AY 2023/2024

POLITECNICO
MILANO 1863

RASD: Requirement Analysis and Specification Document

Tommaso Capacci Gabriele Ginestroni

Professor
Elisabetta Di Nitto

Version 1
December 22, 2023

Contents

1 Introduction

1.1 Purpose

Traditional classroom teaching focuses heavily on theory and concepts, without providing sufficient opportunities for students to gain hands-on coding experience. Additionally, the student-instructor relationship is usually limited to formal lectures and exams, missing some kind of continuous evaluation, which is essential to stimulate learning.

CodeKataBattle aims to fill these gaps by facilitating competitive programming challenges that motivate students to put concepts into practice. The aim of the product is to provide an alternative to evaluate students with respect to their coding skills in a more enjoyable way: this is achieved by allowing educators to create tournaments in which students can partecipate and compete. Through the integration with static analysis tools, the automated scoring system provides reliable evaluation so students can measure their improvements. By collaborating in teams, they also learn key skills like communication and source control. Indeed, partecipating to battles, students will gain experience in using GitHub as source control system.

These kind of evaluation also teaches how the test-first approach can be useful when it comes to developing a new piece of software.

For instructors, CodeKataBattle enables closer mentorship driven by qualitative code reviews, by including optional manual evaluation. In this way, educators will have the opportunity to enhance their code review skills while increasing the engagement with their students. Overall, the platform creates a virtuous cycle of learning powered by practice, feedback and community.

1.1.1 Goals

The main objectives of our system are the following:

- **G1: Allow students to participate to collaborative programming challenges**
  This is the main goal of the system. Students will be able to form teams to partecipate to Code Kata Battles and collaborate through GitHub.
- **G2: Allow educators to create programming challenges**
  Educators (and only them) are allowed to create tournaments, which are composed by a non predefined number of programming challenges

called Code Kata Battles. Whenever a new tournament is created, all the students of the platform are notified and can partecipate to it. After the creation of a tournament, the educator can add collaborators that will help him in the management of the tournament. Both the creator of the tournament and the collaborators that are invited to it can create challenges within the context of a tournament.

- **G3: Allow the system to automatically evaluate students' submissions**
  Students submissions are automatically analyzed and ranked by the system, combining functional aspects, timeliness and quality level of sources. Functional aspects are measured in terms of number of unit test cases passed out of all test cases. Timeliness instead is measured in terms of elapsed time passed since the start of the battle. Finally, quality level of sources is measured in terms of code quality, whose aspects can be selected by the educator at battle creation time, and is computed by integrating with static analysis tools.

- **G4: Allow educators to manually evaluate students' submissions**
  Educators will have the possibility to enable manual evaluation of students' submissions for a battle. If the option is enabled, the educator who created the battle, at the end of the submission phase of a battle, can review the code of all the teams and assign a score. In this case, the system will combine the automatically computed score with the manual one to produce the final battle rank.

- **G5: Allow students to track their performance**
  Teams will be able to see their current rank evolving during the battle, updated for each new submission. Additionally, at the end of each battle, the platform updates a personal tournament score of each student, that is the sum of all battle scores received in that tournament. This score is used to compute the ranking of students partecipating to the tournament. In this way, the student can track his performance during the tournament as well.

1.2 Scope

The Code Kata Battle (CKB) platform aims at providing a collaborative environment for students to enhance their software development skills through structured practice sessions called "code kata battles". CKB facilitates an engaging learning experience by enabling students to partecipate in friendly competition while refining their programming skills.

The system should allow 2 types of accesses, one for **educators** and the other for normal **students**: educators will have the possibility to create **tournaments** and **battles**, while students will have the possibility to subscribe to tournaments and provide solutions for the battles that compose them.

Students can decide whether to participate to a battle by creating a new team or joining one, in respect with the constraints decided by the educator at battle creation time. Teams can be created by students in the context of a battle so that other students subscribed to the same tournament can freely decide to join and team up to produce a solution.

The system will facilitate the collaboration between team members by integrating with **GitHub** through its *GitHub Actions*. In this way, students will be able to collaborate on the same project and share their code.

Once the students start developing their solutions, the platform will also start evaluating them. Evaluation is performed in 2 ways:

- Mandatory **automated evaluation**; it includes:
  - functional aspects measured in terms of number of test cases that are correctly solved; unit test cases are provided by the educator when uploading the project files related to the battle
  - timeliness, measured in terms of time passed between the start of the battle and the last commit
  - quality level of the sources, extracted through **external static analysis tools** that consider multiple aspects selected by the educator at battle creation time (e.g. security, reliability, and maintainability)
- Optional **manual evaluation**: personal score assigned by the educator, who checks and evaluates the work done by students

Lastly, once the deadline for the submission of solutions expires (and after the manual evaluation has been performed in case it was expected), the system

assigns to every team that participated in the battle an integer score between 0 and 100. This will concur both to the team's battle rank and to the personal score of each user in the context of the tournament the battle belongs to. At any point in time every user subscribed to CKB can see the list of ongoing tournaments and the rank of enrolled users. For a smoother experience, students will receive email notifications about the most important events, such as the availability of a new tournament and battle, or the publication of the final rank of a battle.

1.2.1 Phenomena

According to the paper "The World and the Machine" by M.Jackson and P.Zave, in order to correctly describe the application domain, the phenomena involved in it must be identified and categorized. The following table describes the world, shared and the machine phenomena, including the reference to which part controls the phenomena.

Phenomenon	Controlled	Shared
Educator decides to challenge his students	W	N
Student wants to improve his coding skills	W	N
Students communicate with each other	W	N
Student invites another student to join his team	W	N
Student forks the GitHub repository of the code kata	W	N
Student setups GitHub Actions workflow on the team's repository	W	N
Educator reviews team's last submission code	W	N
User subscribes to the platform	W	Y
User logs in	W	Y
Educator creates a tournament	W	Y
Educator adds another educator as tournament collaborator	W	Y
Educator creates a battle within a tournament	W	Y
Educator closes the battle's consolidation stage	W	Y
Educator terminates a tournament	W	Y
Educator submits manual optional evaluation	W	Y
Student creates a team	W	Y
Student joins a team	W	Y
User checks list of ongoing tournaments and ranks	W	Y
Student commits a new submission on GitHub	W	Y
Student checks the updated score of a battle	W	Y
System closes tournament's subscription phase	M	Y
System closes battle's team formation phase	M	Y
System closes battle's submission phase	M	Y
System creates GitHub repository of the battle	M	Y
System sends the link of the GitHub repository of the battle to the students	M	Y
System sends notifications to users	M	Y
System computes and updates battle score of a team	M	Y
System updates battle ranking of teams	M	Y
System computes and updates student's tournament personal score	M	Y
System updates tournament ranking of students	M	Y
System evaluates quality level of a submission through static analysis tools	M	Y
System pulls new submission from GitHub	M	Y
System checks login data	M	N
System periodically checks for expired deadlines	M	N
System evaluates functional aspects of a submission	M	N
System evaluates timeliness of a submission	M	N

Table 1: Phenomena table

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **User:** anyone that has registered to the platform
- **Student:** the first kind of users and, basically, the people this product is designed for. Their objective is to submit solutions to battles
- **Team:** students can decide to group up and form a team to partecipate to a battle. The score assigned to the submission of a team will be assigned also to each one of its members
- **Educator:** the second kind of users. They create tournaments, set-up battles, and eventually, evaluate the solutions that teams of students have submitted during the challenge
- **Tournament:** collection of coding exercises (battles). Interested students can subscribe to it and participate to its battles as soon as they are published
- **Code Kata Battle (or battle):** the atomic unit of a tournament. Usually students are asked to implement an algorithm or to develope a simple project that solves the task. Each battle belongs to a specific tournament: students submitting solutions for a battle will obtain a score that will be used to compute both the team's rank for the battle and the members' tournament rank
- **Code Kata:** description and software project necessary for the battle, including test cases and build automation scripts. These are uploaded by the educator at battle creation time
- **Tournament collaborator:** other educator that is added by the tournament creator to help him in the management of the tournament. He can create battles and evaluate their submissions
- **GitHub:** web-based hosting service for version control, mostly used for programming. It offers both distributed version control and source code management functionalities

- **GitHub repository:** a repository is a storage space where some project files are stored. It can be either public or private. In the context of the platform, each battle is associated with a GitHub repository that is created by the system and shared with the teams
- **GitHub collaborator:** person who is granted access to a GitHub repository with write permission
- **GitHub Actions:** GitHub feature that allows to automate tasks directly on GitHub, such as building and testing code, or deploying applications. In the context of the platform, GitHub Actions is used to automatically notify the system when a new submission is pushed to the repository of a team
- **Test cases:** each battle is associated with a set of test cases, which are input-output value pairs that describe the correct behavior of the ideal solution
- **Static analysis:** is the analysis of programs performed without executing them, usually achieved by applying formal methods directly to the source code. In the context of the platform this kind of analysis is used to extract additional information about the level of security, reliability and maintainability of a battle submission
- **Functional analysis:** measures the correctness of a solution in terms of passed test cases
- **Timeliness:** measures the time passed between the start of the battle and the last commit of the submission
- **Score:** to each solution is assigned a score which is computed taking into account timeliness, functional and static analysis and, eventually, manual score assigned by the educator that created the challenge. The score is a natural number between 0 and 100 (the higher the better)
- **Rank:** during a battle, students can visualize the ranking of teams taking part to the battle. Moreover, at the end of each battle, the platform updates the personal tournament score of each student. Specifically, the score is computed as the sum of all the battles scores received in that tournament. This overall score is used to fill out a ranking of

all the students participating to the tournament which is accessible by any time and by any user subscribed to the platform

- **Notification:** it's an email alert that is sent to users to inform them that a certain event occurred such as the creation of a new tournament and battle or the publication of the final rank of a battle

1.3.2 Acronyms

- **RASD:** Requirements Analysis and Specification Document
- **CKB:** Code Kata Battles
- **API:** Application Programming Interface
- **UML:** Unified Modeling Language
- **GDPR:** General Data Protection Regulation
- **HTML:** HyperText Markup Language
- **CSS:** Cascading Style Sheets
- **JSON:** JavaScript Object Notation
- **REST:** REpresentational State Transfer
- **URL:** Uniform Resource Locator
- **HTTPS:** HyperText Transfer Protocol Secure

1.3.3 Abbreviations

- **Gn:** Goal number "n"
- **Dn:** Domain Assumption number "n"
- **Rn:** Requirement number "n"
- **UCn:** Use Case number "n"

1.4 Revision History

- 22 December 2023, version 1.0: first release

1.5 Reference Documents

- Specification document: "Assignment RDD AY 2023-2024"
- RASD reference template: “02g.CreatingRASD”
- Paper: "Jackson and Zave: the world and the machine"
- UML official specification https://www.omg.org/spec/UML
- Alloy official documentation: https://alloytools.org/documentation.html
- GitHub API official documentation: https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api?apiVersion=2022-11-28
- General Data Protection Regulation (GDPR) official text: https://gdpr-info.eu

1.6 Document Structure

- **Section 1: Introduction**
  This section offers a brief overview of the product and its purpose. It also contains the list of definitions, acronyms and abbreviations that could be found in this document.
- **Section 2: Overall Description**
  The second chapter contains some scenarios to better understand the expected functionalities and offers a more detailed description of the product domain through state machine diagrams and UML class diagram. It contains also the hardware and the software constraints of the system. Finally, there is an overview of all the product features and the actors they are built for.
- **Section 3: Specific Requirements**
  This section contains a description of functional requirement through some use cases and activity diagrams.
- **Section 4: Formal Analysis through Alloy**
  The fourth chapter is a formal analysis of the model, made through the Alloy, including a graphic representation of it obtained from Alloy Tool.

- **Section 5: Effort spent**
  The fifth and last chapter contains the time spent by each contributor of this document.

2 Overall Description

2.1 Product perspective

2.1.1 Scenarios

1. **Educator creates a tournament**
   Alan is a university professor. He has just finished explaining a very hard topic that occupied multiple lectures. He would like to test its class about it but exams dates are far away ahead and he thinks that trying to evaluate their understanding with a quick quiz performed in class before or after his next lecture would not be a good idea given the depth of the topic. Anyway he knows of a site that allows educators to easily organize challenges to which students can take part and decides to give it a try.
   He connects to the website of the platform and signs up as an educator. After that, he navigates to the "Tournaments" section and selects the option to create a new one: inputs the name of the tournament and a deadline for the subscription. All students subscribed to the CKB platform are now notified and can join the tournament.
   Since Alan's schedule is often very busy, he then decides to grant the permission to create battles to his teaching assistant Thomas, which has already subscribed to the platform.

2. **Educator creates a battle**
   Linus is a famous youtuber that publishes video courses on programming languages. He wants to increase the engagement with his subscribers, so for this reason he has already subscribed to the CKB platform and created a tournament. In his last video lecture, he has started a new playlist on Java language, explaining some basic concepts typical of object oriented programming.
   To challenge his viewers, he decides to create a new coding battle. He now connects to the website of the platform and log-ins with his credential. He clicks on the tournament called "LinusTechChallenges" he previously created and selects the option to generate a new battle. The platform then asks him to specify some settings for this specific challenge: he selects "Java" as programming language, uploads the "code kata" which includes a textual description of the challenge and all the

files related to the software project like test cases and build automation scripts, selects set the minimum and maximum number of students per group, fixes the registration and final submission deadlines. Since he has many subscribers and cannot manually review every solution, he also specifies that a final manual evaluation is not needed. However, Linus decides to enable all the available aspects the static analysis should focus on (e.g. security, reliability, maintainability). At this point he confirms the creation of the battle and the system automatically notifies all the students subscribed to the tournament about the new coding battle.

3. **Student subscribes to a tournament**
   Mark is struggling to keep the pace of its professor's lectures. Moreover, since his anxiety always penalizes him, he is afraid he will organize some activity in class to evaluate his knowledge about the last, complex topic. One day, while looking at the academic mail, he notices that the feared professor is contacting its students to inform them of an alternative evaluation method that would allow them to avoid part of the final exam and which involves to write code to solve problems related with the subject and compete with other students. Mark immediately decides to go for it and subscribes (as a student) to the platform indicated by the professor. Once logged in, he goes to the "Tournaments" section and selects the option to search for a specific tournament: at this point he searches the tournament by the name communicated by the professor, selects it and subscribes to it. Once a new battle will be available, he will be notified by the system. He is now ready to participate to the battles and prove what he has learned.

4. **Students form a team**
   Liam is a software engineer that has subscribed to CKB because a famous IT company is offering some job positions to the most talented developers discovered through the platform. For this reason, he already joined the tournament created by the company. Since the company is interested in developers with the ability to work in teams, its human resources department has set "2" as minimum number of team members for their first battle. So, Liam decides to tackle the first challenge with a trusted former university colleague of his. He navigates to the "Tournaments" section, searches and selects the ongoing tournament of

the company and clicks on the first battle called "Challenge 1". The system shows 2 options: "Create a team" and "Join a team". Liam selects the first option and chooses a name for his team, checking the "private" flag option. The systems generates an invite code that he can now share privately to his friend in order to join the team.

5. **Student submits a solution**
   Jonas is subscribed to the CKB platform in order to participate in quizzes organized by his computer science professor. He is very happy with this kind of continual evaluation because it's a great way to understand those difficult concepts taught during theory classes and skip some exercises at the final written exam.
   When a new programming challenge is announced, Jonas gets excited to test his skills. He forms a team with two other classmates to collaborate on solving the exercise. After working hard to come up with a new solution, Jonas submits it to the CKB platform by committing the solution on their GitHub repository before the deadline. Jonas and his team mates have monitored their score during the whole battle, but are not satisfied with rank of their final submission computed with the automated evaluation.
   The next day, after the consolidation stage, Jonas is thrilled to see his final team rank near the top of the leaderboard! Indeed the professor liked a lot their innovative solution and decided to reward them during the manual evaluation.

6. **Educator manually evaluates solutions**
   Oliver is an instructor and he is using CKB to evaluate the students enrolled in his online programming course. He enjoys the platform because it organizes the students' GitHub repository in a convenient and central way. Indeed, once the submission phase ends, during the consolidation stage Oliver can select the current battle and go through the whole list of teams. For each team, the system shows the GitHub repository link and the scores related to the automated evaluation. After inspecting the code in the repo, Oliver inputs in the same page a natural number between 0 and 100 as manual evaluation score (the higher the better). Once he has finished reviewing the code of all the teams, he clicks on the "Terminate battle" field next to the current battle tab and all students involved are notified about the final mark.

2.1.2 Domain Class diagram

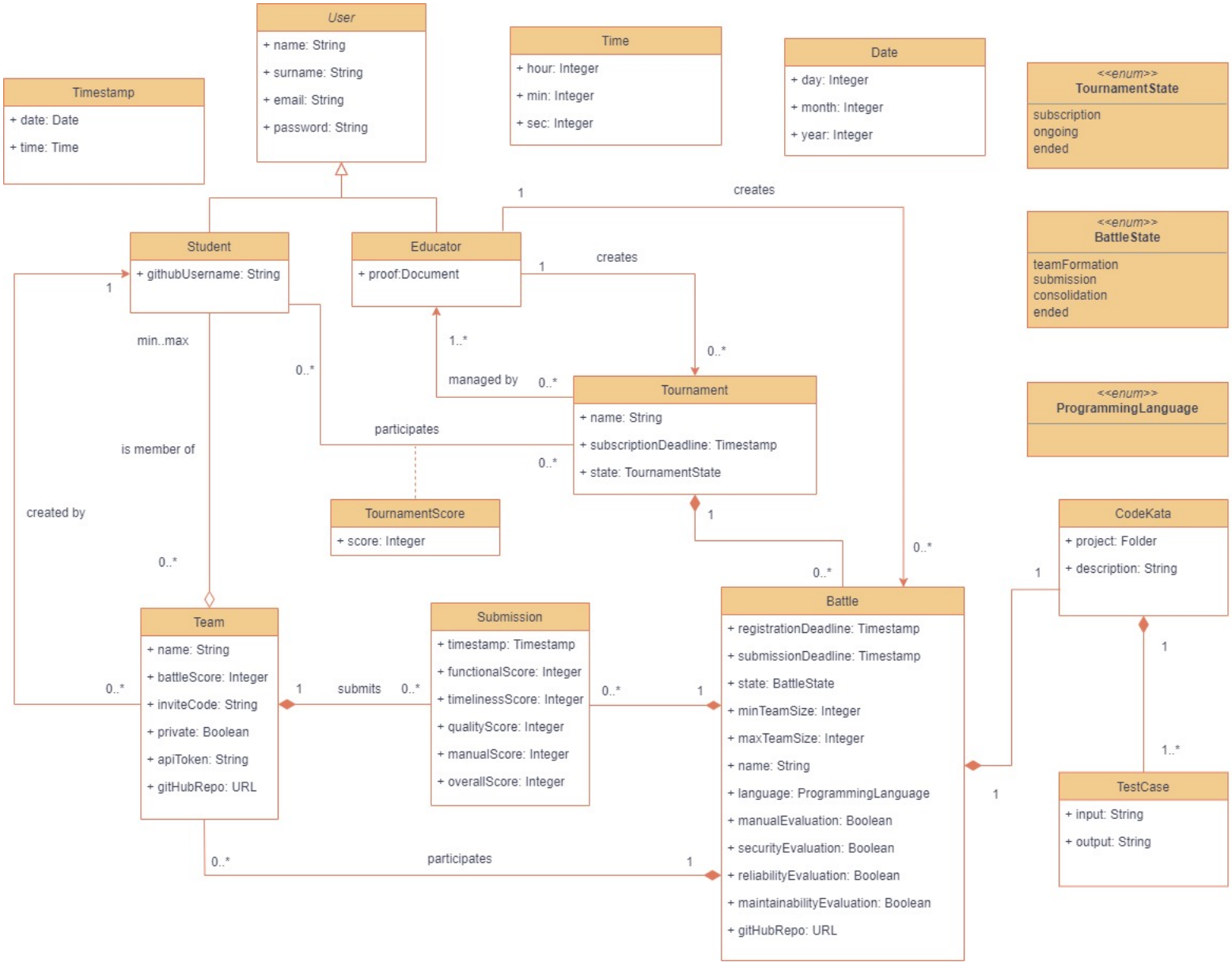

Figure 1: Domain-level UML diagram

Additional notes on the class diagram:

- the "proofDocument" attribute of the "Educator" class contains a reference to the file through which the educator was verified. The educator verification process will not be described in this document, as it is not a functionality that directly involves the end-users of the system.
- the "battleScore" attribute of the Team entity contains the overall score of the latest submission. This is the score that is used to compute the rank of teams participating in the battle.

