Software Engineering 2 - Prof. Matteo Camilli
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

CKB - CodeKataBattle

RASD
Requirement Analysis and Specification Document

December 22, 2023

Emanuele Pocelli (10726303)
Fabrizio Sordetti (10730069)
Andrea Varesi (10724377)

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

The system has, as main function, to deliver a platform where Students can challenge themselves, alone or in team, writing code to solve exercises assigned by Educators, to improve their software development skills. Students have to follow a test-first approach. The platform is used by Educators to create, manage and close tournaments, each one is composed by battles, while Students have to solve the exercises published for every battle filling them with their own code. Each exercise includes a brief textual description and a software project with build automation scripts that contains a set of test cases that the program must pass, but without the program implementation.
CKB offres also the possibility for each team to know their rank (calculated automatically by the platform) within the context of each battle, and the general rank of that tournament, in addition to a set of badges the Students can achieve, to increase the level of challenge.

1.1.1 Goals

[$G1$] Educator can manage (create battles and badges, grant permission to other Educators, and close) each tournament he/she creates.
[$G2$] Educator can manage (deliver exercise, set options, evaluate teams submissions and check rank) each battle he/she creates.
[$G3$] Student can subscribe to a tournament.
[$G4$] Student can manage (subscribe and check rank) their participation to a battle.
[$G5$] Both educators and students can check rank for each tournament.
[$G6$] Both educators and students can visualize the profile of a Student.
[$G7$] Both educators and students can see the list of ongoing tournaments.

1.2 Scope

In recent years, it has gained more and more ground the possibility to train and improve writing code skills through platforms, such as CodeKataBattle, that assign to people who want to challenge themselves exercises to fulfill autonomously, in a programming language of choice.
The philosophy behind these platforms is called *codekata*, a method that aims to refine people's software development skills training and solving excercises over and over again, using feedback to get better every time.
CodeKataBattle deals with two main users on its platform:

- Educators
- Students

Each Educator can create a tournament, either creating battles by him/herself or granting to other Educators the permission to create battles within the context of a specific tournament.
Once a new tournament is created, all the Students subscribed to CKB are notified and can decide whether to subscribe to that tournament or not. If a Student decides to subscribe to a tournament, he/she will be notified whenever a new battle is created in the context of that tournament.
An Educator that has the permission to do it can create a new battle, and delivers the exercise associated to it. Moreover, the Educator can set the rules for that specific battle.
If a Student subscribes to a specific battle, he/she is asked to build his/her own team, with reference to the limits imposed by the Educator. When the deadline expires, for each team is created a GitHub repository, and each team is asked to modify settings, so that for every push the team does, CKB is notified.
If the platform is triggered, it pulls the leatest sources of the team from which it has received the notification and automatically evaluates it. Then, it updates the team rank in the context of the battle.
When the submission deadline expires, there is an intermediate phase, called consolidation stage, in which Educators manually evaluate team projects, if it is required by the battle's settings.
After that, the students partecipating to that battle are notified by the platform and they can visualize their final rank, as well as thier tournament rank, which is updated by CKB whenever a battle ends in the context of that specific torunament.
In every tournament, Students can achieve some Badges, defined by Educators when they create a new tournament.
Every user subscribed to the platform can visualize the list of ongoing tournaments, the corresponding tournament rank and other Student's profile, including the badges achieved.

1.2.1 World Phenomena

[$WP1$] Students work on the assigned exercise.
[$WP2$] Students fork the GitHub repository of the code kata and set up an automated workflow that informs the CKB platform as soon as students push a new commit into the main branch of their repository.

