Requirement Analysis and Specification Document

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	YOUR NAMES
Version:	1.0
Date:	31-January-2016
Download page:	LINK TO YOUR REPOSITORY
Copyright:	Copyright © 2017, YOUR NAMES – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

In today's digital age, education has evolved significantly, with online learning becoming more and more important, especially in subjects like computer science. However, only theoretical learning often can be insufficient in practical fields like coding. This is where the CodeKataBattle (CKB) project steps in, addressing the need for a hands-on, collaborative, and engaging approach to learning programming and software development.

Although theoretical knowledge is essential in computer science, it is the practical application and coding practice that strengthen students' understanding and skills. Hence, CKB is designed to support students with a platform where they can actively practice coding, experiment, learn from their mistakes, and improve their skills. Moreover, it encourages students to work together in teams, promoting teamwork and communication, essential qualities in the software development field. For educators, CKB offers a valuable teaching tool, enabling them to create coding challenges in accordance with their curricula, making learning more engaging, and helping assess students' progress.

In summary, the CodeKataBattle project serves as a bridge between theoretical learning and practical application in computer science education. It offers students a platform to practice coding, collaborate, and improve their skills, and provides educators with effective teaching tools and assessment methods. The aim of CKB is to make computer science education more engaging, practical, and rewarding for both students and educators.

1.1.1 Goals

1. Students are able to find, join, or create teams for specific code kata battles within a tournament.

2. Educators can create and manage code kata battles, including the provision of detailed descriptions, project templates with test cases, and build automation scripts.

3. Educators have the capability to set up and customize various battle parameters such as team size limits, registration deadlines, submission deadlines, and scoring configurations.

4. Students commit their code implementations to a specific GitHub repository for each battle, triggering automated testing and scoring through the platform.

5. The platform automatically calculates and updates battle scores based on predefined criteria including the number of passed tests, commit timeliness, and code quality.

6. Educators can manually score submissions, considering aspects outside of automated evaluations.

7. Students and educators can view the evolving ranks and scores of teams during a battle.

8. The final battle ranks are determined after considering both automated and manual evaluations, and made visible to all participants upon completion of each battle.

9. Cumulative personal tournament scores are updated and displayed for each student, contributing to an overall tournament ranking.

10. The platform notifies students of new tournaments, battle opportunities, and updates on their tournament standings.

11. Students can view their rankings on their profiles.

1.2 Scope

1.2.1 Product Identification

- The **target audience** of the platform is mainly **students and educators** in computer science and related fields.

- CKB is an **interactive web-based coding platform** specifically designed to improve the coding skills of students by participating in battles designed by educators. It is mainly an **educational tool** where practice can be applied.

- CKB is also a **competitive learning environment**. By organizing the coding exercises in battle format, the students are expected to be more motivated to perform their best in challenges.

- The platform **promotes teamwork and collaboration** by allowing students to form groups to tackle battles together. This reflects real-world software development scenarios where collaborative team dynamics are key.

- The platform has an **educator-centric control**. The educators play a crucial role in the platform because they are the ones creating, managing, and scoring the battles and tournaments.

- CKB enables students to **use professional tools and practices** such as version control, and fork-pull-push mechanisms so that they can enhance their real-world software development skills with the help of Github.

- The platform makes use of **automated testing** to assess student submissions, ensuring objective evaluation of functional correctness and code quality. Additionally, it allows educators to perform manual evaluations for aspects that require subjective judgment.

- CKB is designed to provide **instant feedback** on submissions and **real-time updates** of team scores and rankings.

- The platform offers **flexibility** to educators in terms of the choice of programming languages, difficulty levels of challenges, and the scope of coding tasks, making it adaptable to various learning curves and educational needs.

1.2.2 Domain Analysis

The CKB platform operates within the domain of educational technology, specifically tailored for coding and software development education. From this perspective, we have the following users:

Educators: Educators use CKB to **create and manage coding exercises**, known as code katas, in a battle format. They have the capability to set parameters for these exercises, including difficulty levels, deadlines, and specific technical requirements. Educators can **evaluate student submissions** both automatically (using the platform's tools) and manually, **providing feedback and scores**.

Students: Students exploit the platform to enhance their coding skills by **participating** in these code battles. They work individually or in teams to solve the challenges set by educators, fostering both individual and collaborative learning experiences. Students use the platform **to submit their code, receive real-time feedback, and track their progress in coding proficiency**.