2.1.3 State diagrams

In order to provide a complete description of the application domain, here are included the state diagrams relative to tournaments and battles.

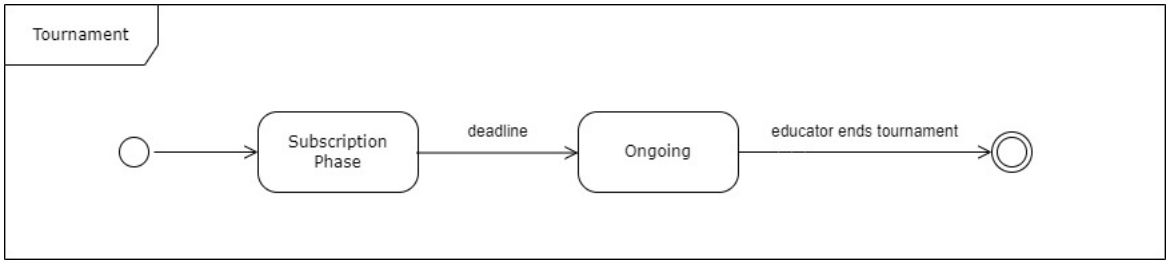

Figure 2: Tournament state diagram

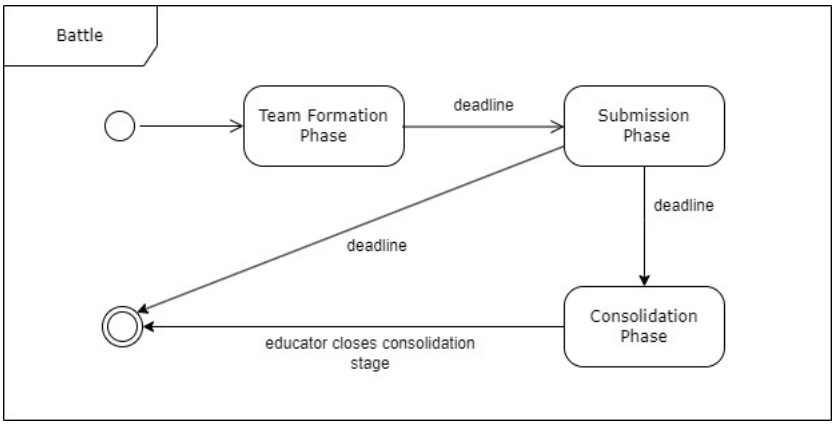

Figure 3: Battle state diagram

In particular, in the case of battles, which transition is taken after the submission stage only depends on whether the manual evaluation was enabled or not. The Team Formation phase will be also referred to as Registration phase in the rest of the document.

2.2 Product functions

2.2.1 Sign up and Log in

These functions are used to respectively subscribe to the platform and access the personal area. This will be available to all kind of users, with a little difference between students and educators. Specifically, during the sign up process, the system will start by asking the user if he/she wants to sign up as a student or as an educator.

In the student's case, the system will ask the new user to enter his/her first and family name, a GitHub username, an email and a password: if the provided email is not already used for another account (of any type) and it really exists a GitHub account associated to the provided username, the system registers the new user. Alternatively, the system allows users to sign up through external identity providers like Google, Facebook and GitHub.

In the educator's case instead, the system will ask the same information but, instead of the GitHub username (which is not needed in the case of an educator), it will ask him/her a form to explain why is he/she subscribing and a document that proves its identity.

At this point, since notifications will be send via mail in any case, in order to use the system the new user has to verify its email: a verification email is sent to the provided email address containing a link that the user will have to click to confirm it.

Obviously in the future, in order to log in, the user will be asked to select the identity provider he/she originally signed up with or to provide the email and password entered during the sign up phase: if these match, the system will grant access.

2.2.2 Manage tournaments

This functionality is available only to educators and basically represents the main functionality that the platform provides.

In order to initiate a new competition, each educator can decide to create a new tournament. To do this the educator log in to its personal area and then goes to the page listing all the tournaments that he's ever created (even the ones that have already terminated: the system persists their final ranks and other useful information, see "Create battles" function) or that he has been granted access to. Here the educator can click a button to create a new tournament: a new page is displayed, in which the user has to provide a new (unique) name for the tournament, fix a subscription deadline and, eventually, grant the possibility to other educators to create battles in the context of the new tournament (this is done by fulfilling a search box with their emails such that the system can send them a notification). This is the only setting that can be changed at any time, even while the tournament is active.

2.2.3 Join a tournament

This functionality is available only to students. In order to subscribe to a new tournament a student can find inside its personal area a page listing all the tournaments he/she has ever participated to. In this page there's also a button that allows the student to join a new tournament: a new page is displayed, in which the user has to provide the name (or part of it) of the tournament of interest. The system searches among all the (active) tournaments those whose name is compatible with what the user's query and returns the list of these.

At this point the student can refine his/her search or select the right tournament from the list and, by doing this, subscribe to it.

2.2.4 Create a battle

This functionality is available only to educators. An educator can select an active tournament from the page showing the list of the tournaments he/she has access to: inside the new page that is displayed educators can find data relative to the tournament (e.g. the state of the tournament and how much time before the next) and the students subscribed to it (e.g. actual ranking and list of subscribed students) and also a button that allows them to create a new code kata battle.

By clicking it, the user is brought to a new webpage in which has to provide multiple information:

- Battle name
- Programming language of use
- Upload Code Kata, including project description, automation scripts and test cases
- Set minimum and maximum number of students per group
- Set registration and final submission deadline
- Specify scoring configuration (see "Evaluate a submission" function)

At this point the battle is created and the registration phase starts.

2.2.5 Join a battle

This functionality is available only to students. A student that is subscribed to a tournament is notified upon the creation of a new battle in the context of it. In the case the user is interested into participating, he/she can subscribe to it by creating a team or joining one that has already been created.

As first thing he/she has to access the page relative to the teams for that battle. This can be done in 2 ways:

- Directly by clicking the link inside the email sent to notify the creation of the battle.
- Go to the page displaying his/her list of tournaments and select the one of the new battle: inside a new page there will be shown the actual rank and the list of battles up to the last, the active one, which can be selected. If a user selects a battle it has already subscribed to, he/she is brought to the page devoted to its team.

Inside a new page it will be displayed the list of available teams and 2 buttons.

The first button allows the student to create a new group: upon click a new page will be shown in which the student has to fill a form to enter all the necessary information. Specifically, the student has to:

- Provide a (unique) name for the team
- Set the visibility of the group: the creator can toggle a checkbox to specify that the team will be private

At this point the system creates 2 random strings that will be univocally associated with the newly created team:

- an invite code: this will represent the only way to access private teams and, in general, an alternative way to join teams.
- an API token: this will be requested to be included inside the GitHub Actions workflow in order to allow the system to understand which team a solution belongs to.

This functionality is particularly important because students that want to participate by their own, need to create a group (in order to be sure, they can set it as private and simply keep the invitation code for theirselves): anyway, this scenario is available only when the minimum number of students per group is set to 1.

The second button instead allows the student to join an already existing group: this is achieved by allowing the student to search groups by name (in case of private teams, in a following page the student will be asked to provide the right invite code) or by directly providing their invite code: if the specified group does not already contain the maximum number of students for the battle, the system adds the student to the group.

Once the registration deadline is met, the system will prepare the development environment: specifically, it will create a new public repository on the platform's GitHub account and send the relative link to the members of every team subscribed to the battle. Then, the system will need *only* one member per team to fork the repository, copy the relative link on the team's settings, set up GitHub Actions and invite his/her teammates (if there are) as collaborators.

2.2.6 Submit a solution

Once the registration phase is over, the submission phase starts: at this point each team should have its own GitHub repository in which the code for their solutions has to be collected. GitHub Actions will enable the necessary workflow: the submission of a new solution is performed by committing and pushing over the team's repository. On a new push, an API call will notify the platform of the publication of a new solution for that team and proceed to evaluate it.

2.2.7 Evaluate a submission

The platform provides an automatic evaluation system that is used to score the solutions submitted by the students. Scoring is divided in 2 groups:

- **Automatic evaluation**: this comprises all the aspects that are mandatory to be assessed. These are:
  - Correctness, measured in terms of number of test cases solved correctly out of all.
  - Timeliness, measured in term of time passed between the registration deadline and the last commit.
  - Quality level: the system uses external tools to evaluate the code with respect to 3 possible aspects that are security, reliability and maintainability (educator decides which to include at battle creation time).

  These aspects are quantified and then combined in a way to return an integer score between 0 and 100.

- **Manual evaluation**: this is an optional evaluation phase that educators can decide to reserve time for (must be indicated at battle creation time). Specifically, in the case the educator has specified it, as soon as the mandatory evaluation has been computed the system allows him/her to manually inspect the code provided by each team: by clicking on the name of the team inside the final rank the system displays a new page in which is shown the link to the team's GitHub repository and a form that must be filled with an integer score between 0 and 100.
  The final mark will be computed as the arithmetic average between the automatically assessed score and the one assigned by the educator.

2.2.8 Visualize ranking

At any time, any user of any kind can visualize the ranking of the tournaments/battles it was involved in by reaching the dedicated page inside their personal area and selecting the tournament/battle of interest. Battle ranks are computed considering the latest score for each team, while tournament ranks are computed considering, for each student, the sum of the scores assigned to its teams in the context of that tournament.

2.2.9 Send notification

The system sends notifications (by means of emails) to its users in the following cases:

- on tournament creation, all the students subscribed to the platform are notified
- on battle creation, all the students subscribed to the tournament the new battle will belong to are notified
- as soon as the final battle score is available, all the students belonging to the same team are notified
- on tournament closure, all subscribed students are notified

2.3 User characteristics

2.3.1 User types

As previously mentioned, the system is meant to be used by 2 kind of users:

- **Educators**: this is the type of users that uses the platform to create competitions. They can create tournaments and battles and, eventually, evaluate the solutions provided by students. They must have a good knowledge of how to create valid automation scripts and have a basic understanding of how to use a web browser.
- **Students**: these are the users that will use the platform to be evaluated or simply to take part in a competition. They can subscribe to tournaments and battles, join groups of students and submit solutions. They are required to have a good knowledge of how to properly setup a GitHub repository and of all the programming languages involved in the battles.

2.3.2 Educator requirements

In order to allow for automatic recognition of all the different kinds of files that the educator has to upload, the system requires him/her to upload a zip file containing all the needed files formatted in a specific way. In this section

the focus is on its content rather than the way it must be structured, which instead will be described in the Design Document.

In particular, the zip file must contain:

- a folder that contains all the files needed to build the project (e.g. source code, build automation scripts, etc.). This will basically be what the GitHub repository of the battle will contain, so it must be organized in the exact way the educator wants it to appear on GitHub.
- a file which contains all the test cases, expressed by means of (input, expected output) couples.

This separation also allows the system to understand which files must be uploaded on the GitHub repository and which not (i.e. test cases, since they must not be visible to students).

2.4 Assumptions, dependencies and constraints

2.4.1 Domain assumptions

- **D1:** Users have access to a stable and reliable internet connection to interact with the CKB platform.
- **D2:** Supported programming languages are limited to popular options like Java, Python and C++.
- **D3:** Registered educators are all legitimate and verified.
- **D4:** Educators upload correct test cases and well-structured Code Kata projects.
- **D5:** Students are expected to engage in Code Kata Battles with integrity, without resorting to plagiarism or cheating (e.g. inviting more or different collaborators with respect to the ones inside the team).
- **D6:** Maximum number of students per group an educator can set will always be bounded (e.g. less than 4).
- **D7:** Every student has a GitHub account.
- **D8:** Students will fork the Code Kata GitHub repository once ready.

- **D9:** Students know how to setup a GitHub Actions worfkflow.
- **D10:** Only one student per group will perform the steps needed to set-up the team's repository and the GitHub Actions workflow.
- **D11:** Static analysis tools are able to quantify the specified code quality aspects.
- **D12:** Students use external communication channels.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The platform will provide a web interface accessible via browser from both students and educators. In this section, mockups of the most important functionalities of the website will be presented. A complete description of the webapp will be provided in the Design Document.

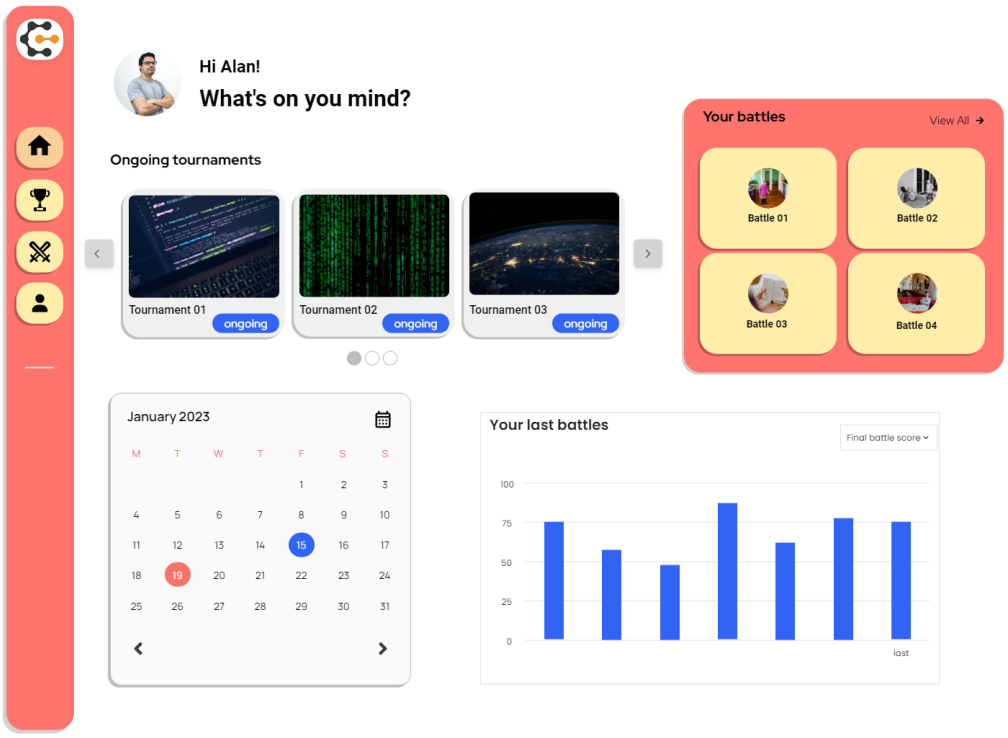

Figure 4: Home page from the perspective of an user logged as student

The picture above shows the home page of the CKB website. It contains different information depending on the role of the user who is visiting it: Figure 4 in particular presents the home page as it appears to a student. It displays the list of all tournaments and battles the student is participating in and a calendar showing upcoming deadlines.

The educator's version of the page shows instead the list of all tournaments and battles he is managing.

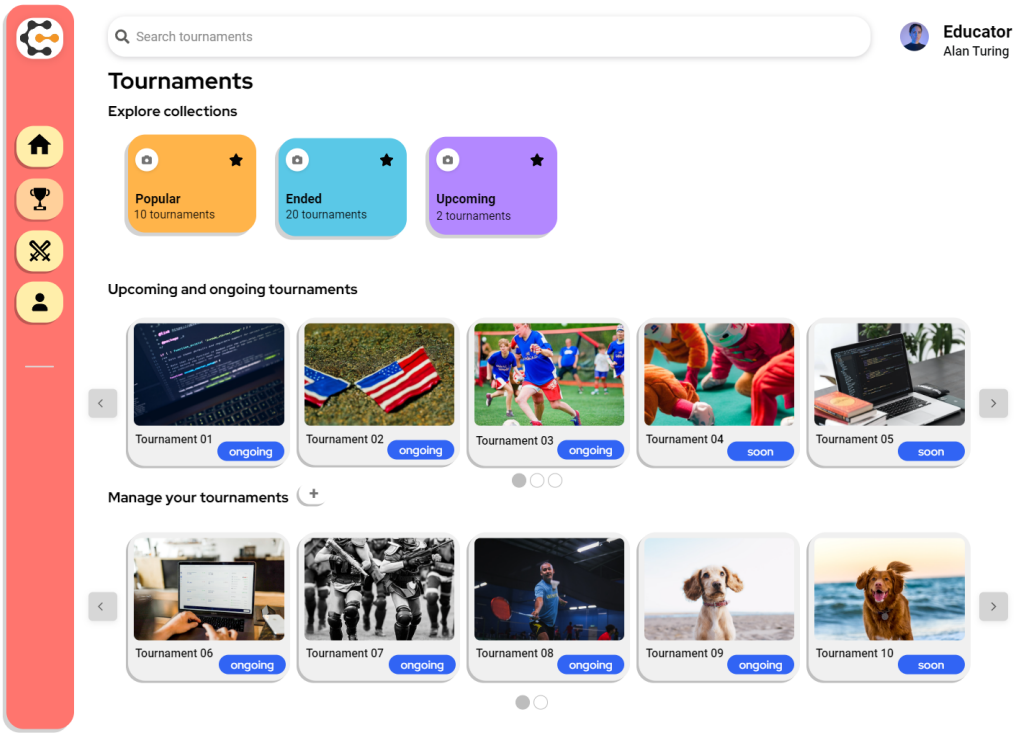

Figure 5: Tournaments page from the perspective of an user logged as educator

The tournaments section contains the list of all past, ongoing and upcoming tournaments. User can click on a specific tournament to navigate to the tournament's specific page.

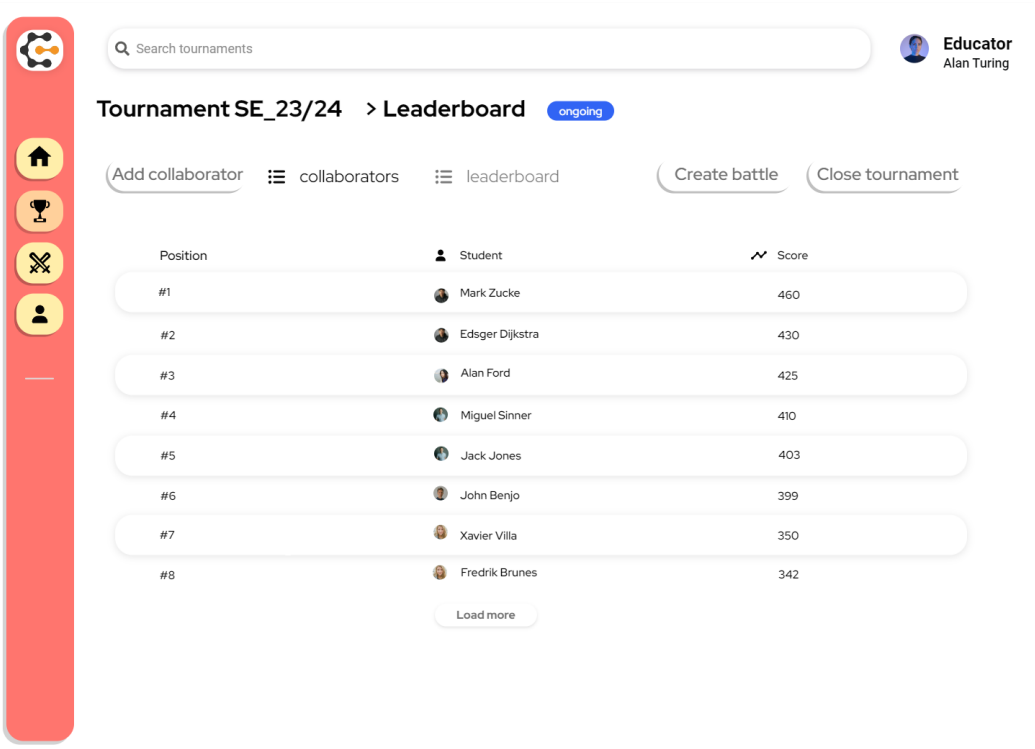

Figure 6: "Leaderboard" tab available to both educators and students to check the ranking in tournament

The tournament page allows the users to visualize the personal rank of students participating in it. Figure 6 shows in particular how the leaderboard will be displayed to an educator.

