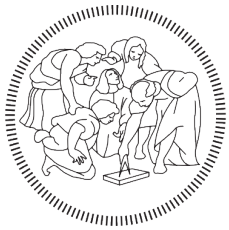


POLITECNICO
MILANO 1863

Requirement Analysis and Specification Document

CodeKataBattle

Authors

Name	Surname	ID
Alessandro	Saccone	11013852
Matteo	Sissa	10972783
Sara	Zappia	11016799

Version 1.0 - 22/12/2023

A.Y. 2023/2024

Contents

1 Introduction

1.1 Purpose

The main idea behind the CodeKataBattle platform comes from the term ”kata”, which is a Japanese word describing detailed patterns of movements repeated many times in order to memorize and master them. This kind of exercise is usually applied in learning karate. CodeKataBattle aims at exporting the same learning process also in the domain of coding and programming.

The software described in this document involves two main actors, which are students and educators. Educators can propose code kata battles (or simply battles for brevity), which are problems and teasers to be addressed by coding with a specific programming language, while students can engage in these battles in order to find solutions to them. Code kata battles are published in the context of tournaments in which students compete with each other.

The main goals of this system are tightly related to the users of the application. On the educator's side, CodeKataBattle allows to easily organize tournaments and coding battles, also for a great number of students. On the student's side, CodeKataBattle creates an enjoyable and playful environment to foster and improve the student's coding skills.

It is possible to summarize the most important objectives of the software to be developed in the following list of goals:

- G1) The system allows students to practice and improve their coding skills by developing solutions to code kata battles.
- G2) The system provides to educators a platform to publish code kata battles and easily manage tournaments and battles for students.
- G3) The system makes the task of coding battles' solutions as fun as possible, introducing elements like competitiveness, rankings, teamwork and badges (or rewards).

1.2 Scope

CodeKataBattle is a software application designed for students and educators in the context of coding challenges, called ”code kata battles”.

As a first basic feature of the application, CodeKataBattle allows both students and educators to log in the application using their GitHub credentials. Since all users are required to own a GitHub account in order to successfully interact with the CodeKataBattle platform, the authentication process is delegated to the GitHub system. When a user accesses CodeKataBattle for the first time, the system saves his/her username and generates an associated CodeKataBattle account.

Once signed in, educators have the possibility to create new tournaments, setting various parameters while doing it. Every tournament has a name, a description and a registration deadline, by which the students are asked to subscribe to the tournament if interested. When the tournament is generated on the platform, CodeKataBattle takes care of notifying all the students with an account on the system of the new tournament.

A tournament is composed of various code kata battles that educators can design and publish. The educator who creates a tournament can grant the permission of publishing battles inside that tournament to other educators on the platform.

In order to create a new battle in any tournament, an educator has to follow a series of steps. First off, every battle needs a name and a textual description of the problem to be solved. Moreover, the educator has to produce and upload on the platform the build automation scripts and the test cases that will be employed in order to build the students' code solutions and verify their correctness.
Educators are also asked to set some parameters for the battle, such as the registration deadline (by which students can sign up for the battle), the submission deadline (by which solutions for the battle can be handed in), the minimum and maximum number of students per team.
The system gives the option to decide whether a ”consolidation stage” is requested for the battle. This is a phase occurring after the submission deadline, during which the educator that created the battle is asked to analyze the students' code solutions and assign a personal score to them. This scores contribute to the final ranking of the battle, along with other parameters.
Finally, the setup of a new battle includes establishing which criteria the application should employ in order to automatically evaluate the students' code solutions while the battle is going on. For instance, valid criteria may be security, reliability, robustness and so on. These evaluations are carried out leveraging an external static analysis tool.

Once a battle is published, students that are subscribed to the tournament in which the battle resides are notified by the system of the new coding challenge and are able to join it by the registration deadline set by the educator. Students can join a battle in two ways: either forming a team with other students or on their own. The platform provides a way to allow a student to invite other students in the same tournament to join a battle together as a team.

When the registration deadline of a battle passes, CodeKataBattle automatically generates a new GitHub repository dedicated to that battle and sends a notification with the link to the remote repository to all the students participating in the battle.
At this point, students are expected to fork the main branch of this repository in order to create a personal repository to submit their code solutions. Besides, students should write an automated GitHub workflow (through GitHub Actions) in order to make the GitHub platform send a notification to CodeKataBattle every time a new commit is added on their repository.
Once this setup phase has been completed, students can write their solutions and push them on their GitHub repository. GitHub will notify CodeKataBattle and this notification will cause the CodeKataBattle application to pull the new source code from the specific GitHub repository where the new

commits occurred. The system can then analyze the new code solution in order to compute the score to assign to it. These calculations are based on a series of factors, such as the number of test cases (provided by the educator) passed, the time went by from the beginning of the battle and the analyses run on the source code leveraging an external static analysis tool, which will take into account only the evaluation criteria specified by the educator at battle creation time (for instance security, reliability...). Thanks to this procedure for automatic recalculation of the score of the students' solutions, CodeKataBattle will constantly maintain the ranking of teams participating in a battle up-to-date, so that students and educators involved in the coding challenge can see it evolving.

After the submission deadline of a battle, the system will no longer accept any additional solution for that battle, i.e. it will no longer listen to GitHub notifications for that battle. If the consolidation stage was required by the educator during the creation of the battle, CodeKataBattle will start a timer in order to set a specific time frame for the educator to evaluate the students' code solutions. When the timer goes off, if the educator hasn't assessed all the solutions yet, the system won't take into account the personal scores of the educator in order to compute the final ranking. On the other hand, if all teams received a personal evaluation from the educator, the scores assigned by the educator will be incorporated with the scores automatically calculated by the system while the battle was going on. In any case, when the consolidation time frame is over, the system is able to draw the final classification of all teams participating in the battle and publish it. CodeKataBattle will also take care of notifying all students members of these teams of the availability of the ranking on the platform.

Different from the ranking of a battle is the ranking of a tournament. Every tournament has a ranking of individual students (not teams). The position of each student in this ranking is the result of the sum of points accumulated by the student in the battles of the tournament the student has participated in. CodeKataBattle takes care of automatically updating the tournament ranking every time a battle of the tournament terminates.

The educator that created a tournament is also responsible for closing it. When a tournament gets closed, the final ranking of the tournament is shown and all students who signed up for that tournament are notified by the system.

Briefly taking a look at the application from the student's point of view, it is possible to summarize the functioning of the platform in a few steps. A student can display the list of all the tournaments available on CodeKataBattle and navigate it to see which ones s/he might be interested in. Then, s/he can subscribe to a tournament and consequently receive notifications by the system of any upcoming battle within that tournament. At this point, the student is able to join any battle in the tournaments s/he's subscribed to, either on his/her own or by inviting other students (always respecting the constraints on the minimum and maximum number of students per team set by the educator who created the battle). After receiving the link of the GitHub repository, students participating in a battle can then fork the main branch of the GitHub repository and start pushing their code solutions there...

Another relevant aspect of the CodeKataBattle system is related to the badges or rewards that are assigned to students participating in a tournament when the tournament ends. At tournament creation time, the educator is able to define these rewards based on specific achievements of students in the tournament, such as the number of commits performed, or the number of battles a student has joined. When the tournament is closed, CodeKataBattle calculates the set of students that are eligible for receiving a specific reward and assigns it to them. Every user on the CodeKataBattle platform has a personal profile and on the personal profile of each student, the list of badges earned by the student is made visible to all users of the platform. Badges and rewards play a relevant role in CodeKataBattle in order to create a playful environment to make coding problems solutions more enjoyable and entertaining.

1.2.1 World and Machine phenomena

This section summarizes the previous description into lists of phenomena (events) that occur in the world of interest for the system to be developed. Phenomena have to be interpreted simply as events occurring in the whole domain that is being analyzed in the document, so they have been stripped of any constraint (that will be better specified in the requirements section).
Phenomena can be divided into:

- **World phenomena**: events happening outside the system and on which the system has no control.
- **Machine phenomena**: events happening internally in the system, independent from the outside world.
- **Shared phenomena**: events that are have an influence on both the system and the world surrounding it. Usually they are further split into two classes:
  - **Machine controlled shared phenomena**: events triggered or initiated by the system with a relevant impact in the domain in which the system works.
  - **World controlled shared phenomena**: events initiated by entities of the world that are impactful for the system.

World phenomena

- WP1) **User** has a personal GitHub account.
- WP2) **Educator** wants to support students in bettering their software development skills.
- WP3) **Educator** comes up with new code kata battles to publish on the system.
- WP4) **Educator** writes the textual description of a new battle s/he wants to publish.
- WP5) **Educator** designs the test cases and the build automation scripts of a new battle s/he wants to publish.
- WP6) **Student** wants to improve their software development skills.
- WP7) **Student** forks the main GitHub repository that hosts a battle to upload his/her code solutions.
- WP8) **Student** writes an automated workflow through GitHub Actions to setup the interaction between GitHub and the system.
- WP9) **Student** works at developing the code solutions for a battle.
- WP10) **Student** pushes on the forked branch of the GitHub repository hosting a battle the code solutions, generating new commits.

Machine phenomena

- MP1) **The system** pulls new code solutions from GitHub.
- MP2) **The system** runs build automation scripts and test cases on new code solutions.
- MP3) **The system** calculates the percentage of test cases passed by a new code solution pulled from a GitHub repository.
- MP4) **The system** calculates the amount of time passed between the beginning of a battle and the moment in which a new submission has been submitted by a student through GitHub.
- MP5) **The system** runs static analysis on the new code solutions exploiting external static analysis tools.
- MP6) **The system** calculates the score to be assigned to a new code solution.
- MP7) **The system** computes the partial or final ranking of students for a tournament.
- MP8) **The system** computes the partial or final ranking of teams for a battle.
- MP9) **the system** computes the set of students that are eligible for receiving a badge (reward) at the end of a tournament.

World controlled shared phenomena

- WSP1) **User** logs in the system using his personal GitHub account.
- WSP2) **Educator** creates a new tournament with a description.
- WPS3) **Educator** sets the name of the tournament s/he wants to create.
- WSP4) **Educator** defines the badges (rewards) for the tournament s/he wants to create.
- WSP5) **Educator** grants permissions to create battles to other educators.
- WSP6) **Educator** creates a battle in a tournament.
- WPS7) **Educator** sets the name of the battle s/he wants to create.
- WSP8) **Educator** uploads on the system the textual description of the battle s/he wants to create.
- WSP9) **Educator** uploads on the system the test cases and build automation scripts for the battle s/he wants to create.
- WSP10) **Educator** inputs in the system the minimum and maximum number of students per group allowed for the battle s/he wants to create.
- WS11) **Educator** inputs in the system the registration deadline of the battle s/he wants to create.
- WSP12) **Educator** inputs in the system the submission deadline for the students' solutions of the battle s/he wants to create.

- WSP13) **Educator** specifies to the system whether a consolidation stage is required after the submission deadline of the battle s/he wants to create.
- WSP14) **Educator** selects which aspects (reliability, maintainability, security...) should be taken into account by the system when computing the score of the students' code solutions for the battle s/he wants to create.
- WSP15) **Educator** checks the code solutions of the teams participating in a battle and assigns a personal score to each one of them.
- WSP16) **Educator** closes a tournament.
- WSP17) **User** opens the description page of a tournament.
- WSP18) **Student** subscribes to a tournament.
- WSP19) **User** opens the description page of a battle.
- WSP20) **Student** sends invitations to other students to ask to join a battle together as a team.
- WSP21) **Student** accepts or rejects the invite of another student to participate in a battle as a team.
- WSP22) **Student** joins a battle of a tournament either forming a group with other students or on his/her own.
- WSP23) **User** opens his/her personal profile on the platform.
- WSP24) **User** searches another user by name on the platform to see his/her personal profile.
- WSP25) **User** opens the personal profile of another user on the platform.
- WSP26) **User** opens the list of badges s/he own or of the ones owned by another user on the platform.
- WSP27) **User** opens the partial or final ranking of students in a tournament.
- WSP28) **User** opens the partial or final ranking of teams in a battle.
- WSP29) **GitHub** notifies the system every time a new commit is pushed on a repository that stores the code solutions for a battle.

Machine controlled shared phenomena

- MSP1) **The system** shows to a **user** the interface to log in the platform using his/her personal GitHub account.
- MSP2) **The system** shows to a **user** the initial page (home page) of the application.
- MSP3) **The system** shows to an **educator** the interface for creating a new tournament.
- MSP4) **The system** shows to an **educator** the interface for creating a new battle in an ongoing tournament.

- MSP5) **The system** shows to an **educator** the interface to assign personal scores during the consolidation stage of a battle.
- MSP6) **The system** shows to a **user** the list of tournaments available on the platform.
- MSP7) **The system** shows to a **user** the description page of a tournament.
- MSP8) **The system** shows to a **student** the interface to subscribe to a tournament.
- MSP9) **The system** shows to a **user** the list of battles within a tournament.
- MSP10) **The system** shows to a **user** the description page of a battle.
- MSP11) **The system** shows to a **student** the interface to invite other students to join him/her in a battle as a team.
- MSP12) **The system** shows to a **student** the interface to join a battle.
- MSP13) **The system** shows to a **user** his personal profile.
- MSP14) **The system** shows to a **user** the personal profile of another user on the platform.
- MSP15) **The system** shows to a **user** the list of badges owned by a student on the platform.
- MSP16) **The system** shows to a **user** the partial and final ranking of students in a tournament.
- MSP17) **The system** shows to a **user** the partial and final ranking of teams in a battle.
- MSP18) **The system** notifies all **students** subscribed to the platform when a new tournament is created.
- MSP19) **The system** notifies all **students** subscribed to a tournament when a new battle is created in the tournament.
- MSP20) **The system** creates a new **GitHub** repository dedicated to a battle.
- MSP21) **The system** sends the link of the GitHub repository to all the **students** subscribed to the battle.
- MSP22) **The system** shows and keeps updated the scores assigned to the students' code solutions for a battle.
- MSP23) **The system** notifies all **students** that participated in a battle when the final ranking of the battle is available.
- MSP24) **The system** notifies all **students** subscribed to a tournament when the tournament is closed.
- MSP25) **The system** assigns badges (rewards) to **students** when a tournament ends.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

A brief list of the most meaningful and relevant terms and synonyms used in this document is reported here, in order to make reading process smoother and clearer:

Term	Definition
User, Actor	A user of the CodeKataBattle system, which includes both educators and students.
Educator, Professor, Teacher	A type of user of the CodeKataBattle system.
Student	A type of user of the CodeKataBattle system.
System, Platform, Application, Machine, Software	All synonyms to indicate CodeKataBattle.
Battle, Coding challenge, Problem, Teaser	A code kata battle offered on CodeKataBattle.
Consolidation Stage	Phase occurring at the end of a battle (after submission deadline) in which the educator that created the battle is asked to personally assess the students' code solutions.
Registration Deadline	Valid for both tournaments and battles, it is the date by which students are asked to subscribe to a tournament or a battle on CodeKataBattle.
Submission Deadline	In the context of a battle, the submission deadline is the date after which no additional code solution for the battle can be submitted to the system.
Ranking, Classification	Ordered list of teams or students based on achieved scores.
Push	Upload code solutions on GitHub repository.
Pull	Download code solutions from GitHub repository.

Term	Definition
Fork	Creating a personal copy of a GitHub repository, which inherits the structure and content of the original repository at the moment the fork is performed.
GitHub workflow	File containing code (usually written in YAML) to instruct GitHub on actions to take when certain events happen in a GitHub repository. In this project, workflows are used to fire a notification from GitHub to CodeKataBattle when a commit is performed on the repository.
GitHub Actions	Pre-configured actions that can be employed inside GitHub workflows to make GitHub carry out specific operations based on the specific needs of the user.
Build automation scripts	Files containing code useful to automatically build a project or (in this specific case) the students' code solutions for a battle.
Test cases	Files containing code useful to automatically test the students' code solutions for a battle.
Evaluation Criteria	Indicate the set of parameters that are accounted for by an external static analysis tool in order to assign a partial score to the students' code solutions for a battle. Examples of Evaluation Criteria might be security, reliability, robustness and so on.
Badges, Rewards	In a tournament, Badges or Rewards are prizes that are assigned to students when the tournament ends based on personal achievements of the student in that tournament.
Participate, Join, Sign up, Take part, Subscribe	Synonyms employed to describe the action of students engaging with tournaments or battles on CodeKataBattle.
Create, Publish, Generate	Synonyms to express the action of educators creating new battles or tournaments on CodeKataBattle.
Evaluate, Assess, Assign score	Synonyms to express the action of associating a score to a battle code solution.

Term	Definition
Submit, Hand in	Synonyms describing the action of a student giving to the system a code solution to a battle.
Show, Display	Synonyms to express the responses of the system to user requests. CodeKataBattle shows or displays something to the users.

1.3.2 Acronyms

A list of acronyms used throughout the document for simplicity and readability:

1. RASD - Requirements And Specification Document
2. CKB - CodeKataBattle

1.4 Reference Documents

Here's a list of reference documents that have been used in order to shape the Requirements Analysis and Specification Document of the CodeKataBattle system. In the following, all external sources of information that have contributed to the design of this document are mentioned.

1. Stakeholders' specification provided by the R&DD assignment for the Software Engineering II course at Politecnico Di Milano for the year 2023/2024.

2. "The World and the Machine", by Michael Jackson, 1995.
   Link: https://ieeexplore.ieee.org/document/5071113

3. "29148-2018, ISO/IEC/IEEE International Standard, Systems and software engineering, Life cycle processes, Requirements engineering", by IEEE, 2018.
   Link: https://ieeexplore.ieee.org/document/8559686

4. UML specifications, version 2.5.1.
   Link: https://www.omg.org/spec/UML/2.5.1/About-UML

5. Alloy documentation, version 6.1.0.8.
   Link: https://alloy.readthedocs.io/en/latest/

1.5 Document Structure

This Requirements and Analysis Specification Document is composed of four major sections.

The first one is the **Introduction**, whose main objective is to introduce the reader to the domain of interest for the system to be developed, mainly using natural language to describe all the most fundamental actors and elements involved in the interactions between the system and the outer world.
The **Purpose** provides the definition of the main goals for the application.
The **Scope** is dedicated to reprocessing the original stakeholders' requirements in a new high-level description of the domain of interest that aims at being as unambiguous and clear as possible. All the most meaningful actors of the world in which the system lives are mentioned and their role explained. Besides, the interactions between these actors and the system are touched at a high-level to clarify what will come next in the document and to justify some of the design choices that are taken in the following paragraphs of the RASD. From the natural language description of the system, world and machine phenomena can be derived and in fact, they come immediately afterwards. These can be interpreted as a schematisation of the previous description in which only the events occurring in the domain of interest are presented (see "The World and the Machine", by Michael Jackson, 1995 for more information).
In the **Definitions** subsection it is possible to find specifications on terminology and vocabulary terms used throughout the document, so that ambiguity shouldn't emerge from reading. From the table provided in this part, synonyms for words employed in the RASD are also listed.

The second major section of this document is the **Overall Description**. This part of the RASD has several goals, which are achieved thanks to its subsections.
The first one is dedicated to scenarios. The **Scenarios** subsection aims at validating the stakeholders' needs by illustrating concrete instances and examples of interactions with the system to be developed.
The **Domain Class Diagram** and **State Charts** are UML diagrams that provide a graphical visualization of the world of interest, consistently with the Introduction section.
In the **Product Functions** chapter, the functional requirements of the system are listed in a schematic way. This analysis derives as a consequence of all the previous sections, in which the world of interest has been described and accurately observed in order to understand what requirements the system should meet.
Finally, the **Assumptions, Dependencies and Constraints** subsection is dedicated to listing all the events and elements of the domain which are not under the system's control or which the system has some dependency over.

The third relevant part of this RASD is **Specific Requirements**, which is more concerned about turning the functional requirements listed in the Product Functions section into schematic and graphical representations.
The **External Interface Requirements** subsection deals with the interfaces and modes of interactions between the system and external users or other software products.
The **Functional Requirements** chapter offers a schematic view of the functional requirements listed in the Product Functions section, by means of use cases, use case diagrams and sequence diagrams. A mapping between these graphical representations and the associated requirements is also provided.
The part named **Design Constraints** specifies any constraint that the system has to respect when being developed, while the last section called **Software System Attributes** lists a series of qualities that the software to be implemented must have and the way to achieve them (for instance reliability, availability...).

The final section called **Alloy** provides the study of the system through Alloy, which is a tool for analyzing systems and seeing if they are designed correctly.

2 Overall Description

2.1 Product Perspective

2.1.1 Scenarios

This section is dedicated to scenarios, which should be thought of as concrete instances of interactions between the external actors of the world and the system to be developed. They are presented in the form of simple short stories, because their main intent is to build a bridge between the developer/designer of the system and the stakeholders, who usually don't share the same technical knowledge. By means of concrete descriptions, scenarios allow the developer to illustrate simple use examples of the system to the stakeholders, in order to validate their needs and be sure about what they are really looking for. This is also the reason why the following scenarios are in some way **original** and **inventive**, and include many details about the idea that they attempt to convey to the reader.