In summary, the domain of the CKB platform is focused on interactive, competitive coding education, enabling students to have an engaging learning experience while offering educators powerful tools for managing and evaluating coding exercises.

1.2.3 World Phenomena

1. Educators prepare the necessary documents for code battles including descriptions, test cases and build automation scripts.

2. Students fork the GitHub repository.

3. Students set up an automated workflow through GitHub Actions.

4. Students write code on their devices.

1.2.4 Shared Phenomena

World Controlled

1. Educators upload coding challenges, including descriptions and test cases, to the platform.

2. Educators create tournaments and give permission to their colleagues to create battles for that tournament.

3. Educators set specific rules and criteria such as deadlines, number of team members, and additional configurations for scoring for code kata battles.

4. Educators decide on the inclusion of manual scoring components for battles.

5. Students invite peers to form teams within the platform or join individually.

6. Students submit their code solutions by pushing the code via GitHub.

Machine Controlled

7. The CKB platform sends notifications to students about new battles and tournaments, final battle ranks and final tournament ranks.

8. The platform generates and manages GitHub repositories for each battle and sends links to students that are participating.

9. The CKB platform automatically evaluates code submissions against test cases.

10. The platform updates battle scores and battle rankings in real-time.

11. The platform displays leaderboards (i.e. the rank of the sum of all battle scores received in that tournament).

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Student**: An individual enrolled in an educational program or course who uses the platform to participate in coding exercises and improve software development skills.

- **Educator**: A person, such as a teacher or an instructor, responsible for creating coding challenges and managing learning activities on the platform.

- **Automated Testing**: A process where the CKB platform automatically executes predefined tests on student code submissions to assess their functionality and correctness without manual intervention.

- **Manual Scoring**: The process where educators evaluate student code submissions subjectively, complementing the automated testing system.

- **Battle**: A competitive coding challenge on the platform where students or teams of students solve specific programming problems within set parameters and time frames.

- **Tournament**: A series of code kata battles organized and managed by educators on the CKB platform, that ranks students or teams based on cumulative scores from individual battles.

- **Ranking**: A system within the CKB platform that orders participating students or teams based on their performance in individual code kata battles, determined by scores from automated and manual evaluations.

- **Leaderboard**: A feature on CKB that displays the standings of students or teams based on their performance overall in a tournament.

1.3.2 Acronyms

- **CKB**: CodeKataBattle

1.3.3 Abbreviations

- G_x: x-th Goal

- WP_x: x-th World Phenomena

- SP_x: x-th Shared Phenomena

2 Overall Description

2.1 Product Perspective

2.1.1 Scenarios

1. **Educator creates a tournament**
   Professor Emanuelle, a computer science educator giving an "Introduction to Programming" course in Politecnico di Milano, decides to improve her student's understanding of programming through coding exercises. She is already registered in CodeKataBattle as an educator, so she logs in to the platform with his credentials, which are email and password. Then, she clicks the "Create Tournament" and a form screen is prompted. She fills out the form respectively:
   - entering tournament title, "Winter Semester Coding Exercise 1",
   - adding a description of the tournament,
   - choosing a deadline for the subscription. After clicking the "Next" button
   Professor Emanuelle is asked to select colleagues for permission to create battles in this tournament. She selects 2 of her colleagues because they all together gives the same lecture. Finally, she clicks on the "Create" button, and tournament is created. She is directed to the "My tournaments" section after creation.

2. **Student Registers for tournament**
   Emre is a Bachelor's student in Computer Science at Politecnico di Milano. At the start of the winter semester, he registered CodeKataBattle as a requirement of his "Introduction to Programming" course. In the middle of the semester, he gets an email notification about a tournament created by Professor Emanuelle on the platform. Then, He clicks the link in the email. Because he has already logged in to the platform with his email and password, he is redirected to the tournament's page, where he finds detailed information about Emanuelle's tournament. After reading the description, he thinks that this tournament will be very helpful for him to understand the topics covered in the lecture. So he clicks to the "Register" button to enroll tournament. Some registration details consisting of descriptions, educators, and tournament creator are shown on the screen. Also, he is informed that he will get email notifications about upcoming battles, battle rankings and tournament rankings. Finally, Emre clicks on the "Accept and Register" button and registers the tournament.