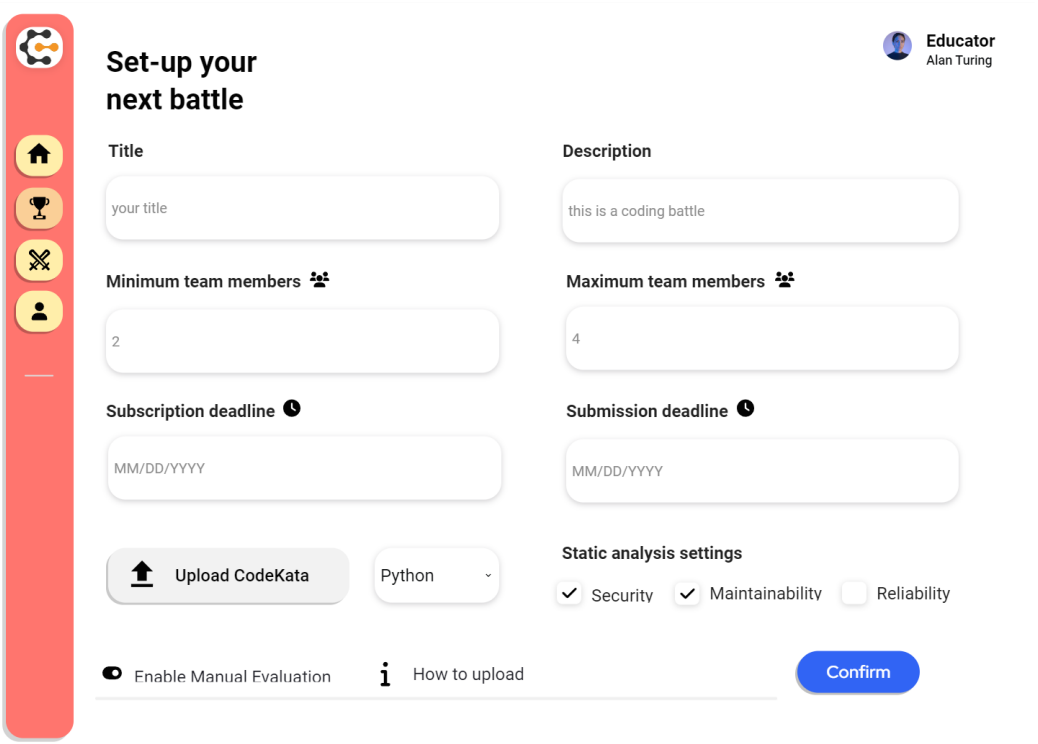

Figure 7: Page used from educators to create a new battle

The create battle functionality is available only to educators and allows them to insert all the relevant information about the challenge and to upload the Code Kata files.

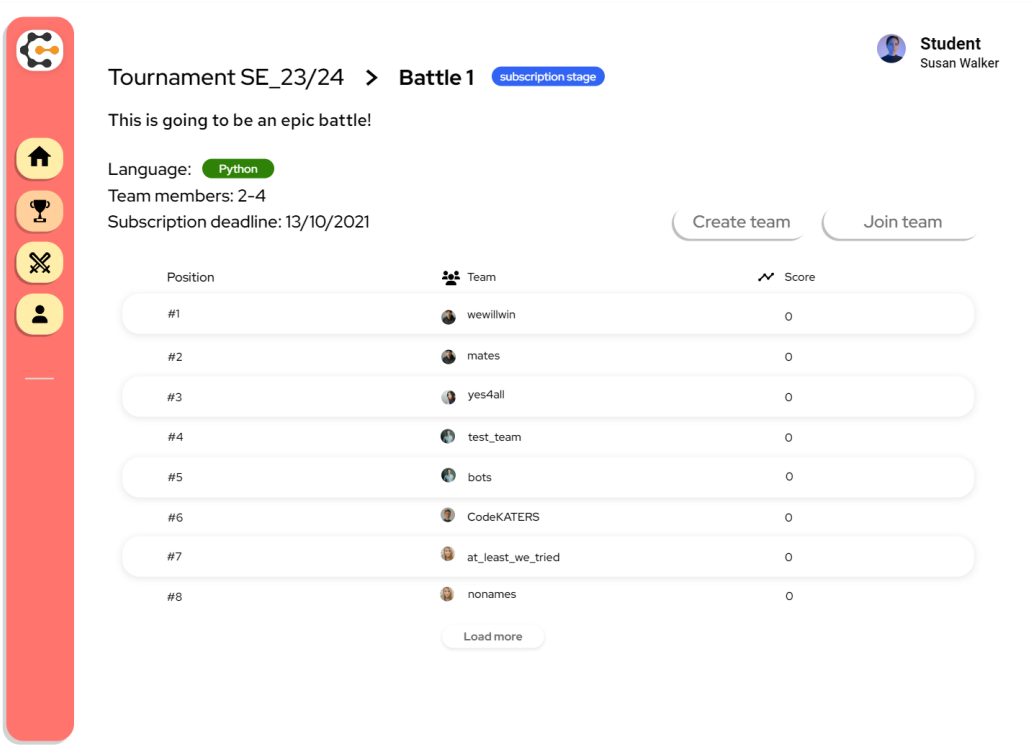

Figure 8: Page used from students to subscribe to a battle

Another key service of the platform is provided by the battle page. It allows educators to access the battle management functionality. Students, through this page, can instead subscribe to the battle. This section is also used to visualize the battle's leaderboard to all the users.

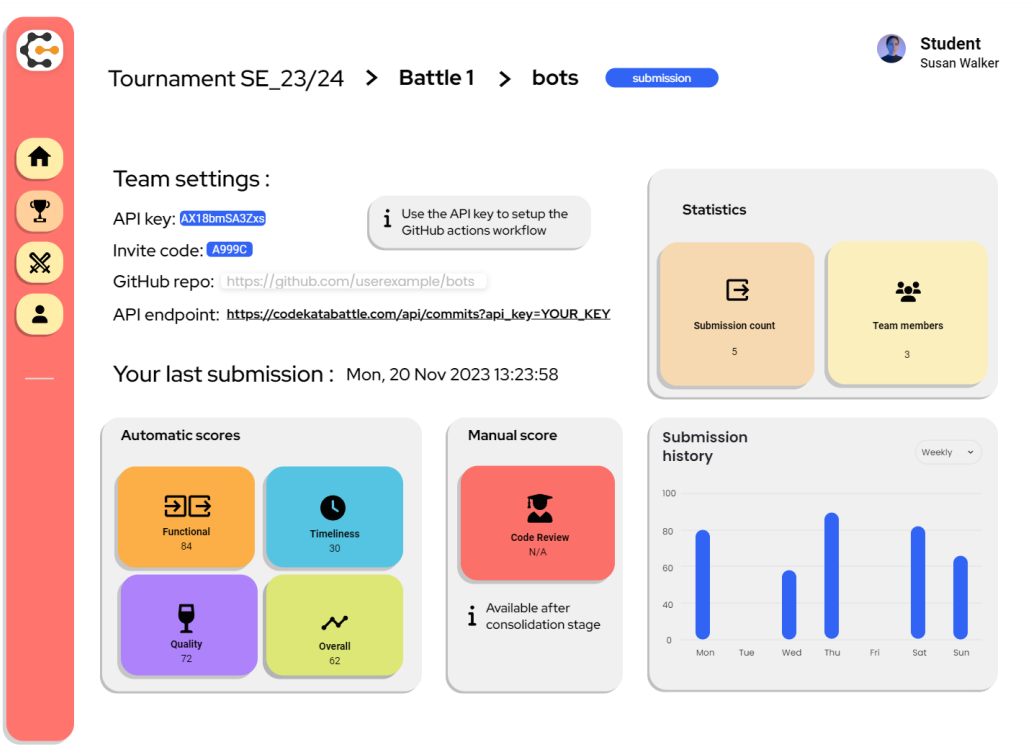

Figure 9: Page used from students to manage the settings of their team and to check submissions information

The team page is used by students to check and configure the settings of their team (e.g. GitHub repository related information). It also allows them to visualize details about their last submission.

Educators will also access the same page to review the code and assign a manual score to the team.

3.1.2 Hardware Interfaces

The software does not require any other hardware interface different than personal computers or smartphones equipped with modern browser from which users can access the platform.

3.1.3 Software Interfaces

Some of the functionalities provided by the system exploit specific interfaces to work. These are:

- **GitHub interface**: the system must be able to interface with GitHub API to create the original repository for each battle and to pull code submissions.
- **Static analysis tools**: some of the aspects that have to be evaluated require the execution of static analysis methods over the provided code. System achieves this by interfacing with external tools that provide this functionality. These services must support the set of languages allowed by the CKB platform and the analysis of specific code quality aspects such as security, maintainability and reliability.

Moreover, in order to be notified about the submission of a new solution, the system itself must expose a **REST endpoint** that will be invoked by the automated GitHub Actions workflow set up by each team on their repository.

3.1.4 Communication Interfaces

Given that the majority of the system's services are provided through the platform's website and REST API calls, no other protocol than HTTP should be needed. Anyway, it is worth mentioning that part of the communication between the system and user is achieved through notifications sent via mail using an external mail service provider.

3.2 Functional Requirements

The main requirements the system has to fulfill are the following:

- **R1:** System shall allow user to register as educator or student.
- **R2:** System shall allow user to log in.
- **R3:** System shall allow educator to create a tournament.
- **R4:** System shall allow educator (tournament creator) to set the name of the tournament.
- **R5:** System shall allow educator (tournament creator) to set the subscription deadline of the tournament.
- **R6:** System shall notify all students enrolled to the platform about a newly created tournament.
- **R7:** System shall allow educator (tournament creator) to end the tournament.
- **R8:** System shall allow educator (tournament creator) to add another educator as collaborator to the tournament.
- **R9:** System shall allow educator (tournament creator or collaborator) to create a battle for the tournament.
- **R10:** System shall allow educator (battle creator) to upload the Code Kata of the battle.
- **R11:** System shall allow educator (battle creator) to set minimum and maximum number of students per group allowed for the battle.
- **R12:** System shall allow educator (battle creator) to set the registration deadline of the battle.
- **R13:** System shall allow educator (battle creator) to set the submission deadline of the battle.
- **R14:** System shall allow educator (battle creator) to enable/disable optional manual evaluation for the battle.

- **R15:** System shall allow educator (battle creator) to select relevant aspects of code quality to be extracted through static analysis.
- **R16:** System shall notify all users enrolled to a tournament about a newly created battle.
- **R17:** The system shall be able to perform static analysis on submitted code utilizing third-party static code analysis tools and services.
- **R18:** System shall create the GitHub repository of a battle as soon as the registration phase closes, sending its link to all involved students.
- **R19:** System shall assign to each submission sent before the submission deadline a score (natural number between 0 and 100) combining correctness, timeliness and code quality *as soon as possible*.
- **R20:** System shall allow educator (battle creator) to set a manual score (natural number between 0 and 100) to the last valid team's submission during the consolidation stage of the battle.
- **R21:** System shall compute a final score combining automatic and manual score (if available) for each team if manual evaluation is enabled.
- **R22:** System shall update the battle teams' ranks right after a new score is available.
- **R23:** System shall notify all students participating to the battle about the availability of the final battle rank.
- **R24:** System shall update the tournament students' scores as the sum of all the battle scores received during the tournament once the final battle rank becomes available.
- **R25:** System shall notify all involved users about the availability of the tournament student's final rank.
- **R26:** System shall allow all users to see the list of ongoing tournaments.
- **R27:** System shall allow all users to see all tournaments' ranks.
- **R28:** System shall allow all users to see all battles' ranks.

- **R29:** System should manage every ranking (tournament or battle) in a way that it represents the correct order of students/teams within its context, from the ones with the highest score to the ones with the lowest.
- **R30:** System shall allow student to enroll to a tournament within the subscription deadline.
- **R31:** System shall allow student to create a team within the registration deadline.
- **R32:** System shall allow student (team creator) to set the team name.
- **R33:** System shall allow student (team creator) to set the privacy of the group.
- **R34:** System shall allow student to join a team *only* before the registration deadline.
- **R35:** System shall allow *only* students who know the correct invite code to join private groups.
- **R36:** System shall allow team members to set their the repository URL.
- **R37:** System shall generate a unique API token for each team.
- **R38:** System should offer an external API which, when invoked, will notify the platform about a new commit on a team's GitHub repository.
- **R39:** System shall allow the educator to access the code of the submitted solutions.
- **R40:** System shall generate a unique invite code for each team.
- **R41:** System shall allow educator (battle creator) to close the consolidation stage of the battle if manual evaluation is enabled.
- **R42:** System shall allow students to see the latest score of their team.

3.2.1 Use cases

Shared perspective

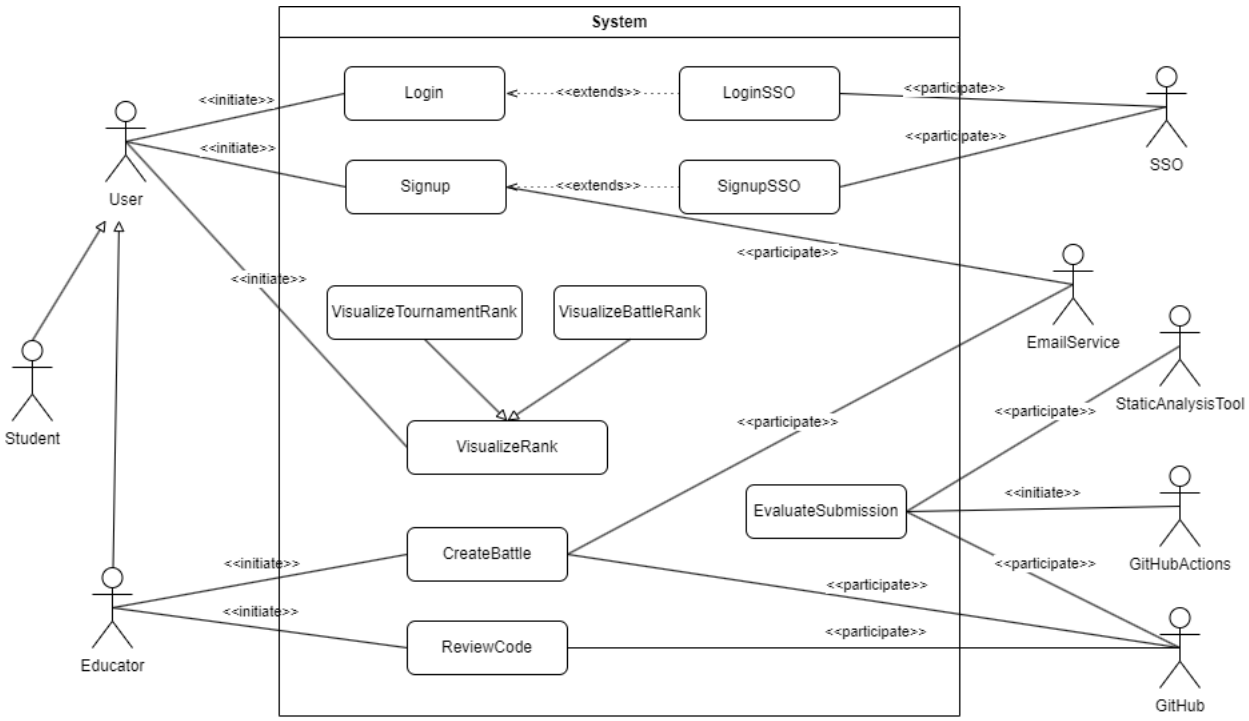

Figure 10: Shared scenarios use case diagram

ID	UC1
Name	Signup
Actors	User, Email Service
Entry conditions	• User is connected to the platform's website • User selected the option to register to the service
Event flow	1. User inserts the required information 2. User flags the "Sign up as student" checkbox if he/she wants to sign up as a student, otherwise leaves it unchecked. In the case of a student, is also asked to provide its GitHub username 3. User clicks on the "Sign up" button 4. System checks user data 5. System sends a confirmation email to the user 6. User confirms the email address by clicking on the link inside the verfication email
Exit conditions	• System adds the new user to its internal database • User is logged in and redirected to the Home page
Exceptions	• The email provided by the user is already in use for another account of the same type (student or educator) • Some text field is left empty • In the case of a student, does not exist any GitHub account corresponding to the provided username In all previous cases the registration is rejected and an error message is displayed on the page

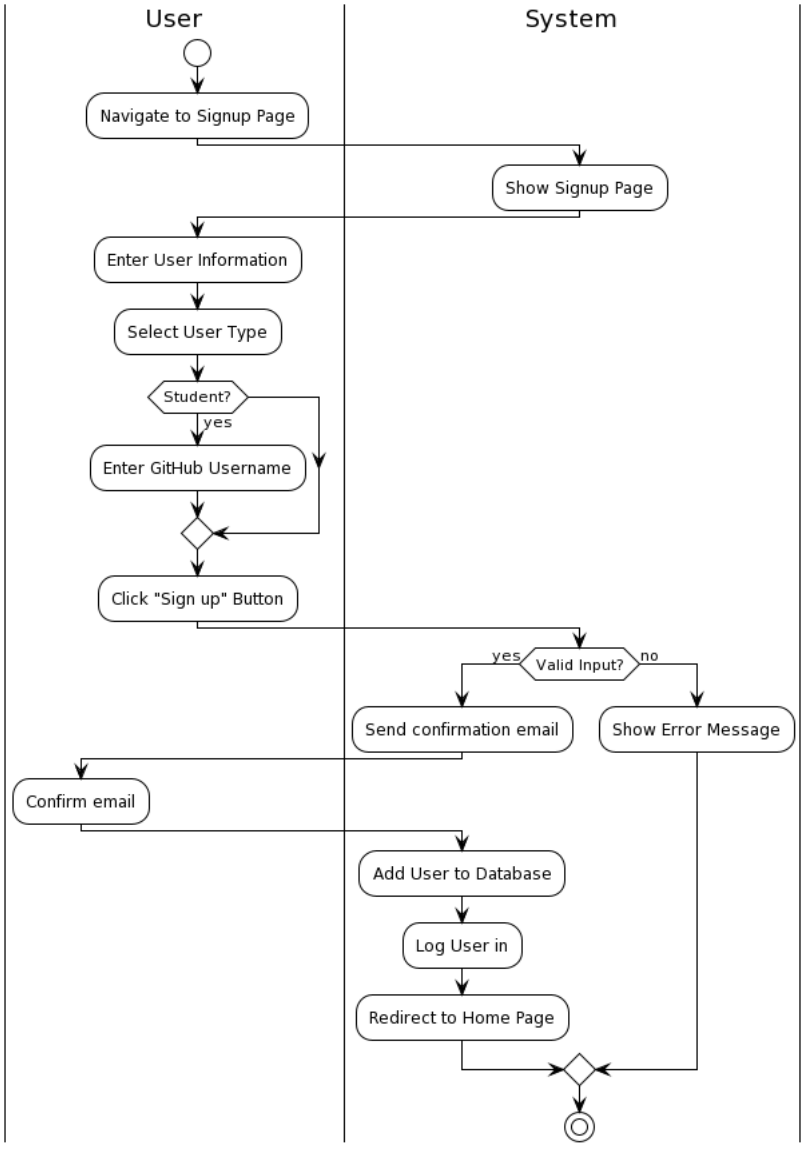

Figure 11: Sign up use case activity diagram

ID	UC2
Name	SignupSSO
Actors	User, Identity provider
Entry conditions	• User is connected to the platform's website • User selected the option to register to the service
Event flow	1. User clicks on the icon of the identity provider he/she wants to sign up through 2. SSO provides to the system the information it needs for the user's registration 3. Inside a new page, the user needs to click one of two buttons to specify whether he/she wants to sign up as a student or as an educator. In the case of a student, is also asked to provide its GitHub username 4. System checks user data
Exit conditions	• System adds the new user to its internal database • User is logged in and redirected to the Home page
Exceptions	• The email provided by the user is already in use for another account of the same type (student or educator) • In the case of a student, does not exist any GitHub account corresponding to the provided username In all previous cases the registration is rejected and an error message is displayed on the page