As a brief remark on the following scenarios, it is possible to notice that the entire setup of the forked GitHub repository that the student is required to do (forking the main GitHub repository, writing the GitHub Actions workflow) is cut out from this section, as it is an interaction between the student and GitHub, which doesn't involve the system to be developed, therefore not an interaction that has to be observed here.

SCENARIO 1 - Educator logs in the system
Emanuele is a professor at Politecnico di Milano in the Computer Science department. He has just discovered a new application called CodeKataBattle that allows educators to organize tournaments of students to make them compete on coding battles.
Emanuele is very enthusiast about the idea because he thinks it might be a great opportunity for his students to better their software development and problem solving skills. He therefore opens CodeKataBattle on his laptop. The first interface showing up is the log in interface, in which the application asks Emanuele to sign in using his GitHub personal account. He clicks on the button to be redirected to the GitHub authentication page and uses his credentials as required. GitHub calls back the CodeKataBattle platform and now Emanuele is logged in and can start experimenting with this brand new system.

SCENARIO 2 - Student logs in the system
Matteo is a student enrolled in a Computer Science master's degree at the University of Barcelona. He's passionate about coding and software development in general. One day, he comes across a new app called CodeKataBattle that allows students to compete against each other in coding battles organized by educators from many universities spread across the world.
This application immediately catches his attention, so he decides to open it on his laptop. The first interface showing up is the log in interface, in which the CodeKataBattle platform asks Matteo to sign in with his personal GitHub account. Matteo clicks on the button that redirects to the GitHub authentication page and logs in as requested. Now, he's signed in the platform and can start playing around with it to see if he likes it.

SCENARIO 3 - Educator creates a new tournament
Mario is a renowned university professor in the field of Software Engineering. He has been using the CodeKataBattle application for a while now, but he's never organized a tournament on this platform. One day, he decides to create his first tournament on CodeKataBattle . So, he opens the application

and signs in. Among the various buttons that are available on the educators' home page of the platform, he clicks on the one to create a new tournament. As a consequence, CodeKataBattle opens up a window in which various parameters can be set for the new tournament. Mario can set the name of the tournament, a description of it and the registration deadline by which students are asked to subscribe to the tournament if interested. Moreover, there is a section dedicated to the definition of badges (rewards) to be assigned to students at the end of the tournament based on some goals or achievements. Mario can declare on CodeKataBattle what kind of rewards he wants allocate and for each reward the achievement(s) that have to be accomplished by a student in order to earn the badge.
After filling in the all fields of the form, Mario can confirm the creation of the tournament and the application will show him the newly-created tournament page. CodeKataBattle will also send a notification to all the student on the platform informing them of the new available tournament.

SCENARIO 4 - Educator creates a new battle
Alex is a professor at the University of London specialized in artificial intelligence. Lately, he's been using the CodeKataBattle platform to help his students practice on some algorithms that he's explaining during the lectures. One morning, he comes up with a new idea for a code kata battle that may be very beneficial for the students that are going to take his exam in the following months. Thus, he seats at the desk in his bedroom and starts writing a textual description of the battle, along with the build automation scripts and test cases that are required by CodeKataBattle in order to build and test the students' solutions to the exercise. Once done with these tasks, Alex takes his laptop and opens the CodeKataBattle application. He logs in through his GitHub account and clicks from the home page the button to create a new battle. A new interface pops up displaying all the tournaments in which Alex has permissions to publish battles (so the tournaments Alex created and the ones other educators granted Alex permissions on). Alex selects one of these tournaments and the form to set the battle's parameters and characteristics shows up. Through this interface, Alex is able to upload the textual description, build automation scripts and test cases that he previously designed; he also set the battle's name, the registration and submission deadlines, the minimum and maximum number of students per team allowed. Moreover, the aspects on which CodeKataBattle has to base its automatic evaluations of the students' solutions (reliability, maintainability...) can be defined and it is possible to establish whether to require a consolidation stage at the end of the battle (in order to allow Alex to assign personal scores). Once all of these parameters have been set, Alex clicks on the confirmation button and the page dedicated to the newly created battle pops up on his display. Immediately after this, CodeKataBattle will fire a notification message to all the students subscribed to Alex's tournament in order to inform them of the new available battle.

SCENARIO 5 - Educator grants the permission to publish battles in his/her tournament to another educator
George is a professor at MIT's Software Engineering department. Lately, he's been actively using the CodeKataBattle software because he initiated a tournament of coding challenges for students centered around the Python programming language. He's published many battles over the last week, and that's why he's running out of ideas for new problems to propose. That's why he decides to ask for some help from one of his closest colleagues, Laura. In order to allow Laura to publish battles inside George's tournament, George has to grant her permissions to do so. Thus, George opens the CodeKataBattle application and clicks on the button to display the tournaments he created. Among these, he selects the one in which he wants Laura to help him. The home page of the selected tournament pops up and there is a button to grant publishing permissions to another educator. Once clicked, George is able to search for Laura's account through her username on the platform and then send her a request to be added as an educator with publishing permissions inside his tournament. Once

these steps have been carried out, Laura is able to see George's request on her profile and accept it. George's tournament will then appear in the list of tournaments in which she can create battles. Now George will no longer be the only one keeping the tournament alive with new battles.

SCENARIO 6 - Student subscribes to a tournament
Alessandro is an artificial intelligence student at the Technical University of Munich. It has been a while since the last time he coded a program of any kind, so he feels a little bit rusty on that. He would like to brush up on this skill and he decides to use the new app CodeKataBattle for this purpose. After logging in the system, Alessandro selects the option for adding a new tournament from the home page and CodeKataBattle displays as a consequence a list of all the available tournaments on the platform. Thus, Alessandro starts browsing the list, searching for something that he might find interesting. At some point, he stumbles upon a tournament created by one of his favorite professors at his university, so he clicks on it and the home page of the tournament pops up. George clicks on the button to subscribe and confirms his choice. Consequently, CodeKataBattle moves the selected tournament in the list of tournaments Alessandro is subscribed to. From this moment on, Alessandro will be notified by the CodeKataBattle app every time a new battle will be published inside the tournament.

SCENARIO 7- Student joins a battle on his/her own (without a team)
Lorenzo is a student at the University of Pisa specialized in data management. He's been eagerly trying to reach the first position in a tournament of the CodeKataBattle application, but he's still some points behind the first in the ranking. He doesn't want to give up, so he opens CodeKataBattle and navigates to the tournament in which he's competing. In order to get additional points he has to join another battle. So he commences the search for the battle, reading some descriptions of the battles published in the tournament whose registration deadline hasn't passed yet. He eventually finds one that might be suitable for him, so he clicks on the button to join it. CodeKataBattle promptly shows Lorenzo the description page of the battle, where there's a button to join it. Once clicked, an interface pops up in which it is possible to select whether to join the battle as a single player or with other students in a team. Since Lorenzo usually prefers to work on his own, he opts for the single player and confirms. The battle is therefore added to the list of battles Lorenzo is participating in. Lorenzo will be able to submit his code solutions when the registration deadline passes and CodeKataBattle will have created the GitHub repository dedicated to the battle.

SCENARIO 8 - Student joins a battle with other students as a team
Lucia is a student at the Politecnico di Torino university and she's enrolled in the cybersecurity program. Her group of university friends has been talking incessantly about creating a team on the new CodeKataBattle application to solve together a coding battle proposed by one of their professors specialized in cryptography called Adrian. She lets her friends persuade her to do that, so she opens the CodeKataBattle application on her laptop and navigates to the tournament that contains the interesting battle. She and her friends are already subscribed to such tournament, since it is a very popular one organized inside the university, therefore she doesn't have to sign up for that. Then, Lucia spots the coding challenge her friends were talking about and clicks on it. CodeKataBattle shows the description page of the battle, with a button to join it. When Lucia clicks on the button, CodeKataBattle promptly displays the interface to take part in the battle, in which it is possible to select whether to sign up as a single player or with other students as a team. Obviously she picks the latter option and a new window pops up, in which Lucia has the possibility to set the team's name and search by username her friends on the platform to send them a request to join the battle in her team. Once this process is carried out, Lucia confirms. The battle is therefore added to the list of battles Lucia

is participating in. Lucia's friends will be able to take part in the battle as soon as they accept the request sent by the system. At that point, the battle will also be added by CodeKataBattle to the list of battles in which they are competing. Lucia and her friends will be able to submit their code solutions as soon as the registration deadline for the battle passes and CodeKataBattle creates the dedicated GitHub repository.

SCENARIO 9 - CodeKataBattle creates the GitHub repository dedicated to a battle
The battle "TrainWithAndrea" has been created a couple of days ago by an educator called Andrea. At 5pm the registration deadline for this battle is over and consequently CodeKataBattle sends a request to the GitHub platform to create a new repository with the same name as the battle "TrainWithAndrea". GitHub generates this new repository and sends back the confirmation to CodeKataBattle . Immediately, the system fires a notification to all the students that joined Andrea's battle to inform them that the battle is now open and it is possible to push on GitHub the code solutions to the challenge. In the notification, the link that points at the remote GitHub repository is also attached. In a few hours, a lot of solutions have already been pushed on GitHub by the students taking part in the "TrainWithAndrea" battle.

SCENARIO 10 - Student pushes a code solution on his/her forked repository on GitHub
Francesco is a student at the University of Edinburgh where he studies computer engineering. In the last couple of days, he has been absorbed by the task of solving a battle on CodeKataBattle proposed by one of his professors. Francesco would like to make a good impression on this professor, as he is very renowned in the entire country for his research activity. As soon as the lectures at the university end, Francesco runs back home, and starts working again on the coding challenge. Suddenly, a new possible solution pops up in his head, so he writes it down and pushes it on his forked repository on GitHub (related to the main repository dedicated to the battle). As soon as the solution is uploaded, the GitHub Actions workflow that Francesco previously wrote triggers the GitHub platform to fire a notification to CodeKataBattle to inform the system of the new commit on the remote repository. Immediately, CodeKataBattle downloads Francesco's new solution and automatically computes the score to assign to it. Then, CodeKataBattle uploads Francesco's score on the platform and finally the system recalculates and shows the partial ranking of the battle, positioning Francesco according to the maximum score he obtained so far with one of the solutions produced by him. Francesco is very happy to see that his new solution scored very high and he managed to climb the ranking up to the second position.

SCENARIO 11 - CodeKataBattle calculates the new score of a student's solution pushed on GitHub
A battle named "SortTheMatrix" has been going on for days. Matilde is a student participating in the battle and she's quite convinced about her new solution to the challenge, since she's been working on it for days. So she pushes it on her forked GitHub repository and waits for the system to compute the new score. CodeKataBattle first downloads Matilde's solution from GitHub and uses the build automation scripts provided by the educator who created the battle to build the solution. If this initial step is successful, the system runs the test cases on the solution and derives the number of test cases passed. Then, CodeKataBattle calculates the time passed from the beginning of the battle to the current moment and stores this information as well. Finally, the platform leverages an external static analysis tool in order to evaluate Matilde's solution on some criteria that have been established at battle creation time by the educator (maintainability, security...). By combining the number of test cases passed, the time went by from the beginning of the battle and the points assigned by the static

analysis tool, CodeKataBattle is able to compute the score for the new code solution as a number between 0 and 100.

SCENARIO 12 - Educator manually evaluates the students' code solutions of one of his/her battles during the consolidation stage

Michele is a meticulous computer science teacher in a little high school in Rome. Lately, he's been publishing several battles on the CodeKataBattle platform for his students to let them exercise on some new algorithms he explained during the lectures. Since he wants to personally assess the code that his students write, he always specifies at battle creation time that a consolidation stage is required for his battles. Today, Michele knows that the submission deadline of one of his battles will pass at 6pm, and that after that time, the CodeKataBattle will allow him to assign a personal evaluation to his students' solutions. At 6:30pm, when Michele comes home from work, he makes himself a cup of tea and opens the CodeKataBattle platform. In the list of battles that he created, he clicks on the one that requires the consolidation stage to be carried out. A new interface shows up, where the list of teams subscribed to the battle is shown and for each team there is a field on the side where it is possible to type in the score Michele wants to assign. Within this interface, there is also a timer that specifies how much time Michele has to assess the code solutions. After the timer goes off, if Michele hasn't provided a score for each team in the battle, the consolidation stage won't be taken into account by CodeKataBattle in order to compute the final ranking of the battle. All the source code for the solutions is available for Michele on the GitHub platform, so he can review it all and complete the consolidation stage for the battle by the specified time frame. When the timer goes off, CodeKataBattle is able to draw the final ranking of teams for the battle and publish it on the platform. CodeKataBattle will also send a notification to all the participants of the battle to notify them of the available hierarchy of teams.

SCENARIO 13 - A battle terminates and the ranking of teams is published

Clara, a data analytics student enrolled in a Master's program at ETH Zurich university is impatiently waiting for the end of a battle on the CodeKataBattle platform. She's worked really hard over the last weekend in order to reach the first position in the ranking with a solution that scored 95/100. At 1pm, the submission deadline for the battle has finally passed. Since no consolidation stage was required by the professor who published the battle, CodeKataBattle immediately calculates the total ranking of teams and fires a notification to all the students participating in the battle. Also Clara receives the alert stating that the battle was over and that the final ranking was available on the app. She enthusiastically opens CodeKataBattle on her laptop just to find out that during the last 10 minutes before the submission deadline, another competitor handed in a solution that received a score higher than hers.

SCENARIO 14 - Educator closes a tournament that s/he previously created

Olivia is a famous professor in the field of Bioinformatics. Over a month ago, she decided to create a new tournament on the CodeKataBattle platform in which she's been started publishing coding battles that are biology-themed. Many students from all over the city in which Olivia lives liked the idea of this tournament and decided to take part in it. Now, since the tournament has been going on for over a month, the challenges are becoming more and more repetitive, therefore Olivia decides to officially close it. She opens the CodeKataBattle application on her laptop and logs in. In the home page, she clicks on a button to display the list of tournaments that she created. She selects the tournament she wants to close, clicks the button to close the tournament and confirms her choice. The CodeKataBattle platform immediately sends a notification to all the students subscribed to the tournament informing them that the tournament is now closed and the final ranking of students is available.

Since CodeKataBattle always maintains updated the ranking of students in a tournament, there is no need to recalculate the ranking here, just to display it. Moreover, since Olivia defined some badges at tournament creation time, CodeKataBattle evaluates the sets of students that are eligible to receive these badges and assigns them accordingly. The personal profile of students that received a reward from the tournament is automatically updated by the app.

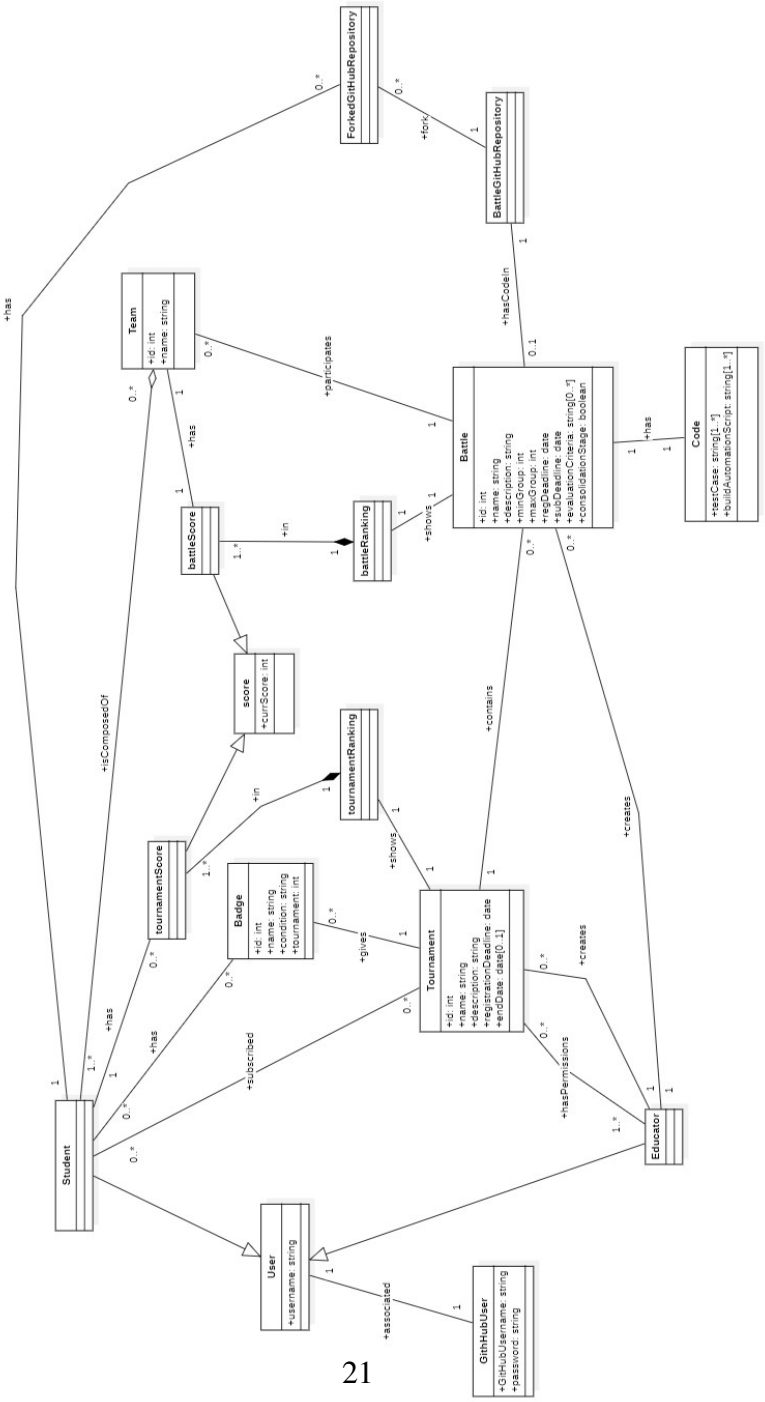
SCENARIO 15 - User opens the ranking of a tournament on the CodeKataBattle platform
Christian is a student at Princeton University where he specializes in game design and development. He knows that some of his classmates are participating in a tournament on the CodeKataBattle platform organized by some of the professor of the engineering department and he's very curious about the ranking of this tournament. So, he opens CodeKataBattle , logs in with his personal GitHub account and in the home page that is displayed right after the log in he can see all the tournaments available on CodeKataBattle . He scrolls a little bit down and eventually finds the tournament he was looking for. So he clicks on it and in the dedicated page, the total ranking of students is shown, updated with the very last scores assigned to the students' solutions. Christian can finally see how his classmates are positioned.

SCENARIO 16 - User opens the ranking of a battle s/he is involved in
Emily is a web development student at the University of Toronto. She's been working with her team on the solution for a battle on the CodeKataBattle platform for a while now, but she cannot get a score higher than 55/100. Since she doesn't know if that is going to be enough to reach the top 10 positions of the battle ranking, she decides to give a look at the partial ranking for the competition, to have an idea of how well the other teams are doing so far. So she opens the CodeKataBattle app on her laptop and navigates to the list of tournaments she's subscribed to. Once she's selected the correct tournament, she can see the list of battles belonging to that tournament that she signed up for and among them she picks the one she wants to see the ranking of. Consequently, CodeKataBattle checks whether Emily is actually involved in the battle or not and displays the ranking only in the affirmative case. At the end, Emily discovers that she and her team are in the very last position!

SCENARIO 17 - User searches by username the personal profile of another user on the platform and looks at his/her badges
Filippo, a student at the University of Bologna, where he's conducting some important research on Cloud Computing, has been using the CodeKataBattle application to compete with other classmates on coding challenges. Since he's very competitive, he wants to give a look at his friends' profiles on the platform to check if they have obtained more badges than him. So he opens the CodeKataBattle application on his laptop, logs in and clicks on the search box in the home page to type in the username of one of his friends. Once he finds the correct user on the application, he clicks on it and CodeKataBattle displays the personal profile of this user. Filippo is able now to inspect the set of badges earned by his friend and compare them with the ones he obtained so far.