3. **Educator Sets Up a Battle**
   A week ago Professor Mottola, was invited to a tournament created by his colleague in the CodeKataBattle platform. He accepted the invitation and learned about the tournament from its description. He also gives the "Introduction to Programming" course and wants to create a new exercise about the topic he showed in the last lecture, which is "Recursion". After the subscription deadline passed, he logs back into CKB, he selects the "Create Battle" option within the "Winter Semester Coding Exercise 1" tournament. He is presented with a detailed form where he inputs the battle's title, "Check if String is Palindrome" and provides a thorough description that includes the battle's focus on recursion, expected coding languages (Java and Python, both accepted), and software project including necessary scripts for build automation (Gradle for Java, no need for Python) and test cases. To encourage collaboration, he sets the minimum and maximum group size to three and five students, respectively. he then specifies the registration deadline, two weeks from the current date, and a final submission deadline, giving students a month to work on their solutions. Finally, he configures additional scoring parameters as **efficiency**, choosing from a list of aspects including reliability, maintainability, etc. Then he clicks on the "Create" button and he is redirected to battle main page illustrating information about the battle. After a couple of minutes, he got an email saying that an email about this battle was sent to all students in the tournament.

4. **Student Joins for Battle**

   Samet got a notification from the tournament he enrolled in saying that Professor Mottola has created a battle with the name "Check if String is Palindrome". He clicks on the link and reads the description carefully. He clicks on the "Register" button then a screen is shown asking Samet whether he wants to invite other students to form a team or not. Samet selects "Yes", then, he names her team 'Code Warriors' as a first step. Then he chooses students from a list of students registered for the tournament, considering minimum and maximum group size. Samet knows Emre, Jack, and Luca also registered for the tournament, so he sends them invitations. After clicking "Complete", he is redirected to the battle information page. On this page, there is a section showing group status, which is pending until all invitations are answered. After a while, Samet sees that Emre and Luca accepted but Jack rejected it. Because the minimum group size is fulfilled, Samet clicks on the "Finalize" button indicating the final decision. Thanks to the help of instructions during the process, Samet understands that if he wants to decline registration, he should have clicked on the "Decline" button. After every member accepts the invitation to finalize registration, the process ends and the status becomes "Registered". They are given a competitor id.

5. **Students Sets Up Environment for CodeKataBattle**

   With the 'Code Warriors' team formed and the registration deadline for the "Check if String is Palindrome" battle passed, the CKB platform takes its next automated step. It creates a unique GitHub repository for the battle, containing the provided code kata with its test cases and build scripts. The system then sends an email to Emre, Samet, and Luca with the repository link and instructions. The team members collaboratively decide to schedule a virtual meeting to set up their working environment. During the meeting, they fork the repository to their group account and set up GitHub Actions in order to make proper API calls with the competitor id they have. This setup is crucial for automating their workflow and ensuring that every code push not only updates their repository but also notifies the CKB platform. They test the setup by pushing a minor change, and upon seeing that the CKB platform acknowledges their commit, they know their system is correctly configured. This marks the start of their coding journey in the battle.

6. **Students Solve Battle Challenge**

   After forking the project and setting environment, they started to think about the solution of the project. As they understand from the description they should return "True" or "False" regarding whether the given string is a palindrome. They implement the algorithm and commit/push the code to the repository. Each push prompts the platform to pull the latest code, run tests, and analyze the quality of their solutions using static analysis tools. After a while, they revisit the ranking on the battle's page to see their score of the last push. They see their scores in 3 different categories:
   - functional aspects: 32/80 (4/10 test cases passed)
   - timeliness: 5/5
   - quality level of the sources: 8/10 (Aspects: Efficiency)
   In the rankings, they are informed via tooltips about the scaling of their scores and the calculation methods of them. So they decided to focus more on code and try to find why some test cases are not passed. The team keeps an eye on the CKB dashboard, which updates their battle score after each commit. They note improvements in their score as they refine their solutions, ensuring more test cases pass and optimizing their code for better quality. This iterative process of coding, committing, and refining continues, with the team members frequently discussing strategies and sharing insights to improve their solutions. After looking at some exercises related to recursions they finally find the wrong part in the algorithm they implemented. After changes they commit and push the code. They see in the rankings that they got 93 points from battle. This iterative process of coding, committing, and refining continues, with the team members frequently discussing strategies and sharing insights to improve their solutions. They think that this is the most efficient

algorithm they can implement. So they decide not to do anything else until code kata battle deadline.

