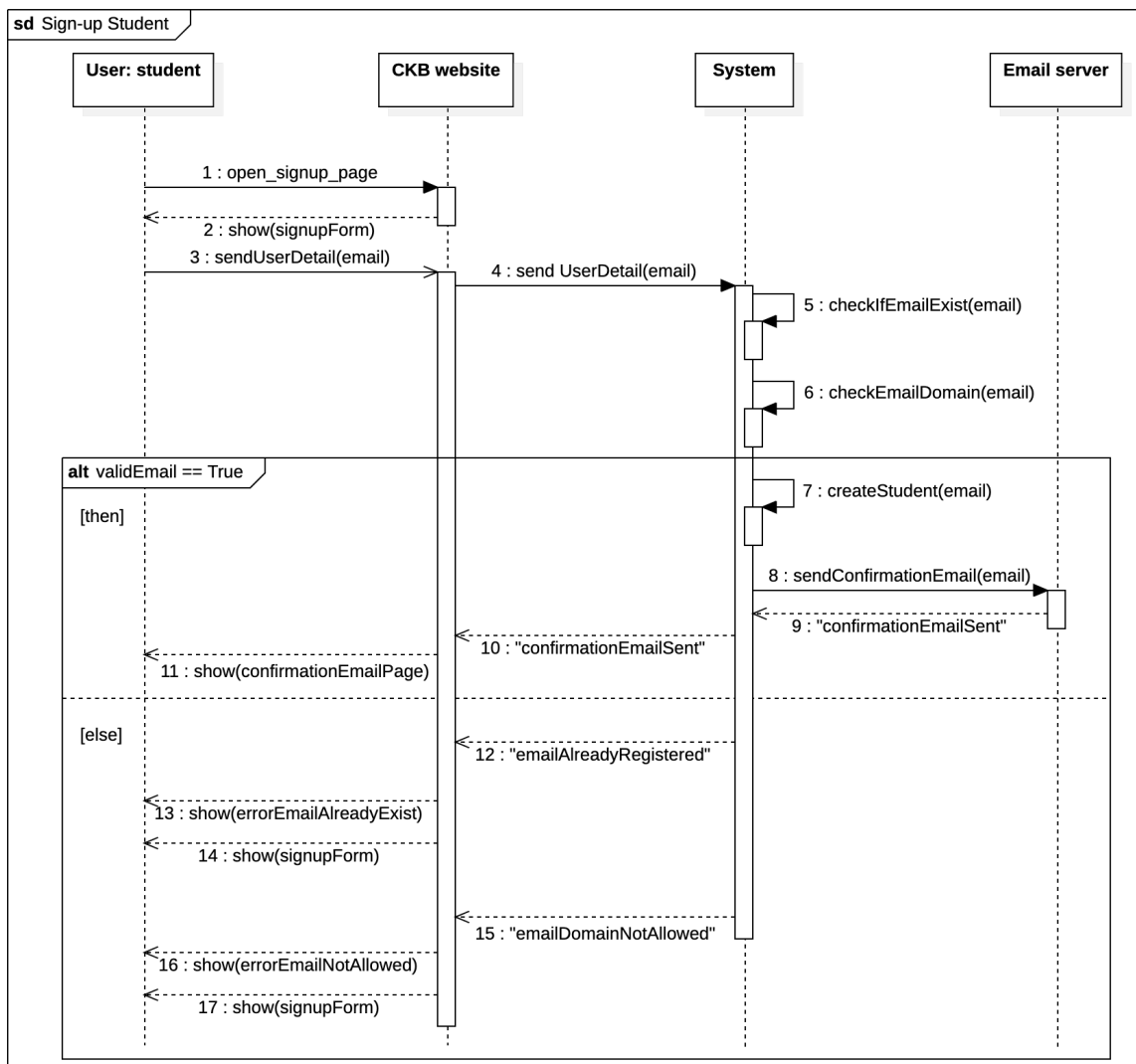


Figure 11: Sequence diagram UC 2: Sign-up Student

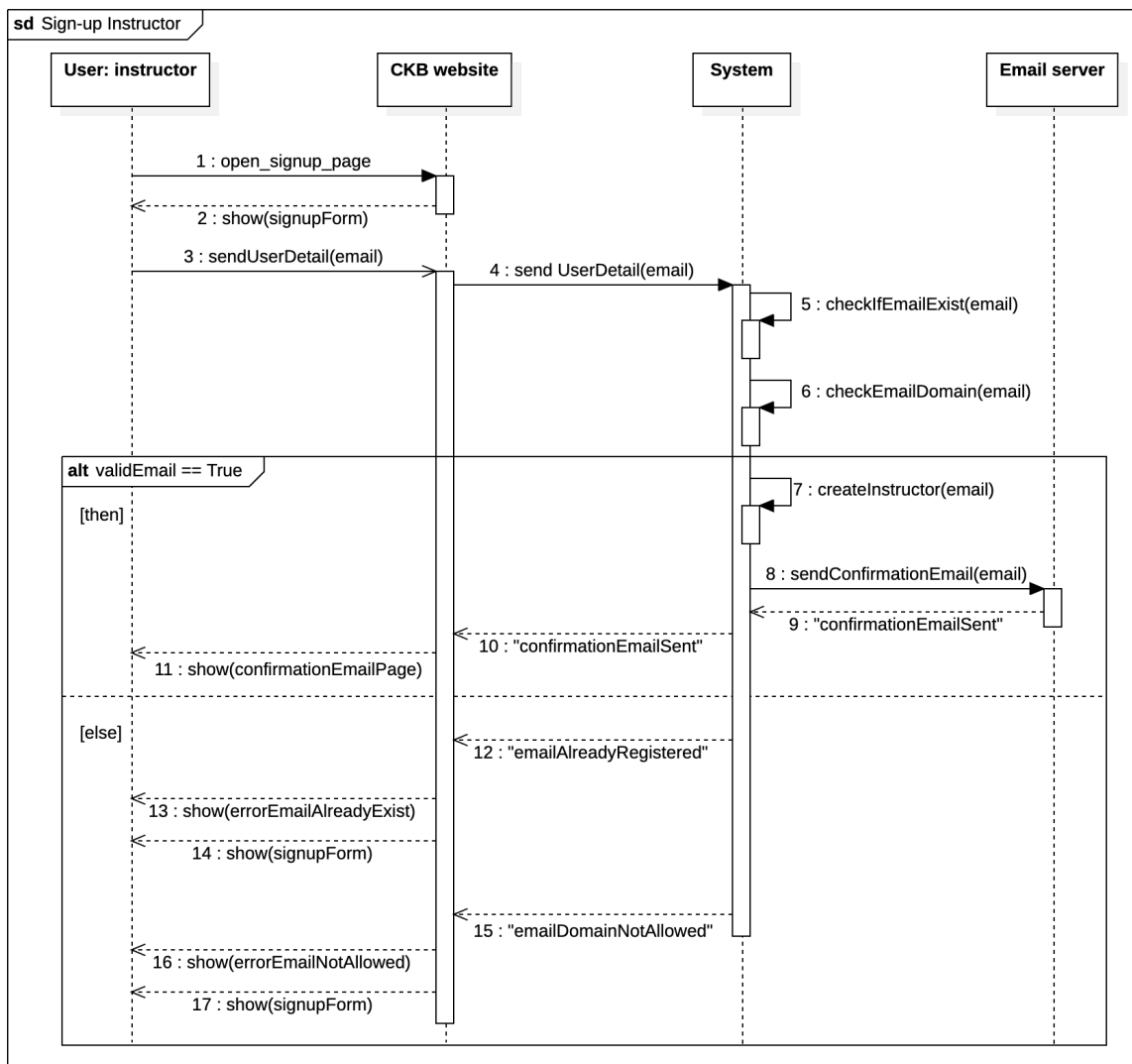


Figure 12: Sequence diagram UC 3: Sign-up Instructor

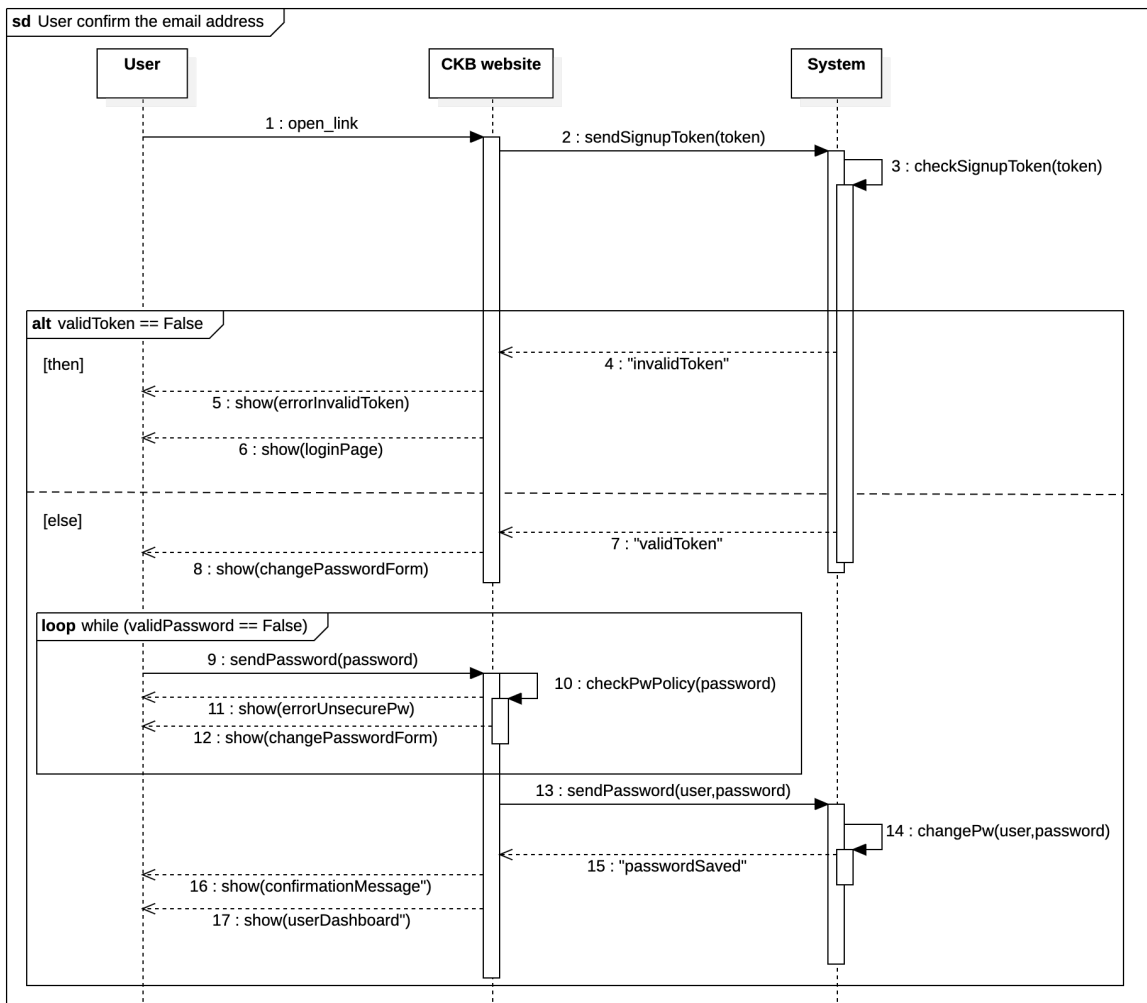


Figure 13: Sequence diagram UC 4: User confirms the email address

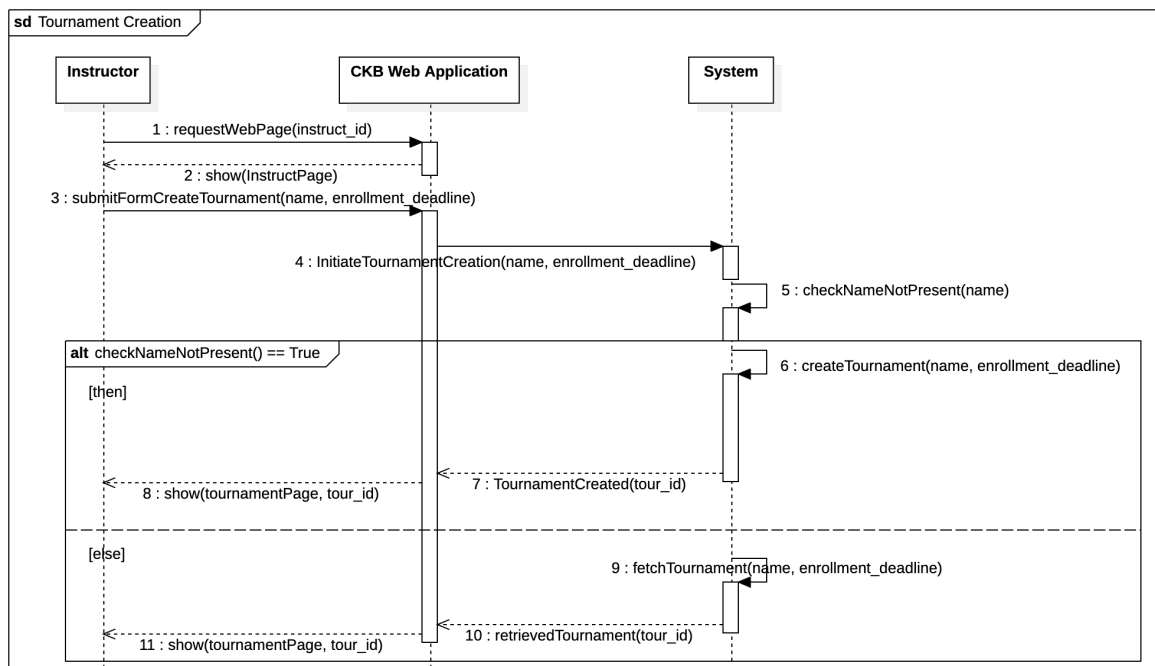


Figure 14: Sequence diagram UC 5: Tournament Creation

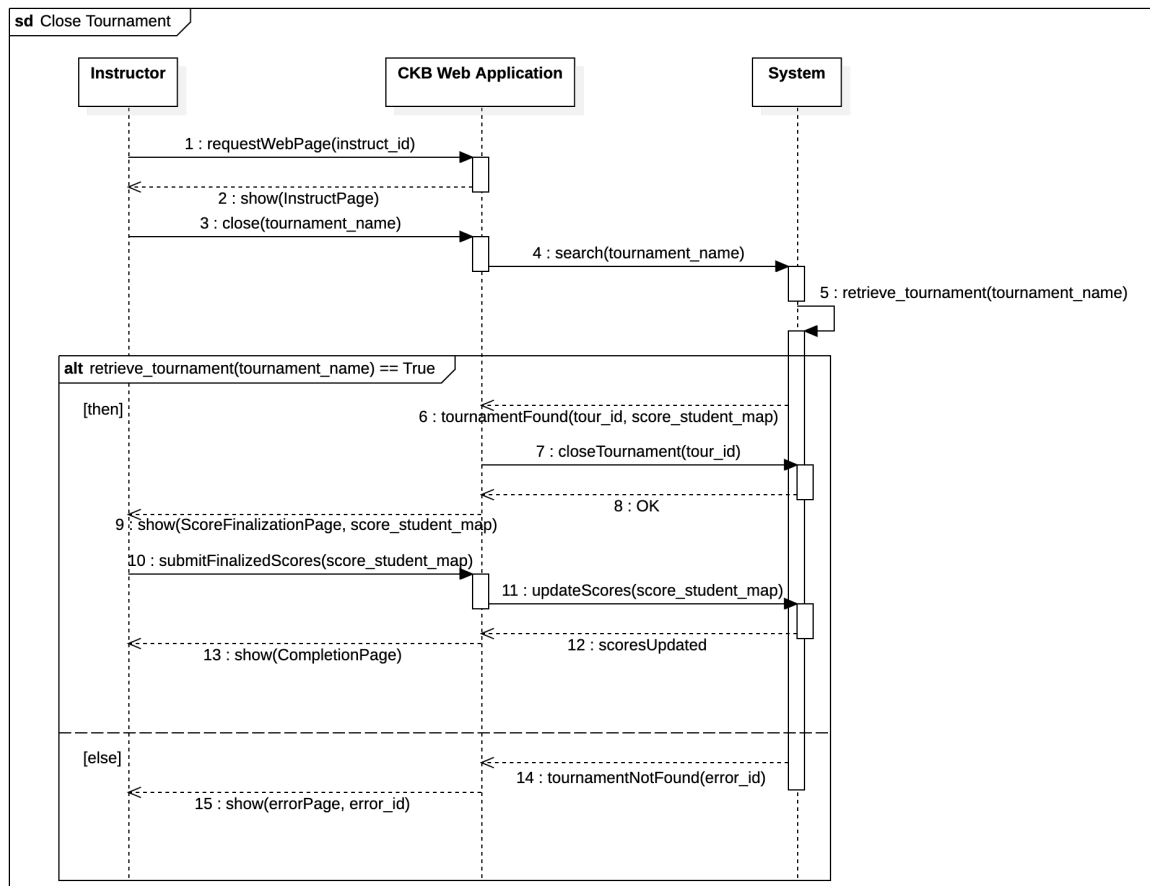


Figure 15: Sequence diagram UC 6: Tournament Close

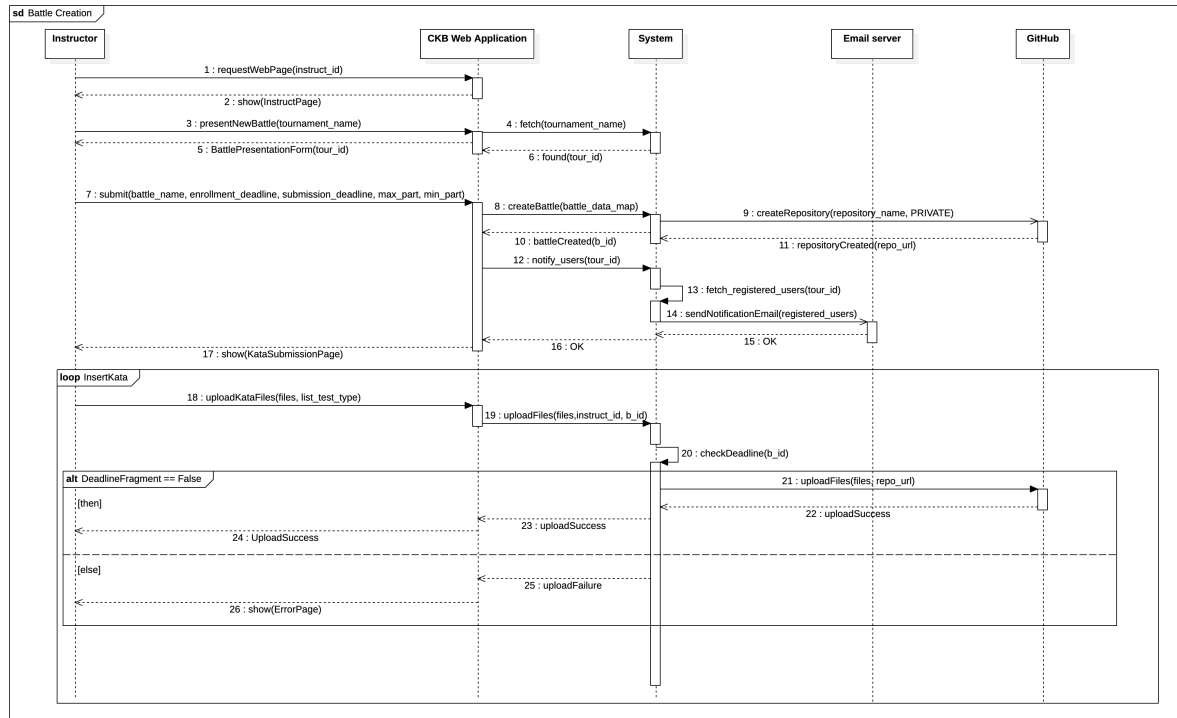


Figure 16: Sequence diagram UC 7: Battle Creation

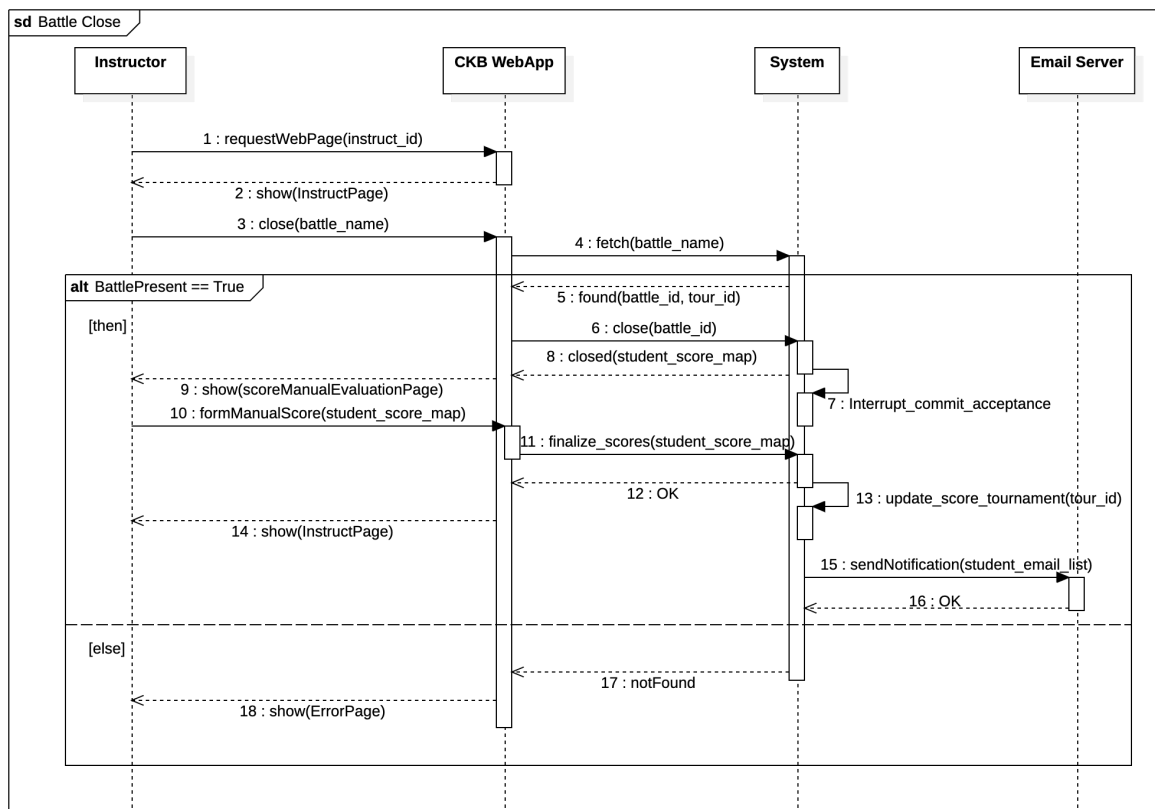


Figure 17: Sequence diagram UC 8: Battle Close

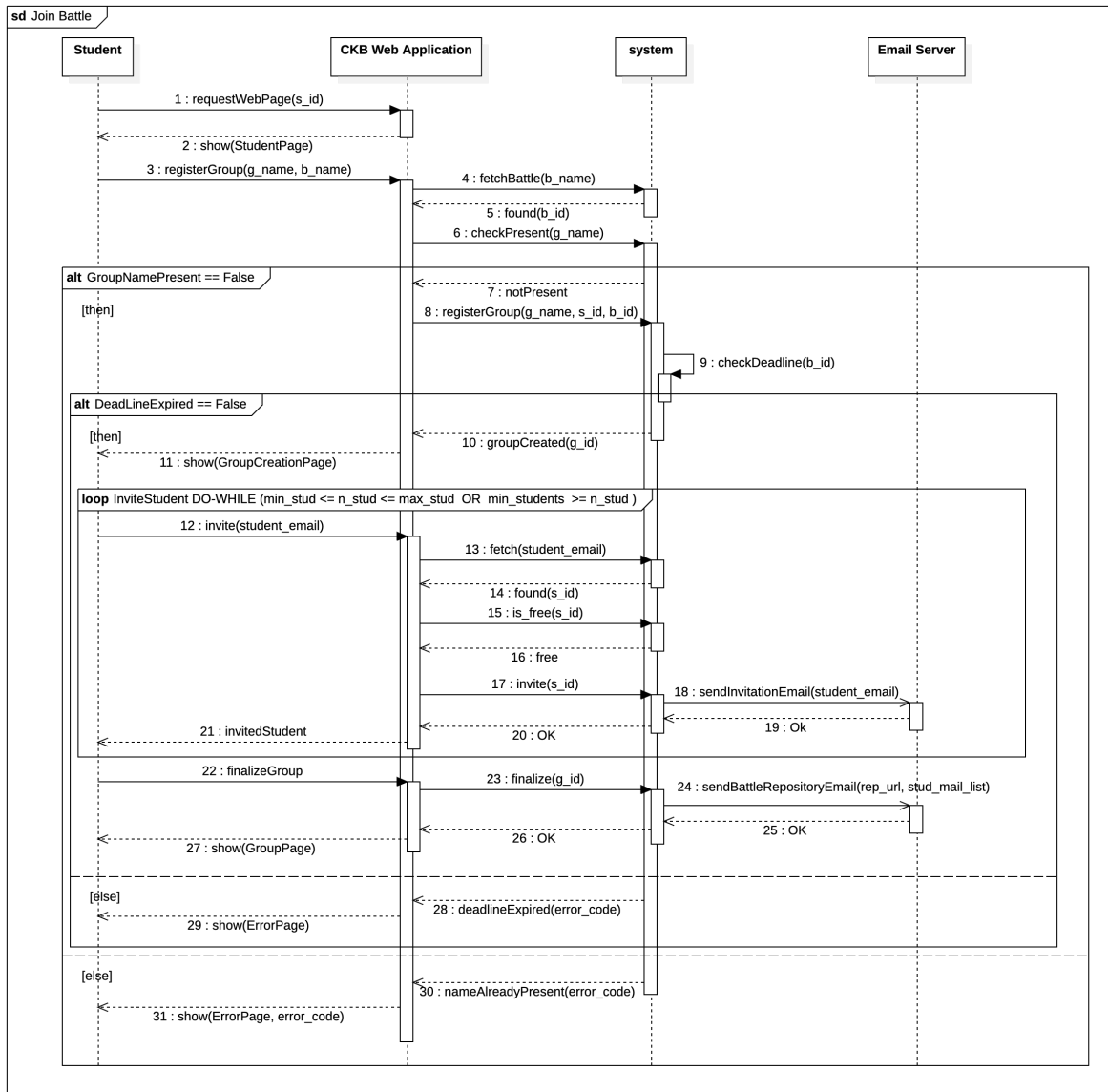


Figure 18: Sequence diagram UC 9: Joining a Battle

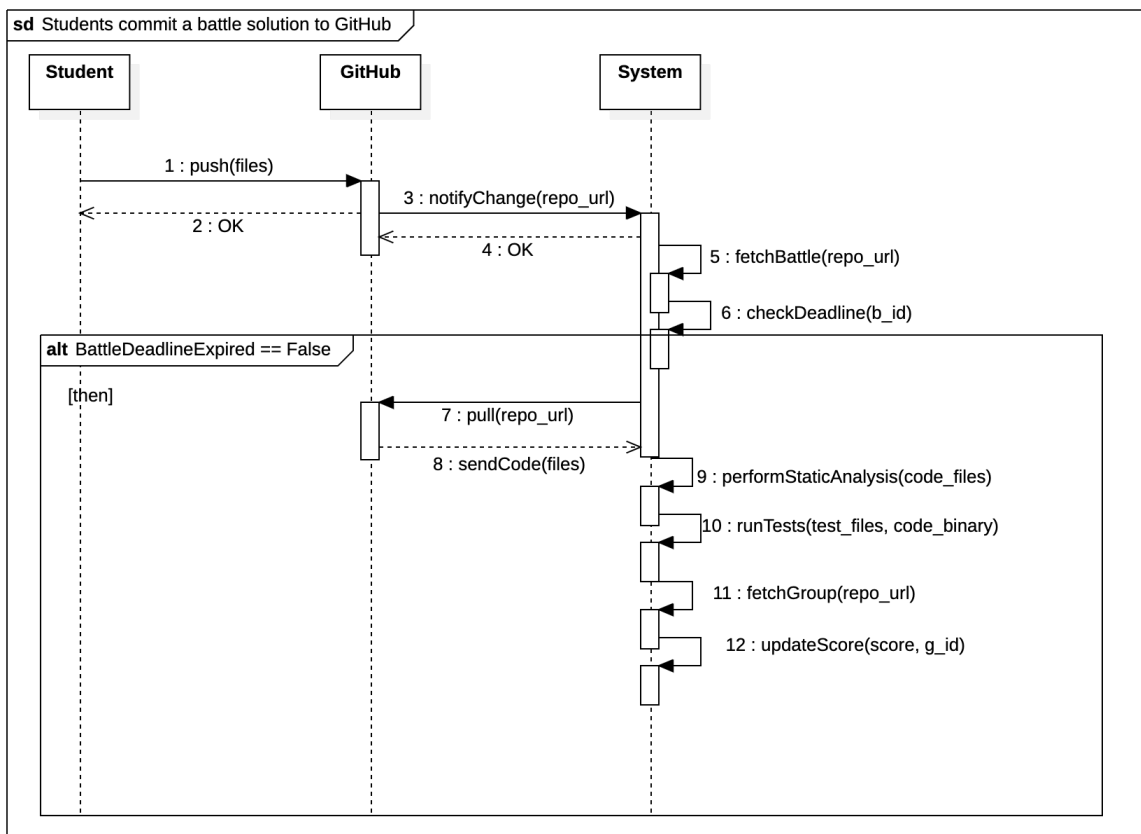


Figure 19: Sequence diagram UC 10: Students commit a battle solution to GitHub

3.2.3 Requirements Mapping

[G 1] Students are able to practice on katas by participating in tournaments in groups of predefined minimum and maximum size.	
R 1 The system allows students to sign-up. R 3 The system allows students to log-in. R 11 The system allows students to form teams. R 11 The system provides visibility of tournament ranks to all platform users. R 16 The system allows students to FORK the provided repository. R 18 The system automatically updates students' rank based on the number of correct solutions provided and the manual evaluation. R 21 The system allows students to join battles.	