2.1.2 Domain Class Diagrams

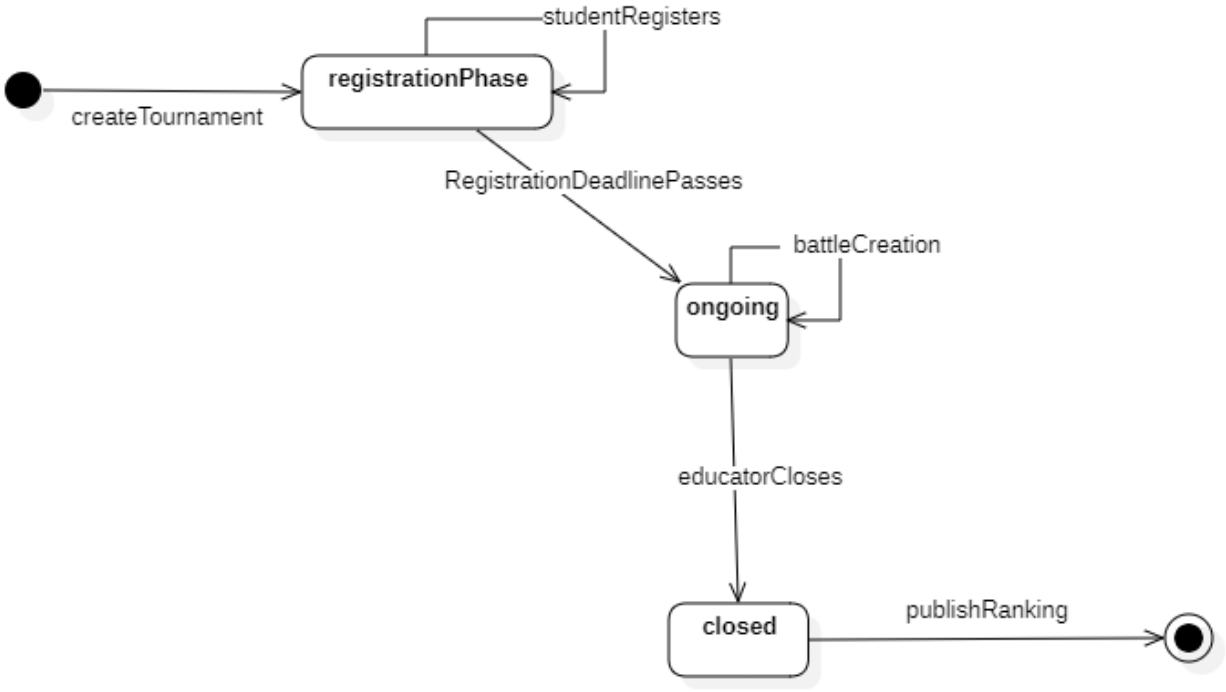
The following class diagram provides a high-level view of the domain of interest for the CodeKata-Battle platform. It is possible to notice elements that are external to the system (GitHub repositories, Users, Students, Educators) as well as the most meaningful internal ones (Tournaments, Battles, Scores, Rankings, Badges...). This diagram pursues the objective of showing the multiplicity relations between all of these components, along with the most relevant pieces of information (attributes) that characterize them. As a remark to understand some of the multiplicity relations stated in the class diagram, only GitHub users that have already accessed CodeKataBattle once are considered, thus the 1:1 relationship between GitHubUser and User. Besides, the set of external GitHub repositories is restricted to those associated to a battle and the ones forked by the students from there.



2.1.3 State Charts

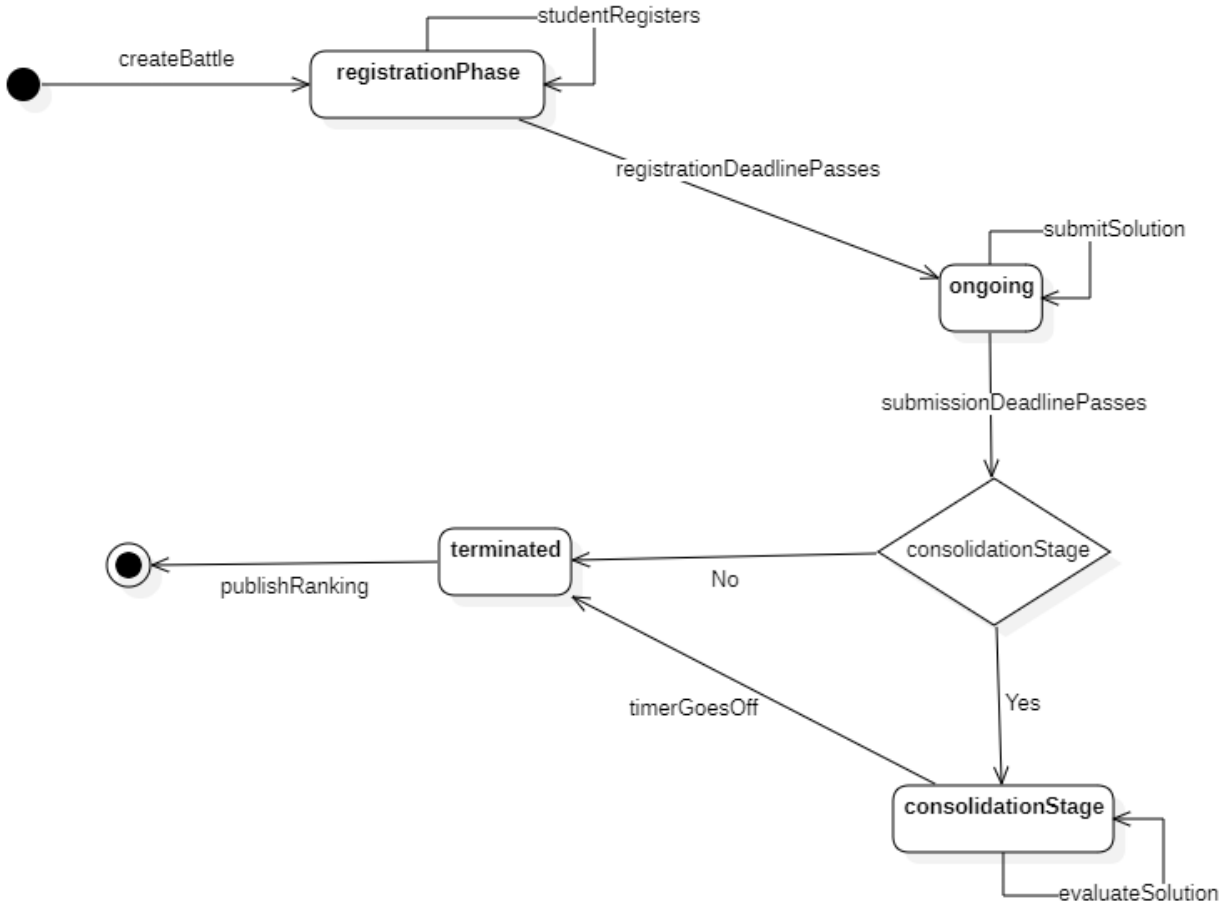
In this section, state charts are employed in order to provide support to the reader in understanding how the internal states of some domain components change over time. In particular, the focus is on the Tournament and Battle entities.

Tournament State Chart



After an educator creates a new tournament, all students with an account on CodeKataBattle are notified and can subscribe to it. This is the **registrationPhase** state. When the registration deadline passes, the tournament starts and students can no longer sign up. The state is now **ongoing** and this is the phase in which battles are created inside the tournament by educators. When the educator that created the tournament decides to close it, the tournament moves to the **closed** state. At this point, no more battle can be published in the tournament and the final ranking of fstudents is released.

Battle State Chart



When a battle is created by an educator, it is in the **registrationPhase** state, during which students can join the battle. When the registration deadline passes, the battle moves on to the **ongoing** state. In this state, students can no longer sign up for the battle and the students who decided to participate are able to submit their code solutions through GitHub. When the submission deadline is met, there are two possible cases. If the educator who created the battle didn't request the consolidation stage for the battle, then the battle can simply terminate (**terminated** state) and the final ranking is published. On the other hand, if the professor who created the battle requested the consolidation stage, then the state shifts to **consolidationStage**. In this state, CodeKataBattle initiates a timer to set a time frame for the educator to manually evaluate the students' code solutions. When the timer goes off, if all teams of the battle received an evaluation from the educator, then the scores assigned during the consolidation stage are taken into account to compute the final ranking, otherwise CodeKataBattle will ignore the consolidation stage and publish the ranking based on the scores obtained in the **ongoing** state.

2.2 Product Functions

2.2.1 Requirements

This section shows a comprehensive list of functional requirements that CodeKataBattle has to meet while being developed. Functional requirements are the actions and choices that the system under analysis has to take in order to meet the stakeholders' needs. These requirements are the result of an elicitation and abstraction process that comes from the previous sections of this document. The analysis of the relevant phenomena for the domain of interest, the validation of the stakeholders' needs by means of scenarios and the graphical illustrations provided by the several diagrams lay solid foundations for the design of the following part of the document.

Requirements for Goal G1

(R1.1) The system allows students to log in the platform using their GitHub account.
(R1.2) The system notifies all students on the platform every time a new tournament is created.
(R1.3) When the student requires it, the system shows the list of tournaments available on the platform for subscription (whose registration deadline hasn't passed yet).
(R1.4) When the student requires it, the system shows the list of tournaments the student is subscribed to.
(R1.5) The system allows students to subscribe to tournaments, by the end of the registration deadlines set for the tournaments.
(R1.6) The system notifies all students subscribed to a tournament every time a new battle within that tournament is published.
(R1.7) The system allows students subscribed to a tournament to join battle(s) within that tournament, by the end of the registration deadline set for the battle(s).
(R1.8) The system allows students subscribed to a tournament to join a battle of the tournament either on their own or creating teams with other students.

(R1.9) The system manages all the interactions with the GitHub platform in order to allow students to submit their code solutions through GitHub. • (R1.9.1) The system creates a new GitHub repository dedicated to a battle right after the registration deadline for that battle passes. • (R1.9.2) The system sends the link of the GitHub repository dedicated to a battle to all the students subscribed to that battle. • (R1.9.3) The system accepts notifications from GitHub in order to know when a new commit is performed by a student in any GitHub repository forked by the one created by the system itself. • (R1.9.4) The system pulls from the GitHub repositories of ongoing battles the new students' code solutions, every time it receives a notification from GitHub of a new commit in those repositories.
(R1.10) The system calculates and then publishes the new score of a team's solution every time it receives a notification from GitHub of a new commit performed by a member of the team.
(R1.11) The system notifies all students participating in a battle when the final ranking of teams is available on the platform.
(R1.12) The system notifies all students subscribed to a tournament when the tournament is closed by the educator who created it.

Requirements for Goal G2

(R2.1) The system allows educators to log in the platform using their GitHub account.
(R2.2) The system allows educators to create new tournaments. • (R2.2.1) The system allows educators to set the name of the tournament they want to create. • (R2.2.2) The system allows educators to set a registration deadline for the tournament they want to create.
(R2.3) The system allows the educator that created a tournament to grant the permission of publishing battles within his/her tournament to other educators on the platform.

(R2.4) The system allows educators to create new battle(s) in the tournaments they have the permissions to do so.

- (R2.4.1) The system allows educators to write on the platform the textual description of the battle they want to create.
- (R2.4.2) The system allows educators to upload the build automation scripts designed for the battle they want to create.
- (R2.4.3) The system allows educators to upload the test cases designed for the battle they want to create.
- (R2.4.4) The system allows educators to set the minimum and maximum number of members for each team of students participating in the battle they want to create.
- (R2.4.5) The system allows educators to decide whether to require a consolidation stage or not at the end of the battle they want to create.
- (R2.4.6) The system allows educators to specify what evaluation criteria (reliability, security...) should be used by the system in order to compute the partial scores of the students' code solutions through an external static analysis tools.
- (R2.4.7) The system allows educators to set a registration deadline for the battle they want to create.
- (R2.4.8) The system allows educators to set a submission deadline for the battle they want to create.

(R2.5) The system doesn't take into account any new code solution for a battle after the submission deadline of that battle.

(R2.6) After the submission deadline of a battle and only if a consolidation stage had been requested by the educator at battle creation time, the system sets a time frame for the educator that created the battle to allow him/her to assign personal scores to the students' code solutions.

(R2.7) The system takes into account the personal scores assigned by the educator during the consolidation stage of a battle only if the educator assigned a score to all teams within the imposed time frame.

(R2.8) At the end of a battle and after the consolidation stage (if requested), the system automatically calculates and publishes the final rank of all teams that participated in that battle.

(R2.9) When a tournament is closed, the system automatically publishes the rank of all students that participated in the tournament.

Requirements for Goal G3

(R3.1) The system allows students subscribed to the same tournament to invite each other in order to join battles together as a team, respecting the minimum and maximum number of students permitted for a team in the battle.

(R3.2) The system allows students to accept or reject invitations from other student asking to join a battle together as a team.
(R3.3) The system allows students to set the name of their team when they join a battle with other students.
(R3.4) The system calculates and then publishes the score assigned to a new code solution of a team in a battle, every time GitHub notifies the platform of a new commit performed by a member of such team.
(R3.5) The system constantly keeps updated the total ranking of teams for a battle, which evolves based on the new code solutions that are uploaded by students on the GitHub repository.
(R3.6) The system calculates and publishes the final ranking of teams when a battle ends.
(R3.7) The system allows only the educators and students involved in a battle to see the partial or final ranking of teams for that battle.
(R3.8) The system automatically calculates and keeps updated the total ranking of students subscribed to a tournament, based on the scores each student received in the battles he participated in.
(R3.9) The system allows all users on the platform to see the partial or final rankings of tournaments.
(R3.10) The system allows educators to define customary badges (rewards) for the students participating in their tournaments, at tournament creation time.
(R3.11) The system assigns badges (rewards) to students that participated in a tournament, when the tournament is closed by the educator that created it.
(R3.12) The system allows all users on the platform to search for the personal profile of a student and see the corresponding account.
(R3.13) The system allows all users on the platform to see the list of badges owned by a student for the tournaments s/he participated in.

2.3 User Characteristics

2.3.1 Student

The user, in this case, is an individual possessing expertise and education in programming. Proficient in GitHub operations, the user is familiar with actions such as repository forking and making commits through pushes on the GitHub platform. A moderate knowledge of GitHub Actions is also required in order to setup the interaction between GitHub and CodeKataBattle .

2.3.2 Educator

The educator is an individual with advanced education in programming, equipped with the proficiency to teach effectively. This is especially necessary in order to write the build automation scripts and test cases that have to be uploaded on CodeKataBattle every time a new battle is created. Additionally, the educator has the capability of thinking and designing code kata battles, adjusting them to specific difficulty levels suitable for students.

2.4 Assumptions, Dependencies and Constraints

2.4.1 Domain Assumptions

D1 The GitHub platform is up and running and provides all the functionalities that are expected by CodeKataBattle to work correctly (creation of repositories, forking of repositories, notification system...).

D2 All users of CodeKataBattle have a personal GitHub account or are able to create a new one.

D3 Student correctly forks the main branch of the GitHub repository dedicated to a battle in order to submit his/her code solutions.

D4 Student correctly writes the automated GitHub workflow to send notifications from GitHub to the system every time a commit is performed on the forked repository.

D5 Educator correctly writes the build automation scripts for the battles s/he creates on the platform.

D6 Educator correctly writes the test cases for the battles s/he creates on the platform.

D7 The network allows notifications sent by CodeKataBattle to successfully reach the users, as well as the interaction with the GitHub platform to work correctly.

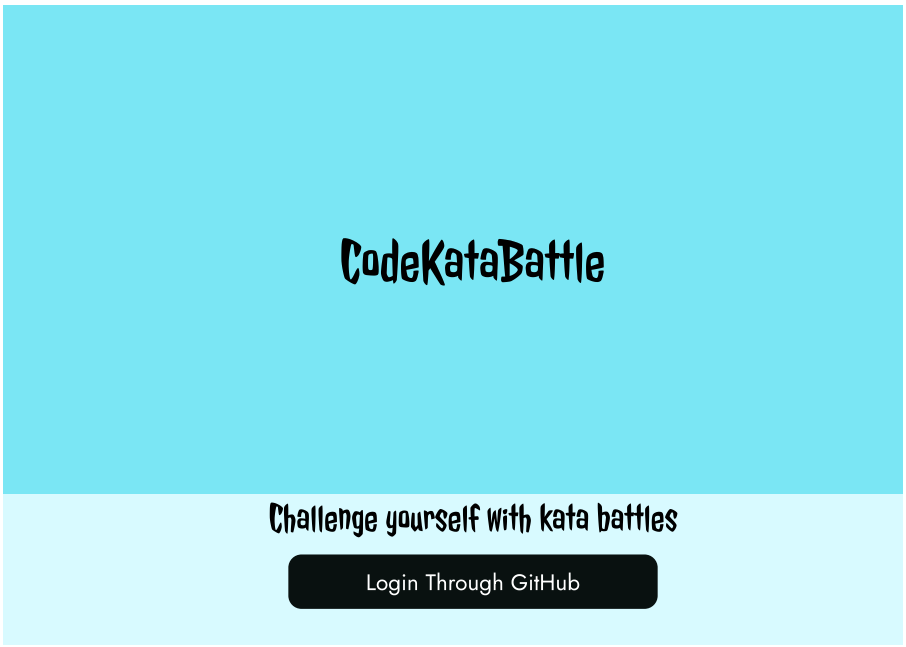
3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interface

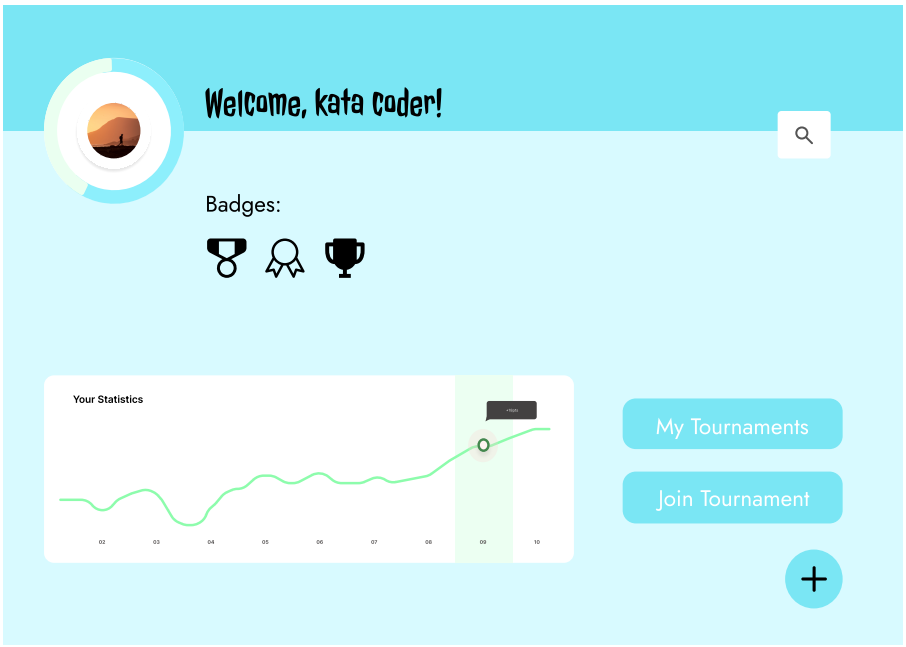
This section is dedicated to the illustration of some mockups of the most relevant graphical user interfaces that CodeKataBattle uses to interact with external users (educators and students). The aim of these representations is to specify the logical characteristics of the presented interfaces as well as introducing some guidelines on the style and look that the final product will have. This information can be used in order to shape the design of the CodeKataBattle platform in more advanced stages of the development.

Log In Interface



Log In Interface for both Educators and Students, who can access CodeKataBattle through their GitHub account by clicking on the button ”Login Through GitHub”

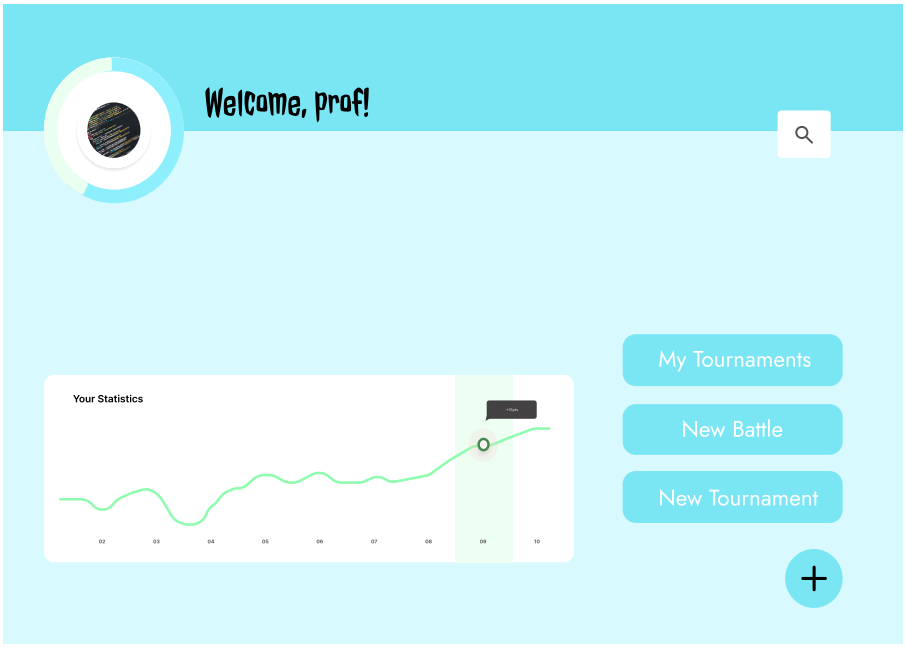
Home Page Student



Home page for a student, in which it is possible to:

- See personal profile of the student, with a customary image.
- See the list of badges earned by the student in tournaments s/he participated in.
- Ask for the list of tournaments the student is subscribed to.
- Subscribe to a new tournament with the ”Join Tournament” button.
- Search for a user on the platform (with the magnifying glass in the top right corner).