7. **Educator Evaluates Submission Manually**

After the submission deadline expires, there is a consolidation stage enabling educators to assign optional points for teams. Professor Mottola begins her manual evaluation of the submissions for the "Check if String is Palindrome" battle. He logs into CKB and accesses the educator's dashboard from the battle's page, where he can review each team's submission. On this page code of the submission and other materials from the repository are shown to the educator. He starts from the latest submission and gives extra points if he finds the solution by analyzing 'Code Warriors', impressed by their timely submission and high score, viewable in the rankings table. Professor Mottola examines their code, focusing on the efficiency of their solution, which got 8/10 from the analysis tool. Considering these factors, she awards them a high personal score. This score reflects her assessment of their problem-solving skills and coding proficiency, adding a crucial human element to the automated evaluation.

8. **Educators Analyze Tournament Leaderboard**

Professor Emanuelle have created a tournament called "Winter Semester Coding Exercise 1" at the beginning of the semester. Some other colleagues of hers registered for the tournament as educators and created koda battles during the semester. During the semester, she has viewed the leaderboard and at the end of every week, she has exported the result. Eventually, at the end of the semester, Professor Emanuelle closes the tournament. She then goes to the leaderboard screen and starts to see the general success of her students by comparing leaderboard exports from the very first battle. In this way, she notices that the leaderboard becomes more competitive as we get closer to the end of the year. She interprets this situation as students improving their programming skills battle by battle. After analyzing the leaderboard she searched other ongoing tournaments of other professors from the university to see the success of their students in different tournaments created by teachers of different courses.

2.1.2 Domain Class Diagram

In the figure below, the Domain Class Diagram for the CodeKataBattle (CKB) platform is illustrated. This diagram is a crucial element in understanding the structure of the classes and relationships within the CKB system. It visually represents the key classes, their attributes, and the interactions among them in the context of the platform. This diagram is designed to provide a clear and concise overview of the domain model, offering insights into how the different components of the CKB platform interact to facilitate a competitive and educational coding environment.
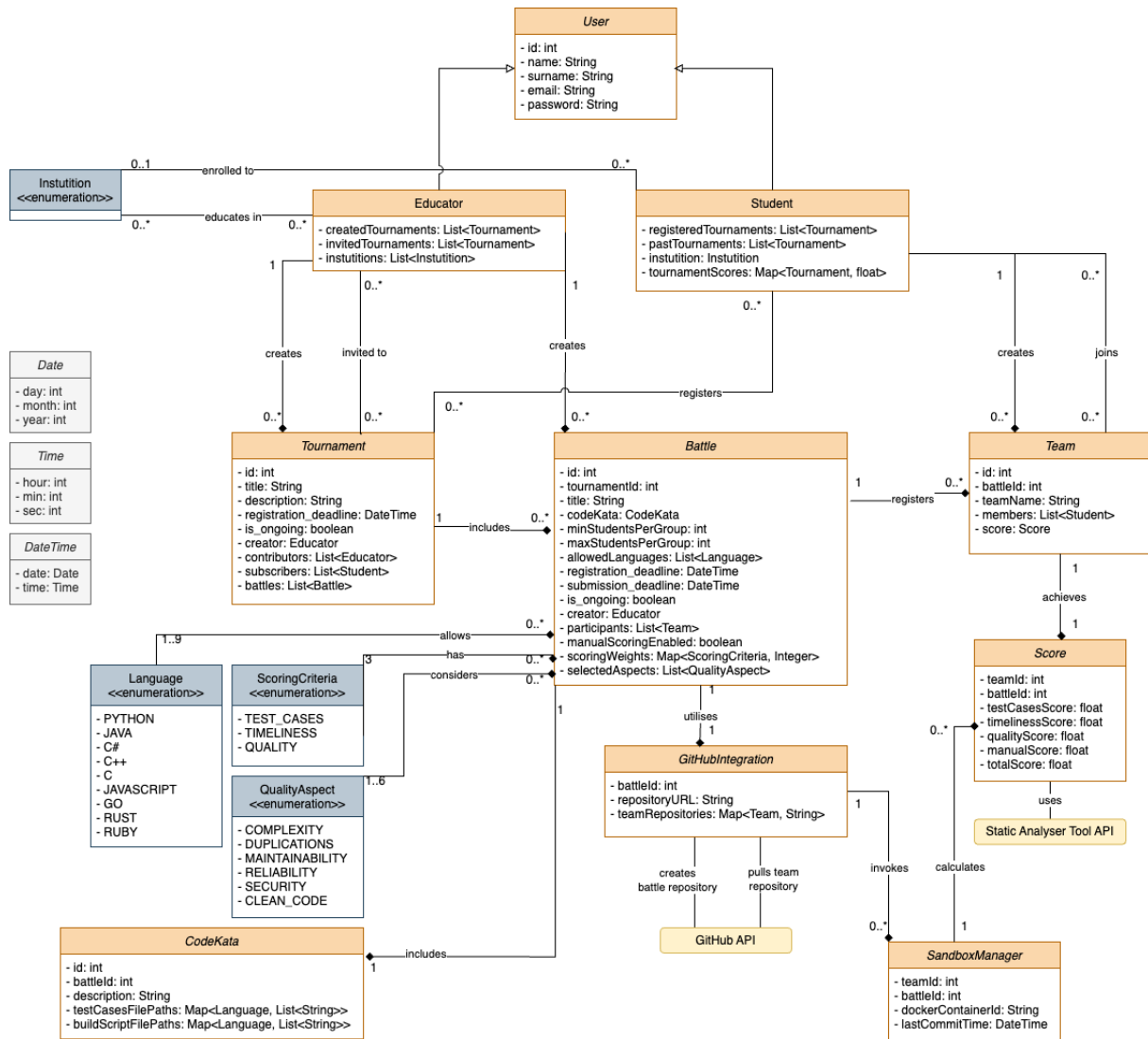
The descriptions below are the explanations of the important classes that take place in the diagram:

- **User:** Serves as a base for different types of users. It encapsulates common attributes that are shared across students and educators. This class is essential for authentication.

- **Educator:** The objects of this class have the ability to create tournaments and battles. They can also be invited to a tournament by the creator of that tournament.

- **Students:** The objects of this class have the ability to register for the tournaments, and they can create teams for battles or join teams that they are invited to.

- **Tournament:** Tournaments are a vital part of the project. They are created by educators and they include coding battles for teams of students to engage and solve.

- **Battle:** Battles are created by educators who take part in a tournament. During battle creation, educators set some of the fields of the battle object.

  - **Scoring Criteria:** There exists 3 scoring criteria whose percentages are decided by the educator during battle creation.
  - **Language:** There are 9 languages allowed in the system. The educator chooses the allowed languages during battle creation. At least one language must be chosen.
  - **Quality Aspect:** There are 6 quality aspects that can be selected by the educator during battle creation time. At least one quality aspect must be chosen.
  - **CodeKata:** It is the most important part of the battle. The test cases and the build scripts for each allowed language should be uploaded by the educator to the system. The paths to the files in the file system are stored in the fields.

- **Team:** Teams are created by students, and the teams are able to register for the battles. They have a score for the battle.

- **GitHub Integration:** This class has one object for each battle and it stores the link to the GitHub repository for the battle after creating it by interacting with the GitHub API. It also stores the links to the repositories of the participating teams after they push their code and trigger a function that belongs to this class.

- **Sandbox Manager:** There exists a Sandbox Manager for each team in the corresponding battle. If there exists a docker container (the team previously pushed a code and their GitHub actions already communicated with our RESTful APIs), the same container is called, and the code is run on that container. If there is not an existing container (the team pushes code for the first time), a new container is created and assigned to the team for the rest of the battle. The Sandbox Manager interacts with the Score class after running the code to calculate the score of the team after each push/commit.

- **Score:** The score of a team for each criterion is calculated for the battle. This class uses an external Static Analyser Tool to calculate the score of the quality aspect of the code. The timeliness and test case scores are handled internally.

2.1.3 Statecharts

The state charts presented in this section elucidate the operation of the CodeKataBattle (CKB) project, illustrating the state transitions for a code kata battle and a tournament within the platform. Especially for tournaments, a state chart is very crucial to gain a better understanding of the tournament.