DA 1 Users have an internet connection. DA 2 The notifications sent to the students arrive at a reasonable time before the deadline for subscriptions. DA 5 The students are able to form groups outside the platform. DA 6 Users have an institutional email address complying with the whitelist of domains specified by the platform. DA 7 Instructors' and students' emails have a different domain. DA 8 CKB platform is an open-source project that can be cloned and hosted by each university. So that they can specify the whitelist of email domains that can subscribe to the platform.

[G 2] Students obtain an automatic evaluation of their proposed solutions.	
R 13 The system provides visibility of tournament ranks to all platform users. R 14 The system creates a new repository for each team that joins a battle. R 15 The system automatically evaluates submissions based on functional aspects, timeliness, and code quality. R 16 The system allows students to FORK the provided repository. R 17 The system allows students to provide solutions to katas within the deadline through COMMIT actions. R 18 The system automatically updates students' rank based on the number of correct solutions provided and the manual evaluation. R 22 The system is informed every time a PUSH action is performed on a repository within the deadline of the battle. R 23 The system performs a PULL action every time a PUSH has occurred.	DA 1 Users have an internet connection. DA 2 The notifications sent to the students arrive at a reasonable time before the deadline for subscriptions. DA 3 The students write their solutions in a programming language for which test cases are provided by the instructor. DA 4 The students follow a test-first approach. DA 8 CKB platform is an open-source project that can be cloned and hosted by each university. So that they can specify the whitelist of email domains that can subscribe to the platform. DA 9 During the entire duration of the battle, the GitHub services are fully operative. DA 10 Students have a valid GitHub account. DA 11 Students are able to fork the repository and set up the GitHub action that notifies the CKB platform when a new PUSH is made.

[G 3] Instructors are able to create, manage, and award tournaments.	
<p>R 2 The system allows instructors to sign-up.</p> <p>R 4 The system allows instructors to log in.</p> <p>R 5 The system allows instructors to create a tournament.</p> <p>R 6 The system allows the instructors to insert the deadline for enrollment in the tournament.</p> <p>R 7 The system allows the instructors to insert the name for the tournament.</p> <p>R 8 The system allows instructors to close a tournament.</p> <p>R 9 The system allows instructors to create a battle in a tournament.</p> <p>R 10 The system allows instructors to manage a battle in a tournament.