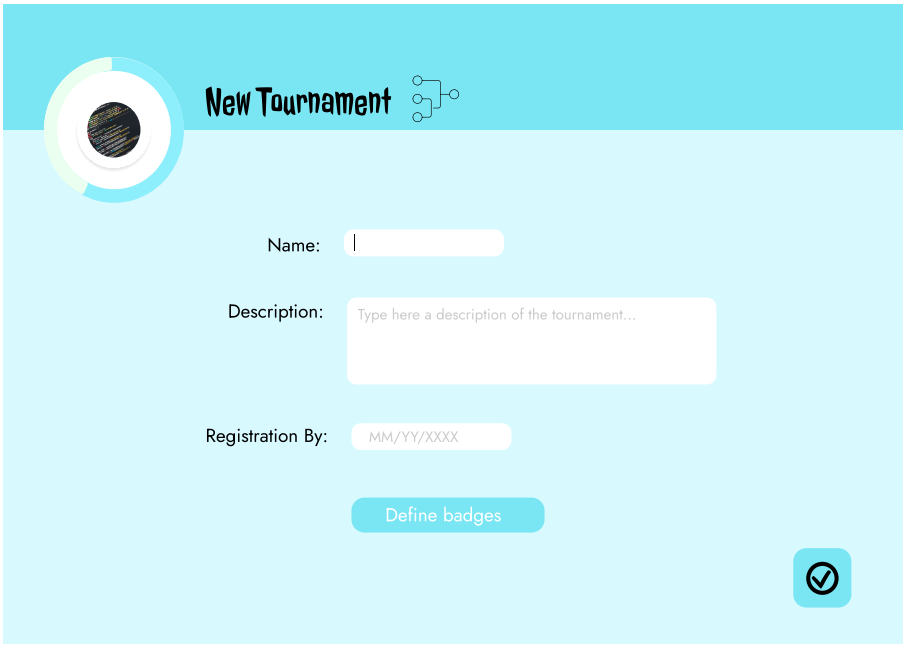
Home Page Educator



Home page for an educator, in which it is possible to:

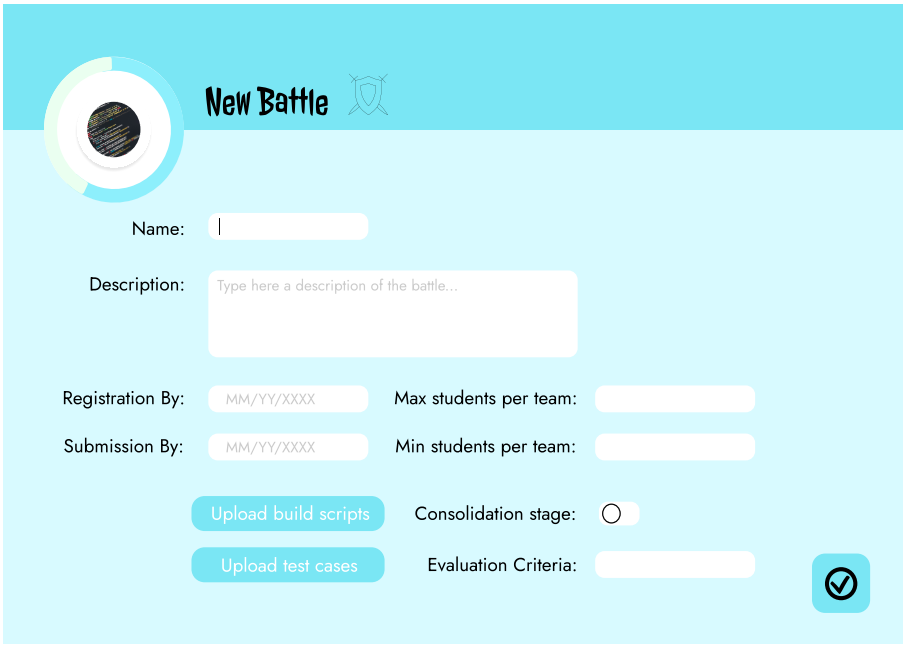
- See personal profile of the educator, with a customary image.
- Ask for the list of tournaments the educator created (or has permissions to publish battles in), with the ”My Tournaments” button.
- Create a new battle with the ”New Battle” button.
- Create a new tournament with the ”New Tournament” button.
- Search for a user on the platform (with the magnifying glass in the top right corner).

Tournament Creation Form



This interface pops up when the educator clicks on ”New Tournament” from the home page. Thanks to this form, all relevant parameters for the creation of a tournament can be set.

Tournament Creation Form



This interface pops up when the educator clicks on ”New Tournament” from the home page. Thanks to this form, all relevant parameters for the creation of a tournament can be set.

3.1.2 Hardware Interfaces

CodeKataBattle is developed on a server machine that will offer the services described in this document to the users (clients) through a web page. The server machine requires some fundamental hardware components with their corresponding hardware interfaces:

- A processor (CPU): The server hosting the web application needs a capable CPU to handle concurrent user requests and process data.
- Main memory (RAM).
- Storage (Hard Drive/SSD).
- Network Interface Card (NIC): A reliable network interface is essential for communication between the server and users' devices.

Moreover, since CodeKataBattle will tightly work with the Internet, the following hardware components will be also needed on the server side:

- Internet Connection: either Wi-Fi or Ethernet, a high-speed and reliable connection is needed in order to provide the promised services in an efficient manner.
- Firewall and Security Appliances: for security of data over the network.
- Load Balancers: for scalability.

The end-users (clients) also need some hardware components in order to integrate with the CodeKataBattle environment:

- Computer or Device.
- Internet connection (Wi-Fi, Ethernet...).
- Network Interface Card to manage the connection.

3.1.3 Software Interfaces

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

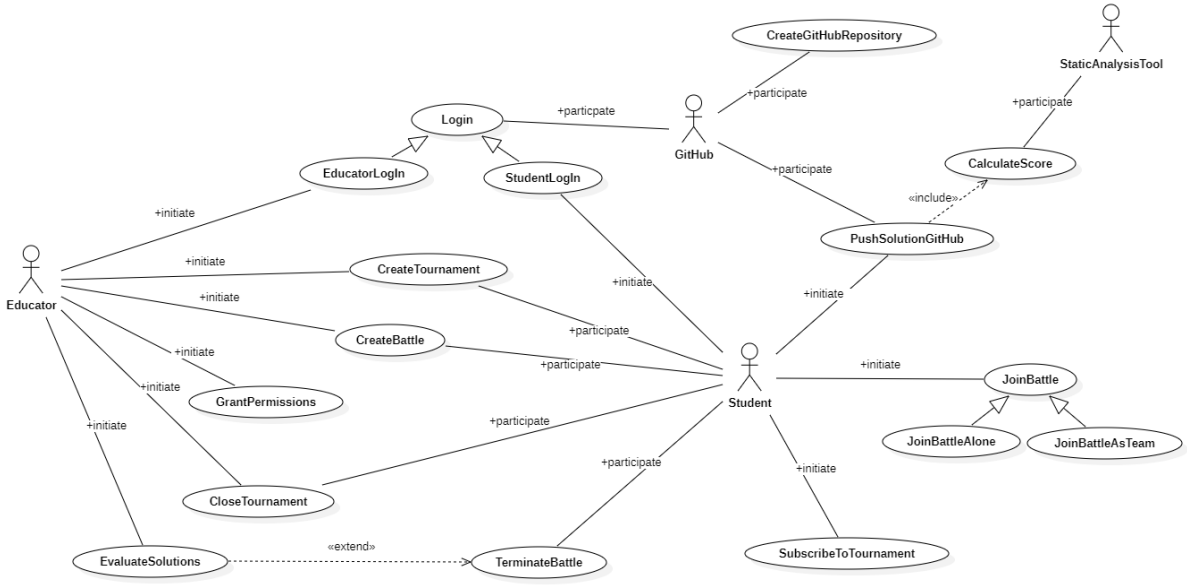
- GitHub OAuth interface: the authentication process is delegated to the GitHub platform by means of the OAuth standard.
- GitHub API Integration: the system communicates with GitHub's API to allow the creation of GithHub repositories for each battle.
- GitHub Actions workflows: CodeKataBattle exposes an API that allows a student of the application to setup the GitHub Actions workflow in such a way that GitHub will send notifications to CodeKataBattle when a new commit is pushed by the student to his/her forked repository.
- Client-server interactions occur by means of a HTTPS protocol to exchange data and information.

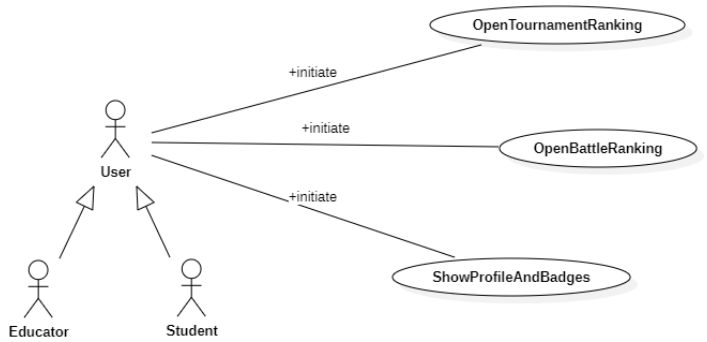
- Database interface: CodeKataBattle needs a database to store and manipulate user's data, therefore a software interface in between CodeKataBattle and the database has to be developed to allow the interaction.

3.2 Functional Requirements

3.2.1 Use Case Diagrams

Use case diagrams are a type of dynamic UML diagram that aim at illustrating the interactions between different actors (users or external systems) and the system. They provide a high-level view of the ways the system collaborates with external entities, further details on each specific use case will be provided in the following sections of this document.





The representation has been split into two different diagrams just for ease of reading. The Actors that are shown in the second diagram are the same as the ones in the diagram above.
Some of the use cases illustrated in the first diagram have relationships with other use cases. In particular:

- **EvaluateSolutions** is an extension of **TerminateBattle**, meaning that EvaluateSolutions offers additional functionalities to the TerminateBattle use case when the consolidation stage is requested by the educator for his/her battle. It is not mandatory to have a consolidation stage for a battle, therefore the relationship is just an ¡¡extend¿¿ link.

- **PushSoluitonGitHub** includes **CalculateScore**. This is due to the fact that every time a student pushes a new code solution on GitHub, the process to compute the score to assign to that solution is triggered in the CodeKataBattle system. Therefore, PushSolutionGitHub leverages every time the use case CalculateScore and cannot be completed in a successful way without the included use case.

It is not the purpose of these diagrams to show the sequencing and temporal constraints that hold in the system between the various use cases. These analyses will be better shown through sequence and activity diagrams down the line of this document.

3.2.2 UseCases

Starting from the scenarios presented in section 2.1.1 and through a process of abstraction, the following use cases can be derived. Also use cases are descriptions of interactions between the system under analysis and external entities of the world surrounding it, but they abstract from all the specific details that were used to give a tangible and representative idea to scenarios. This section also provide a detailed explanation of all the use cases that were mentioned in the use case diagrams in section 3.2.1.

[UC1] - EducatorLogsIn

Name	EducatorLogIn
Actors	Educator, GitHub.
Entry Condition	Educator has opened CodeKataBattle on his/her laptop.
Event Flow	1 - CodeKataBattle shows the log in interface with the button to sign in with a GitHub account. 2 - Educator clicks on the button to sign in using his/her GitHub account. 3 - CodeKataBattle redirects the educator to the GitHub login page. 4 - Educator inserts his/her GitHub credentials and confirms.
Exit Condition	CodeKataBattle shows the initial (home) page of the application for an educator.
Exceptions	1 - Educator inserts wrong credentials to access his/her GitHub account: the GitHub authentication page will return an error to the educator, asking him/her to retry.

[UC2] - StudentLogsIn

Name	StudentLogIn
Actors	Student, GitHub.
Entry Condition	Student has opened CodeKataBattle on his/her laptop.
Event Flow	1 - CodeKataBattle shows the log in interface with the button to sign in with a GitHub account. 2 - Student clicks on the button to sign in using his/her GitHub account. 3 - CodeKataBattle redirects the student to the GitHub login page. 4 - Student inserts his/her GitHub credentials and confirms.
Exit Condition	CodeKataBattle shows the initial (home) page of the application for an student.
Exceptions	1 - Student inserts wrong credentials to access his/her GitHub account: the GitHub authentication page will return an error to the student, asking him/her to retry.

[UC3] - CreateTournament

Name	CreateTournament
Actors	Educator, Student.
Entry Condition	Educator is logged in the CodeKataBattle platform.
Event Flow	1 - Educator clicks on the button to create a new tournament. 2 - CodeKataBattle shows the form to fill with the information for the tournament to be created. 3 - Educator types in the name of the tournament. 4 - Educator writes down a description of the tournament. 5 - Educator defines the registration deadline by which students are asked to register for the tournament if interested. 6 - Educator defines the badges (rewards) to be assigned to students at the end of the tournament based on some customary achievements. 7 - Educator confirms the information filled in the form. 8 - CodeKataBattle shows the initial page of the tournament to the educator. 9 - CodeKataBattle sends a notification to all the students on the platform to inform them of the new tournament available.
Exit Condition	The new tournament is added to the list of tournaments created by the educator and all users on the platform can see the new tournament.
Exceptions	1 - The registration deadline set by the educator is on the same day or before the day in which the educator is trying to create the new tournament: the system does not allow the educator to confirm the creation of the tournament and reports an error message. 2 - Educator confirms the form to create the new tournament leaving some mandatory information unspecified: in this case CodeKataBattle will return an error message to the educator stating that some information is missing and the tournament cannot be created.
Special Requirements	Mandatory data in the tournament creation form is composed of name and registration deadline of the tournament.

[UC4] - CreateBattle

Name	CreateBattle
Actors	Educator, Student.
Entry Condition	Educator is logged in the CodeKataBattle platform and has the permissions to publish a battle in the tournament which the battle will reside in. S/he has already written the build automation scripts and test cases for the battle to be created.

Event Flow	1 - Educator clicks on the button to create a new battle from the home page of CodeKataBattle . 2 - CodeKataBattle displays the entire list of tournaments in which the educator has permissions to publish battles. 3 - Educator selects the tournament in which s/he wants to create the new battle. 4 - CodeKataBattle shows the form to fill in with all the relevant data associated with the battle. 5 - Educator types in the name of the battle. 6 - Educator writes down the textual description of the battle. 7 - Educator uploads the build automation scripts related to the battle. 8 - Educator uploads the test cases related to the battle. 9 - Educator sets the registration deadline for the battle, by which students are asked to join if interested. 10 - Educator sets the submission deadline for the battle, after which CodeKataBattle stops accepting additional solutions to the battle. 11 - Educator sets the minimum and maximum number of students allowed per team. 12 - Educator selects the aspects on which CodeKataBattle automatically evaluates the students' code solutions, leveraging an external static analysis tool. For instance maintainability, security... 13 - Educator specifies whether a consolidation stage for personal evaluation of the students' code solutions is required at the end of the battle. 14 - Educator clicks on the button to confirm the data inserted in the form. 15 - CodeKataBattle shows the educator the presentation page of the newly created battle. 16 - CodeKataBattle sends a notification to all the students subscribed to the tournament in which the battle has been created, informing them of the new battle available.
Exit Condition	The new battle is added on the list of battles created by the educator and students are able to see this new battle in the tournament in which it resides.
Exceptions	1 - The registration deadline set by the educator is on the same day or before the day in which the educator is trying to create the new battle: the system does not allow the educator to confirm the creation of the battle and reports an error message. 2 - The submission deadline set by the educator is on the same day or before the day of the registration deadline: the system does not allow the educator to confirm the creation of the tournament and reports an error message. 3 - Educator confirms the creation of the battle without uploading either the build automation scripts or the test cases for the battle: the system doesn't create any new battle and reports an error message to the educator stating the reason why the battle cannot be instantiated. 4 - Educator confirms the creation of the battle without specifying some mandatory information for the battle: the system doesn't create any new battle and reports an error message to the educator stating the reason why the battle cannot be instantiated.

Special Requirements	Mandatory data in the battle form include the battle's name, the textual description of the problem to be solved, the registration and submission deadlines, the minimum and maximum number of students per team.

[UC5] - GrantPermissions

Name	GrantPermissions
Actors	Educator.
Entry Condition	Educator A is logged in the system. Educator A created a tournament on the platform.
Event Flow	1 - Educator A selects from the home page of CodeKataBattle the button to visualize the list the tournaments that s/he created. 2 - CodeKataBattle displays the list of tournaments created by educator A. 3 - Educator A clicks on the tournament in which s/he wants to grant permissions to educator B. 4 - CodeKataBattle opens the description page of the selected tournament. 5 - Educator A clicks on the button to grant the permissions to publish battles to another educator in the tournament. 6 - CodeKataBattle displays an interface to search for an educator by username. 7 - Educator A types in educator B's username, finds him/her and confirms the choice. 8 - CodeKataBattle sends a request to educator B to inform him/her of the possibility to publish battles in A's tournament. 9 - Educator B accepts the request.
Exit Condition	The system adds the tournament to the list of tournaments in which educator B has permissions to publish battles.
Exceptions	1 - Educator A tries to grant publishing permissions to educator B who already had permissions on that tournament: CodeKataBattle doesn't modify anything on B's account and reports to A that B already had publishing permissions on that tournament. 2 - Educator A tries to grant publishing permissions to educator B on a closed tournament: CodeKataBattle doesn't modify anything on B's account and reports to A the tournament is already terminated.

[UC6] - SubscribeToTournament

Name	SubscribeToTournament
Actors	Student.

Entry Condition	The student is logged in the system.
Event Flow	1 - Student selects the option to add a new tournament from the home page of CodeKataBattle . 2 - CodeKataBattle displays the list of available tournaments on the platform in the home page. 3 - Student browses the list of available tournaments and reads some of their descriptions to get an idea on them. 4 - Student clicks on the tournament s/he wants to subscribe to. 5 - CodeKataBattle shows the home page of the selected tournament. 6 - Student clicks on the button to subscribe to the tournament and confirms his choice.
Exit Condition	The system adds the tournament to the list of tournaments the student is subscribed to.
Exceptions	1 - Student tries to subscribe to a tournament whose registration deadline has already passed: CodeKataBattle reports an error message to the student stating that it is not possible to carry out the operation.

[UC7] - JoinBattleAlone

Name	JoinBattleAlone
Actors	Student.
Entry Condition	The student is logged in the platform and is subscribed to the tournament in which the battle s/he wants to join resides.
Event Flow	1 - Student opens the list of tournaments s/he's subscribed to from the home page of CodeKataBattle . 2 - CodeKataBattle displays the list of tournaments the student is subscribed to. 3 - Student clicks on the tournament in which the battle s/he wants to join resides. 4 - CodeKataBattle shows the home page of the selected tournament. 5 - Student selects the battle s/he wants to join and clicks on it. 6 - CodeKataBattle shows the description page of the selected battle with the button to join it. 7 - Student clicks the button to join the battle. 8 - CodeKataBattle shows the interface to join a battle. 9 - Student selects the option to participate as a single player and confirms.
Exit Condition	CodeKataBattle adds the battle to the list of battles the student is taking part to.

Exceptions	1 - Student tries to join a battle whose registration deadline has passed: CodeKataBattle shows an error message stating that it is not possible to carry out that operation and doesn't allow the student to join the battle. 2 - Student tries to join a battle in a tournament s/he is not subscribed to: CodeKataBattle doesn't allow this operation and reports a warning message stating that it is necessary to first subscribe to the tournament in order to participate to its battles.