Tournament
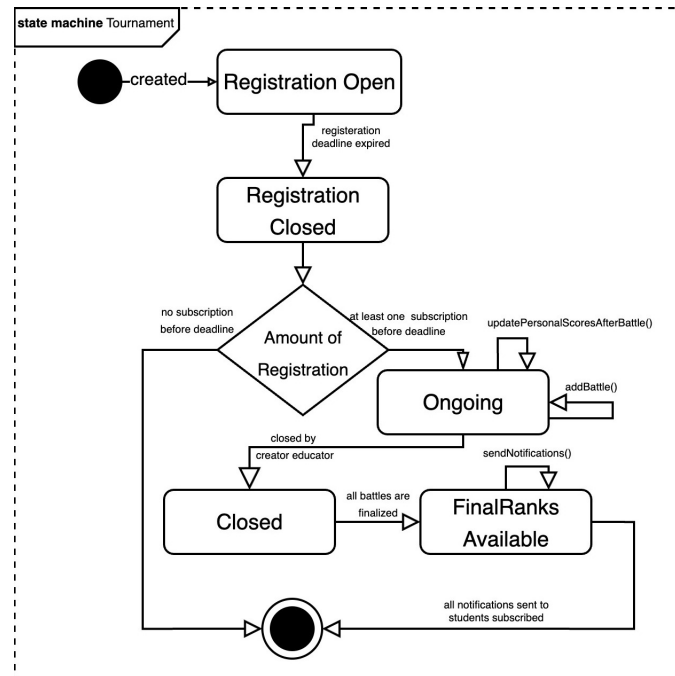
Firstly, the tournament is in the very first state called "Registration Open", which indicates that Tournament is created and registration is open for students and invited educators. Until the registration deadline, they can register. When the deadline expires registration is closed, so the tournament goes into the "Registration Closed" state. In this state, the process controls the amount of registration for the tournament. If there is no registration it goes to "Final" state. Otherwise, the tournament starts with the "Ongoing" state. In this state, throughout the tournament, educators can create battles, and personal scores are updated after every battle. If the creator educator closes the tournament, it goes to the "Closed" state. However, there can be some battles in the consolidation stages and it is mandatory to finalize them to calculate final ranks. After every battle is finalized, the tournament transitions to the "FinalRankAvailable" state in which notifications about final rankings are sent to the students. After finishing this notification process tournament reaches the "Final" state. There won't be any rank changes or notifications hereafter.

Battle
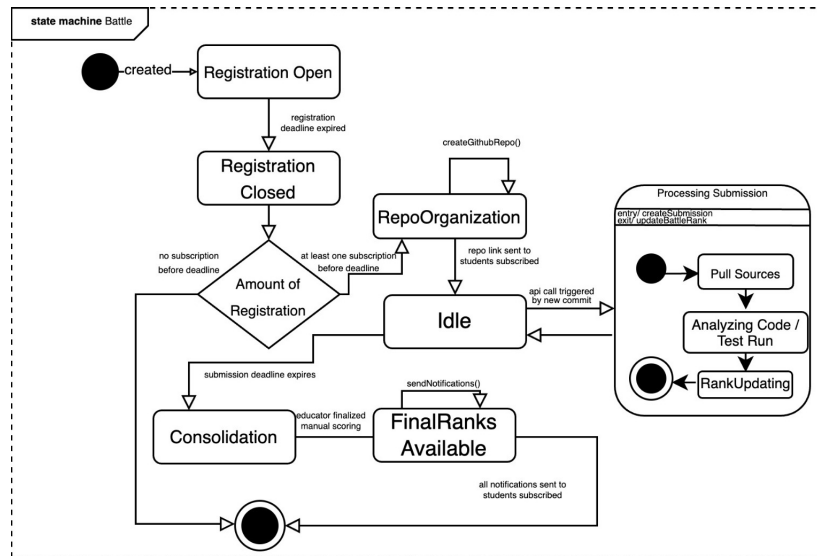
Firstly tournament is created with the "Registration Open" state. In this state, students can register to battle with teams or individually. When the deadline expires, Battle goes into the "Registration Closed" state. At this time, it is calculated that if there is any registration for battle. If no, it goes to the "Final" state with no registration and any other action. If there are registrations, it goes to a new state called as RepoOrganization in which GitHub repository for the battle is created with proper instructions. Sending the link and instructions triggers the process to go into the "Idle" state. This means that Battle has started and waits for API calls from forked repositories, triggered by a new commit. An API call from repositories of competitor leads to go into "Processing Submission. Here with the "createSubmision" input 3 sequential states follow each other, respectively, "Pull Sources", "Analyzing Code / Test Run" and "RankUpdating". In this 3 state code is pulled and analyzed. Then, the score and rankings are updated. Eventually, it returns to the "Idle" state. Until the submission deadline, students' commit can change their scores. When the deadline expires, Battle goes into the "Consolidation Stage" state in which educators can change rankings via manual scorings. When they are finalized, Battle is closed, indicating as state "FinalRanksAvailable". Immediately, the notification sending process starts. After all notifications are sent, Battle goes into the Final state.