1.2.2 Shared Phenomena - controlled by the World

[$SP1$] Educator creates a new tournament.
[$SP2$] Educator creates a new battle.
[$SP3$] Educator grants another Educator the permission to create battles within the context of a specific tournament.
[$SP4$] Educator delivers the exercise for a specific battle.
[$SP5$] Educator sets the minimum and maximum number of students per group for a specific battle.
[$SP6$] Educator sets the registration deadline for a specific battle.
[$SP7$] Educator sets the final submission deadline for a specific battle.
[$SP8$] Educator eventually sets the possibility of manually evaluating the projects for a specific battle.
[$SP9$] Student can subscribe to a tournament.
[$SP10$] Student can subscribe to a battle.
[$SP11$] Student can form a team for a specific battle.
[$SP12$] Both students and educators involved in the battle can see the current rank evolving during the battle.
[$SP13$] Educator manually evaluates the projects for a specific battle (consolidation stage).
[$SP14$] Both Educators and Students subscribed to CKB can see the list of ongoing tournaments as well as the corresponding tournament rank.
[$SP15$] Educator can define a Badge for a specific tournament.
[$SP16$] Both Educators and Students can visualize another Student's profile.

1.2.3 Shared Phenomena - controlled by the Machine

[$SP17$] CKB sends a notification to all Students subscribed to the platform whenever a new tournament is created.
[$SP18$] CKB sends a notification to all Students subscribed to a specific tournament whenever a new battle is created in the context of that specific tournament.
[$SP19$] At the end of the consolidation stage, all students participating in the battle are notified when the final battle rank becomes available.
[$SP20$] When an educator closes a tournament, as soon as the final tournament rank becomes available, the CKB platform notifies all students subscribed to that tournament.
[$SP21$] CKB can assign a Badge to a Student.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Programming language**: set of rules that allows string values to be converted into various ways of generating machine code, or, in the case of visual programming languages, graphical elements.
- **Automation scripts**: a specific type of automation script used in software development to automate the process of building, compiling, and packaging software applications.
- **Test-first approach**: is a software development process relying on software requirements being converted to test cases before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases.
- **GitHub**: a platform and cloud-based service for software development and version control, allowing developers to store and manage their code.
- **Badge**: elements in the form of rewards that represent the achievements of individual students.

1.3.2 Acronyms

- **CKB**: CodeKataBattle
- **RASD**: Requirement Analysis and Specification Document
- **UML**: Unified Modelling Language
- **UI**: User Interface
- **OWASP**: Open Web Application Security Project
- **WCAG**: Web Content Accessibility Guidelines

1.3.3 Abbreviations

- $[Gn]$ - the n-th goal of the system
- $[WPn]$ - the n-th world phenomena
- $[SPn]$ - the n-th shared phenomena
- $[UCn]$ - the n-th use case
- $[Rn]$ - the n-th functional requirement

1.4 Revision history

- Version 1.0 (22/12/2023)
- Version 1.0.1 (04/01/2024) minor spelling mistakes

1.5 Reference Documents

This document is based on:

- The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2023/2024
- Slides of Software Engineering 2 course on WeBeep;
- Official link of Codewars (https://www.codewars.com/), a platform similar to CodeKata-Battle
- Other informations that helped the development of the project:
  - In-depth analysis on codekata;
  - In-depth analysis on test-driven development;

1.6 Document Structure

This document is divided in 6 chapters, more in detail:

1. **Introduction**: this chapter describes the reasons for which this project is being developed, highlighting above all the goals aimed to reach. Moreover, it analyses the environment and the shared phenomena between the world and the machine.
2. **Overall description**: it is an initial analysis about the system to be developed, with examples of possible scenarios. In particular it focuses, through diagrams (e.g. class diagrams, state diagrams), on the elements in the system's domain and the interaction between them. This section also includes domain assumptions and the most relevant requirements of the system.
3. **Specific requirements**: this chapter aims to describe all the system's requirements, both functional and non-functional, and eventual constraints. Diagrams are used to integrate requirements with scenarios and generalize them into use cases.
4. **Formal analysis using Alloy**: this chapter describes, with Alloy language, the entire model in a formal way. There is also an overview about what can be verified and some examples of the world obtained running the code.
5. **Effort spent**: this section shows the time spent (in hours) by each member to work on this document.
6. **References**: it contains the references to any documents and to the Software used in this document.