[UC8] - JoinBattleAsTeam

Name	JoinBattleAsTeam
Actors	Student.
Entry Condition	Students A, B, C are logged in the platform and are all subscribed to the tournament in which the battle they want to join resides.
Event flow	1 - Student A opens the list of tournaments s/he's subscribed to from the home page of CodeKataBattle . 2 - CodeKataBattle displays the list of tournaments the student A is subscribed to. 3 - Student A clicks on the tournament in which the battle s/he wants to join resides. 4 - CodeKataBattle shows the home page of the selected tournament. 5 - Student A selects the battle s/he wants to join and clicks on it. 6 - CodeKataBattle shows the description page of the selected battle with the button to join it. 7 - Student A clicks the button to join the battle. 8 - CodeKataBattle shows the interface for joining a battle. 9 - Students A opts for participating to the battle as a team. 10 - CodeKataBattle shows an interface in which it is possible to type in the team's name and search for other students by username to invite them in the team. 11 - Student A types in the team's name. 12 - Student A types in B and C's usernames and selects them on the interface. 13 - Student A confirms his/her choices. 14 - CodeKataBattle sends a request to students B and C asking them to participate in the team created by A.
Exit condition	CodeKataBattle adds the battle to the list of battles in which A participates, as well as to the list of battles in which B participates and C participates. CodeKataBattle also creates a new team grouping A, B and C together.

Exceptions	1 - Student A tries to join a battle whose registration deadline has passed: CodeKataBattle shows an error message stating that it is not possible to carry out that operation and doesn't allow the student to join the battle. 2 - Student A tries to join a battle in a tournament s/he is not subscribed to: CodeKataBattle doesn't allow this operation and reports a warning message stating that it is necessary to first subscribe to the tournament in order to participate in its battles. 3 - Student A tries to invite a student that is not subscribed to the tournament in which the battle resides: CodeKataBattle won't allow this action by limiting the search space for the students to invite to the set of students subscribed to the tournament in which the battle resides. 4 - Students B or C do not accept the request to join the team before the registration deadline of the battle: CodeKataBattle excludes them from the battle. The request is no longer valid.

[UC9] - CreateGitHubRepository

Name	CreateGitHubRepository
Actors	GitHub, Student.
Entry Condition	There is a battle on CodeKataBattle whose registration deadline has just passed.
Event Flow	1 - CodeKataBattle sends a request to GitHub to generate a new repository dedicated to the battle whose registration deadline has just passed. 2 - GitHub creates the new repository. 3 - GitHub sends back a confirmation message to CodeKataBattle . 4 - CodeKataBattle sends a notification to all the students who joined the battle to inform them that the battle they're subscribed to is now open and it is possible to submit code solutions through GitHub. The system also includes the link to the remote GitHub repository in the notification message.
Exit Condition	All students subscribed to the battle receive the notification and are able to start pushing code solutions for the ongoing battle.
Exceptions	1 - GitHub doesn't respond to the request and doesn't create the new repository: CodeKataBattle sends again the request after waiting for a fixed time interval.

[UC10] - PushSolutionGitHub

Name	PushSolutionGitHub
Actors	Student, GitHub.

Entry Condition	Student is logged in the system and has already joined a battle that is ongoing (the registration deadline is passed, while the submission deadline hasn't).
Event Flow	1 - Student pushes a new code solution on the GitHub repository that s/he forked from the main GitHub repository dedicated to the battle. 2 - GitHub sends a notification to CodeKataBattle with the information relative to the new commit performed by the student. 3 - CodeKataBattle downloads the new code solution from GitHub. 4 - CodeKataBattle computes the score to assign to the new code solution (see UC11). 5 - CodeKataBattle shows on the platform the score assigned to the new solution. 6 - CodeKataBattle updates the ranking of teams in the battle accordingly to the student's new score.
Exit Condition	The student is able to see on CodeKataBattle the score of his/her solution and the new ranking of the battle.
Exceptions	-

[UC11] - CalculateScore

Name	CalculateScore
Actors	StaticAnalysisTool.
Entry Condition	A student has just pushed a new code solution on his/her GitHub repository linked to a battle, CodeKataBattle has received the notification from GitHub and downloaded the code of the student to be evaluated.
Event Flow	1 - CodeKataBattle uses the build automation scripts to build the student's code solution. 2 - CodeKataBattle runs the test cases on the student's solution and counts how many of them are passed. 3 - CodeKataBattle calculates the time went by from the beginning of the battle to the moment in which the solution was submitted and stores the information. 4 - CodeKataBattle leverages the StaticAnalysisTool to evaluate the student's solution based on some criteria (such as maintainability, security...) defined by the educator that created the battle at battle creation time. 5 - CodeKataBattle calculates the total score (between 0 and 100) to assign to the student's solution based on the information gathered at points 2, 3, 4.
Exit Condition	CodeKataBattle saves the total score computed for the student's solution
Exceptions	1 - The building process of the code solution fails: CodeKataBattle terminates the computation and assigns a score of 0 to the solution.

[UC12] - EvaluateSolutions

Name	EvaluateSolutions
Actors	Educator.
Entry Condition	Educator is logged in CodeKataBattle . Educator has already created a battle and required a consolidation stage at the end of the battle. The submission deadline of the battle has just passed.
Event Flow	1 - Educator navigates on the system to the battle for which the consolidation stage has to be carried out and clicks on the button to assess the teams' solutions. 2 - CodeKataBattle shows an interface in which for each team participating in the battle it is possible to specify a score to assign to it. On the interface there is also a timer illustrating the maximum amount of time to complete the task. 3 - Educator reads the teams' solutions one by one and inputs in the system the corresponding scores. 4 - After having assessed all the solutions, the educator confirms his/her choices.
Exit Condition	CodeKataBattle saves the scores assigned by the educator in order to calculate the final ranking for the battle.
Exceptions	1 - Educator closes the consolidation stage interface before having assessed all teams' solutions: CodeKataBattle won't publish any final ranking yet and will save the scores assigned so far by the educator. 2 - Educator doesn't complete the task of assigning personal scores before the timer goes off: CodeKataBattle will discard all the scores assigned by the educator and will base the final ranking only on the automatic evaluations performed during the battle.

[UC13] - TerminateBattle

Name	TerminateBattle
Actors	Educator, Student.
Entry Condition	There is a battle created on CodeKataBattle whose submission deadline has just passed.

Event Flow	1 - CodeKataBattle detects that the submission deadline of a battle is passed. 2 - If a consolidation stage was required by the educator who created the battle, CodeKataBattle initiates the process to carry out the personal evaluation of the teams' solutions by the educator (see UC10). 3 - CodeKataBattle computes the final ranking of teams for the battle, considering both the scores automatically assigned by the platform during the battle and possibly the personal scores provided by the educator (if the consolidation stage was required and successfully completed). 4 - CodeKataBattle publishes the final ranking of the battle on the platform. 5 - CodeKataBattle sends a notification to all the students participating in the battle, informing them of the availability of the final ranking for the battle. 6 - CodeKataBattle automatically updates the ranking of the tournament that contains the battle, adding to each student's previous points, the score obtained during the battle.
Exit Condition	All students involved in the battle and the educator that created the battle are able to visualize the final ranking of the battle and all users can see the updated tournament ranking.

[UC14] - CloseTournament

Name	CloseTournament
Actors	Educator, Student.
Entry Condition	Educator is logged in CodeKataBattle . Educator created a tournament on the platform.
Event Flow	1 - Educator retrieves from the home page of the CodeKataBattle platform the list of tournaments that s/he created. 2 - CodeKataBattle displays the list of tournaments the educator created. 3 - Educator selects from the list the tournament that s/he wants to close. 4 - CodeKataBattle shows the home page of the tournament. 5 - Educator clicks on the button to close the tournament and confirms the choice. 6 - CodeKataBattle sends a notification to all the students subscribed to the tournament informing them that the tournaments has been closed. 7 - CodeKataBattle assigns the badges (rewards) to all the students that completed some achievements in the tournament (defined by the educator that created the tournament at tournament creation time).
Exit Condition	CodeKataBattle removes the tournament from the list of available tournaments on the platform, so that it will no longer be displayed to the users of the application. All personal profiles of students are updated if they received some badges at the end of the tournament.

Exceptions	1 - Educator tries to close a tournament before the registration deadline of the tournament passes: CodeKataBattle won't allow the action and reports an error message stating that it is not possible to close a tournament that hasn't even started yet. 2 - Educator tries to close a tournament when some battles are still going on inside the tournament: CodeKataBattle won't allow the educator to close the tournament yet, until all battles are over. At the same time though, CodeKataBattle will no longer let any educator publish battles in that tournament. As soon as all the battles that are still going on are concluded, also the tournament will close automatically.

[UC15] - OpenTournamentRanking

Name	OpenTournamentRanking
Actors	User (Student or Educator).
Entry Condition	User is logged in the platform and there is an ongoing tournament on CodeKataBattle whose registration deadline is passed.
Event Flow	1 - User browses the home page of CodeKataBattle to search for a tournament. S/he clicks on the tournament s/he wants to see the ranking of. 2 - CodeKataBattle displays the home page of the selected tournament. 3 - User clicks on the button to display the tournament's ranking. 4 - CodeKataBattle displays the tournament's ranking.
Exit Condition	User is able to see the tournament's ranking on his/her screen.
Exceptions	1 - User attempts to see the ranking of a tournament whose registration deadline hasn't passed yet: CodeKataBattle doesn't allow this action simply by not providing any button on the interface to access the ranking.

[UC16] - OpenBattleRanking

Name	OpenBattleRanking
Actors	User (Student or Educator).
Entry Condition	User is logged in the platform. There is at least one tournament on CodeKataBattle in which at least a battle has been published. The battle's registration deadline has already passed.

Event Flow	1 - User selects from the home page a tournament s/he is involved in (if the user is a student, then a tournament s/he's subscribed to, if the user is an educator, a tournament in which s/he has permissions to publish battles). 2 - CodeKataBattle shows the selected tournament's home page. 3 - User selects from the tournament's home page a battle s/he's involved in (if the user is a student, then a battle s/he joined, if the user is an educator, then a battle s/he created). 4 - CodeKataBattle shows the battle's description page. 5 - User clicks on the button to see the battle's ranking. 6 - CodeKataBattle checks if the user that is trying to access the battle's ranking is involved in the battle or not (if the user is a student, CodeKataBattle verifies if the student is participating in the battle, if the user is an educator, CodeKataBattle verifies if the educator has created the battle or not). 7 - CodeKataBattle shows the battle's ranking if the user requesting it is involved in the battle.
Exit Condition	User sees the ranking on his screen.
Exceptions	1 - User tries to access a battle's ranking without being involved in it: CodeKataBattle will detect the fact that the user is not involved in the battle (with the checks performed on point 6) and won't show any ranking to him/her. Instead, an error message is reported to the user stating that s/he doesn't have the permissions to access the ranking.

[UC17] - ShowProfileAndBadges

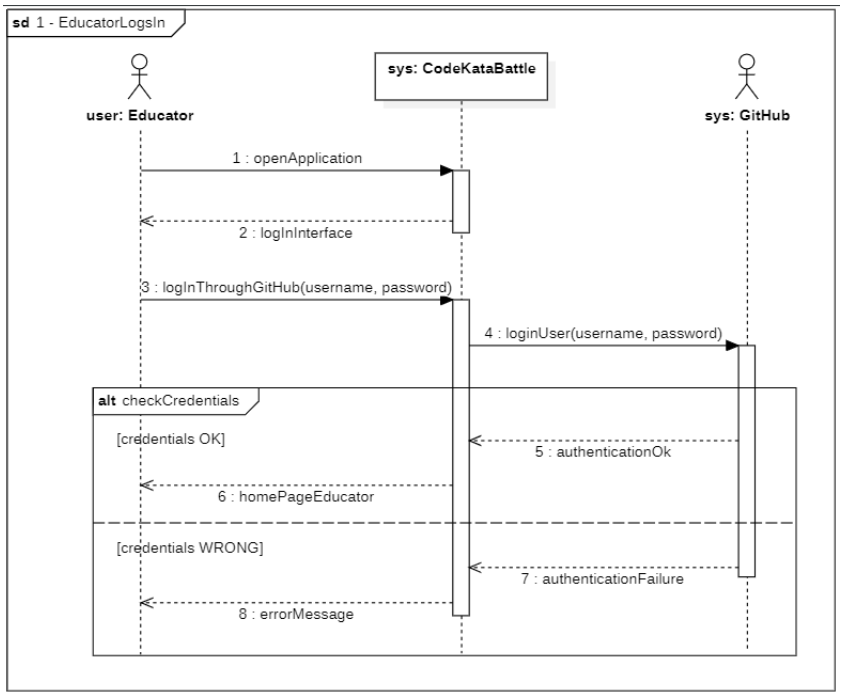
Name	ShowProfileAndBadges
Actors	User (Student or Educator).
Entry Condition	User A is logged in the platform. User B has logged in the platform at least once (to have a profile on the CodeKataBattle system).
Event Flow	1 - From the home page of CodeKataBattle, user A clicks on the button to search for user B by username. 2 - CodeKataBattle shows the interface for searching users on the platform. 3 - User A types in the username of B. 4 - CodeKataBattle shows the results of the search by username. 5 - User A clicks on B's profile. 6 - CodeKataBattle displays B's personal profile.
Exit Condition	User A sees B's profile on the screen and is also able to inspect the badges that B earned.

Exceptions	1 - User A searches for a username that doesn't exist on the system: CodeKataBattle will return no results from the search, so A is forced to change the username s/he's looking for.

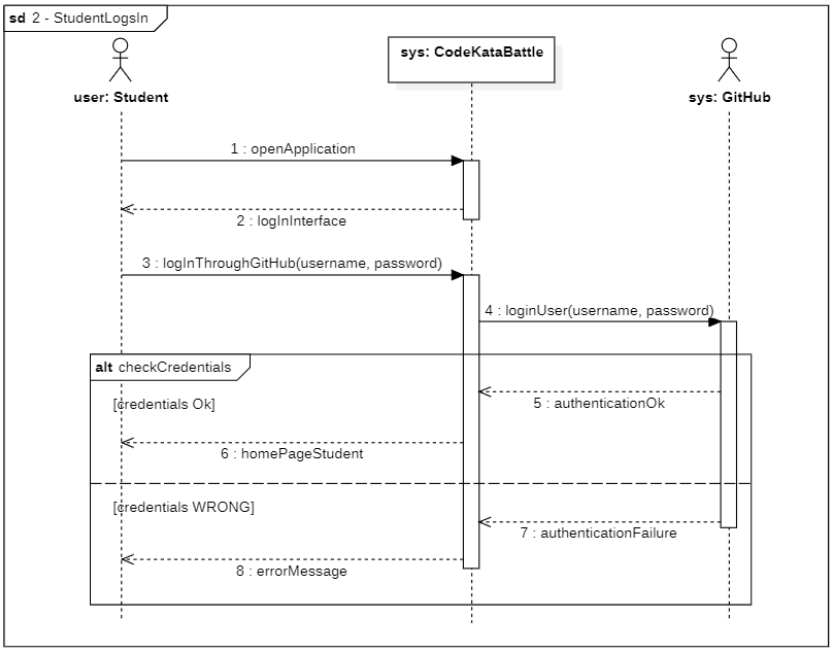
3.2.3 Sequence Diagrams

Sequence diagrams are employed to provide a dynamic view of the system under development. The focus of these diagrams is around data and information exchanged between CodeKataBattle and external entities and the temporal sequence in which these message transfers occur. A sequence diagram for each one of the use cases listed in section 3.2.2 is provided.