2.2 Product Functions

The major functions of the project will be organised according to the user types. We will have three sections, first for the common function both regarding the educators and the students. The second and the third will be specific to each user type respectively.

2.2.1 Common Functions

- **Account Registration:** Both students and educators can be registered to the platform. During registration, the user indicated which type of user they are along with their credentials.

- **Viewing Leaderboards and Rankings:** Both user types can view leaderboards that display rankings of participants based on performance in battles and overall tournament standings.

2.2.2 Educator Functions

- **Creating a Tournament:** Educators can create and configure tournaments, setting specific parameters such as time constraints. In these tournaments, multiple battles can take place.

- **Creating a Battle:** Educators can create and configure coding battles in the tournament that they have created or they have been permitted by their fellow educators. During battle creation, specific parameters such as coding languages, and team sizes are set by the educator. On top of that, the description of the battle and the test file including test cases, automated build scripts or manual evaluation choice should be provided.

- **Giving Permission to Other Educators:** This function can be utilised both in the creation of a tournament and during the management of a tournament. Educators are able to permit fellow educators to create battles in the tournaments that they have created.

- **Scoring:** This function is vital since it is used to compute the points of the students' submissions in the battles. The output of this method determines the ranking of the students. Both automated and manual scoring reply to this function call.

2.2.3 Student Functions

- **Join a Tournament:** Students can join tournaments that the educators are created.

- **Join a Battle:** Students can join battles which is a part of the tournament they already joined. When joining the battle, students can send requests to other students to form a team if they want or need to.

- **Send Request for Teaming:** This function can be called during students join the battle. Students send requests to other students using their usernames, and if they accept the invitation, they are now a part of the team for the battle they are joining.

- **Submit Solution:** Students submit their solutions for each code battle. After submission successfully completed, the scoring function is called.

2.3   User Characteristics

2.3.1   Educators

A. **Profile**

- Typically instructors, professors, or teachers in computer science or related fields, with varying levels of experience in coding and software development education.

B. **Needs**

- Tools for creating, managing, and monitoring coding challenges and tournaments.
- Flexibility in setting parameters for battles, including team sizes, deadlines, and testing criteria.
- Ability to assess student work both automatically or manually.
- Resources to track student progress and engagement in coding exercises.

C. **Interactions**

- Making students engage with battles for educational purposes.
- Collaboration with other educators within or across institutions.

2.3.2   Students

A. **Profile**

- Individuals enrolled in computer science and similar courses or those seeking to enhance their coding skills, ranging from beginners to advanced levels.

B. **Needs**

- Access to a variety of coding challenges that help to improve the student's skills.
- Tools for collaborative coding and team formation.
- Real-time feedback on code submissions for iterative learning.
- Opportunities to apply theoretical knowledge in practical scenarios.

C. **Interactions**

- Active participation in coding battles and tournaments.
- Collaboration with peers for team-based challenges.
- Utilisation of platform resources for self-directed learning and improvement.

2.4 Assumptions, Dependencies, and Constraints

2.4.1 Domain Assumptions

1. Educators and students have basic proficiency in using web-based platforms and are familiar with basic operations such as account creation, logging in, and navigating through a digital interface.

2. Educators have the necessary skills to create and manage coding challenges, including the ability to correctly write problem descriptions, test cases, and understand code quality metrics.