2 Overall Description

2.1 Product perspective

2.1.1 Scenarios

A. Registration
Emanuele is a Computer Science and Engineering student. He wants to showcase and develop his programming skills. He heard about CKB and decides to give it a try. He goes to the CKB website and creates an account providing his personal info, his academic status (student/educator), and setting up a password. After that, he can join tournaments and finally start showing off his abilities while becoming a code master in the process.

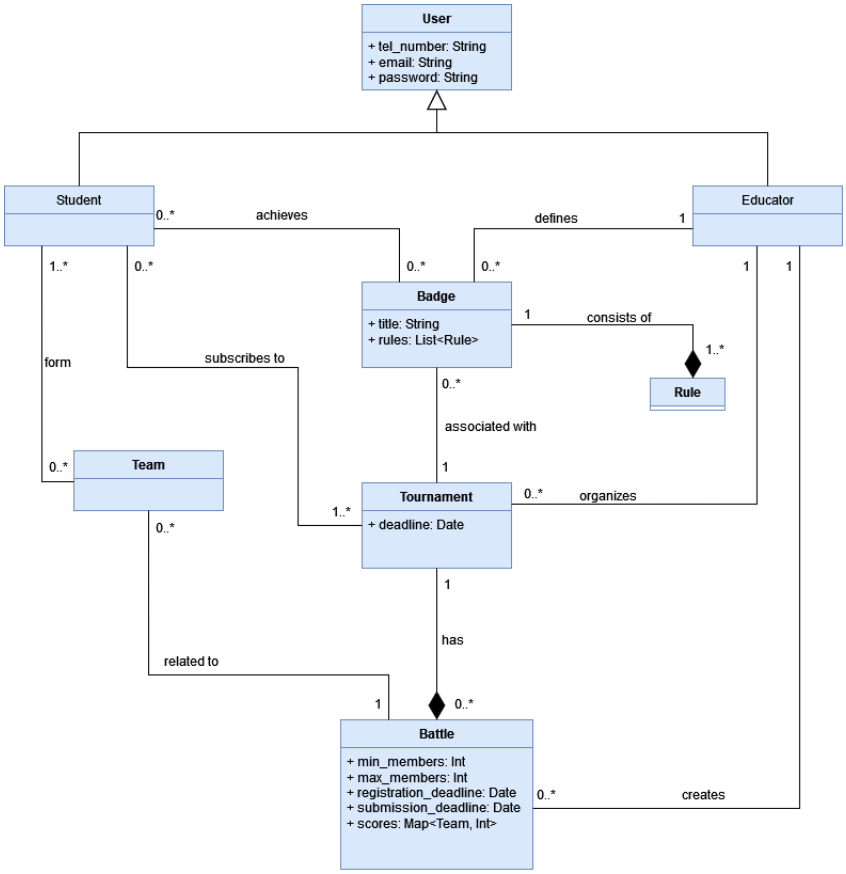
B. Tournament creation
Matteo is a Google hiring manager. He is looking for new talented and enthusiastic software engineers. He creates a new tournament specifying the deadline for the subscription and grants his team permission to create new battles. Users who enabled notifications will receive an email notifying them about the creation of this new tournament. Matteo finds a lot of passionate coders and engages with them in a wide variety of battles. He can then pick those who performed brilliantly and can offer them a job at Google.

C. Battle creation
Giuseppe is a recruiter from Google with an 'educator' account in CKB. He has been granted permission to create battles by his superior, Matteo. He publishes the code kata (that includes everything to make the battle work, from the description to test cases and automation scripts) and chooses edits the battle configuration settings such as minimum and maximum number of students per team, submission deadline and more. Students subscribed to this tournament can form teams and start working on the project. After the submission deadline, teams will receive an automatic evaluation. Giuseppe is allowed to review the evaluations if need be, and finally publish the final grades for each team.

D. Badge creation
Matteo wants to add achievements to his tournament to hype up the students. He chooses a few conditions that he finds interesting selects the titles and creates the badges. At the end of the tournament, all the system checks the badges conditions to each user. Ultimately, the badges will be assigned to the students and they will be able to show them in their personal profile.