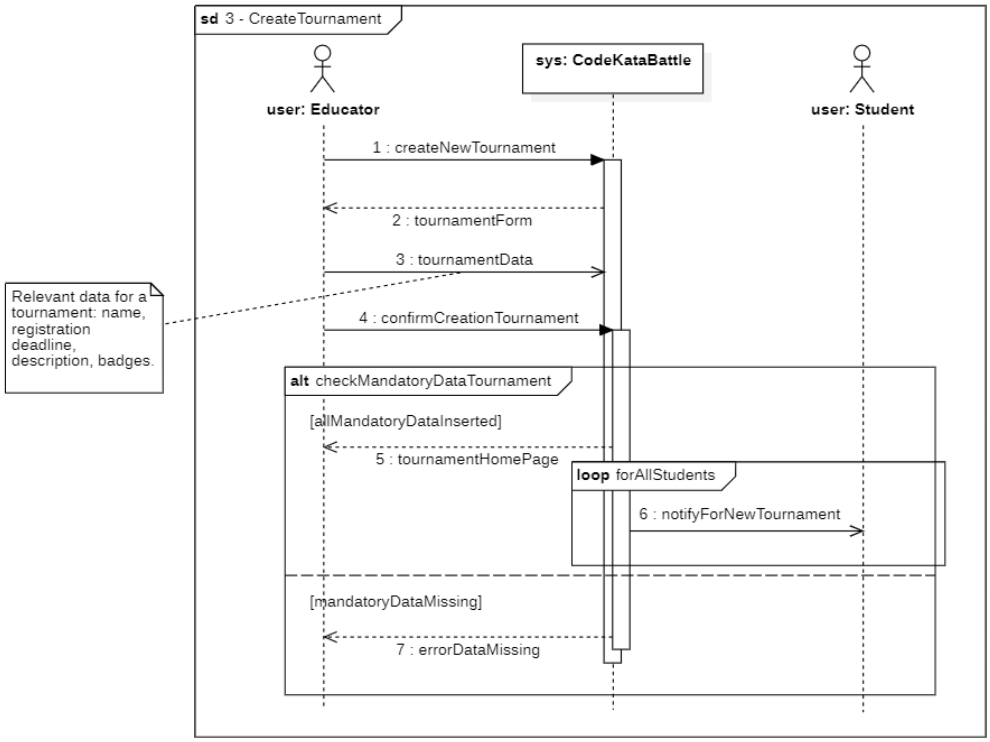
1 - EducatorLogsIn



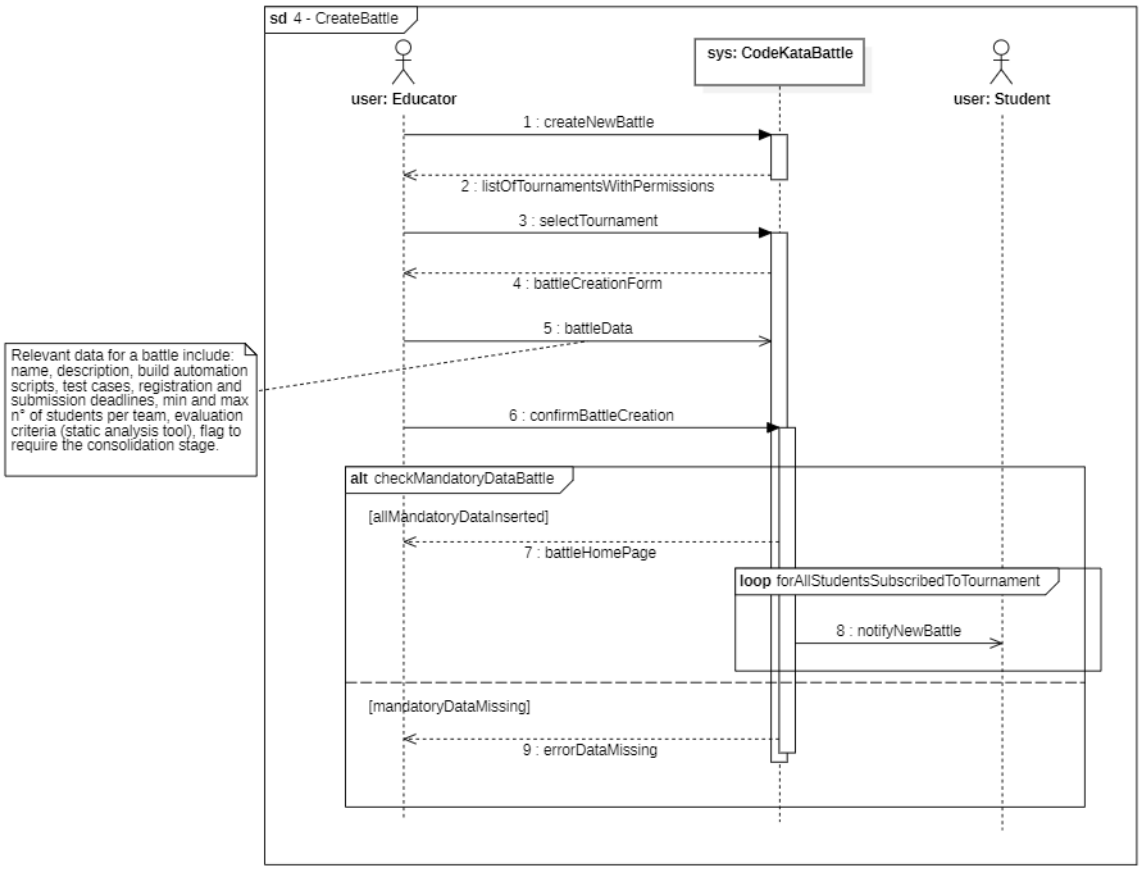
2 - StudentLogsIn



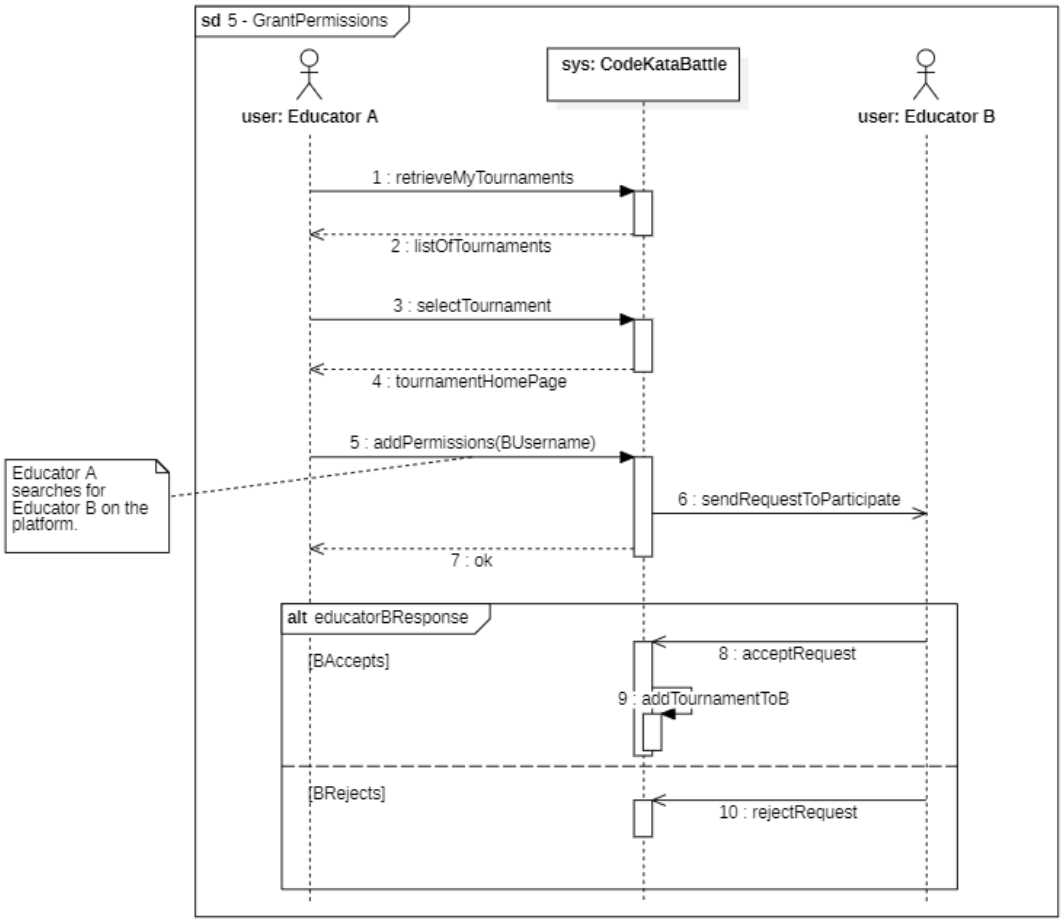
3 - CreateTournament



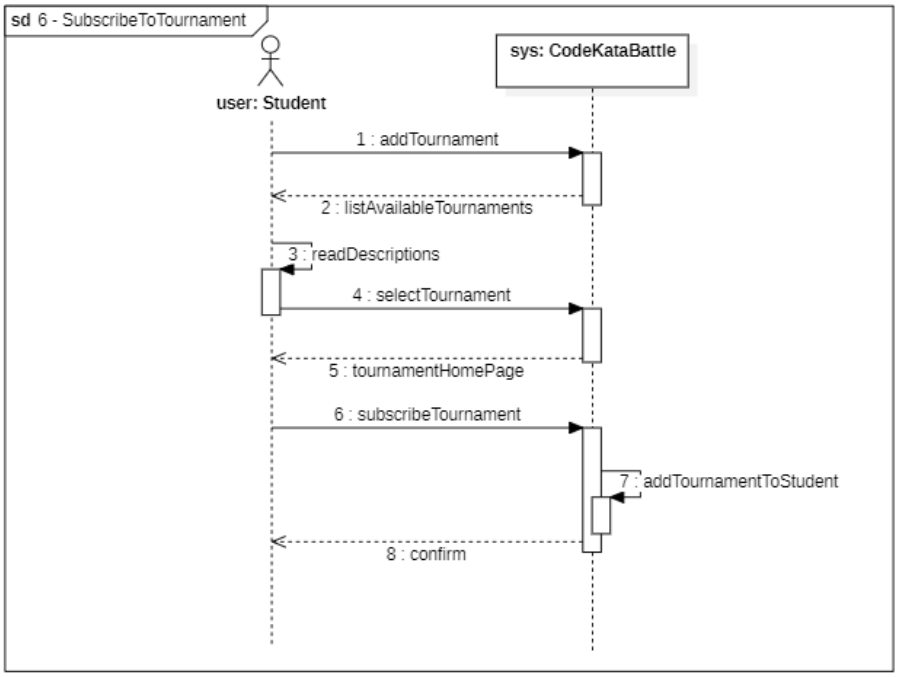
4 - CreateBattle



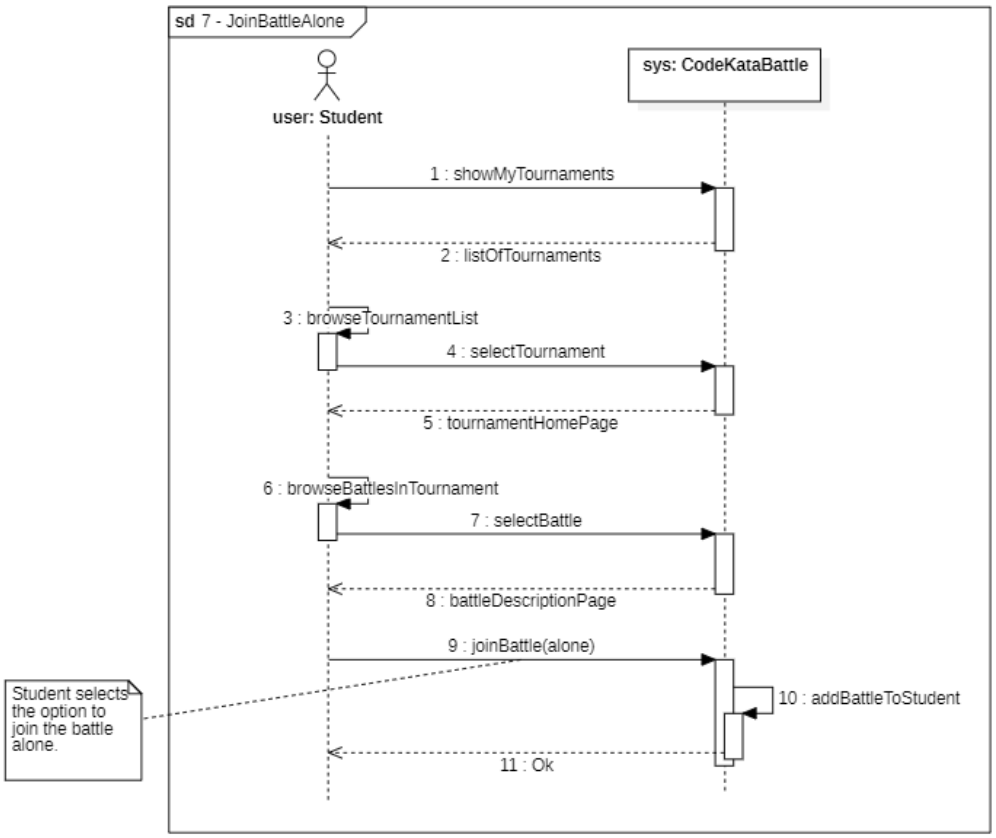
5 - GrantPermissions



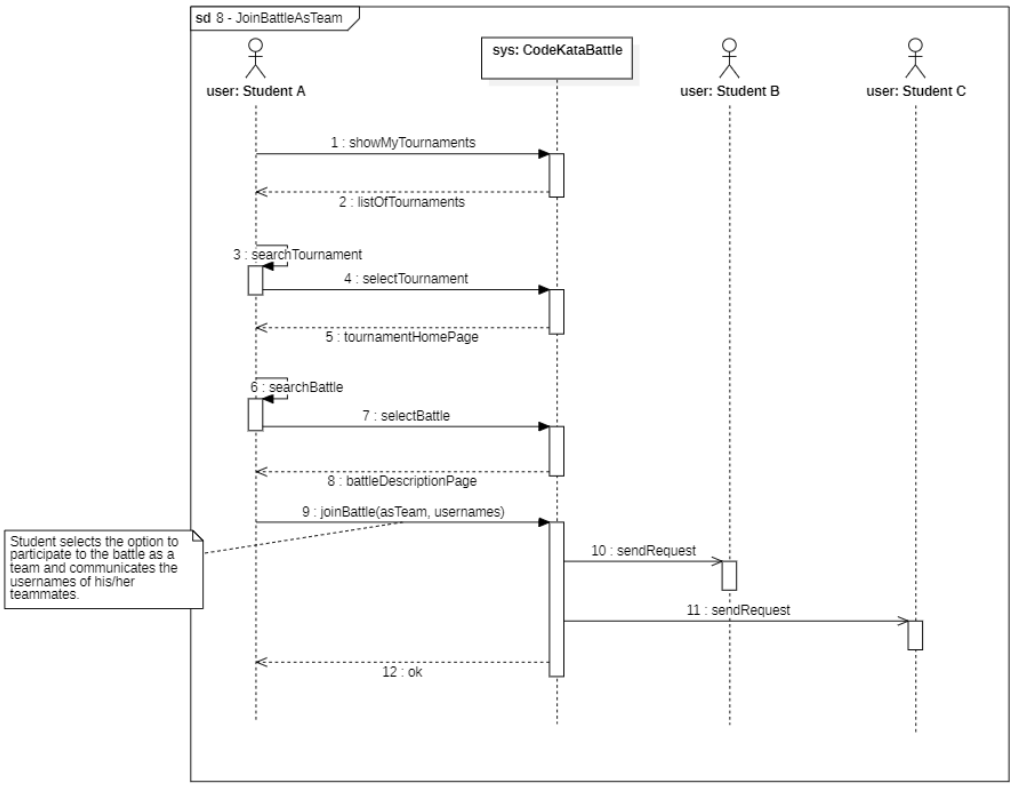
6 - SubscribeToTournament



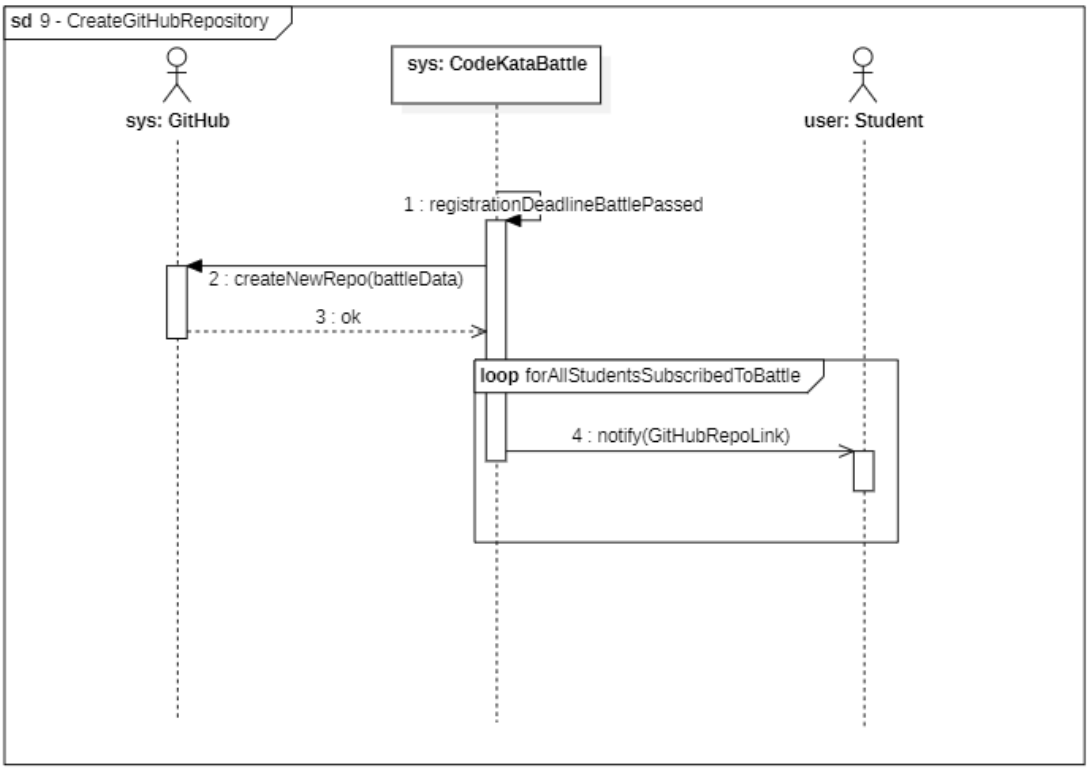
7 - JoinBattleAlone



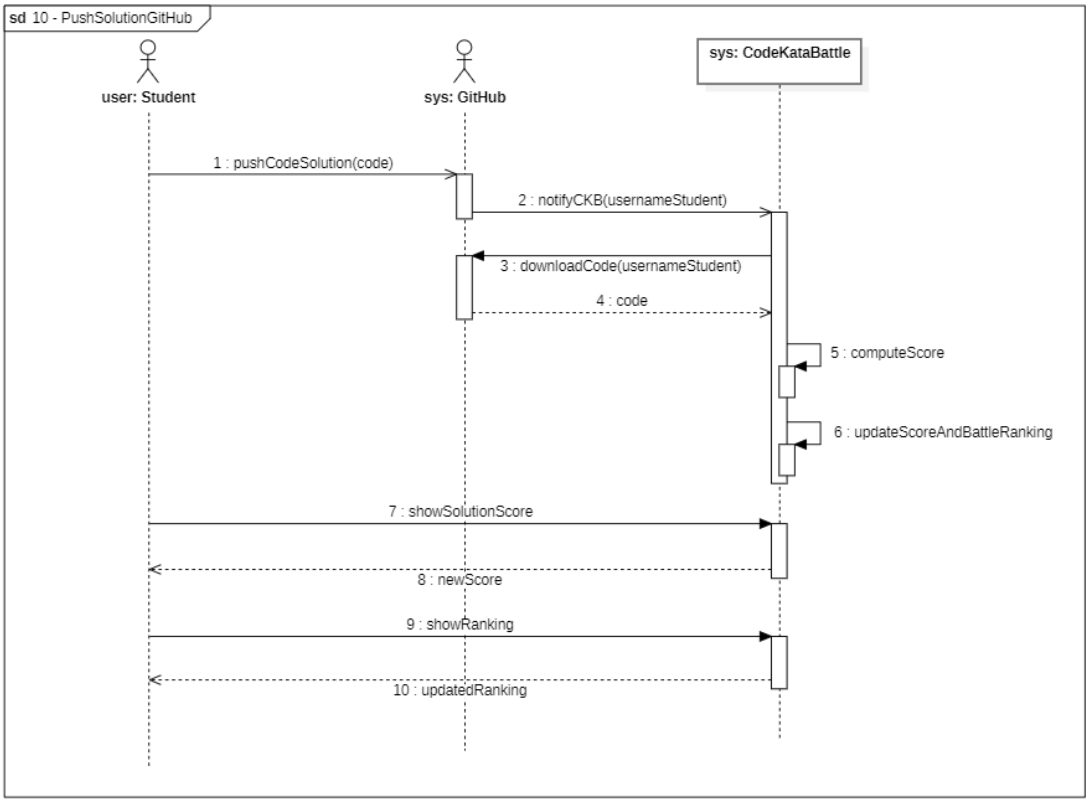
8 - JoinBattleAsTeam



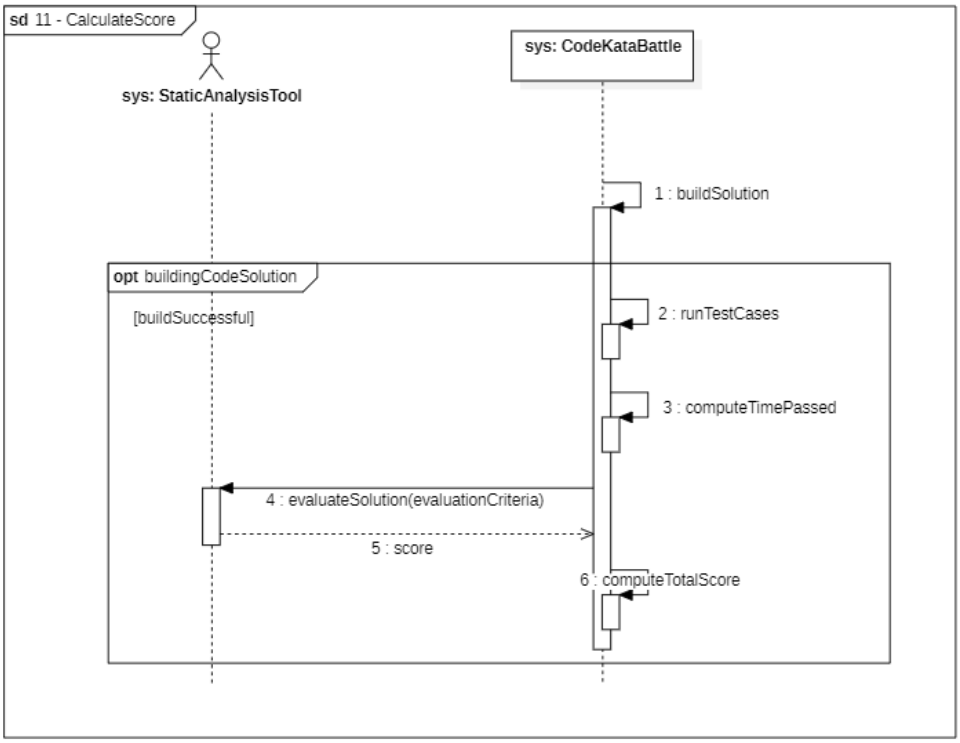
9 - CreateGitHubRepository



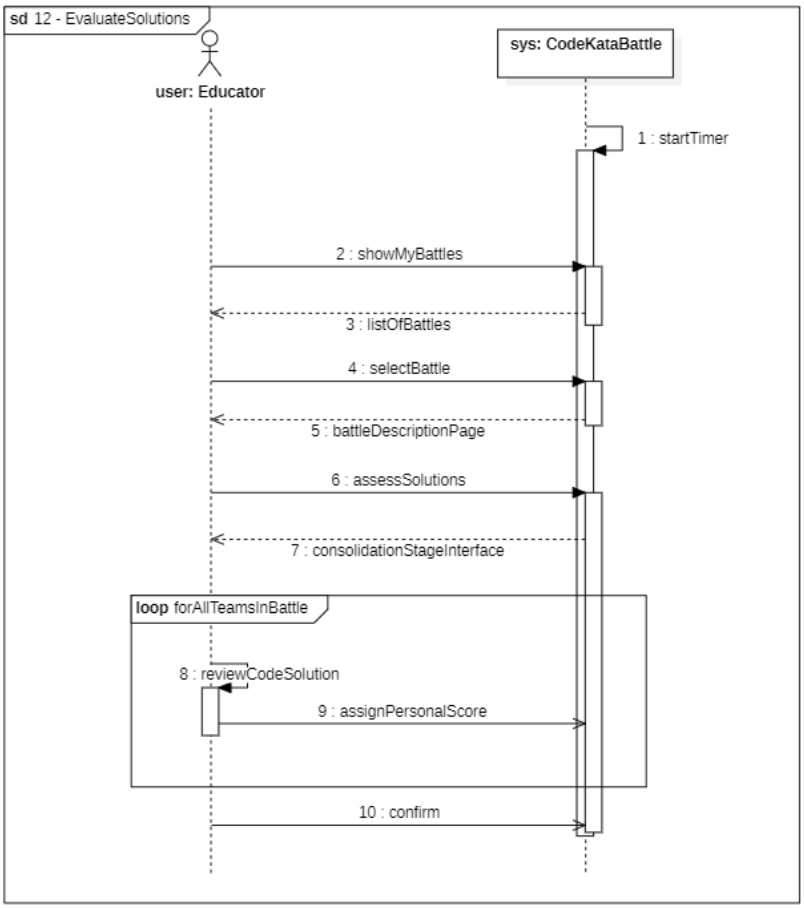
10 - PushSolutionGitHub



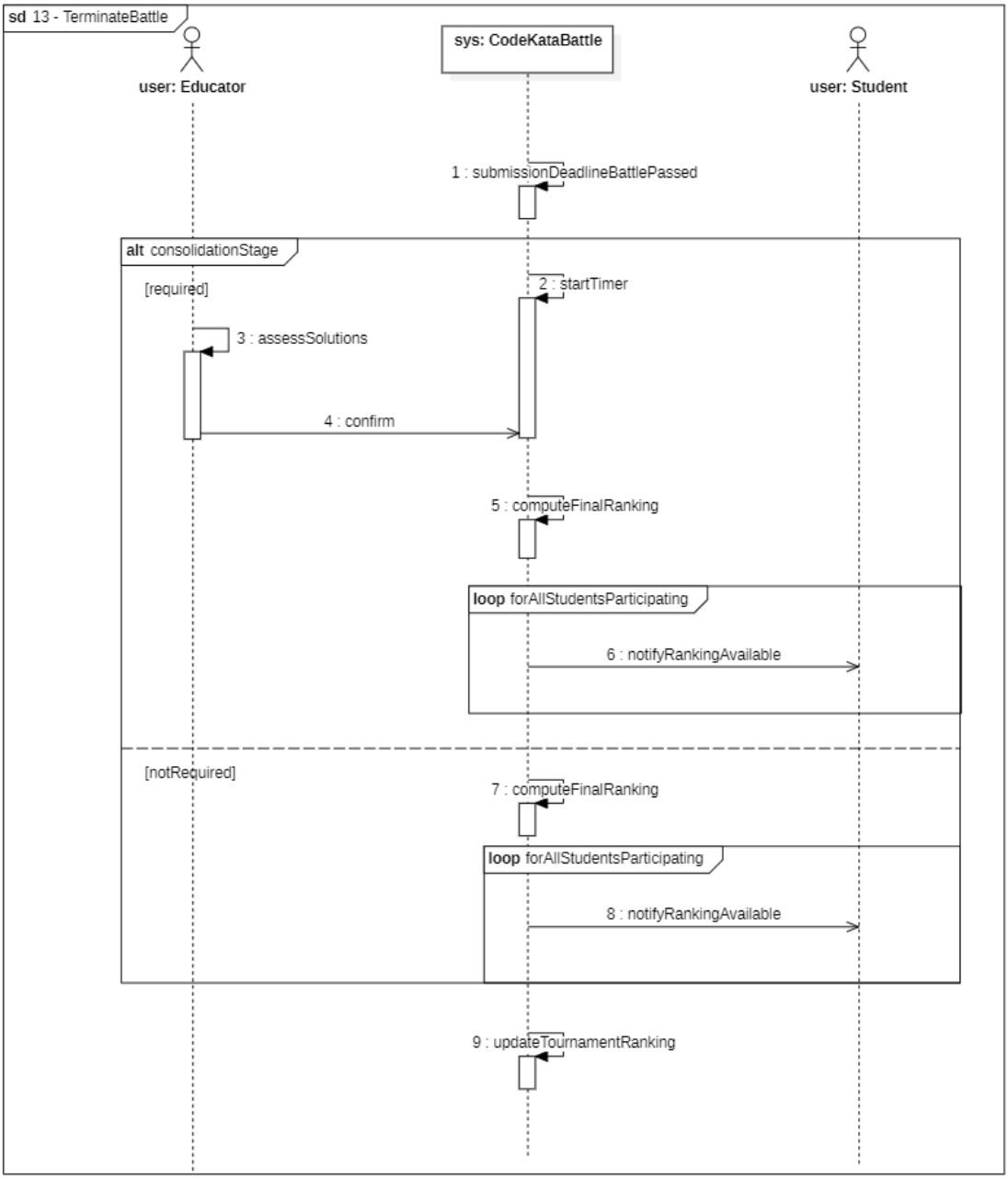
11 - CalculateScore



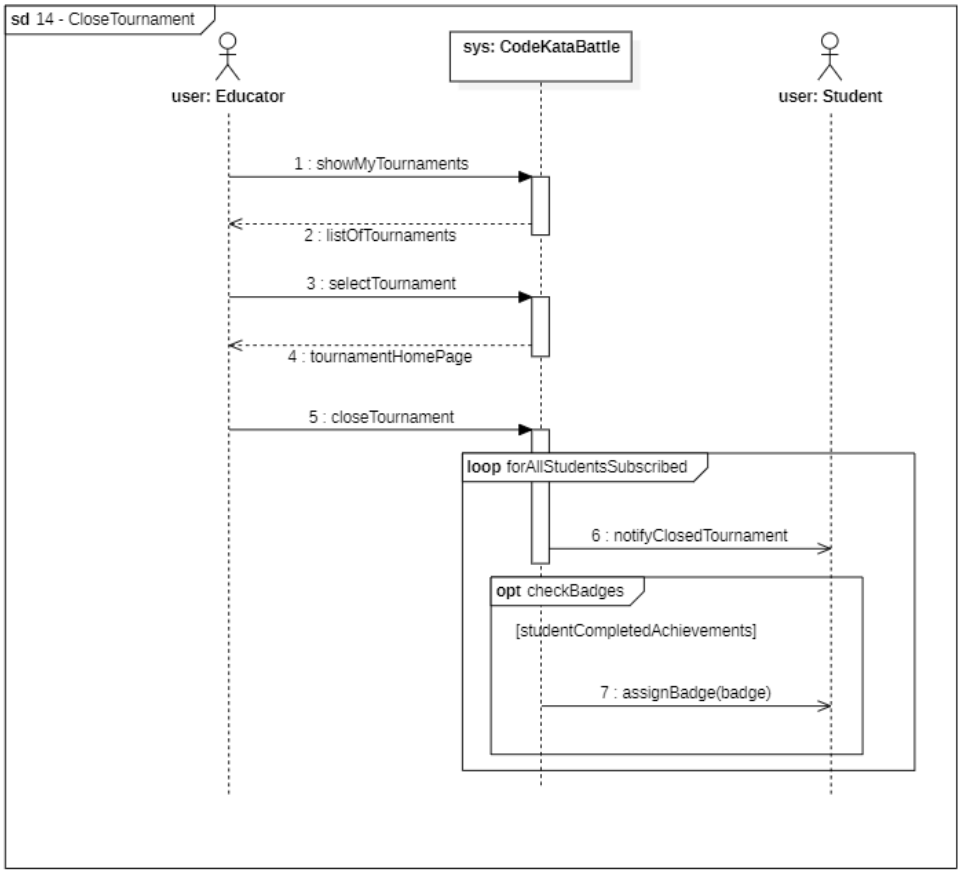
12 - EvaluateSolutions



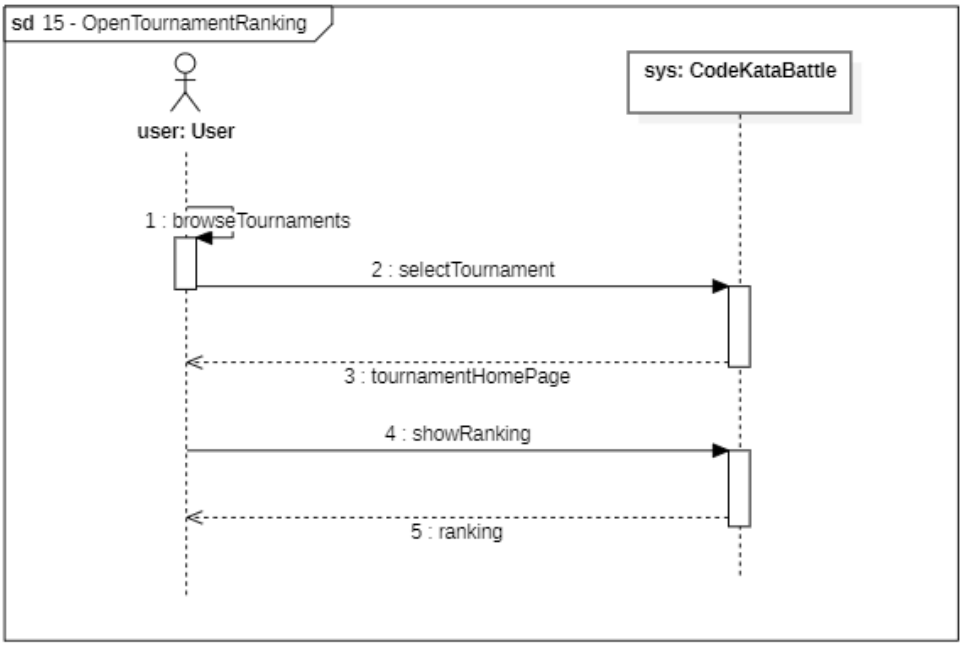
13 - TerminateBattle



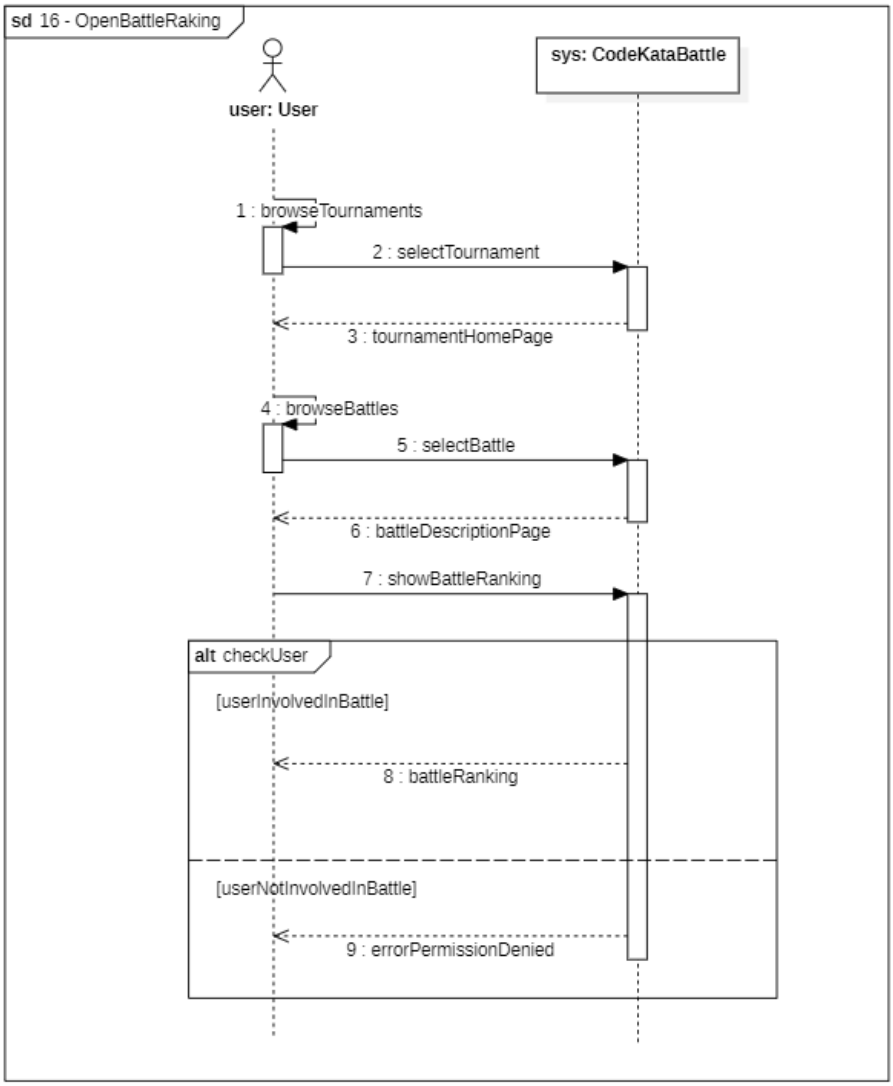
14 - CloseTournament



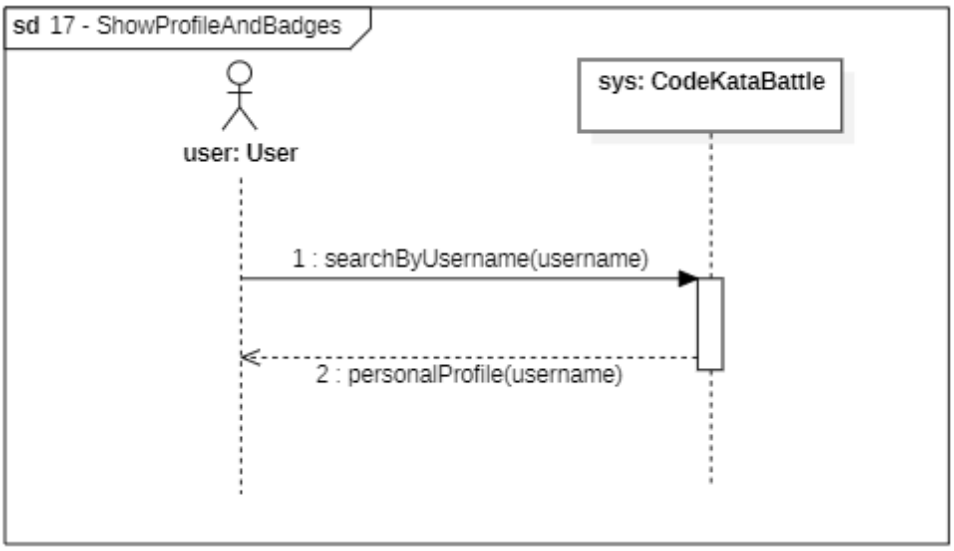
15 - OpenTournamentRanking



16 - OpenBattleRanking



17 - ShowProfileAndBadges



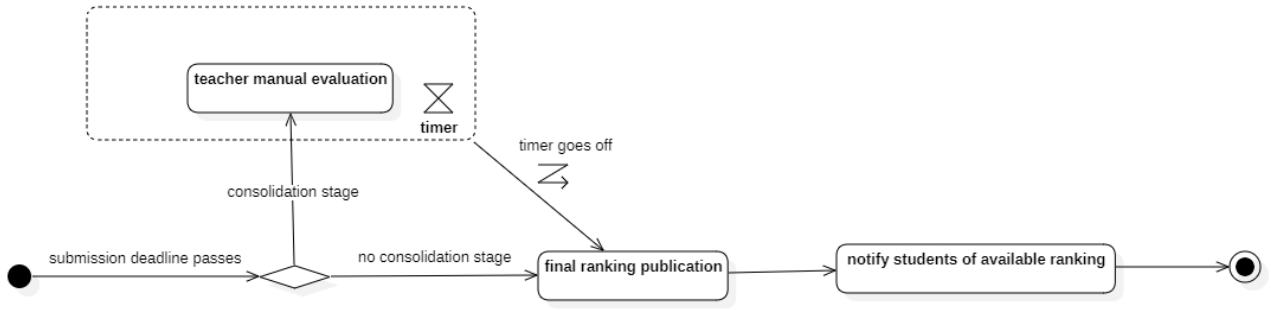
3.2.4 Activity Diagrams

Activity diagrams are employed to give insights to the reader about specific sequences of operations occurring in some parts of the system being described. In this case, they illustrate the procedures actuated to calculate the score of a code solution submitted through GitHub and to terminate a battle.

Computing Score

The first activity diagram regards the process for computing the score of a code solution to a battle submitted by a student through GitHub. The procedure is initiated when a new commit is pushed by a student in the main branch of his forked GitHub repository. GitHub sends a notification to CodeKataBattle , which causes the system to download the latest code solution. Since the final score is obtained by combining results of three different analyses (static analysis, timeliness and number of test cases passed) the diagram shows how these three computations are carried out in parallel.

TerminateBattle



This activity diagram delineates the process that leads to terminating a battle and publish the conclusive battle results. The procedure is initiated upon the expiration of the battle submission deadline. The diagram is of interest because it highlights the workflow of when a battle expects a manual evaluation. The **manual evaluation** activity is executed by an educator. If this activity doesn't terminate before the time set by the timer, the system autonomously proceeds ignoring the consolidation stage scores and publishing the final ranking based on the scores assigned during the battle.

3.2.5 Requirements Mapping

This table shows a mapping between:

- Requirements, presented in section 2.2.1.
- Goals, presented in section 1.1.
- Use Cases, presented in section 3.2.2.
- Sequence Diagrams, presented in section 3.2.3.

Requirement	Goal	UseCase	SequenceDiagram
R1.1	G1	UC2	SD2
R1.2	G1	UC3	SD3
R1.3	G1	UC6	SD6
R1.4	G1	UC7-8	SD7-8
R1.5	G1	UC6	SD6
R1.6	G1	UC4	SD4
R1.7	G1	UC7-8	SD7-8
R1.8	G1	UC7-8	SD7-8
R1.9	G1	UC9-10	SD9-10
R1.10	G1	UC11	SD11
R1.11	G1	UC13	SD13
R1.12	G1	UC14	SD14
R2.1	G2	UC1	SD1
R2.2	G2	UC3	SD3
R2.3	G2	UC5	SD5
R2.4	G2	UC4	SD4
R2.5	G2	UC13	SD13
R2.6	G2	UC12	SD12

Requirement	Goal	UseCase	SequenceDiagram
R2.7	G2	UC12-13	SD12-13
R2.8	G2	UC13	SD13
R2.9	G2	UC14	SD14
R3.1	G3	UC8	SD8
R3.2	G3	UC8	SD8
R3.3	G3	UC8	SD8
R3.4	G3	UC11	SD11
R3.5	G3	UC10	SD10
R3.6	G3	UC13	SD13
R3.7	G3	UC16	SD16
R3.8	G3	UC13	SD13
R3.9	G3	UC15	SD15
R3.10	G3	UC3	SD3
R3.11	G3	UC14	SD14
R3.12	G3	UC17	SD17
R3.13	G3	UC17	SD17

3.3 Performance Requirements

The CodeKataBattle application must deliver efficient performance to meet the demands of students and educators engaging in coding challenges. The system should aim for low latency, ensuring that the time between user interactions and system responses does not exceed a few seconds under normal internet conditions, but it's not a strict condition due to the nature of the application. It is acknowledged that users with slower internet connections may experience increased response times due to external factors beyond the system's control. The application should include mechanisms to detect and handle such situations effectively.

