AY 2023/2024

Politecnico di Milano

RASD: Requirement Analysis and Specification Document

Tommaso Capacci Gabriele Ginestroni

Professor
Elisabetta Di Nitto

Version 1.2
November 16, 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 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 programming challenges in teams**
  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 tournaments**
  Only educators are allowed to create tournaments, which are composed by a non predefined number of 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.

- **G3: Allow educators to create programming challenges**
  The creator of the tournament (or invited collaborators) can create programming challenges (Code Kata Battles) within the context of a tournament.

- **G4: Let the system to automatically analyze and rank submissions of students**
  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.

- **G5: Allow educators to manually inspect and review submissions of students**
  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.

- **G6: Allow students to track their performance**
  Teams will we 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, we can identify the application domains. The following table describes the world, shared and the machine phenomena, including the reference to which part controls the phenomena.

Phenomenon	Controlled	Shared
Teachers test their students	W	N
Evaluator decides to challenge his students	W	N
Student wants to improve his coding skills	W	N
Students may communicate with each other	W	N
Student invites other student to join his team	W	N
Student/educator subscribes to the platform	W	Y
Student/educator logins	W	Y
Educator creates a tournament	W	Y
Educator adds other 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
Student checks list of ongoing tournaments and ranks	W	Y
Student forks the GitHub repository of the code kata	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
Platform creates GitHub repository and sends the link to the students of the teams subscribed to the battle	M	Y
Platform sends notification 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	M	Y
System checks login data	M	N
System pulls new submission from GitHub	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) about specific topics of a subject. 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 collaborator:** person who is granted the access to a GitHub repository with write permission
- **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

- **CKB:** Code Kata Battles
- **API:** Application Programming Interface
- **UML:** Unified Modeling Language

1.3.3 Abbreviations

- **Gn:** Goal number "n"

- **Dn:** Domain Assumption number “n”
- **Rn:** Requirement number “n”

1.4 Revision History

- December 22, 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 costraints 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