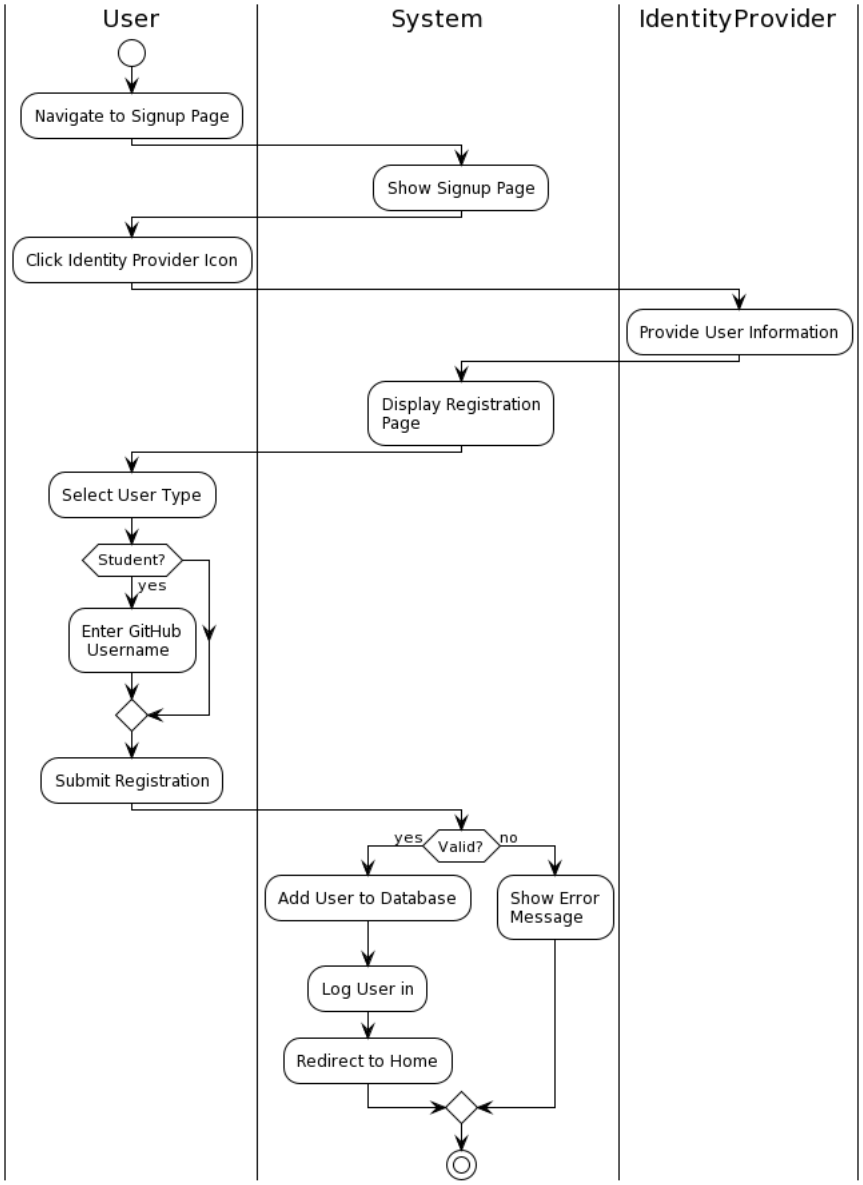

Figure 12: Sign up through SSO use case activity diagram

ID	UC3
Name	Login
Actors	User
Entry conditions	• User is connected to the platform's website • User selected the option to log in • User has already registered to the service
Event flow	1. User fills username and password fields 2. User clicks on the "Log in" button 3. System checks the validity of the provided credentials
Exit conditions	• User is logged in and redirected to the Home page
Exceptions	• The provided credentials do not correspond to any of the accounts registered on the platform • Some text field is left empty In all previous cases the log in request is rejected and an error message is displayed on the page

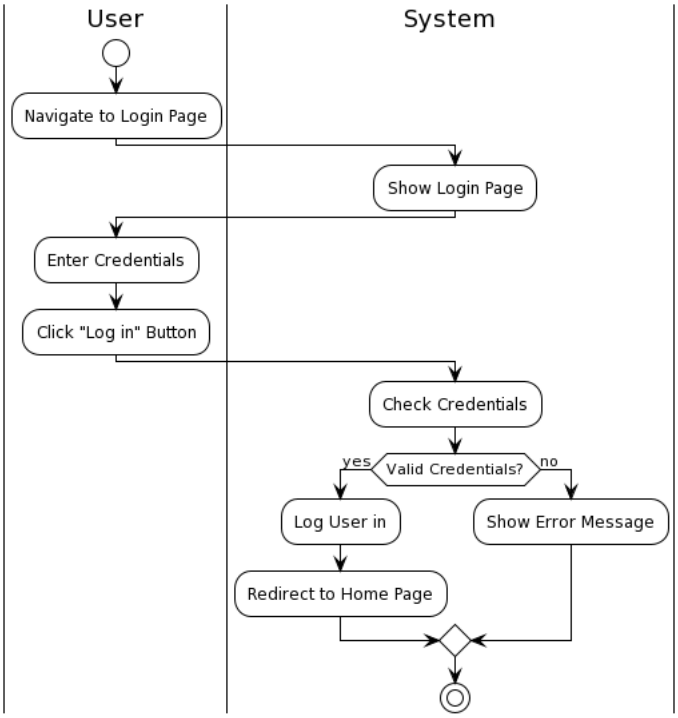

Figure 13: Log in use case activity diagram

ID	UC4
Name	LoginSSO
Actors	User, Identity provider
Entry conditions	• User is connected to the platform's website • User selected the option to log in • User has already registered to the service
Event flow	1. User clicks on the icon of the Identity provider he/she signed up with 2. SSO provides to the system the information it needs for the user's authentication 3. System checks user validity
Exit conditions	• User is logged in and redirected to the Home page
Exceptions	• User selects a different identity provider with respect to the one he/she initially used to register to the platform In the previous case the log in request is rejected and an error message is displayed on the page

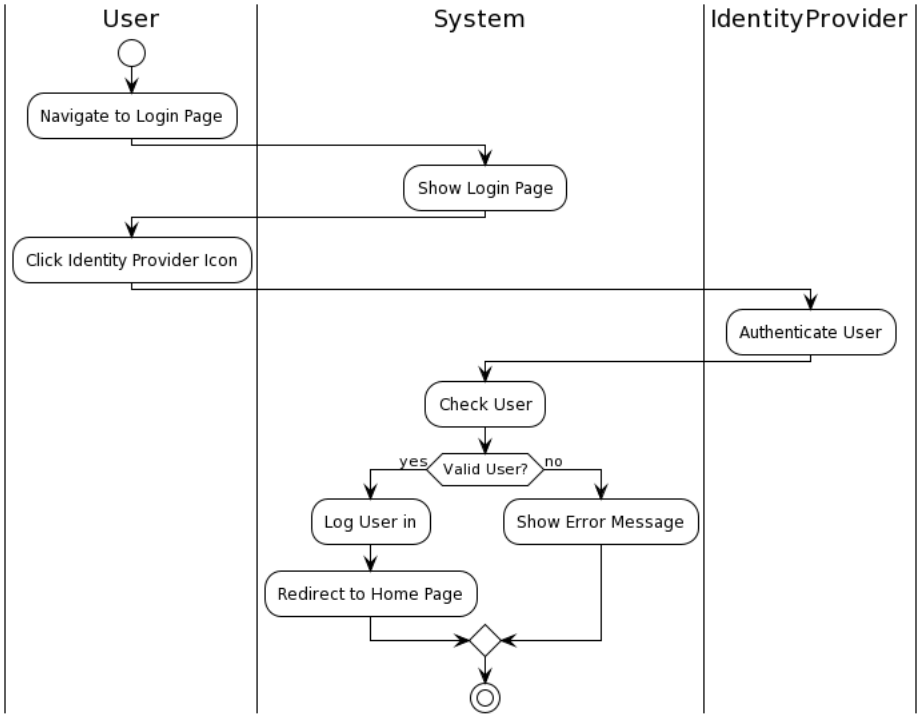

Figure 14: Log in through SSO use case activity diagram

ID	UC5
Name	CreateBattle
Actors	Educator, GitHub, Email Service
Entry conditions	• Educator is logged in • Educator has created or is a collaborator of the tournament in the context of which he/she wants to create the new battle in • Educator has already prepared the Code Kata zip file
Event flow	1. Educator navigates to the tournaments page 2. Educator selects the tournament from his/her "Manage your tournaments" list 3. Educator clicks the "Create battle" button 4. Educator inserts all the needed information and uploads the Code Kata 5. Educator clicks the "Confirm" button 6. Educator is redirected to the tournament's page 7. System checks the validity of the provided information and project files 8. System notifies all the students enrolled into the tournament about the new battle via the Email Service

Exit conditions	• System adds the new battle to the tournament's battles list and students can subscribe to it until the registration deadline • System creates the repository for the challenge via GitHub API when the subscription deadline is met. System sends an email to all the students subscribed to the battle with the URL of the repository to fork • Emails to students are correctly dispatched
Exceptions	• The provided title has already been used for another battle of the same tournament • Some text field is left empty • The uploaded Code Kata is not correctly structured In all previous cases the battle creation request is refused and an error message is shown in the page

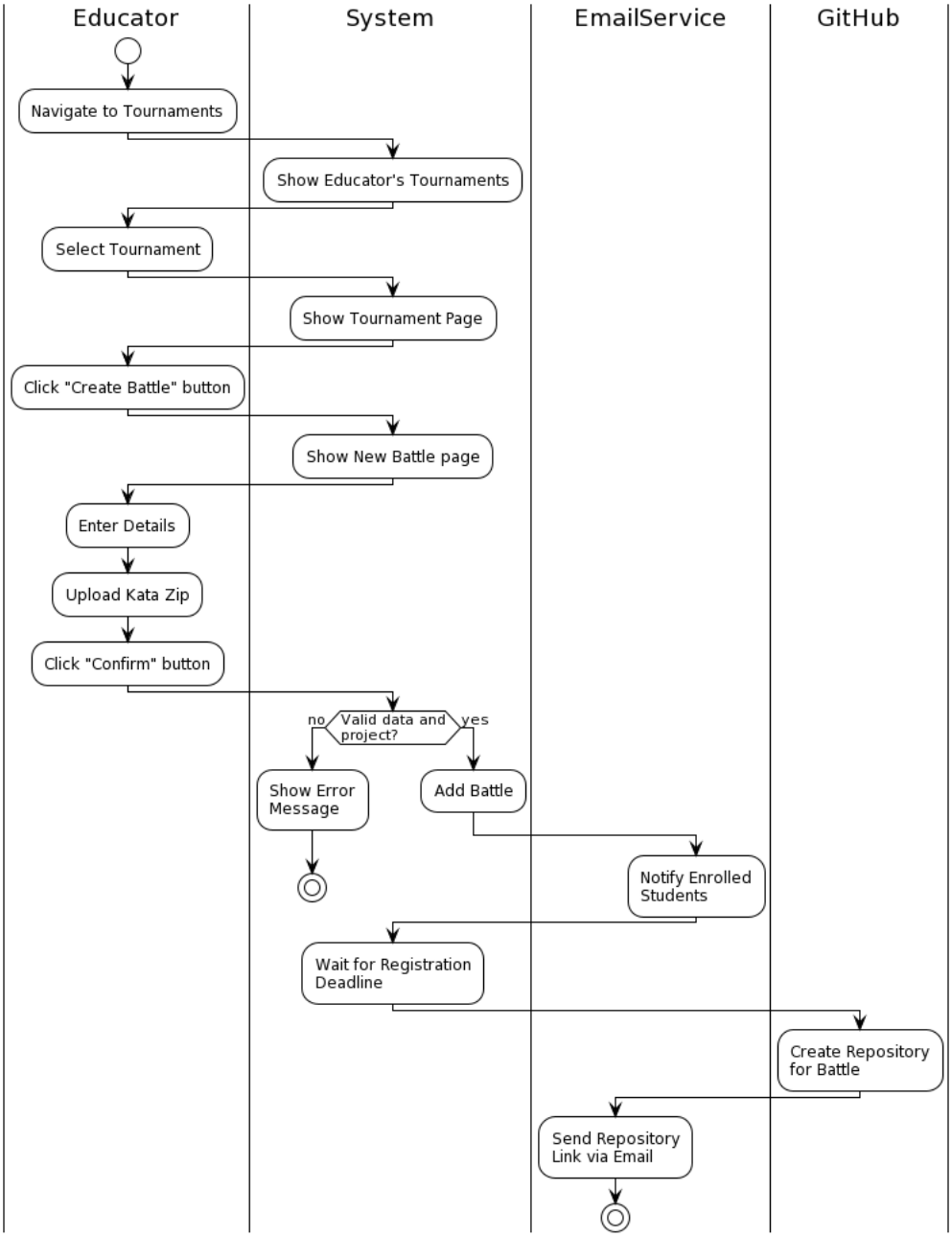

Figure 15: Battle creation use case activity diagram

ID	UC6
Name	VisualizeTournamentRank
Actors	User
Entry conditions	• User is logged in
Event flow	1. User navigates to the tournaments page 2. User inputs the name of the tournament of interest in the dedicated searchbox 3. User selects the right tournament 4. User clicks the "Leaderboard" button
Exit conditions	• System shows the page containing the tournament's ranking
Alternative	1a. User directly selects the tournament from the ones shown inside his/her tournaments page or home page
Exceptions	• Tournaments state is not "Ongoing" or "Ended" In the previous case the "Leaderboard" button will not be clickable

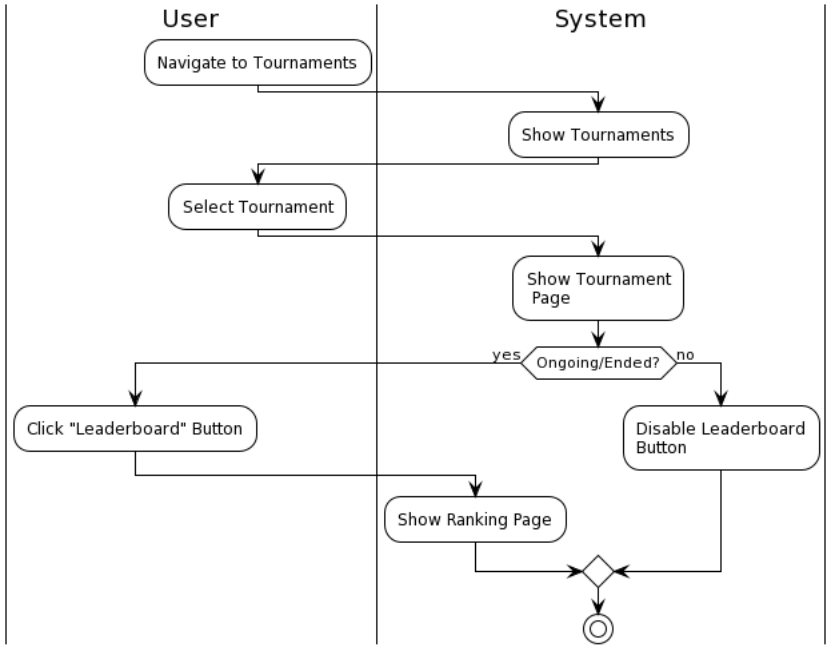

Figure 16: Tournament ranking's visualization use case activity diagram

ID	UC7
Name	VisualizeBattleRank
Actors	User
Entry conditions	• User is logged in
Event flow	1. User navigates to the tournaments page 2. User inputs the name of the tournament of interest in the dedicated searchbox 3. User selects the right tournament 4. User selects the battle of interest from the tournament's battle list
Exit conditions	• System shows the page containing the battle's ranking
Alternative	1a. User directly selects the battle from the ones shown in the context of his/her home page
Exceptions	• Battle is in Registration Phase In the previous case the battle's ranking will simply show the list of teams that subscribed till now

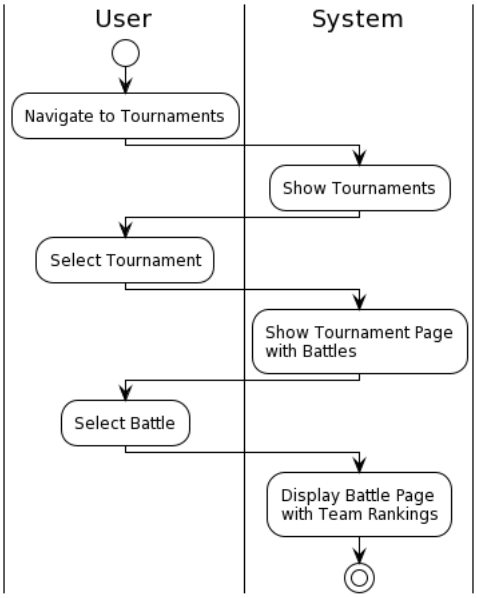

Figure 17: Battle ranking's visualization use case activity diagram

ID	UC8
Name	EvaluateSubmission
Actors	GitHub Actions, GitHub, Static Analysis tool
Entry conditions	• Team has forked the battle's repository (once per team) • GitHub Actions workflow is setup (once per team) • Team's repository is setup in the team's settings page (once per team) • A new commit is pushed into team's GitHub repository
Event flow	1. GitHub Actions performs a request directed to the system's API endpoint, including the relative API token of the team as an additional parameter of the request 2. System checks the validity of the token and retrieves the corresponding repository URL 3. System pulls the code of the new submission directly from GitHub via API 4. System computes a score for timeliness by comparing the timestamp of the request with the one corresponding to the start of the submission phase of the battle 5. System computes a score for correctness by executing all the test cases provided and comparing the results with the expected ones 6. If specified from the battle creator, the system provides the code for the evaluation to the external static analysis tool which will assign a score for each of the code quality aspects selected at battle creation time

Exit conditions	• System computes an overall score by averaging the results of all the previous evaluations and updates the team's score and rank
Exceptions	• GitHub Actions workflow is not correctly setup, the system's endpoint receives a request with a token that does not correspond to any of the teams subscribed to the battle • Repository's url is not correctly setup in the team's settings page, system is not able to retrieve the code In all previous cases the new submission is simply ignored by the system