2.1.2 Class diagram

The presented UML class diagram illustrates a conceptual, high-level model of the software. Due to its nature, it might model entities that won't necessarily be part of the final system under development. At this stage, it lacks references to methods and other low-level details, as these will be elaborated during the subsequent design phase.



The main entities are:

- **User**
  A user can be either a student or an educator. Users can perform different actions according to this: educators create tournaments and battles, while students join them.
- **Tournament**
  Tournaments are created by educators and students can subscribe to them. They consist of different battles. The creator of the tournament can grant other educators privileges to define battles. Each student has a score consisting of the cumulative sum of scores obtained in all battless. Badges can also be assigned to a specific tournament.
- **Battles**
  Battles are created by educators with privileges in the scope of a tournament. Numerous settings are available for customization, such as the minimum and maximum settings for a team, deadlines and more. Students subscribed to a tournament need to form a team for each battle. At the end of the battle, each team is assigned a score between 0 and 100.
- **Badge**
  Badges are gamification objects that students can collect and show off in their profile. Educators can create badges and associate them to a specific tournament. Each badge has a set of rules that students need to satisfy in order to achieve It.

2.1.3 State diagrams

The purpose of the following UML state diagrams is to provide additional information about the behaviour of the system. These are the scenarios that we found more interesting.

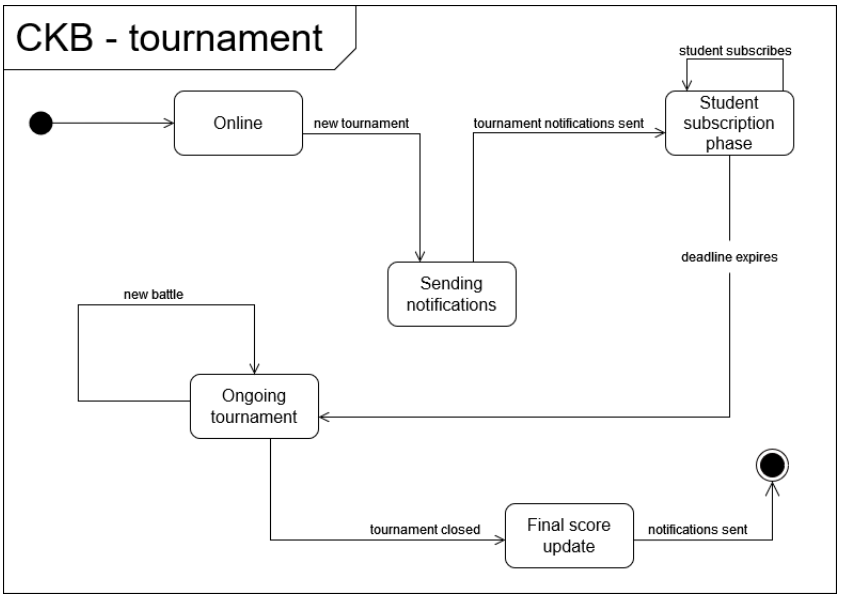


Figure 1: Tournament timeline

This diagram describes what happens inside the system throughout a tournament, from the creation to the conclusion. At first, the system is waiting for new events. When an educator creates a new tournament, the system responds by sending notifications to all students. After that, the system enters a waiting state, allowing students to subscribe to the tournament until the specified deadline. When the tournament is finally open, authorized educators can create new battles. This process is seen more in-depth in the next diagram. After each battle, the system automatically updates the score for each student. New battles can be created until the creator closes the tournament. Students who took part in this tournament are sent a notification about their score, their ranking and eventual badges they may have achieved. At any moment, the creator can grant other educators permission to create battles in the scope of his tournament.