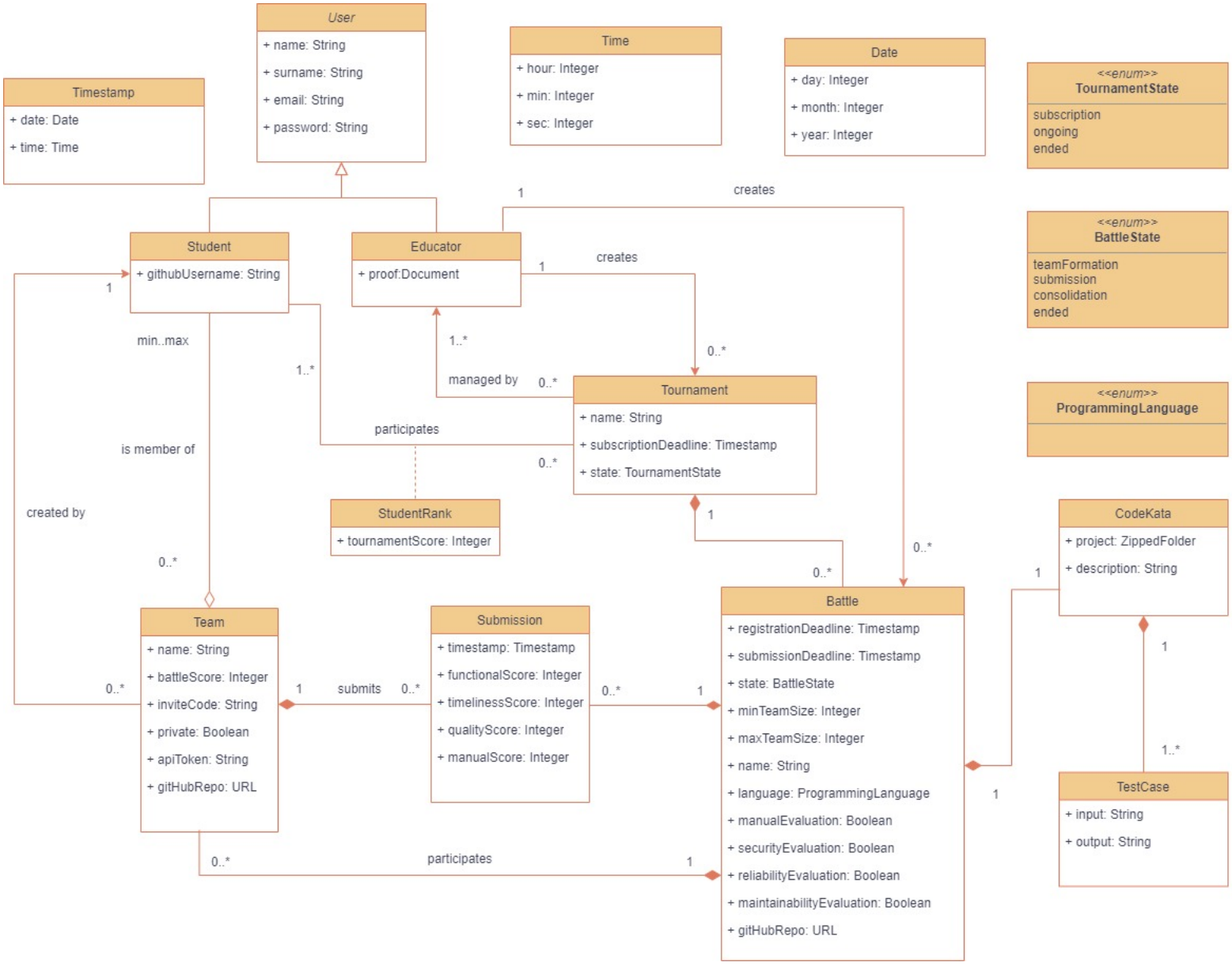

Figure 1: Domain-level UML diagram

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.

2.1.3 State diagrams

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

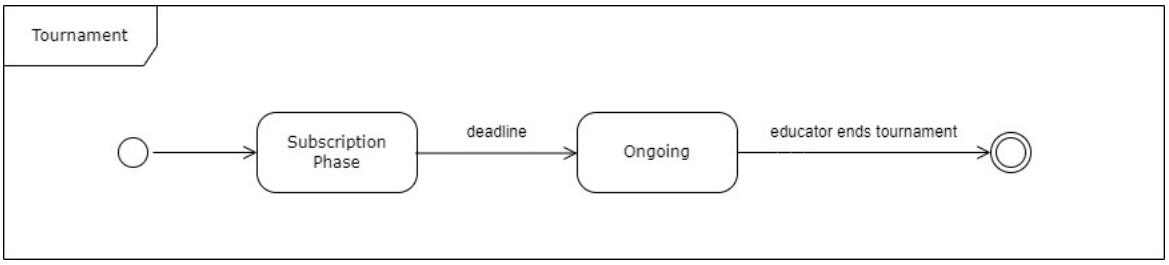

Figure 2: Tournament state diagram

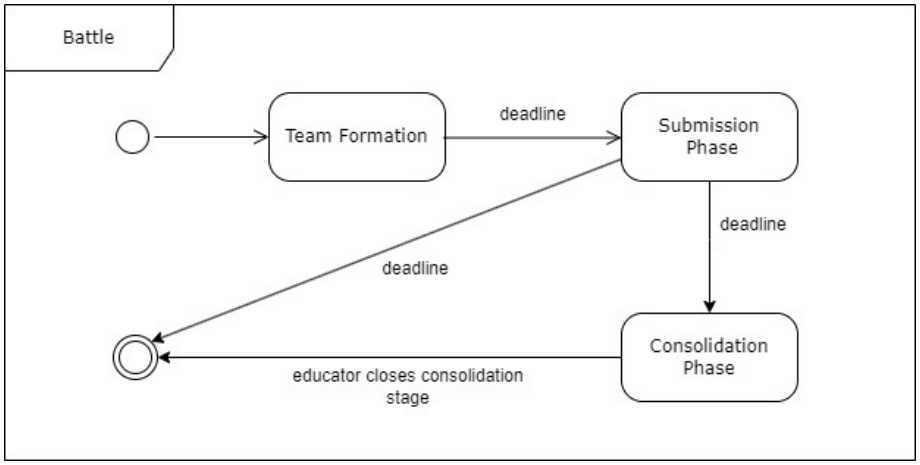

Figure 3: Battle state diagram

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.

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 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 an invitation, which they can accept or not). 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. Once the registration deadline is met, the system will prepare the development environment for each student: specifically, it will create a new public repository on the platform GitHub account and send the relative link to all team leaders, which will need to fork the repository, set up GitHub Actions and invite his/her teammates (if there are) as collaborators.

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 notification sent through the email.
- 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.

In this case the student that created the team will be marked as team leader (see "Create battles" functionality). 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 we must keep in mind that 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.

2.2.6 Submit a solution

Once the registration phase is over, the submission phase starts: the system provides each student/team a GitHub repository in which the solution code has to be collected (teams of students have to manually guarantee access to all needed collaborators). Then, GitHub Actions has to be set up in each of these repositories in order to enable the necessary workflow: in this way the submission of a new solution is performed by committing and pushing over the repository as we would normally use GitHub for. On push, an API call will notify the platform of the publication of a new solution for that student/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 highest score of each team, while the tournament ranks are computed considering, for each student, the sum of the scores assigned to its teams during battles 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

As we've already said many times, 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.