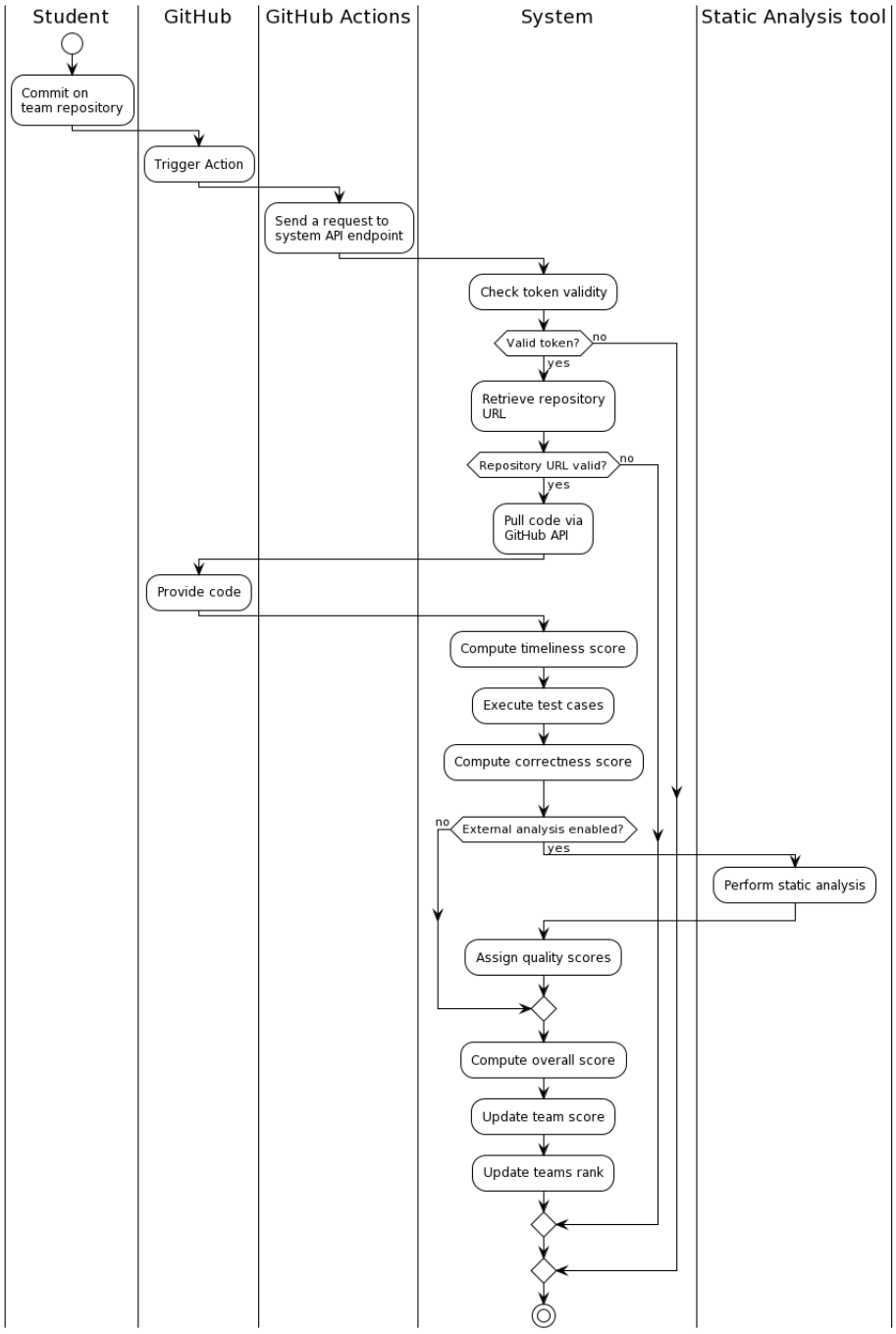

Figure 18: Submission evaluation use case activity diagram

ID	UC9
Name	ReviewCode
Actors	Educator, GitHub
Entry conditions	• Educator is logged in • Battle is in Consolidation Stage
Event flow	1. Educator navigates to the tournament section 2. Educator selects the right tournament 3. Educator selects the battle of interest 4. Educator selects a team from the battle's ranking 5. Educator clicks the link to the team's GitHub repository to inspect the code 6. Educator provides a score by inputting it inside the dedicated textbox 7. Educator clicks the "Confirm" button
Exit conditions	• System saves the new manual score in its database and computes the final battle score as the average between the new score and the automatic evaluation results
Alternative	1a. Educator directly selects the battle from the ones shown in the context of his/her home page
Exceptions	• The score text field is left empty or contains something that is not an integer number between 0 and 100 In this case the score registration request is rejected and an error message is shown

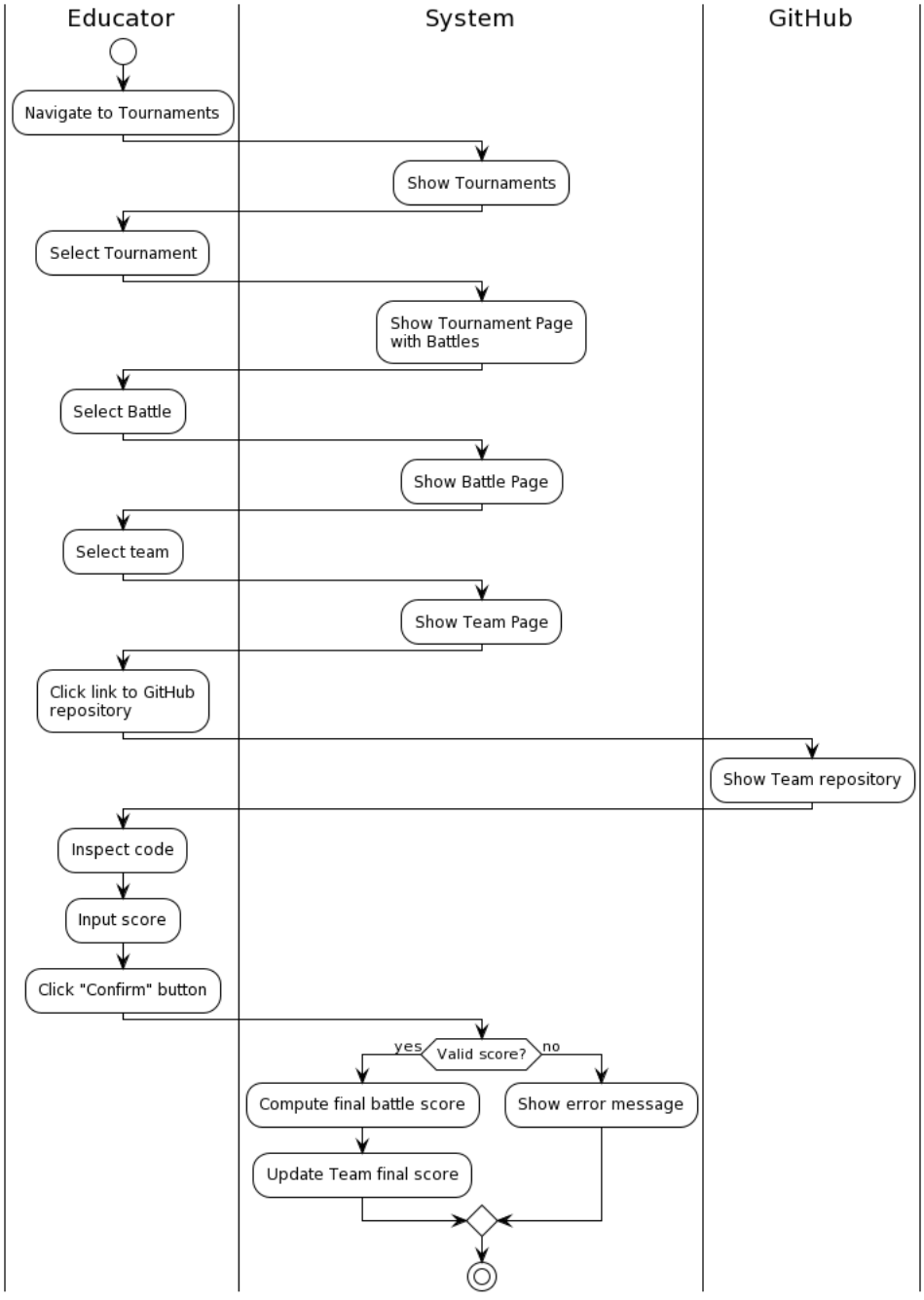

Figure 19: Code review use case activity diagram

Student perspective

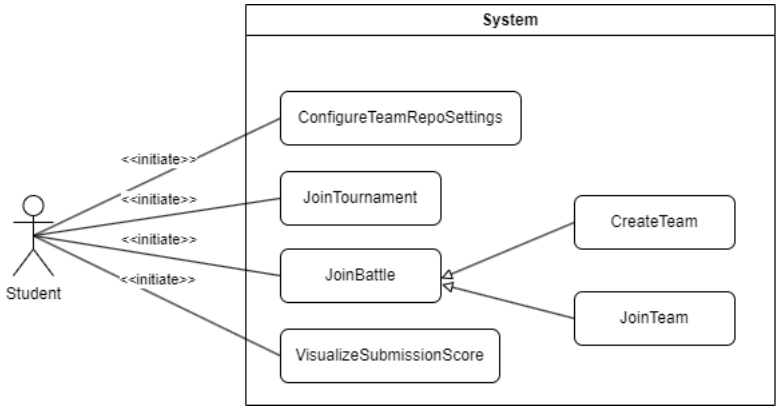

Figure 20: Students' scenarios use case diagram

ID	UC10
Name	JoinTournament
Actors	Student
Entry conditions	• Student is logged in
Event flow	1. Student navigates to the tournaments section 2. System shows the list of tournaments the student is enrolled in and the list of all available tournaments in the platform 3. Student clicks on one of the tournaments in the list of available tournaments 4. Student is redirected to the page of the tournament containing the list of all students currently enrolled 5. Student clicks on "Enroll"
Exit conditions	• Student is correctly enrolled into the tournament

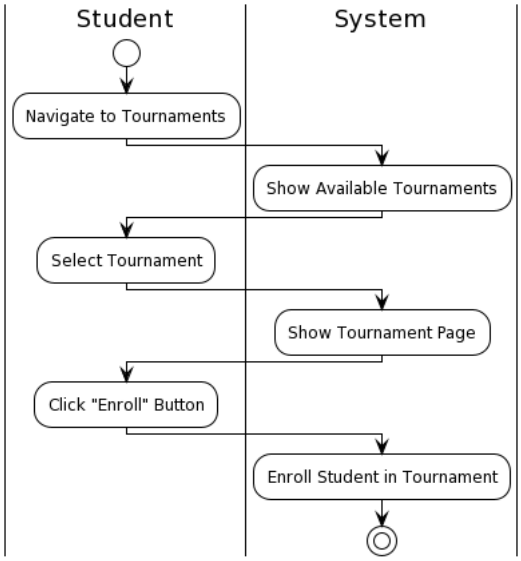

Figure 21: Joining a tournament use case activity diagram

ID	UC11
Name	CreateTeam
Actors	Student
Entry conditions	• Student is logged in • Studen has already enrolled to the tournament
Event flow	1. Student navigates to the tournaments section 2. System shows the list of tournaments the student is enrolled in 3. Student clicks on one of the tournaments in the list 4. Student is redirected to the page of the tournament containing the list of all the battles 5. Student clicks on the battle he wants to participate in 6. System redirects the student to the page of the battle 7. Student clicks on the "Create Team" button 8. System shows a pop-up with the team settings 9. Student inputs the name of the team, selects the desired privacy setting and clicks on "Confirm"

Exit conditions	• Student is correctly enrolled into the battle with its new team • System generates a unique invite API token and invite code valid for the team • The battle's page now shows a "Your team" button to manage the team settings
Exceptions	• Student inputs a team name already in use for that battle In the previous case, the student is not able to create a team and a pop-up with a specific error message is shown

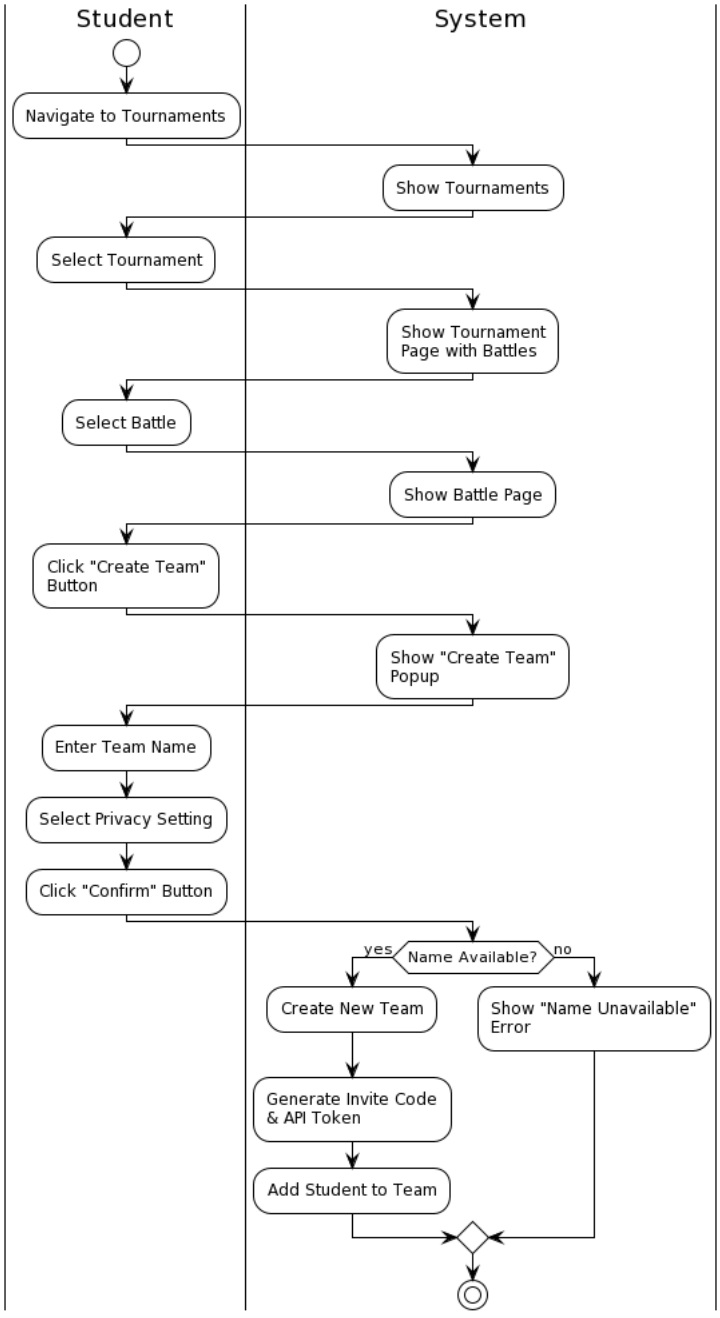

Figure 22: Team creation use case activity diagram

ID	UC12
Name	JoinTeam
Actors	Student
Entry conditions	• Student is logged in • Studen has already enrolled to the tournament
Event flow	1. Student navigates to the tournaments section 2. System shows the list of tournaments the student is enrolled in 3. Student clicks on one of the tournaments in the list 4. Student is redirected to the page of the tournament containing the list of all the battles 5. Student clicks on the battle he wants to participate in 6. System redirects the student to the page of the battle 7. Student clicks on the "Join Team" button 8. System shows a pop-up with 2 options and a input text field. The first options lets the user join a public team by name. The second option instead allows the user to join a team by invite code. The input field instead, asks for the team name or the invite code, depending on the checked option 9. Student selects his preferred option, inputs the requested string and clicks on "Confirm"

Exit conditions	• Student is correctly enrolled into the battle with its new team • The battle's page now shows a "Your team" button to manage the team settings
Exceptions	• Student tries to join a private team by name • Student tries to join a team which has already reached maximum number of members • Student inputs invalid team name or invite code In all the previous cases, the student is not able to join a team and a pop-up with a specific error message is shown

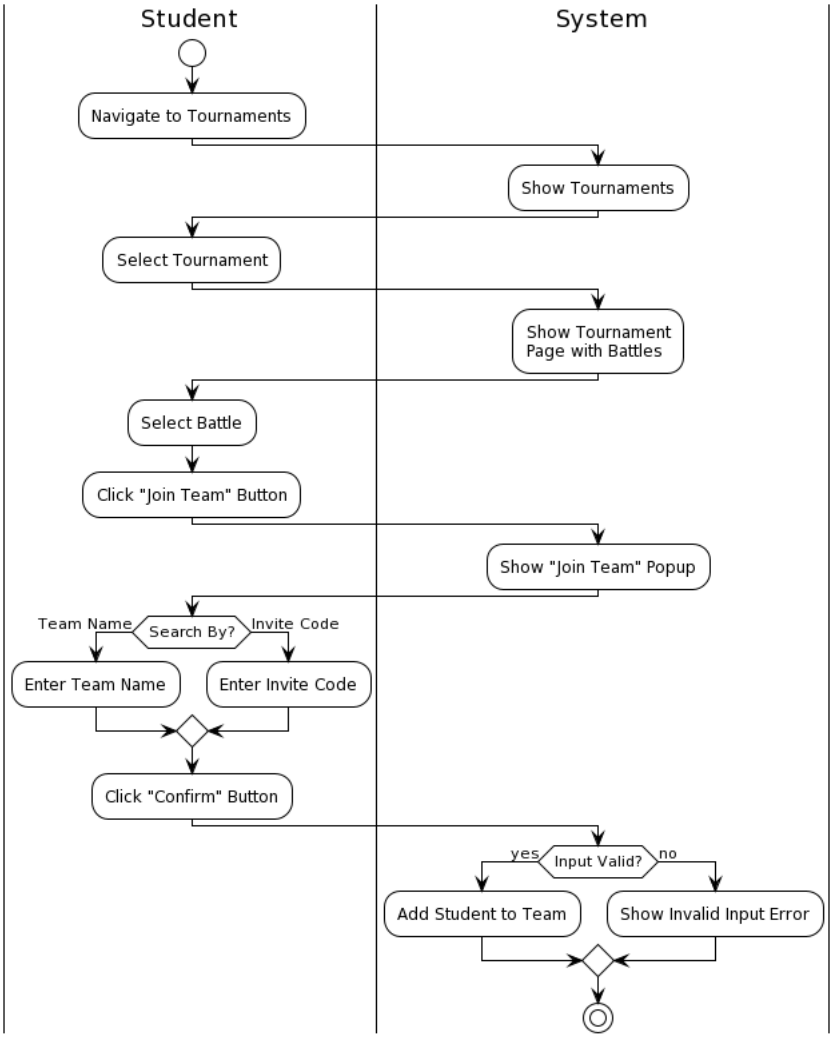

Figure 23: Joining a team use case activity diagram

ID	UC13
Name	ConfigureTeamRepoSettings
Actors	Student
Entry conditions	• Student is logged in • Student is already participating to a battle
Event flow	1. Student navigates to the tournaments section 2. System shows the list of tournaments the student is enrolled in 3. Student clicks on one of the tournaments in the list 4. Student is redirected to the page of the tournament containing the list of all the battles 5. Student clicks on the battle he wants to manage 6. System redirects the student to the page of the battle 7. Student clicks on "Your team" button 8. System redirects the student to the page of the team 9. Student can now visualize the team's settings such as system's API commit endpoint, team's API token to setup the GitHub actions and the invite code. Student can also set the URL of the GitHub repository

Exit conditions	• Systems correctly saves the inputted GitHub repository URL
Exceptions	• URL is invalid In the previous case, the system won't accept the input and an error message is shown next to the input field