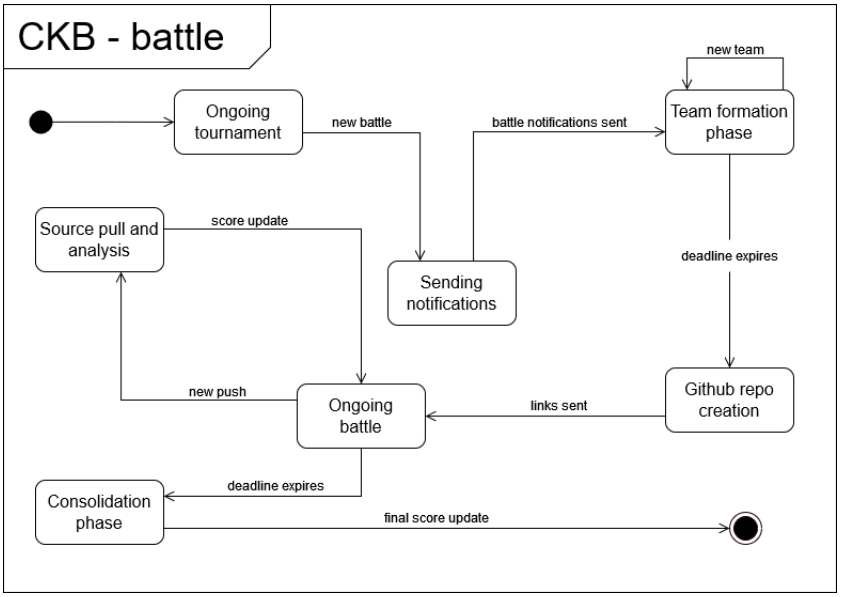


Figure 2: Battle timeline

This diagrams describes the steps of a battle. When a new battle is created in the scope of a tournament, the system sends a notification to all students subscribed to that tournament. Students have some time to form teams and after the deadline, a github repository is automatically created. Teams can then start working on the project. The system is notified through API calls every time a team pushes new code. Following each push, the system downloads and evaluates the sources, then publishes the scores up to that point. The specifics of these calls are beyond the scope of this diagram. After the deadline, there is a consolidation phase, where authorized educators can manually adjust the scores if need be. Finally, the battle officially ends and the students are notified.

2.2 Product Functions

2.2.1 User Registration and Authentication

- **User Registration**: Users can create accounts on the CKB platform by providing necessary information such as academic position (educator or student), email address and password. The registration process includes validation and email verification.
- **Authentication**: Once registered, users can log in with their credentials to access the platform. Authentication ensures that only authorized users can use the platform and the available functions are restricted depending on the user.
- **User Account Management**: Registered users have the ability to manage their account settings, update their profile information, and change passwords as needed.

2.2.2 Creating and Managing Tournaments

- **Tournament Creation**: Educators, who act as administrators, can create new tournaments. They provide details about the tournament, including its name, description, duration, and any specific rules.
- **Tournament Management**: Educators can modify and manage existing tournaments. They can update tournament details, add or remove battles, and set deadlines

for registration and submission. They can add other Educators to administer the tournament, too.

2.2.3 Joining and Forming Teams

- **Joining Battles**: Students can browse and subscribe to available tournaments and join specific battles within those tournaments. They can view information about battles, such as the code kata description and participation rules.
- **Team Formation**: Students can form teams by inviting other students to join them. The system checks that they adhere to the minimum and maximum team size rules set by the educator.

2.2.4 Code Kata Battles

- **Accessing Projects**: Students participating in a battle can download the respective project. Each project consists of a problem description and an initial codebase with missing functionality.

2.2.5 GitHub Integration

- **GitHub Repository Creation**: For each code kata, the platform creates a corresponding GitHub repository for each team. This repository contains the initial codebase and testing scripts.
- **Updating and Testing Projects**: Students set up GitHub Actions to automate the submission process. With each code commit to their repository, GitHub Actions trigger updates and testing on the platform.

2.2.6 Scoring and Ranking

- **Score Calculation**: The platform calculates scores based on the number of passed test cases, timeliness of submissions, and code quality. The higher the number of passing test cases, the higher the score.
- **Ranking System**: Students can view the real-time ranking of their teams and others in the same battle. Rankings are based on scores achieved during the competition.