</p> <p>R 12 The system allows instructors to give access to peers to the tournament management.</p> <p>R 19 The system allows instructors to provide test cases in different programming languages to katas.</p>	<p>DA 1 Users have an internet connection.</p> <p>DA 2 The notifications sent to the students arrive at a reasonable time before the deadline for subscriptions.</p> <p>DA 6 Users have an institutional email address complying with the</p> <p>DA 7 Instructors' and students' emails have a different domain.</p> <p>DA 8 CKB platform is an open-source project that can be cloned and hosted by each university. So that they can specify the whitelist of email domains that can subscribe to the platform.</p>

[G 4] Instructors are able to provide a manual evaluation of the proposed solutions.	
<p>R 2 The system allows instructors to sign-up.</p> <p>R 4 The system allows instructors to log in.</p> <p>R 11 The system allows students to form teams.</p> <p>R 20 The system allows instructors to manually score students' solutions.</p> <p>R 21 The system allows students to join battles.</p> <p>R 18 The system automatically updates students' rank based on the number of correct solutions provided and the manual evaluation.</p>	<p>DA 1 Users have an internet connection.</p> <p>DA 2 The notifications sent to the students arrive at a reasonable time before the deadline for subscriptions.</p> <p>DA 5 The students are able to form groups outside the platform</p> <p>DA 3 The students write their solutions in a programming language for which test cases are provided by the instructor.</p>