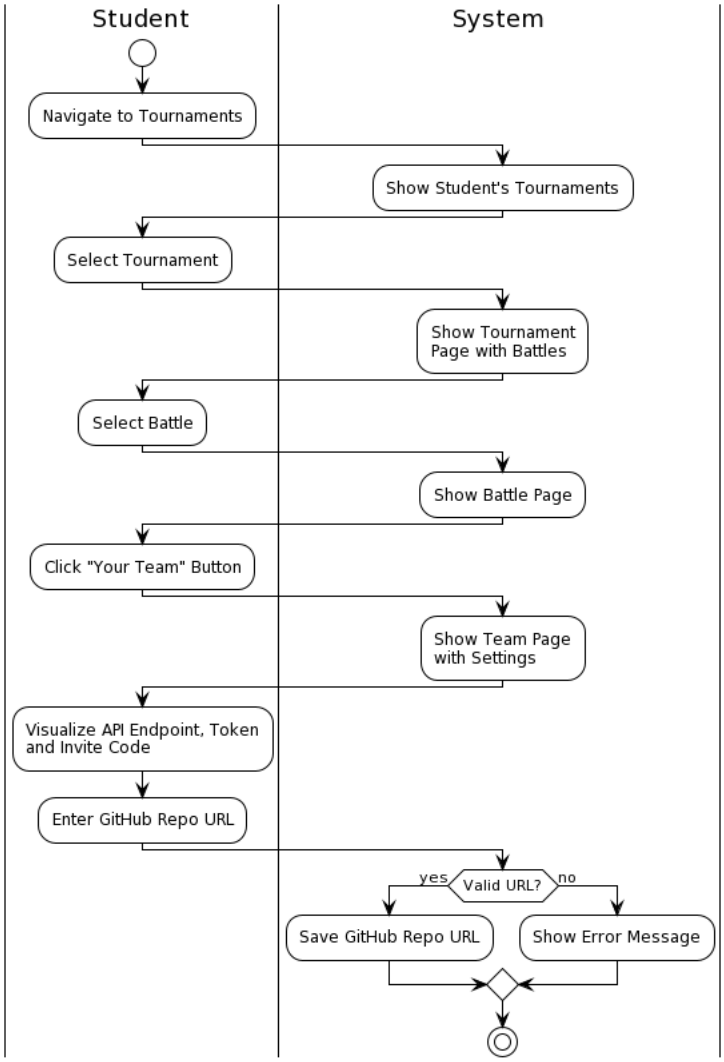

Figure 24: Team settings' configuration use case activity diagram

ID	UC14
Name	VisualizeSubmissionScore
Actors	Student
Entry conditions	• Student is logged in • Student is already participating to a battle
Event flow	1. Student navigates to the tournaments section 2. System shows the list of tournaments the student is enrolled in 3. Student clicks on one of the tournaments in the list 4. Student is redirected to the page of the tournament containing the list of all the battles 5. Student clicks on the battle he wants to check 6. System redirects the student to the page of the battle 7. Student clicks on "Your Team" button 8. System redirects the student to the page of the team
Exit conditions	• System shows all the scores related to the last submission and some statistics about past scores
Exceptions	• Team hasn't submitted any solution yet In the previous case, the team page will show no scores about last submission

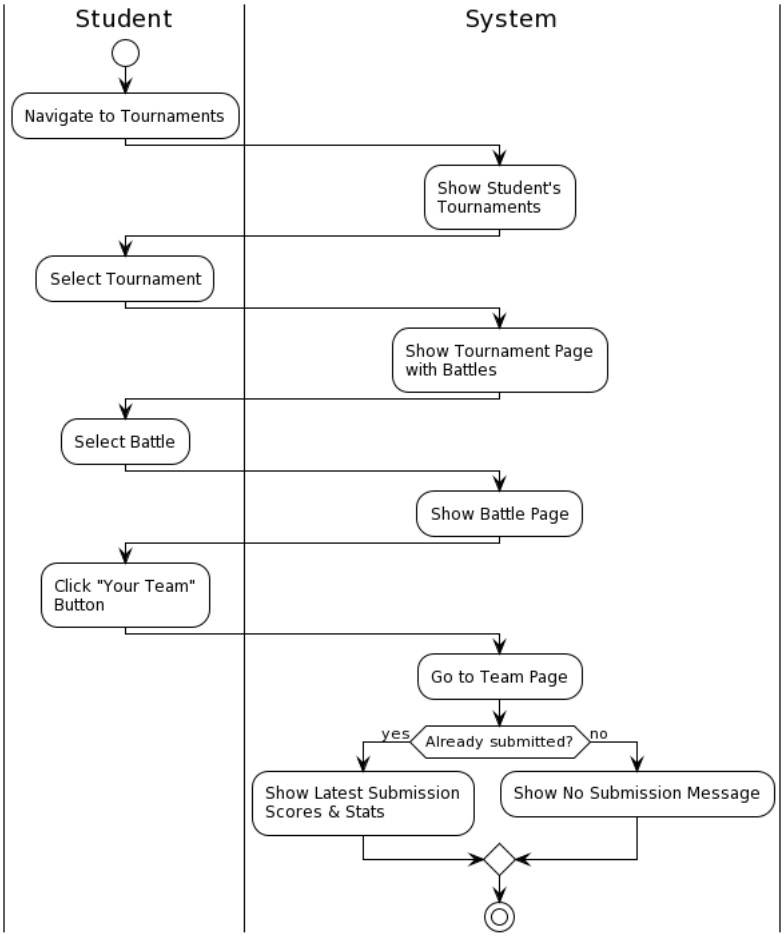

Figure 25: Submission's score visualization use case activity diagram

Educator perspective

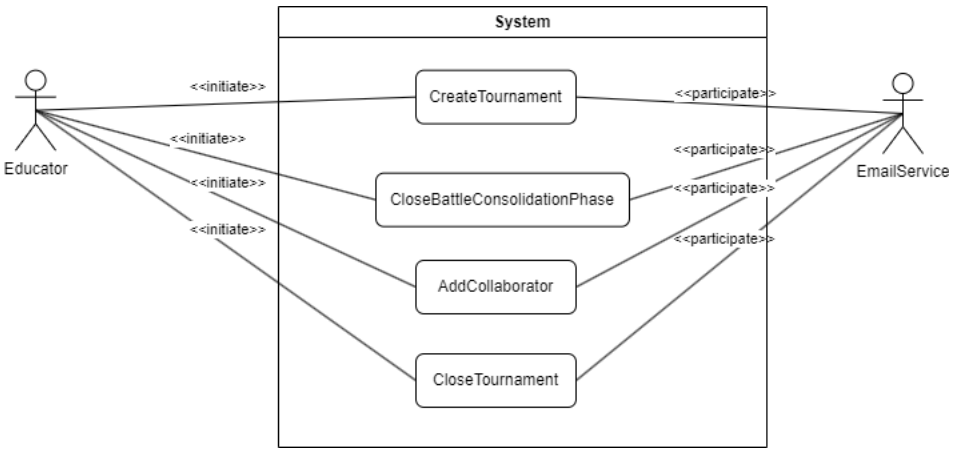

Figure 26: Educators' scenarios use case diagram

ID	UC15
Name	CreateTournament
Actors	Educator, Email Service
Entry conditions	• Educator is logged in
Event flow	1. Educator navigates to the tournaments section 2. System shows the list of ongoing tournaments the educator is managing 3. Educator clicks on the button next to the list to create a new tournament 4. System prompts a pop-up asking the name and the subscription deadline 5. Educator fills in with requested information and clicks on "Save" 6. System notifies all the students of the platform about the new tournament via the Email Service
Exit conditions	• System correctly creates the new tournament • Educator is redirected to the new tournament page • Emails to all the students are dispatched
Exceptions	• Provided name is already in use • Subscription deadline is invalid In all the previous cases, the educator is not able to create the tournament and a pop-up with a specific error message is shown

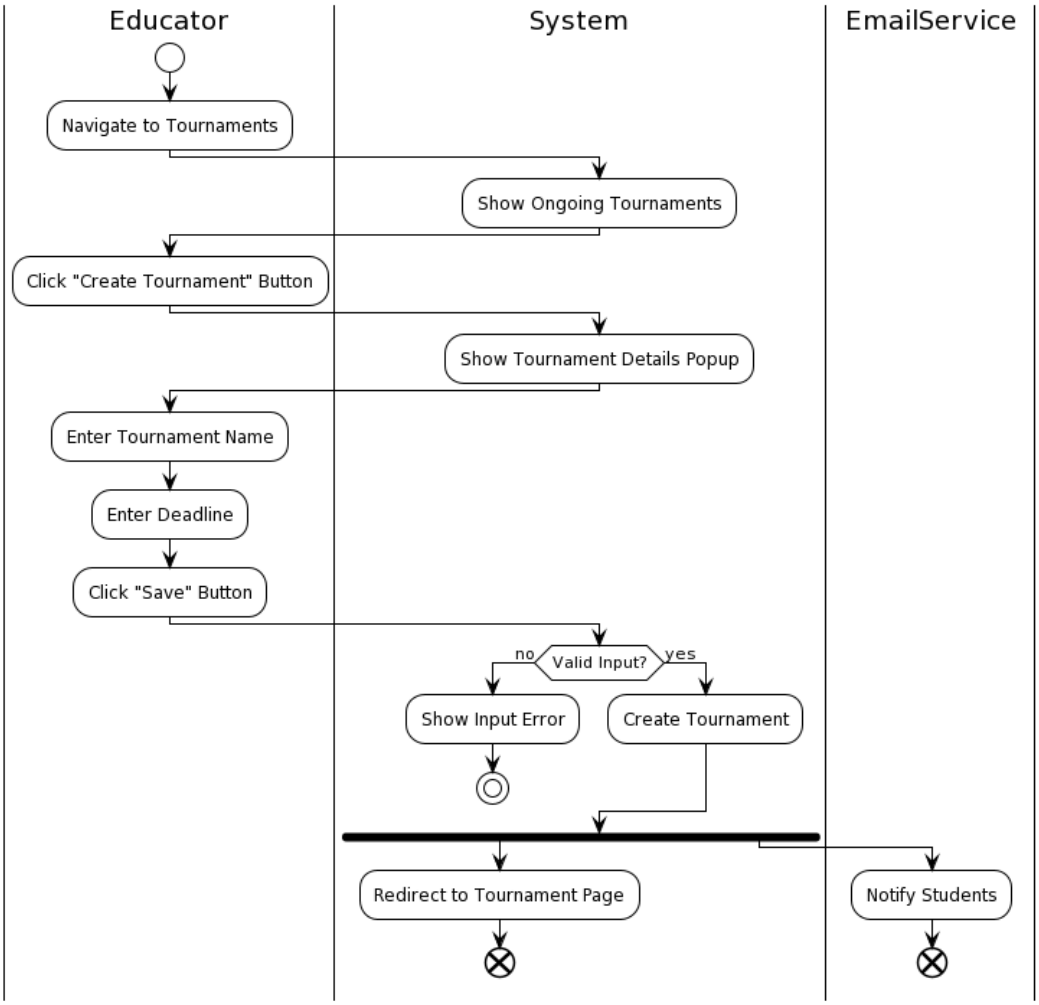

Figure 27: Tournament creation use case activity diagram

ID	UC16
Name	CloseTournament
Actors	Educator, Email Service
Entry conditions	• Educator is logged in • Educator is the creator of an ongoing tournament
Event flow	1. Educator navigates to the tournaments section 2. System shows the list of ongoing tournaments the educator is managing 3. Educator clicks on the tournament he is interested in 4. System redirects the educator to the tournament's page 5. Educator clicks on the "Close Tournament" button 6. Systems notifies all the enrolled students about the end of the tournament
Exit conditions	• Tournament ends correctly • Emails to all the students are dispatched
Exceptions	• Educator is trying to close a tournament with an ongoing battle In the previous case, the educator is not able to close the tournament and a pop-up with a specific error message is shown

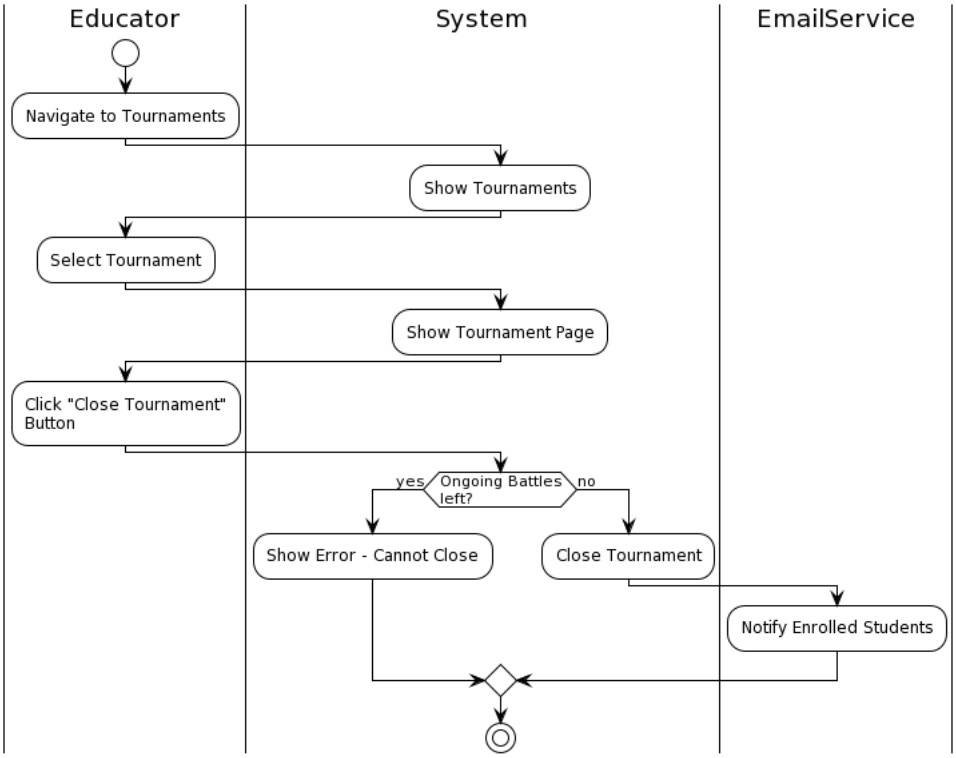

Figure 28: Tournament closure use case activity diagram

ID	UC17
Name	AddCollaborator
Actors	Educator, Email Service
Entry conditions	Educator is logged in and is the creator of a tournament
Event flow	1. Educator navigates to the tournaments section 2. System shows the list of ongoing tournaments the educator is managing 3. Educator clicks on the tournament he is interested in 4. System redirects the educator to the tournament's page 5. Educator clicks on the "Add Collaborator" button 6. System prompts a pop-up asking the email of the collaborator to add 7. Educator fills in the requested information and clicks on "Save" 8. System notifies the new collaborator about the invitation
Exit conditions	• New educator is correctly added to the tournament and can now create battles • Email to the new educator is dispatched
Exceptions	• Email does not belong to any educator registered to the platform In the previous case, the educator is not able to add the collaborator and a pop-up with a specific error message is shown

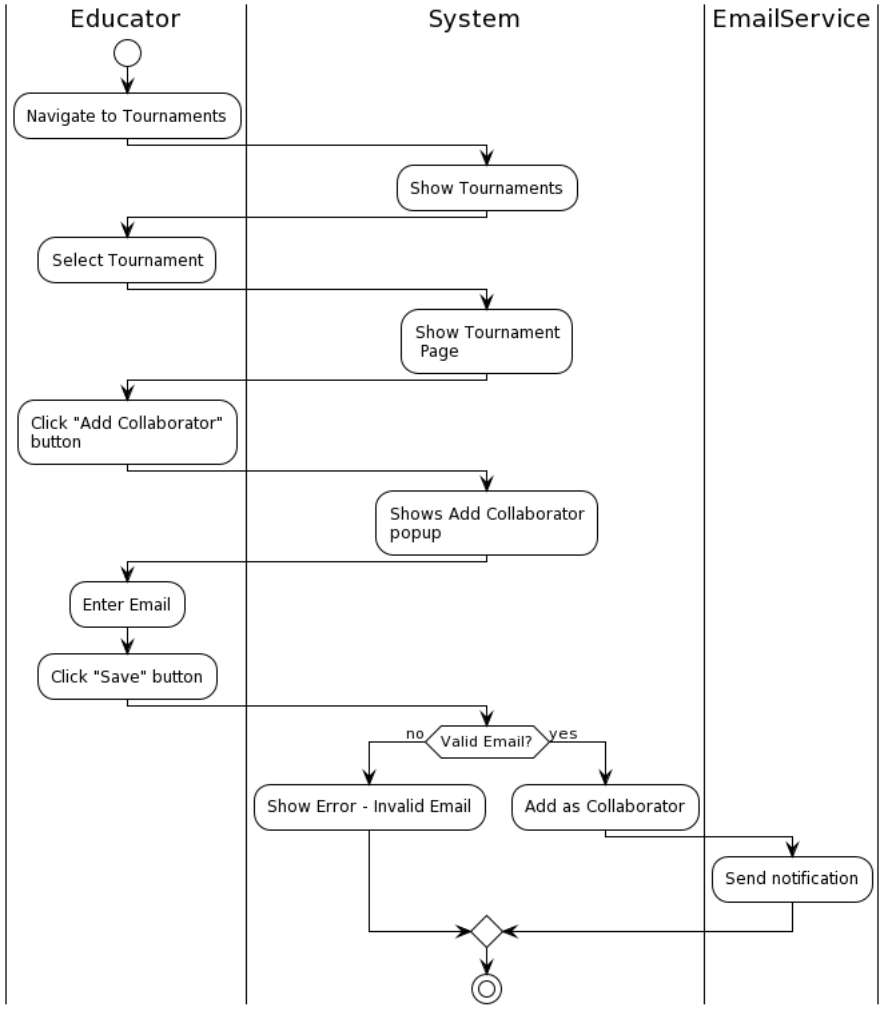

Figure 29: Adding a collaborator to a tournament use case activity diagram

ID	UC18
Name	CloseBattleConsolidationPhase
Actors	Educator, Email Service
Entry conditions	• Educator is logged in • Educator is the creator of a battle with the manual optional evaluation activated • Battle's submission phase has expired
Event flow	1. Educator navigates to the tournaments section 2. System shows the list of ongoing tournaments the educator is managing 3. Educator clicks on the tournament he is interested in 4. System redirects the educator to the tournament page, which contains the list of all the battles of the tournament 5. Educator clicks on the battle he is the creator of 6. System redirects the educator to the battle page 7. Educator clicks on the "Close Consolidation" button 8. System updates the personal tournament scores as the sum of all battle scores of the student obtained in the tournament 9. System updates both the tournament rank and the final battle rank 10. Systems notifies all the enrolled students into the battle about the availability of the battle scores

Exit conditions	• Battle is correctly closed, the new scores and ranks are updated • Emails to all the students are dispatched
Exceptions	• Educator is trying to close the consolidation stage of a battle without having completed the manual code evaluation In the previous case, the educator is not able to close the battle and a pop-up with a specific error message is shown, asking the educator to complete the evaluation first