3. The coding problems and challenges provided by educators are free from errors and ambiguities.

4. Students have at least foundational knowledge in programming and can understand and respond to coding challenges.

5. Students' submissions to the platform are their original work.

6. The users have access to reliable internet connectivity and devices capable of supporting the web-based CKB platform.

7. Students have familiarity with GitHub operations such as forking a repository, setting up GitHub Actions, and committing and pushing their codes.

8. The automated testing and scoring systems within the CKB platform are trusted by users to fairly and accurately assess coding submissions. Similarly, educators are trusted in the case of manual evaluation of submissions.

2.4.2 Dependencies

- **GitHub Services:** The system's functionality for code submission and automated workflow relies on the availability and reliability of GitHub services.

- **Internet Connectivity:** Access to the internet is essential for both users.

- **Web Browser Compatibility:** The platform's user interface and features depend on their compatibility with various web browsers, ensuring all users can access and use the platform effectively.

- **Hosting Services:** Dependence on reliable hosting services (like cloud-based servers) for hosting the web application, database, and related services.

- **Notification Service:** The platform depends on an external notification service to send timely alerts and updates to students and educators.

- **Email Verification Service:** The platform relies on an external email service provider to send verification codes to users during the account registration process.

2.4.3 Constraints

- **Technological Constraints:** Compatibility requirements with various web browsers and devices may limit certain design or feature implementations.

- **Resource Constraints:** Hosting and operational costs may limit the extent of scalability and redundancy features of the platform.

- **Security and Privacy Constraints:** The system must adhere to strict data protection and privacy regulations, such as GDPR, which may limit certain data collection and processing activities.

- **Legal and Compliance Constraints:** Compliance with intellectual property laws, particularly in the use and distribution of coding challenges and educational content, i.e. the rights of the contents are reserved.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.2 Hardware Interfaces

The CodeKataBattle platform operates entirely through web interfaces, eliminating the need for specialized hardware interfaces. Users can access the platform using standard web-enabled devices, such as computers, tablets, and smartphones. The platform is designed to function within a web browser, requiring no specific hardware beyond a device capable of running a modern browser and accessing the internet. This approach ensures broad accessibility without necessitating particular hardware configurations, making the platform versatile and user-friendly across various hardware setups. However, for optimal experience, at least 2GB of RAM and a minimum of 1GB of free space are recommended for browser caching and temporary files.

3.1.3 Software Interfaces

Our platform CKB interfaces with various software systems:

- **Web Browsers:** Platform operates on web browsers. For universal access and easy use, it is compatible with major browsers like Chrome, Edge, Safari, Opera, and Firefox.

- **GitHub API:** Integrated for creating coding repositories and managing code *submission* processes.

- **Sandboxing:** To create an isolated testing environment, our Sandbox Manager will utilise Docker.

- **Static Analysis Tool:** SonarQube API will be integrated for scoring the quality aspect of the code with static analysis.

- **Database Technology:** Platform uses databases such as PostgreSQL.

- **Hosting:** Our platform uses Amazon Web Services EC2 for web server and database hosting.

- **Cloud File Storage:** The files will be stored using Amazon S3 services.

- **Email Service:** Amazon Simple Email Service SES will be used to send automated email notifications.

3.1.4 Communication Interfaces

Our platform utilises various communication systems:

- **HTTPS:** For secure connection over the internet.

- **RESTful APIs:** We have RESTful APIs to communicate with the requests coming from GitHub Actions and the web application.

- **SMTP:** It is used to manage to send automated emails.

3.2   Functional Requirements

3.2.1   Use Case Diagrams

Unregistered User Actor has been used to illustrate the Registration Use Case. Similarly, the Registered User Actor has been used to show the Authentication Use Case. Other user actors are assumed to be registered and authenticated.

Our system is composed of two main user types: Student and Educator. In the use case diagram, we showed common use cases with the Authenticated User actor. Authenticated User, simply every user registered to the platform, can view Tournaments, Battles, and their rankings with different filters and search options. Moreover, Registered User has the capability of viewing their profile and editing settings.

Use case related Educator and Student Actors summarizes the goal of these users in the system.

Invitee Student and Invitee Educator Actors are used to demonstrate use cases for tournament(for educators) and team(for students) invitations.

GitHub System Boundary contains use cases about code submission with the participation of GitHub API Actor.

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