3.3 Performance requirements

The system must guarantee high-performance capabilities, supporting a substantial volume of simultaneously connected users and effectively managing concurrency control. It should expose an API endpoint for the GitHub Action so that the CKB platform will be notified every time a new commit is done. This API endpoint should return a response within the timeout time mentioned in the GitHub action documentation¹, otherwise, the GitHub action will be killed with an error. Additionally, the system must guarantee that the automatic scores of a solution are computed in a reasonable time so that students can follow a test-first approach by trying to improve their solution's scores.

3.4 Design constraints

3.4.1 Standards compliance

About the privacy of data, the CKB platform is subject to the General Data Protection Regulation (GDPR), a regulation in EU law on data protection and privacy for all individuals within the European Union (EU) and the European Economic Area (EEA). The system has to adopt international standards² about date and time use and representation.

3.4.2 Hardware limitations

The user (Student or Instructor) needs the following hardware requirements:

- A stable internet connection.
- A computer that runs an operating system with an internet browser installed.

3.5 Software systems attributes

3.5.1 Reliability

The system must operate without interruption for extended periods. To achieve fault tolerance, its back-end deployment should utilize replication and redundancy. Additionally, the system should maintain offline backups of data storage for use in disaster recovery in the event of data loss.

3.5.2 Availability

Considering that the CKB platform is not related to emergency or critical services, it is required to maintain a system availability of 99.9%. This implies that the Mean Time To Recovery (MTTR), or the average time required to restore service after a fault occurs, should be limited to approximately 0.365 days per year.

3.5.3 Security

The CKB platform does not store sensitive personal information about users; only email and password are collected during the sign-up process. Nevertheless, it is crucial to ensure proper data protection. Passwords in the data store must be encrypted, and they should never be transmitted in an unencrypted form. To prevent traffic sniffing and spoofing, all platform pages should be delivered over the HTTPS protocol, thereby safeguarding against cheating attacks and maintaining privacy and consistency. Additionally, prior to executing automatic code evaluation for new solutions submitted to the repository, the platform should incorporate a service that scans the code. This is to protect the system from attacks and errors that could exploit machine resources.