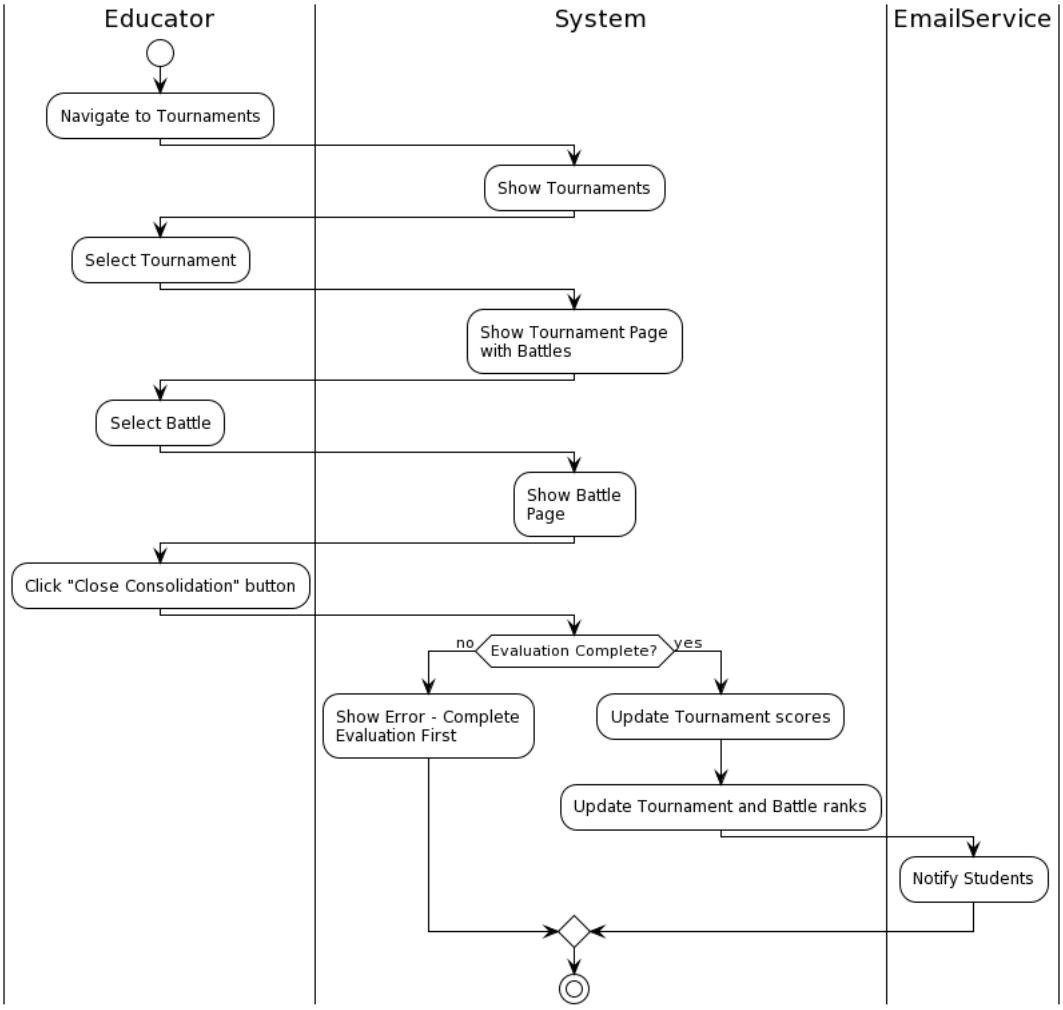

Figure 30: Close the consolidation phase of a battle use case activity diagram

3.2.2 Mapping on Use Cases

Use Case	Requirements
UC1	R1, R2
UC2	R1, R2
UC3	R2
UC4	R2
UC5	R9, R10, R11, R12, R13, R14, R15, R16, R18
UC6	R26, R27
UC7	R26, R28
UC8	R17, R19, R22, R29, R37, R38, R39
UC9	R20, R21, R39
UC10	R26, R30
UC11	R31, R32, R33, R37, R40
UC12	R31, R34, R35
UC13	R36, R37, R38, R40
UC14	R42
UC15	R3, R4, R5, R6
UC16	R7, R25
UC17	R8
UC18	R21, R23, R24, R41

3.2.3 Mapping on Goals

- **G1**: Allow students to participate to collaborative programming challenges

Requirements	**R1**: System shall allow user to register as educator or student **R2**: System shall allow user to log in **R6**: System shall notify all students enrolled to the platform about a newly created tournament **R16**: System shall notify all users enrolled to a tournament about a newly created battle **R18**: System shall create the GitHub repository of a battle as soon as the registration phase closes, sending its link to all involved students **R26**: System shall allow all users to see the list of ongoing tournaments **R30**: System shall allow student to enroll to a tournament within the subscription deadline **R31**: System shall allow student to create a team within the registration deadline **R32**: System shall allow student (team creator) to set the team name **R33**: System shall allow student (team creator) to set the privacy of the group **R34**: System shall allow student to join a team *only* before the registration deadline **R35**: System shall allow *only* students who know the correct invite code to join private groups **R36**: System shall allow team members to set their the repository URL **R37**: System shall generate a unique API token for each team **R38**: System should offer an external API which, when invoked, will notify the platform about a new commit on a team's GitHub repository **R40**: System shall generate a unique invite code for each team

Domain Assumptions	**D1**: Users have access to a stable and reliable internet connection to interact with the CKB platform **D5**: Students are expected to engage in Code Kata Battles with integrity, without resorting to plagiarism or cheating (e.g. inviting more or different collaborators with respect to the ones inside the team) **D6**: Maximum number of students per group an educator can set will always be bounded (e.g. less than 4) **D7**: Every student has a GitHub account **D8**: Students will fork the Code Kata GitHub repository once ready **D9**: Students know how to setup a GitHub Actions worfkflow **D10**: Only one student per group will perform the steps needed to set-up the team's repository and the GitHub Actions workflow **D12**: Students use external communication channels

- **G2**: Allow educators to create programming challenges

Requirements	**R1**: System shall allow user to register as educator or student **R2**: System shall allow user to log in **R3**: System shall allow educator to create a tournament **R4**: System shall allow educator (tournament creator) to set the name of the tournament **R5**: System shall allow educator (tournament creator) to set the subscription deadline of the tournament **R7**: System shall allow educator (tournament creator) to end the tournament **R8**: System shall allow educator (tournament creator) to add another educator as collaborator to the tournament **R9**: System shall allow educator (tournament creator or collaborator) to create a battle for the tournament **R10**: System shall allow educator (battle creator) to upload the Code Kata of the battle **R11**: System shall allow educator (battle creator) to set minimum and maximum number of students per group allowed for the battle **R12**: System shall allow educator (battle creator) to set the registration deadline of the battle **R13**: System shall allow educator (battle creator) to set the submission deadline of the battle **R14**: System shall allow educator (battle creator) to enable/disable optional manual evaluation for the battle **R15**: System shall allow educator (battle creator) to select relevant aspects of code quality to be extracted through static analysis

Domain Assumptions	**D1**: Users have access to a stable and reliable internet connection to interact with the CKB platform **D2**: Supported programming languages are limited to popular options like Java, Python and C++ **D3**: Registered educators are all legitimate and verified **D4**: Educators upload correct test cases and well-structured Code Kata projects **D6**: Maximum number of students per group an educator can set will always be bounded (e.g. less than 4)

- **G3**: Allow the system to automatically evaluate students' submissions

Requirements	**R17**: The system shall be able to perform static analysis on submitted code utilizing third-party static code analysis tools and services **R18**: System shall create the GitHub repository of a battle as soon as the registration phase closes, sending its link to all involved students **R19**: System shall assign to each submission sent before the submission deadline a score (natural number between 0 and 100) combining correctness, timeliness and code quality *as soon as possible* **R21**: System shall compute a final score combining automatic and manual score (if available) for each team if manual evaluation is enabled **R36**: System shall allow team members to set their the repository URL **R37**: System shall generate a unique API token for each team **R38**: System should offer an external API which, when invoked, will notify the platform about a new commit on a team's GitHub repository

Domain Assumptions	**D2**: Supported programming languages are limited to popular options like Java, Python and C++ **D4**: Educators upload correct test cases and well-structured Code Kata projects **D8**: Students will fork the Code Kata GitHub repository once ready **D9**: Students know how to setup a GitHub Actions worfkflow **D10**: Only one student per group will perform the steps needed to set-up the team's repository and the GitHub Actions workflow **D11**: Static analysis tools are able to quantify the specified code quality aspects

- **G4**: Allow educators to manually evaluate students' submissions

Requirements	**R1**: System shall allow user to register as educator or student **R2**: System shall allow user to log in **R13**: System shall allow educator (battle creator) to set the submission deadline of the battle **R14**: System shall allow educator (battle creator) to enable/disable optional manual evaluation for the battle **R20**: System shall allow educator (battle creator) to set a manual score (natural number between 0 and 100) to the last valid team's submission during the consolidation stage of the battle **R39**: System shall allow the educator to access the code of the submitted solutions **R41**: System shall allow educator (battle creator) to close the consolidation stage of the battle if manual evaluation is enabled
Domain Assumptions	**D1**: Users have access to a stable and reliable internet connection to interact with the CKB platform **D8**: Students will fork the Code Kata GitHub repository once ready **D10**: Only one student per group will perform the steps needed to set-up the team's repository and the GitHub Actions workflow

- **G5**: Allow students to track their performance

Requirements	**R1**: System shall allow user to register as educator or student **R2**: System shall allow user to log in **R17**: The system shall be able to perform static analysis on submitted code utilizing third-party static code analysis tools and services **R19**: System shall assign to each submission sent before the submission deadline a score (natural number between 0 and 100) combining correctness, timeliness and code quality *as soon as possible* **R20**: System shall allow educator (battle creator) to set a manual score (natural number between 0 and 100) to the last valid team's submission during the consolidation stage of the battle **R21**: System shall compute a final score combining automatic and manual score (if available) for each team if manual evaluation is enabled **R22**: System shall update the battle teams' ranks right after a new score is available **R23**: System shall notify all students participating to the battle about the availability of the final battle rank **R24**: System shall update the tournament students' scores as the sum of all the battle scores received during the tournament once the final battle rank becomes available **R25**: System shall notify all involved users about the availability of the tournament student's final rank **R27**: System shall allow all users to see all tournaments' ranks **R28**: System shall allow all users to see all battles' ranks **R29**: System should manage every ranking (tournament or battle) in a way that it represents the correct order of students/teams within its context, from the ones with the highest score to the ones with the lowest **R42**: System shall allow students to see the latest score of their team

Domain Assumptions	**D1**: Users have access to a stable and reliable internet connection to interact with the CKB platform

3.3 Performance Requirements

Even though the platform is not mission critical, should guarantee a smooth and appealing experience to all kind of users. Given the complexity of the overall project, several performance requirements for different aspects of the system can be identified:

- *Basic user interaction with the webapp*: response time for user actions like creating/joining a team, loading scoreboards, creating a tournament etc should be under 2 seconds for 90% of requests under normal load.
- *Space requirements*: a reasonable estimate of a project's maximum size is about 40-50 MB. So, the system, to control the space usage, should limit the allowed projects sizes to 80 MB.
- *Upload/download speed requirements*: given the previous estimate of project's size, the system should guarantee a good throughput in terms of upload and download speed of the projects' files. System should be able to reserve on average at least 8 Mb/s to each connection; this means that a project directory of 50 MB would take about 50 seconds to be downloaded. However, bandwidth of educator's uploads should be prioritized. A pessimistic estimate with 1000 concurrent upload/download connections suggests that the systems needs at least a 10 Gb/s internet connection. Aggregating different Internet Service Providers may be considered to achieve greater bandwidths.
- *Computational resources*: system should be sized to handle up to 100 simultaneous code submissions from different teams, providing a score within 20 seconds. Multiple CPUs with worker pooling approach will help in handling this kind of concurrency, dealing also with scalability issues.
- *Notifications*: email notification should be able to reach all the recipients within 2 minutes.

These performance requirements will be taken in consideration also in choosing the external static analysis provider.

Moreover, given the different and independent performance requirements of all the functions involved, it is preferred to support each one with different machines or subsystems. In this way, the overall platform could still work with degraded performance affecting only a part of the functionalities.

The performance of the system could be improved following some basic strategies:

- **Limit inbound API rate**: this will help in reducing commits from the same team (preventing also spam attacks).
- **Bound execution time**: terminate the evaluation of a submission after a timeout. This will help to prevent excessive resource consumption on tasks that should be quick to perform.

In conclusion, it is important to mention again that even though the system is not providing an essential service to the user, it should offer a non stressful educational experience. Indeed, given the strict deadlines system, even a little delay in the submission can compromise the competition (which could potentially reflect into school marks of students), thus the overall purpose of the project.

3.4 Design Constraints

3.4.1 Standards compliance

There are several standards the system must comply with:

- The web interface will comply with the latest versions of HTML, CSS and JavaScript standards.
- API will follow REST architectural constraints and use JSON data formats.
- Coordinated Universal Time (UTC) will be used to store timestamps in order to correctly show deadlines to user located in different time-zones.

3.4.2 Hardware limitations

The CDK platform has no strict hardware requirements, but should perform well on average smartphone or computer equipped with a modern web browser and a reliable internet connection.

3.4.3 Privacy constraints

Since the platform will need to ask the user for personal information like name, surname and email address, it needs to guarantee that this data will be processed and stored accordingly to the GDPR. Moreover, from the moment that the platform will also use cookies (e.g. to keep users authenticated), the system will need to get consent from visitors before placing cookies on their devices, in compliance with the EU ePrivacy regulations.

3.4.4 Quality constraints

User acceptance testing will measure task success rate and user satisfaction.

3.5 Software System Attributes

3.5.1 Reliability

Given the complexity of the project, reaching high reliability levels represents a challenge and a key to success for the product. Indeed, even though the systems performs many interactions with external actors and service providers, consistency and integrity of the overall platform should be guaranteed even in case of errors or unexpected conditions.

3.5.2 Availability

Service should be continuously available (24/7) to users, with little downtime and rapid service recovery. The overall service level target is 99.5%, which corresponds nearly to 50 minutes of downtime per week. Planned maintenances should be communicated to users in advance, given the strict time requirements of the service offered by the platform.

3.5.3 Usability

The platform should be intuitive and minimal, still offering all the functionalities needed by all kind of users.

3.5.4 Security

- Passwords must be securely hashed and salted before storage.
- Encryption should be used for sensitive data.
- Input validation and sanitization should be applied to prevent attacks.
- Adopt HTTPS protocol, which uses encryption for secure communication.
- Deploy firewalls to protect the system from unauthorized access.
- Employ API authentication mechanism to limit access to the API.

3.5.5 Maintainability

The system should be designed to be easily maintained and updated, allowing also new functionalities to be added in future. This can be achieved by adopting a modular architecture, which permits to easily replace or update components without affecting the overall system. The most important aspects related to maintainability and modularity will be addressed in the design document.

3.5.6 Portability

Since the platform is web-based, it will be accessible from any device equipped with a modern web browser. For this reason, the system interface should be responsive and usable on different screen sizes, also on mobile devices.

3.5.7 Scalability

The system should be designed to easily scale horizontally to accommodate growth in traffic.

4 Formal Analysis Using Alloy

An Alloy model is provided to present a formal description of the system. The goals of this kind of analysis are to check the consistency of the model and to find possible errors and corner cases in the requirements.
A formal model is also useful to express complex facts which are not captured by the domain class diagram such as the constraints involving subcriptions, scores, timestamps, battle and tournament states.