2.2.7 Consolidation and Manual Evaluation

- **Consolidation Stage**: After the submission deadline, educators review the code and manually evaluate aspects that cannot be determined automatically, such as code quality and innovation.
- **Final Ranking**: Once the consolidation stage is complete, the final ranking is determined, and students are notified of their positions and scores.

2.2.8 Gamification and Badges

- **Badge Definition**: Educators can create gamification badges with specific titles and rules.
- **Badge Assignment**: The badges are awarded to students based on the rules defined by the educators.

2.2.9 Notification and Communication

- **Notification System**: The platform sends notifications to students about various events, such as new tournament announcements, battle updates, and final tournament rank confirmations.

2.2.10 Profile and Tournament Statistics

- **User Profiles**: Each user has a profile that displays their personal information, tournament participation, and badges earned.
- **Tournament Statistics**: Users can view statistics related to their performance in each tournament, including scores, rankings, and tournament history.

2.3 User characteristics

There are mainly two kinds of users that interact with the system: Educator and Student.

2.3.1 Educator

- **Teaching Level**: Educators can be university professors, instructors, or trainers responsible for organizing and supervising code kata battles.
- **Educational Institution**: They may work in educational institutions and have experience in teaching software development.
- **Platform Experience**: Educators should be familiar with the CodeKataBattle platform to create and manage tournaments, set up battles, and assess student performance.
- **Technical Proficiency**: While not all educators need to be technically proficient, some may have a background in software development and programming.

2.3.2 Student

- **Programming Proficiency**: Students may vary in their level of programming proficiency, ranging from beginners to advanced developers.
- **Educational Level**: Students from various educational backgrounds, such as undergraduate, graduate, or coding boot camp participants, may use the platform.
- Motivation: Students may use the platform for different reasons, including academic requirements, self-improvement, or competition.
- **Preferred Programming Languages**: Users may have preferences for specific programming languages and may select code katas accordingly.
- **Team Formation**: Students can either participate individually or form teams based on the requirements of specific battles.
- **Time Availability**: Students have different schedules and availability for participating in code kata battles.

2.4 Assumptions, dependencies and constraints

2.4.1 Regulatory policies

1. ***Data Privacy and Protection:***

- **General Data Protection Regulation (GDPR)**: The CKB platform will ask for user personal information like name, surname and email address. Email addresses won't be used for commercial purposes. Personal information will be processed in compliance with the GDPR.

2. ***Intellectual Property Rights:***
   - **Copyright and Licensing**: The platform must respect copyright and licensing regulations when handling code katas, user submissions, and other content. It should encourage users to respect the intellectual property rights of others and provide mechanisms for reporting copyright violations.
   - **Plagiarism Prevention**: The platform should have mechanisms in place to detect and prevent plagiarism in code submissions. This includes educating users on proper citation and attribution practices.

2.4.2 Domain Assumptions

The following are the assumptions made for the domain. Such assumptions are properties and/or conditions that the system takes for granted, mostly because they are out of the control of the system itself, and hence need to be verified to assure the correct behavior of CKB.
[$D1$]: Users have reliable and consistent access to the internet to participate in code kata battles and interact with the platform.
[$D2$]: Users have access to suitable devices, such as computers or mobile devices, with compatible web browsers.
[$D3$]: Educators have a basic understanding of the CodeKataBattle platform's features, enabling them to create and manage tournaments and battles effectively. In particular, they have to provide correct tests and rules for projects and badges.
[$D4$]: User-provided data, including user profiles, code submissions, and scoring information, is assumed to be accurate and valid. Users are supposed to providing truthful information.
[$D5$]: Users will use the platform responsibly, adhering to ethical guidelines and academic integrity. Students are expected to complete code katas independently or within the rules of team collaboration defined by educators.
[$D6$]: The platform assumes a high level of system uptime, with minimal downtime, to support user interactions, database storage, and real-time features.
[$D7$]: Some platform functionalities may rely on external services, such as email verification, communication tools and Github integrations. These external services are assumed to be available and functional.
[$D8$]: Users are responsible for maintaining the security and privacy of their accounts, including choosing strong passwords and keeping their login credentials confidential.
[$D9$]: Users are expected to engage in ethical behavior, respecting intellectual property rights, academic integrity, and community guidelines established by the platform.