¹<https://docs.github.com/en/actions>

²Refer to the ISO 8601 standard <https://www.iso.org/iso-8601-date-and-time-format.html>

3.5.4 Maintainability

The system is required to ensure a robust maintainability standard. This necessitates the utilization of suitable design patterns and adherence to high-quality standards. The code should be thoroughly documented, and reliance on hard coding should be minimized. Additionally, a comprehensive testing routine must be established, covering a minimum of 75% of the entire codebase, with the exclusion of interface code.

3.5.5 Portability

The CKB platform should run on any operating system (like Windows, Mac OS, Linux, etc) that supports a web browser.

4 Formal Analysis Using Alloy

The following section represents the alloy model that models the CKB platform.

4.0.1 Alloy model

```
// The prototype instance for simplicity
sig User {
  name: one String
}

sig Student extends User {
  // A student can choose different teams for different battles in a tournament
  teams: set Team,
  battles: set Battle
}

sig Instructor extends User {
  // An Instructor can create many tournaments and battles
  tournaments: set Tournament,
  battles: set Battle
}

sig Tournament {
  // every tournament has a single creator and may have assistant instructors
  creator: one Instructor,
  assistants: set Instructor,
  battles: set Battle,
  students: set Student,
  var status: TournamentStatus
}

sig Battle {
  tournament: one Tournament,
  creator: one Instructor,
  assistants: set Instructor,
  kata: set Kata,
  teams: set Team,
  minStudents: one Int,
  maxStudents: one Int,
  registrationDeadline: one DateTime,
  submissionDeadline: one DateTime,
  scoreConfig: one ScoreConfig,
  var status: BattleStatus
}

sig Team {
  students: set Student,
  battle: one Battle,
  score: one Score,
  enrollmentTime: one DateTime
}

sig Score {
  functional: one Int,
  timeliness: one Int,
  quality: one Int,
  personal: lone Int
}

sig Kata {
  description: one String,
  testCases: set TestCase,
  buildScripts: set BuildScript,
  battle: one Battle
}

sig Solution {
  submissionTime: one DateTime,
  team: one Team,
  kata: one Kata,
```

```

    score: one Score,
    var status: SolutionStatus
}

// A DateTime signature to represent a point in time
sig DateTime {
    year: one Int,
    month: one Int,
    day: one Int,
    hour: one Int,
    minute: one Int
}

sig TestCase {}
sig BuildScript {}
sig ScoreConfig {}

enum BattleStatus {BatOpen, BatClosed}
enum TournamentStatus {TourOpen, TourClosed}
enum SolutionStatus {Submitted, Evaluated}

// ##### PREDICATES #####
↪ #####

// Check if a DateTime is before another
pred isBefore[d1, d2: DateTime] {
    d1.year < d2.year or
    (d1.year = d2.year and d1.month < d2.month) or
    (d1.year = d2.year and d1.month = d2.month and d1.day < d2.day) or
    (d1.year = d2.year and d1.month = d2.month and d1.day = d2.day and d1.hour < d2.hour)
    ↪ or
    (d1.year = d2.year and d1.month = d2.month and d1.day = d2.day and d1.hour = d2.hour
    ↪ and d1.minute < d2.minute)
}

// Close a Battle
pred closeBattle [b: Battle] {
    b.status = BatOpen
    b.status' = BatClosed
}

// Close a Tournament
pred closeTournament [t: Tournament] {
    t.status = TourOpen
    t.status' = TourClosed
}

// ##### FACTS #####

// A fact to ensure DateTime values are valid
fact validDateTime {
    all dt: DateTime | (dt.year > 0) and
    (dt.month ≥ 1 and dt.month ≤ 12) and
    (dt.day ≥ 1 and dt.day ≤ 31) and // it is not completely accurate since months have
    ↪ different number of days, but for easyness of notation we are not accounting
    ↪ for it
    (dt.hour ≥ 0 and dt.hour ≤ 23) and
    (dt.minute ≥ 0 and dt.minute ≤ 59)
}

// A fact to ensure Score values are valid
fact validScoreValues {
    all sc: Score | sc.functional + sc.timeliness + sc.quality + sc.personal ≤ 100 and
    sc.functional + sc.timeliness + sc.quality + sc.personal ≥ 0
}

// Teams can't enroll in a battle after the deadline
fact teamsCantEnrollAfterDeadline {
    all b: Battle, t: Team |
    (t in b.teams) implies isBefore[t.enrollmentTime, b.registrationDeadline]
}

```

```

// Students can be in one team in each battle
fact studentInOneTeamPerBattle {
  all s: Student | all disj t1, t2: s.teams | t1.battle ≠ t2.battle
}

// Solutions cannot be submitted after the deadline
fact solutionsCannotBeSubmittedAfterDeadline {
  all s: Solution | isBefore[s.submissionTime, s.kata.battle.submissionDeadline]
}

// Students cannot send solutions to a battle they are not enrolled in
fact studentsCanNotSendSolutionsIfNotEnrolled {
  all s: Solution | s.team in s.kata.battle.teams
}

//The solution time must be after the registration deadline
fact solutionTimeAfterRegistrationDeadline {
  all s: Solution | isBefore[s.kata.battle.registrationDeadline, s.submissionTime]
}

// The number of members of a team should be between the min and max value specified in
↪ the battle
fact numberTeamMembers {
  all t: Team | #t.students ≥ t.battle.minStudents and #t.students ≤ t.battle.
  ↪ maxStudents
}

// Battle will eventually be closed
fact battleEventuallyClosed {
  all b: Battle | always (b.status = BatOpen implies eventually b.status = BatClosed)
}

// Tournament will eventually be closed
fact tournamentEventuallyClosed {
  all t: Tournament | always (t.status = TourOpen implies eventually t.status =
  ↪ TourClosed)
}

// The Submission deadline cannot be before the registration deadline
fact submissionDeadlineAfterEnrollmentDeadline {
  all b: Battle | isBefore[b.registrationDeadline, b.submissionDeadline]
}

// Battle historically open
fact battleHistoricallyOpen{
  all b: Battle | always (b.status = BatOpen implies historically b.status = BatOpen)
}

// Tournament historically open
fact tournamentHistoricallyOpen {
  all t: Tournament | always (t.status = TourOpen implies historically t.status =
  ↪ TourOpen)
}

// Battle cannot be reopened
fact battleCannotBeReopened {
  all b: Battle | always ( b.status = BatClosed implies after always b.status =
  ↪ BatClosed)
}

// Tournament cannot be reopened
fact tournamentCannotBeReopened {
  all t: Tournament | always ( t.status = TourClosed implies after always t.status =
  ↪ TourClosed)
}

// Kata should have at least one test case
fact allKatasHaveTestCases {
  all k: Kata | some k.testCases
}

// Kata cannot have duplicated test cases