3.4 Design Constraints

3.4.1 Standard Compliance

In accordance with data privacy standards, the CodeKataBattle application operates within the framework of the General Data Protection Regulation (GDPR) in order to guarantee the privacy for students and educators data. This regulation governs data protection and privacy for individuals within the European Union (EU) and the European Economic Area (EEA). The application also adopts international standards for date and time representation.

3.4.2 Hardware Limitations

Students and educators hardware requirements are:

- Internet connection (2G/3G/4G/5G/Wi-Fi)
- iOS/Android smartphone or computer with any modern web browser installed (Chrome, Opera, Firefox, Edge, and others)

3.4.3 Other Constraints

Considering the diverse user base participating in coding challenges, the CodeKataBattle application should prioritize ease of use. The interface should be intuitive, catering to both novice and experienced users. The application must provide straightforward and user-friendly ways for individuals to interact with the platform.
It is mandatory to possess a GitHub account both for students and educators.

3.5 Software System Attributes

3.5.1 Reliability

The CodeKataBattle system must demonstrate high reliability, ensuring continuous operation without interruptions for extended periods. To achieve fault tolerance, the backend deployment should incorporate replication and redundancy, as it will be explained in the Design Document. Offline backups of data storage are essential for disaster recovery in case of data loss.

3.5.2 Availability

Given that CodeKataBattle is not an emergency service, the system can maintain an availability rate less then 99.9 percent. The average time between a fault occurrence and service recovery (MTTR) should be contained at a maximum of 7 days out of 365 per year. This implies that the service could remain highly available still not being completely reliable.

3.5.3 Security

As the system stores sensitive personal information about students and educators, security is of great importance. The central database must be fortified with comprehensive security measures to prevent external and internal attacks. Passwords stored in the data store must be encrypted, and in the case of password recovery, clear-text transmission should be avoided. All the channel of communication (e.g. notifications, mails) are assured to be encrypted in relation to the information contained in their messages (e.g. a password recovery is different from a reminder notification). Communication over the internet must utilize encryption to prevent traffic sniffing and spoofing, ensuring privacy and data consistency.

3.5.4 Maintainability

The CodeKataBattle system must uphold a high level of maintainability. Employing appropriate design patterns, adhering to coding standards, and avoiding hard-coding are imperative, so the precision of this document combined with the detailed descriptions of the Design Document are the key for this aspect. The codebase should be well-documented too, and a testing routine must cover at least 70 percent of the entire codebase, excluding interface code.

3.5.5 Portability

The application must be developed as a web application, so it can be used by smartphones, tablets and computers having modern web browser. In other words, the web application should be compatible with any operating system (e.g., Windows, Mac OS, Linux) that supports a web browser. For this reason it could be developed in a common programming language which guarantees the use of a virtual machine employed in the market (e.g. C, C++, C# don't seem to be a good choice; instead Java or Python do).