2.4 Assumptions, dependencies and constraints

2.4.1 Additional 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.

2.4.2 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:** Users are expected to engage in code kata battles with integrity, without resorting to plagiarism or cheating.
- **D6:** Maximum number of students per groups an educator can set will always be bounded (e.g. less than 4).
- **D7:** Students will fork the code kata GitHub repository once ready and invite only his teammates as collaborators.
- **D8:** Only one student per group will perform the steps needed to set-up the team's repository and the GitHub Actions workflow.
- **D9:** Every student has a GitHub account.
- **D10:** Students know how to setup a GitHub Actions worfkflow.
- **D11:** Static analysis tools are able to quantify the specified code quality aspects.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

TODO
The platform will have a web interface accessible via browser from both students and educators. Key interfaces will include account registration, log in page, tournament/battle creation, team creation/joining and rankings visualization.

3.1.2 Hardware Interfaces

The software does not require any other hardware interface different from normal personal computers 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 has to be able to interface with GitHub to create the original repository for each battle, which will then be forked by the different teams. 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.
- **Static analysis tools**: some of the aspects that have to be evaluated require the execution of some static analysis method over the provided code. In this case, the platform utilizes external services directly provided by third-party organizations.

Each of these interfaces are implemented by correctly setting and utilizing specific API calls to the involved services that will be included inside the platform's inner processes.

3.1.4 Communication Interfaces

From the moment that most of the system's services are provided through the platform's website and REST API calls, no other protocol than normal HTTP should be needed. Anyway, it is worth mentioning that 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 description of the tournament.
- **R6:** System shall allow educator (tournament creator) to set the subscription deadline of the tournament.
- **R7:** System shall notify all students enrolled to the platform about a newly created tournament.
- **R8:** System shall allow educator (tournament creator) to end the tournament.
- **R9:** System shall allow educator (tournament creator) to add another educator as collaborator to the tournament.
- **R10:** System shall allow educator (tournament creator or collaborator) to create a battle for the tournament.
- **R11:** System shall allow educator (tournament creator or collaborator) to upload the Code Kata of the battle.

- **R12:** System shall allow educator (battle creator) to set minimum and maximum number of students per group allowed for the battle.
- **R13:** System shall allow educator (battle creator) to set the registration deadline of the battle.
- **R14:** System shall allow educator (battle creator) to set the submission deadline of the battle.
- **R15:** System shall allow educator (battle creator) to enable/disable optional manual evaluation for the battle.
- **R16:** System shall allow educator (battle creator) to select relevant aspects of quality level to be extracted through static analysis.
- **R17:** System shall notify all users enrolled to a tournament about a newly created battle.
- **R18:** System shall integrate with external static analysis tools through proper API calls.
- **R19:** System shall create the GitHub repository of a battle as soon as the registration phase closes.
- **R20:** 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 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.
- **R22:** System shall compute a final score combining automatic and manual score (if available) for each team after the consolidation stage if expected or after the submission deadline otherwise.
- **R23:** System shall update the battle teams' ranks after a new score is available.
- **R24:** System shall notify all students participating to the battle about the availability of the final battle rank.

- **R25:** 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.
- **R26:** System shall notify all involved users about the availability of the tournament student's final rank.
- **R27:** System shall allow all users to see the list of ongoing tournaments.
- **R28:** System shall allow all users to see all tournaments' ranks.
- **R29:** System shall allow all users subscribed to the same battle the relative battle's rank.
- **R30:** 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.
- **R31:** System shall allow student to enroll to a tournament within the subscription deadline.
- **R32:** System shall allow student to create a team within the registration deadline.
- **R33:** System shall allow student (team creator) to set the team name.
- **R34:** System shall allow student (team creator) to set the privacy of the group.
- **R35:** System shall allow student to join a team *only* before the registration deadline.
- **R36:** System shall allow *only* students who know the correct invite code to join private groups.
- **R37:** System shall allow team members to set their the repository URL.
- **R38:** System shall generate a unique API token for each team.
- **R39:** System should offer an external API which, when invoked, will notify the platform about a new commit on a team's GitHub repository.

3.2.1 Use cases

3.2.2 Requirements mapping

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, we describe different performance requirements for different aspects:

- *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.

4 Formal Analysis Using Alloy

Organize this section according to the rules defined in the project description.

5 Effort Spent

Provide here information about how much effort each group member spent in working at this document. We would appreciate details here.

References