```



```

fact noDuplicateTestCasesInKata {
  all k: Kata | all disj tc1, tc2: k.testCases | tc1 ≠ tc2
}

// If a Battle is closed, it must have been closed sometime in the past
fact ifBattleClosedDidClose {
  all b : Battle | always (b.status = BatClosed implies once closeBattle[b] )
}

// If a Tournament is closed, it must have been closed sometime in the past
fact ifTournamentClosedDidClose {
  all t : Tournament | always (t.status = TourClosed implies once closeTournament[t] )
}

// Solution historically submitted
fact solutionHistoricallySubmitted{
  all s: Solution | always (s.status = Submitted implies historically s.status =
    ↪ Submitted)
}

// Solution will eventually be Evaluated
fact solutionEventuallyEvaluated {
  all s: Solution | always (s.status = Submitted implies eventually s.status =
    ↪ Evaluated)
}

// The creator of a tournament cannot be an assistant instructor
fact noAssistantCreator {
  all t: Tournament | t.creator not in t.assistants
}

pred showStudentSubmitsSolution {
  some s: Solution, st: Student, t: Team, b: Battle, k: Kata, currentTime: DateTime |
    st in t.students and
    t in b.teams and
    s.team = t and
    s.kata = k and
    k.battle = b and
    s.status = Submitted and
    isBefore[currentTime, b.submissionDeadline]
}

run showStudentSubmitsSolution for 5

```

4.0.2 Alloy relaxed version

For the sake of the simulation, we were forced to relax some temporal and numerical constraints. This is the updated version of the model.

```
// The prototype instance for simplicity
sig User {
  //name: one String
}

sig Student extends User {}

sig Instructor extends User {}

sig Tournament {
  // every tournament has a single creator and may have assistant instructors
  creator: one Instructor,
  assistants: set Instructor,
  battles: some Battle
}

sig Battle {
  tournament: one Tournament,
  katas: some Kata,
  teams: some Team,
  minStudents: one Int,
  maxStudents: one Int,
  registrationDeadline: one DateTime,
  submissionDeadline: one DateTime
}

sig Team {
  students: some Student,
  enrollmentTime: one DateTime
}

sig Score{
  score: one Int
}

sig Kata {
}

sig Solution {
  submissionTime: one DateTime,
  team: one Team,
  kata: one Kata,
  score: one Score
}

// A DateTime signature to represent a point in time
sig DateTime {
  time: one Int
}

// ##### PREDICATES #####
↔ #####

pred isBefore[d1, d2: DateTime]{
  d1.time < d2.time
}

// ##### FACTS #####

// A fact to ensure DateTime values are valid
fact validDateTime {
  all dt: DateTime | (dt.time > 0)
}

// A fact to ensure Score values are valid
fact validScoreValues {
  all sc: Score | sc.score ≤ 5 and
    sc.score ≥ 0
}
```

```

}

// Teams can't enroll in a battle after the deadline
fact teamsCantEnrollAfterDeadline {
  all b: Battle, t: Team |
    (t in b.teams) implies isBefore[t.enrollmentTime, b.registrationDeadline]
}

// Students can be in one team in each battle
fact studentInOneTeamPerBattle {
  all s: Student, b: Battle, disj t1, t2: b.teams | s in t1.students implies not s in t2.
  ↪ students
}

// Solutions cannot be submitted after the deadline
fact solutionsCannotBeSubmittedAfterDeadline {
  all s: Solution, b: Battle | s.kata in b.katas and isBefore[s.submissionTime, b.
    ↪ submissionDeadline]
}

// Students cannot send solutions to a battle they are not enrolled in
fact studentsCanNotSendSolutionsIfNotEnrolled {
  all s: Solution, b: Battle | s.kata in b.katas and s.team in b.teams
}

// The solution time must be after the registration deadline
fact solutionTimeAfterRegistrationDeadline {
  all s: Solution, b: Battle | s.kata in b.katas and isBefore[b.registrationDeadline, s
    ↪ .submissionTime]
}

// The number of members of a team should be between the min and max value specified in
  ↪ the battle
fact numberTeamMembers {
  all t: Team, b: Battle | t in b.teams and #t.students ≥ b.minStudents and #t.students
    ↪ ≤ b.maxStudents
}

// The Submission deadline cannot be before the registration deadline
fact submissionDeadlineAfterEnrollmentDeadline {
  all b: Battle | isBefore[b.registrationDeadline, b.submissionDeadline]
}

// The creator of a tournament cannot be an assistant instructor
fact noAssistantCreator {
  all t: Tournament | t.creator not in t.assistants
}

pred show{
  #Battle = 1 and
  #Tournament = 1 and
  #Kata = 5 and
  #Team = 2 and
  #Solution = 2 and
  #Student = 6
}

run show for 10 but exactly 1 Tournament, 5 Kata, 2 Team, 2 Solution, 6 Student, 1 Battle

```



Figure 20: Alloy model

5 Effort Spent

Here we report an estimation of the hours spent in the creation of the RASD document.

name	hours
Alberto Eusebio	32h
Marcello Martini	32h

Table 15: Effort spent in the creation of the document

References

- Diagrams made with StarUML
- Mockups made with Miro
- Alloy models made with Visual Studio Code (Alloy, Alloy VSCode)
- Alloy models ran and checked with alloytools