4 Alloy

In this section it is provided a representation of the world of CKB using the Alloy language. Every run and every check are commented in order to guarantee a syntactical correct compilation of the code: it is up to the reader to decide what to uncomment based on their will.

```
-- SIGNATURES (lexicographically orderded)

sig ABS {} --Automation Build Scripts

sig Badge {
    macrovariables: some Macrovariables,
    students: some Student,
    tournament: one Tournament
}

sig Battle {
    material: one EducatorMaterial,
    teams: set Team,
    educator: one Educator,
    tournament: one Tournament,
    maxTeamSize: one Int,
    minTeamSize: one Int,
    ranking: some Score,
    repositoryLink: one GitHubRepoLink,
    consolidationStage: one ConsolidationStage,
    manualEvaluation: one Boolean,
    regDeadline: one DateTime,
    subDeadline: one DateTime,
    additionalConfig: one EvaluationCriteria,
}{minTeamSize > 0 and minTeamSize < maxTeamSize and subDeadline.value
   > 0}

sig BattleNotification extends Notification {
    battle: one Battle
}

sig Boolean {
    value: one Int
}{value = 0 or value = 1}

sig CodeTest {}

sig ConsolidationStage {
    start: one DateTime, --useful to model the world, not used in
       practice
    end: one DateTime --as the above
    -- a start = 0 and an end = 0 represent that there is no
       ConsolidationState
}

sig DateTime {
```

```
    value: Int
}{value >= 0} --in order to simplify the signature, dates will be
   compared only by one value

sig Description {}

sig Educator extends User {
    battlesCreated: set Battle,
    tournamentsCreated: set Tournament,
    tournamentWithPermission: set Tournament --they include the
       tournament he creates
}

sig EducatorMaterial {
    automationBuildScripts: set ABS,
    textualDescription: one Description,
    tests: one CodeTest
} --in our world it can exist EducatorMaterial without a relation with
    a battle, as in the reality

sig EvaluationCriteria {}

abstract sig Notification {}

sig GitHubName {}

sig GitHubRepoLink {
    identifier: one Int
}{identifier >= 0}

sig Macrovariables {} --it is an implementing choice to decide how to
   model macrovariables

sig GitHubPassword {}

sig Score {
    var value: one Int
}{value >= 0}

sig Student extends User {
    team: set Team, --the partecipation to a tournament is reached
       through team
    tournaments: set Tournament --students without tournaments can
       exists
}

sig Team {
    members: some Student,
    participationToBattle: one Battle,
    score: one Score,
}{#members >= 1} --the student itself is treated as a one member team
```

```
sig Tournament {
    creator: one Educator,
    badges: set Badge,
    var battles: some Battle,
    ended: one Boolean, --if 1 the Date of end is different from 0
    endDate: one DateTime, --the date when the tournament is ended, 0
       if it is not ended
    ranking: some Score
}{ended.value > 0}

sig TournamentNotification extends Notification {
    tournament: one Tournament
}

sig User {
    password: one GitHubPassword,
    username: one GitHubName
}

-----------------------------------------------------------------

-- FACTS (lexicographically ordered)

-- General considerations: in our world GitHubNames and
   GitHubRepoLinks can exist, but it cannot exist a user without a
   proper GitHubName or a Battle without a proper GitHubRepoLink
-- Fact: Badge and tournament are biunivocally related
fact badgesAlwaysAssociatedWithTournament {
    all t: Tournament, b: Badge | b in t.badges implies b.tournament =
        t
}

-- Fact: Battles and tournament are biunivocally related
fact battleAlwaysAssociatedWithTournament {
    all t: Tournament, b: Battle | t in b.tournament implies b.
       tournament = t
    all t: Tournament, b: Battle | b.tournament = t implies b in t.
       battles
}

-- Fact: Every battle created by and educator belongs to the set of
   battle created by that educator
fact battleCreatedForEducators {
    all e: Educator, b: Battle | b.educator = e implies b in e.
       battlesCreated
}

-- Fact: Two educators can't create the same battle
fact battleCreatedByOneEducator {
```

```
    no disj e1, e2: Educator, b: Battle | b in e1.battlesCreated and b
        in e2.battlesCreated
}

-- Fact: A battle must end before the endDate of a tournament
fact battleEndInTournament {
    all b: Battle, t: Tournament | b in t.battles implies b.
       subDeadline.value < t.endDate.value
}

-- Fact: We need to define the correct time order of the consolidation
    stage, when its value is different from the error state (when both
    the start and specifically the end values equal zero)
fact consolidationStageTimeOrder {
    all cs: ConsolidationStage | cs.end.value!=0 implies cs.start.
       value < cs.end.value
}

-- Fact: DifferentUsername, so different GitHubNames
fact differentUsername {
    no disj u1, u2: User | some u1.username & u2.username
}

-- Fact: Every GitHubRepository has a different URI
fact differentURI {
    no disj g1, g2: GitHubRepoLink | some g1.identifier & g2.
       identifier
}

-- Fact: Every battle has a different GitHubRepo
fact differentGitHubRepo {
    no disj b1, b2: Battle | some b1.repositoryLink & b2.
       repositoryLink
}

-- Fact: Educator creates a battle in own tournament or in a
   tournament he has a permission for
fact educatorCreatesBattleInOwnTournament {
    all e: Educator, t: Tournament, b: Battle | b in e.battlesCreated
       or b in e.tournamentWithPermission.battles or b in e.
       tournamentsCreated.battles implies b in t.battles
}

-- Fact: Every tournament that is created by an educator is in his
   tournamentWithPermission
fact everyTournamentCreatedGivesPermissions {
    all t: Tournament, e: Educator | t in e.tournamentsCreated implies
        t in e.tournamentWithPermission
}

-- Fact: Manual evaluation requires the consolidation stage
```

```
fact manualEvaluationRequiresConsolidationStage {
    all b: Battle | (b.manualEvaluation.value = 0 implies b.
       consolidationStage.start.value = 0 and b.consolidationStage.end
       .value = 0) and
                (b.manualEvaluation.value = 1 implies b.
                   consolidationStage.start.value != 0 and b.
                   consolidationStage.end.value != 0)
}

-- Fact: Every user must be a student or an educator
fact noUserWithoutRole {
    Student + Educator = User
}

-- Fact: The registration must be done before the ending of the
   tournament
fact registrationBeforeTournamentEnd {
    all t: Tournament, b: Battle | b.tournament = t implies b.
       regDeadline.value < t.endDate.value
}

-- Fact: Students can't join a tournament more than one time
fact studentJoinsTournamentOnce {
    all t: Tournament, s: Student, team1, team2: Team |
        team1.members = s && team1.members.tournaments = t && team2.
           members = s && team2.members.tournaments = t implies team1
           = team2
}

-- Fact: If a student is a member of a team partecipating to a battle,
    the tournament to which that battle belongs belongs to the
   tournament to which that student is subscribed
fact studentParticipatesToTournament {
    all s: Student, team: Team, b: Battle | s in team.members and b in
        team.participationToBattle implies b.tournament in s.
       tournaments
}

-- Fact: A student can receive a badge only from a tournament he
   partecipated to
fact studentsBadge {
    no disj s: Student, b: Badge | s in b.students and b.tournament
       not in s.team.participationToBattle.tournament
}

-- Fact: The submission deadline of a battle must be always before the
    registration deadline
fact submissionAfterRegistration {
    all b: Battle | b.subDeadline.value < b.regDeadline.value
}
```

```
-- Fact: After a battle all teams receive a score
fact teamsHaveScoreAfterBattle {
    all b: Battle | all t: b.teams | some s: Score | s in t.score
}

-- Fact: Students and team are biunivocally related
fact teamStudent {
    all t: Team, s: Student | t in s.team implies s in t.members
    all t: Team, s: Student | s in t.members implies t in s.team
}

-- Fact: All teams partecipating to a battle are teams of that battle
fact teamBattle {
    all t: Team, b: Battle | t in b.teams implies b in t.
       participationToBattle
}

-- Fact: Every tournament can be created only by one educator
fact tournamentsCreatedByOneEducator {
    no disj e1, e2: Educator, t: Tournament | t in e1.
       tournamentsCreated and t in e2.tournamentsCreated
}

-- Fact: All creators' tournaments are tournaments created by him
fact tournamentCreatedForEducators {
    all t: Tournament, e: Educator | t.creator = e implies t in e.
       tournamentsCreated
}

-- Fact: A tournament has a date of ending only if its boolean value "
   ended" equals one
fact tournamentEndedIfBoolean {
    all t: Tournament | t.ended.value = 1 implies t.endDate.value != 0
}

-- Fact: There are no tournaments with the same battle
fact tournamentHaveDifferentBattles {
    no disj t1, t2: Tournament, b: Battle | b in t1.battles and b in
       t2.battles
}

-- Fact: If an educator creates battles in a tournament, he has the
   permission for it
fact tournamentsWithPermissionsForEducatorsCreatingBattlesInThem {
    all e: Educator, b: Battle | b in e.battlesCreated implies b.
       tournament in e.tournamentWithPermission
}

-- Fact: Every educator material is different
fact uniqueSourceCodeTestABS {
    all c1, c2: EducatorMaterial | c1 != c2 implies {
```

```
        c1.textualDescription != c2.textualDescription &&
        c1.tests != c2.tests &&
        c1.automationBuildScripts != c2.automationBuildScripts
    }
}

-------------------------------------------------------------

-- PREDICATES useful for asserts
pred addBattleToATournament[t: Tournament, b: Battle] {
    t.battles' = t.battles + b
}

pred delBattleFromATournament[t: Tournament, b: Battle] {
    t.battles' = t.battles - b
}

pred addTournament[e: Educator, t: Tournament] {
    e.tournamentsCreated' = (e.tournamentsCreated) + t
}

pred studentLeavesATeam[s: Student, t: Team] {
    s.team' = s.team - t
}

pred battleScoreRefresh[b: Battle, t: Team] {
    (b.ranking.score.value)' = b.ranking.score.value + t.score.value
}

pred tournamentScoreRefresh[t: Tournament, b: Battle] {
   (t.ranking.score.value)' = t.ranking.score.value + b.teams.score.
      value
}
-- The reader should be careful that these predicates are correctly
   generating expected world only when used in the following assertion
   :
-- In fact, it is up to the assertion to avoid the situations where an
    union wants to add an object already part of the set or where a
   subtraction wants to delete an object not belonging to the set

-------------------------------------------------------------

-- ASSERTIONS
-- Assertion: when adding a battle not previously belonging to a
   tournament, the total number of the battles of the tournament is
   increased by one
assert addBattle {
   all t: Tournament, b: Battle | before b not in t.battles and after
      addBattleToATournament[t, b] implies always #t.battles' = (#t.
      battles + 1)
}
```

```
--check addBattle for 5

-- Not considering the case of adding a battle not previously
   belonging to a tournament where the educator who created that
   tournament will have this battle in his created battles, because
   there could be a lot of educators with permissions, not granting he
    is the one who created the battle added

-- Assertion: when deleting a battle previously belonging to a
   tournament, the total number of the battles of the toruanement is
   decreased by one
assert deleteBattle {
   all t: Tournament, b: Battle | before b in t.battles and after
      delBattleFromATournament[t, b] implies always #t.battles' = (#t.
      battles - 1)
}
--check deleteBattle for 5

-- Assertion: when adding a tournament by an educator not previously
   belonging to the tournaments he created, the total number of his
   created tournament is increased by one
assert addTournament {
   all t: Tournament, e: Educator | before t not in e.
      tournamentsCreated and after addTournament[e, t] implies always
      #e.tournamentsCreated' = (#e.tournamentsCreated + 1)
}
--check addTournament for 5

-- Assertion: when deleting a battle previously belonging to a
   tournament, and then adding it to that tournament, the world
   represented after two times will be the initial one
assert delBattleAfterInsert {
   all t: Tournament, b: Battle | before b in t.battles and
      addBattleToATournament[t, b] and after delBattleFromATournament[
      t, b] implies t.battles'' = t.battles
}
--check delBattleAfterInsert for 5

-- Assertion: when a student leaves his team, then he will be no more
   a member of that team
assert studentLeavesHisTeam {
   all s: Student, t: Team | before t in s.team and after
      studentLeavesATeam[s, t] implies t not in s.team'
}
--check studentLeavesHisTeam for 5

-- Assertion: checking that the score is been well updated
assert totalScoreRefresh {
   all b: Battle, t: Team | always (battleScoreRefresh[b, t] and
      after tournamentScoreRefresh[b.tournament, b] implies (b.
         tournament.ranking.value)' = b.tournament.ranking.value + t.
```

```
        score.value )
}
--check totalScoreRefresh for 5

-- Assertion: checking that the start and end value of the
   consolidation stage are different from zero only when the educator
   asked for the manual evaluation
assert consolidationStageOnlyIfManualEvaluation {
   all b: Battle | b.manualEvaluation.value = 0 implies
      b.consolidationStage.start.value = 0 and b.consolidationStage.
         end.value = 0
   all b: Battle | b.manualEvaluation.value = 1 implies
      b.consolidationStage.start.value != 0 and b.consolidationStage.
         end.value != 0
}
--check consolidationStageOnlyIfManualEvaluation for 5

-----------------------------------------------------------------

-- RUN
-- A general world. The objects represented are a few in other to make
    the generated image readable
pred general[] {
    #GitHubName = 4
    #Educator = 1
    #Student = 2
    #Team = 1
    #Tournament = 1
    #Battle = 2
    #GitHubRepoLink = 3
}

-- A world where there are two battle, one with the manual evalution,
   the other without it
pred noManualEvaluation[] {
    #GitHubName = 4
    #Educator = 1
    #Student = 2
    #Team = 1
    #Badge = 2
    #GitHubRepoLink = 3
    #{b1, b2: Battle | b1.manualEvaluation.value = 0 && b2.
       manualEvaluation.value = 1} = 1
}

-- A world where there are educator creating tournaments, and other
   only with permission to a tournament created
pred educatorsWithOnlyPermission[] {
    #Team = 1
    #Tournament = 3
    #Battle = 3
```

```
    #{e1, e2: Educator | #e1.tournamentWithPermission = 1 && #e1.
       tournamentsCreated = 0 and #e2.tournamentsCreated = 1} = 1
}

--run general for 4
--run noManualEvaluation for 4
--run educatorsWithOnlyPermission for 4
```

4.1 Generated World

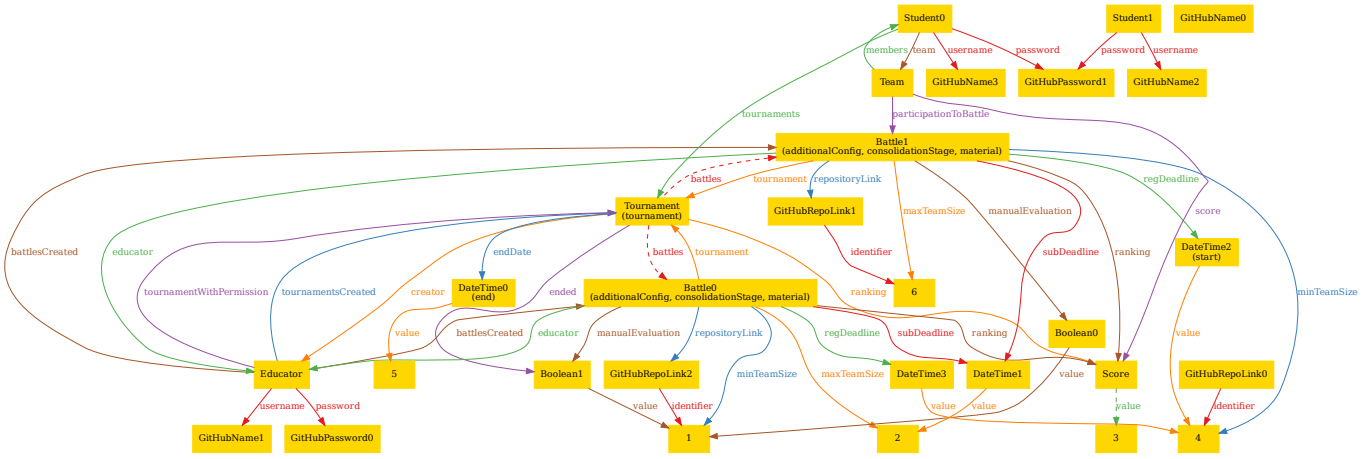


Figure 1: The world generated by running *general* predicate projected over nine signatures (ABS, CodeTest, ConsolidationStage, Description, EducatorMaterial, EvaluationCriteria, Macrovariables, Notification, String) in order to be more comprehensible. This world represents a normal situation of the model

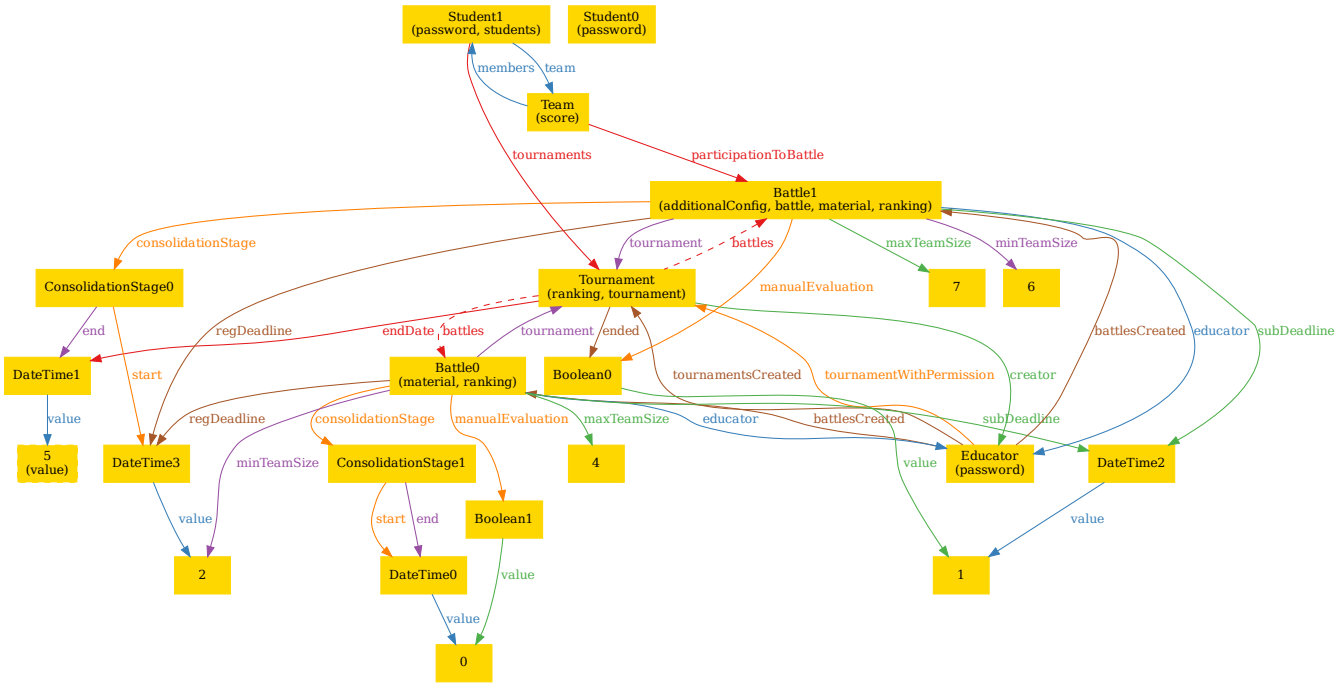


Figure 2: The world generated by running *noManualEvaluation* predicate projected over thirteen signatures (ABS, Badge, CodeTest, Description, EducatorMaterial, EvaluationCriteria, GitHubName, GitHubPassword, GitHubRepoLink, Macrovariables, Notification, Score, String) in order to be more comprehensible. This world represents the situation where there is no manual evaluation for one battle out of two. It is useful in order to see the right aim of the model. Indeed, as you can see, in the case the manualEvaluation is not required (the boolean value equals 0) there is no consolidation stage, and viceversa

Student
(password, students)
Team
(scored)
Educator0
(password)
Educator1
(password)
Educator2
(password, username)
Tournament0
(ended, ranking)
Tournament1
(endDate, ended, ranking)
Tournament2
(endDate, ended, ranking, tournament)
Battle0
(consolidationStage, manualEvaluation, material, ranking)
Battle1
(consolidationStage, manualEvaluation, material, ranking)
Battle2
(additionalConfig, consolidationStage, manualEvaluation, material, ranking, repositoryLink)
members
team
tournaments
tournamentWithPermission
participationToBattle
battlesCreated
educator
tournamentsCreated
creator
battles
tournament

Figure 3: The world generated by running *educatorsWithOnlyPermission* projected over seventeen signatures (ABS, Badge, Boolean, CodeTest, ConsolidationStage, DateTime, Description, Educator-Material, EvaluationCriteria, GitHubName, GitHubPassword, GitHubRepoLink, Int, Macrovariables, Notification, Score, String) in order to be more comprehensible. This world represents the situation where there are educator with permissions for a tournament they haven't created. As you can see, it cannot exist a tournament without a creator, but some educators can have tournaments with only the permission, without having created them

5 Effort spent

Studente	Introduction	Overall Description	Specific Requirements	Alloy
Alessandro	5	12	15	30
Matteo	17	12	9	Cell 4
Sara	0	15	5	0

Table 23: Time spent on every section of the RASD for each student