4.1 Model Code

```
open util/integer

//SIGNATURES

sig Email {}

sig Password {}

sig GithubUsername {}

abstract sig User {

    email: Email,

    password: Password

}

sig Student extends User {

    username: one GithubUsername

}

sig Educator extends User {

    createsTournament: set Tournament,

    createBattle: set Battle

}
```

```
sig Timestamp {
    time: Int
} {
    time ≥ 0
}
one sig SystemTimestamp {
    ts: Timestamp
}
pred earlierThan [t1, t2: Timestamp] {
    int t1.time < int t2.time
}
pred earlierEqual [t1, t2: Timestamp] {
    int t1.time ≤ int t2.time
}
enum TournamentState { Subscription, Ongoing, Ended }
sig Tournament {
    subscriptionDeadline: Timestamp,
    state: TournamentState ,
    isManaged: some Educator,
    battles: set Battle,
    participants: set Student -> one Score
}
sig TestCase {}
sig CodeKata {
testCases: some TestCase
}
enum Bool { True, False }
enum ProgrammingLanguage { Python, Java, C }
enum BattleState { TeamFormation, SubmissionPhase, ConsolidationPhase,
    ↪ Finished }
```

```
sig Battle {
    state: BattleState ,
    participants: set Team,
    registrationDeadline: Timestamp,
    submissionDeadline: Timestamp,
    minTeamSize: Int,
    maxTeamSize: Int,
    programmingLanguage: ProgrammingLanguage,
    codeKata: CodeKata,
    a1Eval: Bool,
    a2Eval: Bool,
    a3Eval: Bool,
    manualEval: Bool
} {
    earlierThan[registrationDeadline,submissionDeadline]
    minTeamSize > 0
    maxTeamSize ≥ minTeamSize
}
sig Score {
    value: Int
} {
    int value ≥ 0
}
sig Submission {
    ts: one Timestamp,
    functionalScore: one Score,
    timelinessScore: one Score,
    a1Score: lone Score,
    a2Score: lone Score,
```

```
    a3Score: lone Score,

    qualityScore: one Score,

    manualScore: lone Score,

    overallScore: one Score,

    battle: Battle

}

sig ApiToken{}

sig InviteCode{}

sig Team {

    submissions: set Submission,

    participates: one Battle,

    members: some Student,

    battleScore: one Score,

    apiToken: one ApiToken,

    inviteCode: one InviteCode

}


//INTEGRITY CONSTRAINTS

// each password is associated to some user (some users may have the same
    ↪ password)
fact eachPasswordToSomeUser{
all p : Password | ( some u : User | u. password = p)
}

// each Int associated to at most 1 timestamp
fact oneTimestampPerInteger{
all i:Int | ( lone t : Timestamp | t.time = i)
}

// each Int associated to at most 1 score
fact oneTimestampPerInteger{
all i:Int | ( lone s : Score | s.value = i)
}

// each username is associated to a unique student
fact eachGitHubUsernameToOneStudent{
all u : GithubUsername | ( one s : Student | s. username = u)
}
```

```
// each ApiToken is associated to a unique team
fact eachApitTokenToOneTeam{
all a : ApiToken | ( one t : Team | t. apiToken= a)
}

// each InviteCode is associated to a unique team
fact eachInviteCodeToOneTeam{
all i : InviteCode | ( one t : Team | t. inviteCode= i)
}

// each email is associated to a unique user
fact eachEmailToOneUser{
all e : Email| ( one u : User | u. email= e)
}

// each tournament has 1 creator
fact eachTournamentHasOneCreator{
all t : Tournament | ( one e : Educator | t in e.createsTournament )
}

// each battle has 1 creator
fact eachBattleHasOneCreator{
all b : Battle | ( one e : Educator | b in e.createBattle)
}

// each battle belongs to only a tournament
fact eachBattleToOneTournament{
all b: Battle| (one t : Tournament | b in t.battles )
}

// each testcase belongs to only a code kata
fact eachTestCaseToOneCodeKata{
all t: TestCase| (one c : CodeKata| t in c.testCases )
}

// each CodeKata belongs to only a battle
fact eachCodeKataToOneBattle{
all c: CodeKata| (one b : Battle| c in b.codeKata )
}

// each Submission belongs to only a Team
fact eachSubmissionToOneTeam{
all s: Submission| (one t : Team| s in t.submissions )
}

// coherence between battle's participants and team's participates
fact coherenceBetweenParticipantsAndParticipate{
all t: Team, b: Battle | t in b.participants ⟺ t.participates = b
}
```

```
//ADDITIONAL CONSTRAINTS

//tournament is on ongoing state if and only if subscription deadline has
    ↪ elapsed
fact tournamentStateCoherentWithSystemTimestamp{
all t:Tournament |
    earlierThan[t.subscriptionDeadline, SystemTimestamp.ts] ⟺ t.state =
        ↪ Ongoing
}

// battle states coherent with system timestamp
fact battleStateCoherentWithSystemTimestamp{
all b:Battle |
    (earlierThan[SystemTimestamp.ts, b.registrationDeadline]
        implies b.state = TeamFormation) and
    ((earlierEqual[ b.registrationDeadline,SystemTimestamp.ts] and
        ↪ earlierThan[ SystemTimestamp.ts, b.submissionDeadline])
        implies b.state = SubmissionPhase) and
    (earlierEqual[ b.submissionDeadline,SystemTimestamp.ts]
        implies (b.state = ConsolidationPhase or b.state = Finished) )
}

//submission ts come before system timestamp
fact submissionTsBeforeSystemTimestamp{
all s:Submission | earlierEqual[s.ts, SystemTimestamp.ts]
}

// each student can subscribe only once to a tournament
fact eachStudentSubscribesMostOnce{
all t:Tournament, s:Student | #t.participants[s] ≤ 1
}

// tournament creator is also managing the tournament (tournament has at
    ↪ least 1 educator managing it)
fact creatorCanManage{
all e: Educator, t:e.createsTournament  | e in t.isManaged
}

// tournament's participants scores are the sum of battle scores obtained by
    ↪  the teams the student is a member of in the battles of that
    ↪ tournament
fun allStudentTeamsInEndedBattles[s:Student]:set Team{
{t:Team | s in t.members and t.participates.state = Finished}
}
fun totalTournamentScore[t: Tournament, s: Student]: Int {
sum te: t.battles.participants&allStudentTeamsInEndedBattles[s] | int te.
    ↪ battleScore.value
}
fact TournamentParticipantsScoreComputation{
all t: Tournament, s: Student | #t.participants[s] = 1 implies int t.
    ↪ participants[s].value = int totalTournamentScore[t,s]
}

// submission's quality score is the sum of all aspects scores
fun add3Scores[a, b, c: Score]: Score {
{i: Score | int i.value = integer/add[ integer/add[a.value, b.value], c.
    ↪ value]}
```

```
}
fact qualityScoreCombinesAspectsScores {
all s:Submission | s.qualityScore = add3Scores[s.a1Score, s.a2Score, s.
    ↪ a3Score]
}

// submission's overall score is the sum of all the scores
fun add4Scores[a, b, c, d: Score]: Score {
{i: Score | int i.value = integer/add[integer/add[integer/add[a.value,b.
    ↪ value],c.value], d.value]}
}
fact OverallScoreCombinesAutoAndManualScore {
all s:Submission | s.overallScore = add4Scores[s.functionalScore, s.
    ↪ timelinessScore, s.qualityScore, s.manualScore]
}

// battleScore is the overall score of the team's last submission
fun lastSubmission[t: Team]: one Submission {
{ s: t.submissions | no su: t.submissions - s | earlierThan[s.ts,su.ts] }
}
fact battleScoreIsLastOverallScore {
all t: Team | (#t.submissions= 0 implies int t.battleScore.value = 0) and
              (#t.submissions >0 implies int t.battleScore.value = int
                  ↪ lastSubmission[t].overallScore.value)
}

// battle registration deadline comes after tournament subscription deadline
fact RegistrationAfterSubscription {
all b:Battle,to:Tournament | b in to.battles implies earlierThan[to.
    ↪ subscriptionDeadline,b.registrationDeadline]
}

// Student participates only to battles that belong to tournaments he is
    ↪ enrolled in
fact OnlyBattlesInEnrolledTournament {
all s:Student, te:Team, to:Tournament | (s in te.members and te.participates
    ↪  in to.battles) implies (#to.participants[s] = 1)
}

// a battle can be associated to a tournament only if the tournament is not
    ↪ in registration phase anymore
fact NoBattleInTournamentSubscription {
no t:Tournament | t.state = Subscription and #t.battles > 0
}

// if a tournament is ended, then all its battles must be ended too
fact AllBattleEndedIfTournamentEnded {
all t:Tournament| t.state = Ended implies (all b: t.battles | b.state =
    ↪ Finished)
}

// if a battle is in teamFormation/submission/consolidation state, then its
    ↪ tournament must be on ongoing state
fact TournamentOngoingIfBattleNotEnded {
all t:Tournament, b:t.battles | b.state ≠ Finished implies (t.state ≠ Ended)
}
```

```
// automated scores of the submission are present only if they are enabled
    ↪ in the battle
fact AutomatedScorePresentIfEnabled {
all s:Submission | (#s.a1Score = 1 ⟺ s.battle.a1Eval = True) and
                   (#s.a2Score = 1 ⟺ s.battle.a2Eval = True) and
                   (#s.a3Score = 1 ⟺ s.battle.a3Eval = True)
}

// a submission can be associated to a battle only if the battle is in the
    ↪ submission/consolidation/ended state
fact noSubmissionInTeamFormationPhase {
no s:Submission | s.battle.state = TeamFormation
}

// a submission can happen only after the registration deadline and before
    ↪ the battle's submission deadline
fact SubmissionInSubmissionPhase {
all s:Submission | earlierThan[s.battle.registrationDeadline,s.ts] and
    ↪ earlierThan[s.ts,s.battle.submissionDeadline]
}

// a submission to a battle can occur only if the team is participating to
    ↪ the battle
fact TeamSubmitsOnlyToCorrectBattle {
all t:Team, s:Submission | (s in t.submissions) implies (s.battle = t.
    ↪ participates)
}

// a battle can be created only by educators managing the tournament the
    ↪ battle belongs to
fact BattleCreatedByManager {
all b:Battle, e:Educator, t:Tournament | (b in e.createBattle and b in t.
    ↪ battles) implies (e in t.isManaged)
}

// team members count respects battle's settings
fact TeamSize {
all t:Team | #t.members ≥ int t.participates.minTeamSize and
             #t.members ≤ int t.participates.maxTeamSize
}

// a student can participate to a battle only with 1 team
fact OnlyOneTeamPerBattle {
no t1,t2:Team,s:Student  | t1 ≠ t2 and
                           s in t1.members and
                           s in t2.members and
                           t1.participates = t2.participates
}

// tournament scores initialized to 0  (if tournament is not started yet or
    ↪ none of its battles are ended)
fact ZeroInitializedTournamentScores {
all s:Student, to:Tournament, b:Battle | (#to.participants[s] = 1
        and (to.state = Subscription or (b in to.battles and b.state ≠
            ↪ Finished) ) ) implies to.participants[s].value = 0
}
```

```
// if a submission comes after another, then its timeliness score should be
    ↪ lower
fact LowerTimelinessScore {
all b:Battle, disj s1,s2:Submission  |  (b in s1.battle and b in s2.battle
    ↪ and earlierThan[s1.ts,s2.ts])
        implies
    int s1.timelinessScore.value > int s2.timelinessScore.value
}

// There no exist 2 concurrent submissions from same team
fact NoConcurrentSubmission{
all t:Team| no disj s1,s2:t.submissions  |  s1.ts.time = s2.ts.time
}

// a student can be enrolled to a tournament at most once
fact NoDoubleEnrollment {
no to:Tournament, s:Student| #to.participants[s] > 1
}

// a battle can enter ConsolidationPhase only if manual evaluation is
    ↪ enabled
fact consolidationPhaseOnlyWithManualEval{
all b: Battle | b.state = ConsolidationPhase implies b.manualEval = True
}

// manual score can be present only if the related battle is in
    ↪ consolidation phase or finished
fact ManualScoreOnlyInConsolidationOrFinished{
all s: Submission | #s.manualScore = 1 implies (s.battle.state =
    ↪ ConsolidationPhase or s.battle.state = Finished)
}

// manual score in the submission is present only if its enabled in the
    ↪ battle
fact ManualScorePresentIfEanbled {
all s:Submission | #s.manualScore = 1 implies s.battle.manualEval = True
}

// manualScore can be present only in the last submission
fact manualScoreOnlyInLastSubmission{
all t: Team, s: t.submissions | #s.manualScore = 1 implies s =
    ↪ lastSubmission[t]
}

// if a battle is ended, manualScore must be present in the last submission
fact manualScoreWhenBattleFinished{
all t: Team | (t.participates.state = Finished and t.participates.manualEval
    ↪  = True)
        implies
    #lastSubmission[t].manualScore = 1
}
```

```
// Show predicates
pred world1{
#Tournament=1
#Educator=2
#Battle=2
#Tournament.isManaged=2
}

pred world2{
#Tournament=1
#Educator=1
#Battle=2
#Team=2
#Submission=3
}

run world1  for 6
```

4.2 Static analysis

```
//Check that all students enrolled in the tournament have 0 tournament score
    ↪  during subcription phase
assert StudentTournamentScoreBeforeStart{
    all t : Tournament, s:Student | (#t.participants[s]=1 and t.state=
        ↪ Subscription) implies int t.participants[s].value = 0
}
check StudentTournamentScoreBeforeStart for 4

//Check that all students enrolled to tournament without any finished battle
    ↪  have 0 tournament score
assert StudentTournamentScoreNoFinishedBattles{
    all t : Tournament, s:Student|(#t.participants[s]=1 and t.state=Ongoing
        ↪ and (no b: t.battles, te:b.participants |  b.state = Finished and
        ↪  s in te.members)) implies int t.participants[s].value = 0
}
check StudentTournamentScoreNoFinishedBattles for 4

//Check that all students enrolled to tournament with a tournament score
    ↪ greater than 0 have participated to at least 1 finished battle in
    ↪ that tournament
assert StudentParticipatesWithTeam{
    all t : Tournament, s:Student| int t.participants[s].value > 0 implies (
        ↪  some b:t.battles,te:t.battles.participants | b.state=Finished
        ↪ and s in te.members)
}
check StudentParticipatesWithTeam for 4

//Check that all teams enrolled to the battle have 0 battle score before
    ↪ their first submission
assert TeamScoreZeroBeforeSubmissions{
    all b : Battle, te:b.participants | (#te.submissions = 0) implies int te
        ↪ .battleScore.value = 0
}
check TeamScoreZeroBeforeSubmissions for 4
```

```
//Check that all teams having battle score greater than 0 have at least 1
    ↪ submission
assert TeamNonZeroScore{
    all b : Battle, te:b.participants |  int te.battleScore.value > 0
        ↪ implies  (#te.submissions > 0)
}
check TeamNonZeroScore for 4

//Check that all teams's latest submission have been manually evaluated if a
    ↪  battle is ended
assert AllTeamsManuallyEvaluated{
    no b : Battle |  b.manualEval = True and b.state=Finished and (some te:b
        ↪ .participants | #lastSubmission[te].manualScore = 0)
}
check AllTeamsManuallyEvaluated for 4
```

The alloy model previously presented focuses on the most important entities and attributes of the domain, which are strictly necessary to describe the main functionalities of the system. Indeed, some concepts such as time and scores have been simplified to be easily handled by the Alloy Analyzer and syntax.
Images of three world examples are shown below, the first focuses on educators, tournaments and battles, while the second and the third ones include all the aspects related to students, teams and submissions.

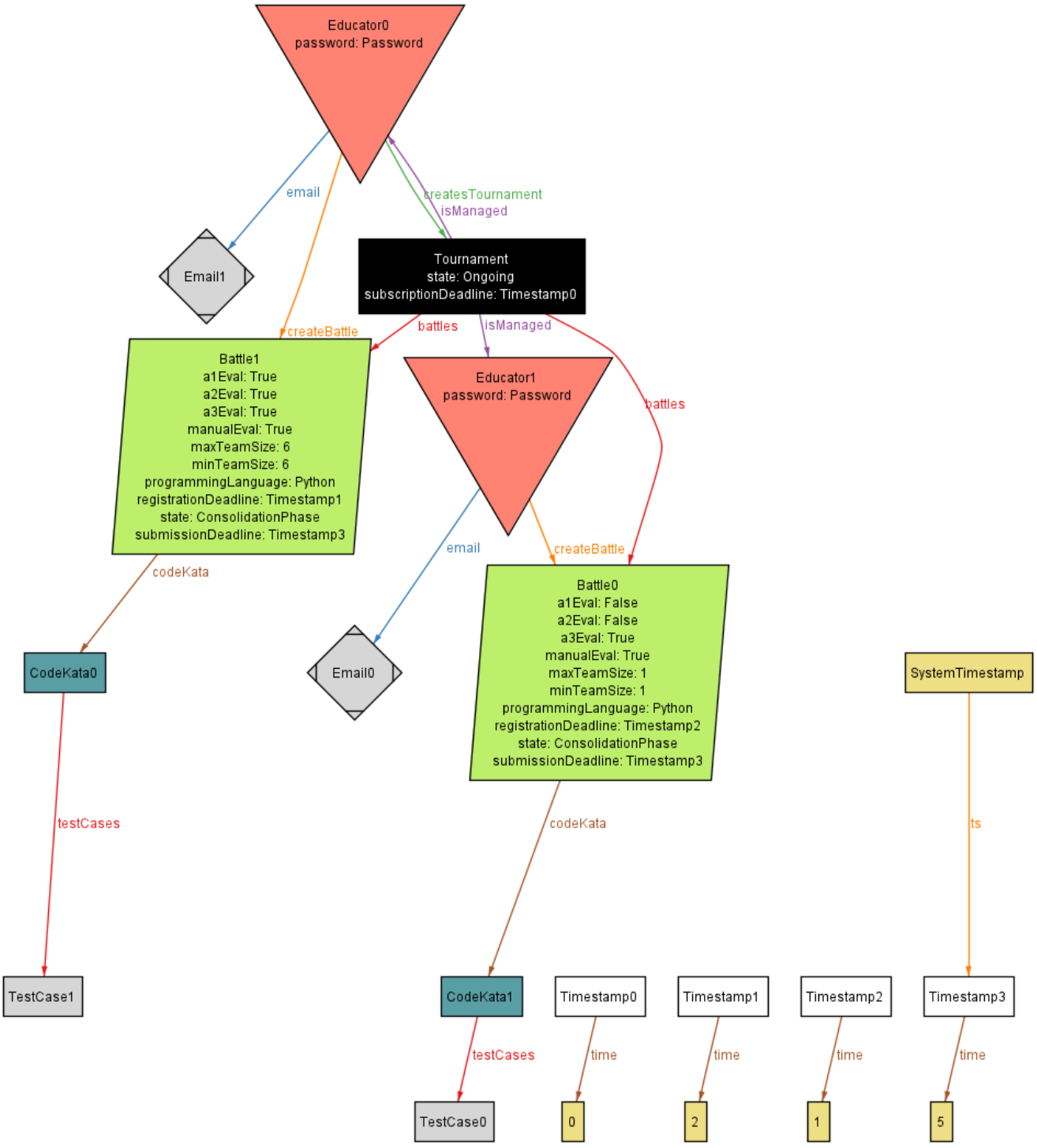

Figure 31: World1, generated with the *world1* predicate

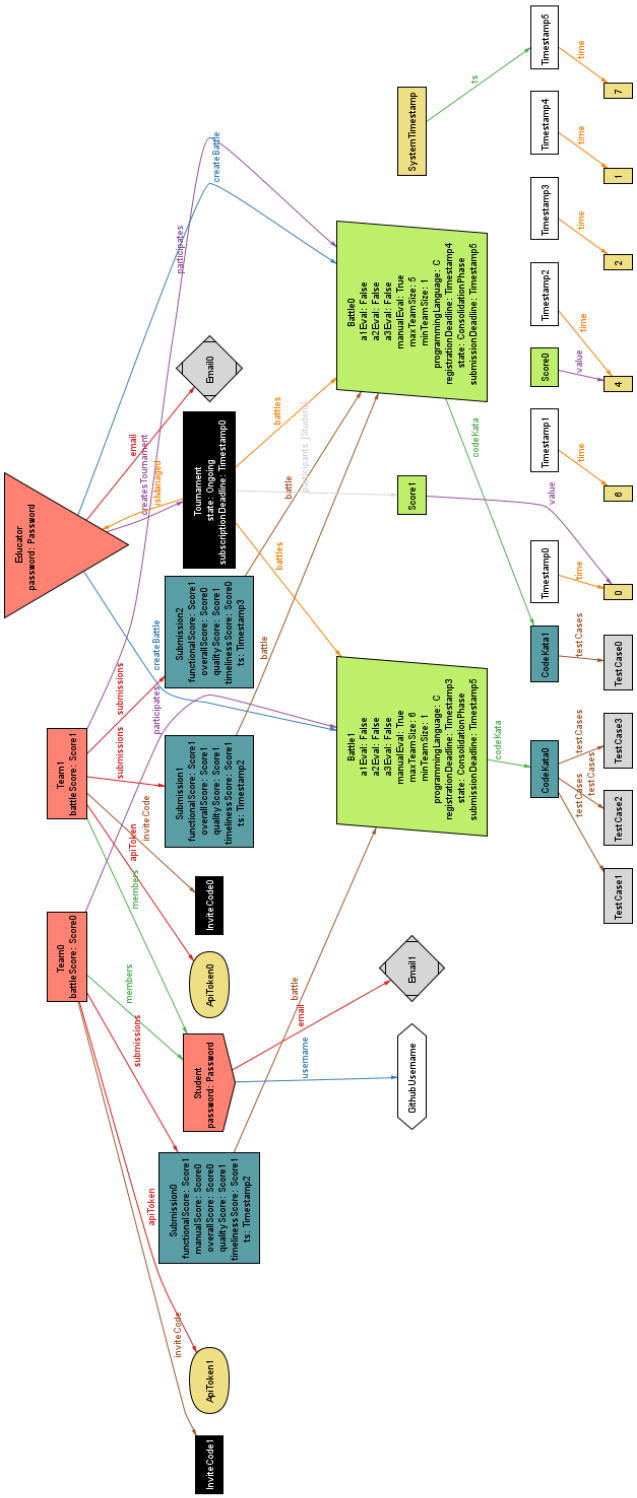

Figure 32: World2, generated with the *world2* predicate

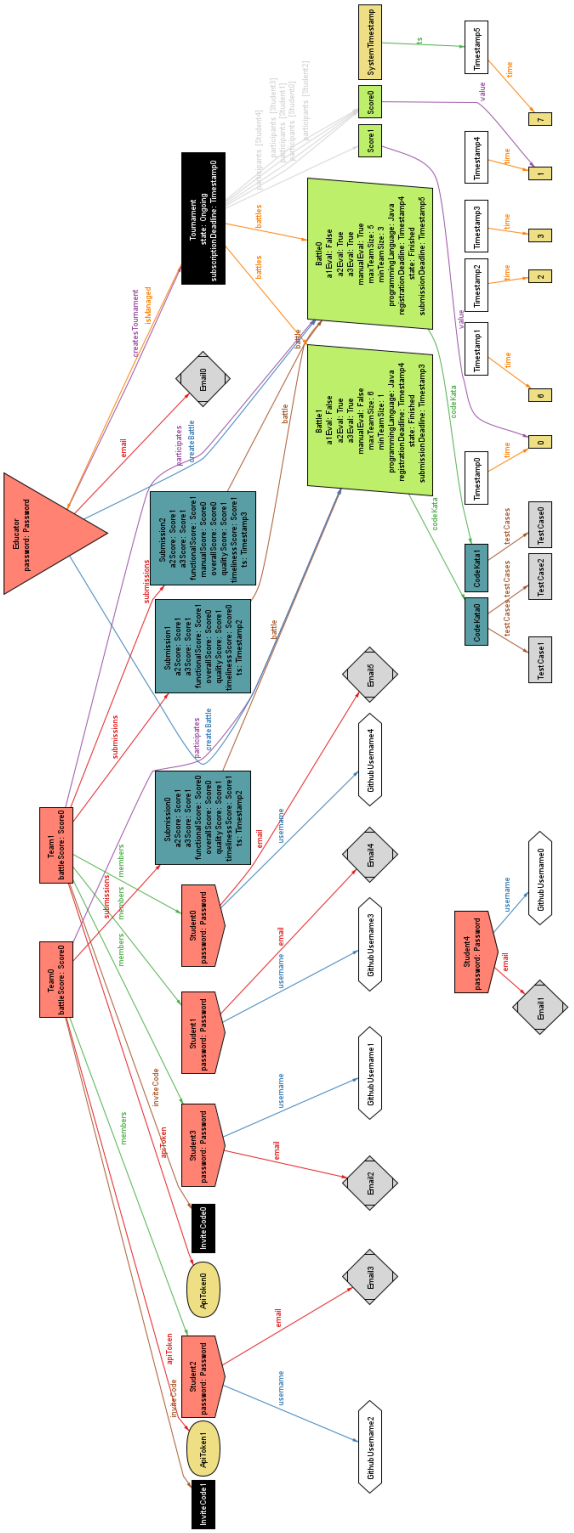

Figure 33: World2, generated with the *world2* predicate

5 Effort Spent

Name and Surname	Section 1	Section 2	Section 3	Section 4
Tommaso Capacci	11h	18h	20h	13h
Gabriele Ginestroni	11h	10h	24h	18h