3 Alloy

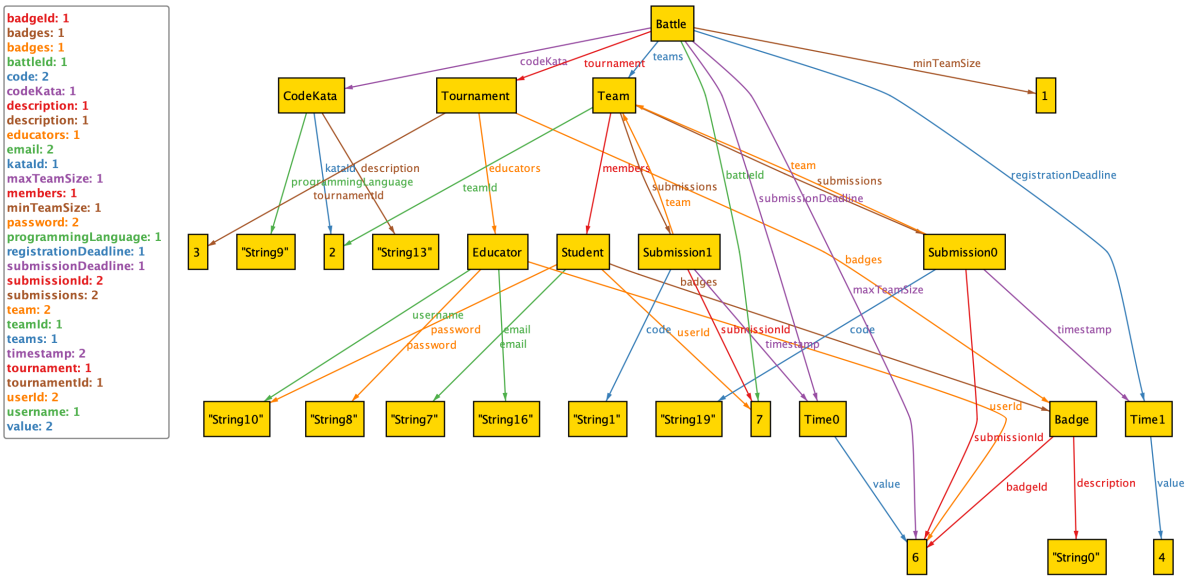
In the following are stated all the signatures defined for the Alloy modelization.

```
-- Alloy model for CodeKataBattle platform


// SIGNATURES

-- Time is used to represent a couple <date,time>
sig Time {
  value: one Int,      -- Value of the <data, time> couple
}

-- Signature representing a user in the system
abstract sig User {
  userId: one Int,                  -- Unique identifier for a user
  username: lone String,        -- User's username
  email: one String,                -- User's email address
  password: one String,         -- User's password
}
sig Educator extends User {}
sig Student extends User {
  badges: some Badge           -- Student's badges
}

-- Signature representing a code kata for programming exercises
sig CodeKata {
  kataId: one Int,                                  -- Unique identifier for a code kata
  description: lone String,                     -- Description of the programming exercise
  programmingLanguage: lone String,    -- Programming language for the exercise
}

-- Signature representing a tournament organized by an educator
sig Tournament {
  tournamentId: one Int,          -- Unique identifier for a tournament
  educators: some Educator,    -- Educator organizing the tournament and Educators managing the
      ↪ tournament
  battles: set Battle,                -- Set of battles within the tournament
  badges: some Badge            -- Some badges that could be associated with the tournament
}

-- Signature representing a battle within a tournament
sig Battle {
  battleId: one Int,                        -- Unique identifier for a battle
  codeKata: one CodeKata,            -- Code kata for the battle
  tournament: one Tournament,     -- Tournament associated to the battle
  teams: set Team,                          -- Set of teams participating in the battle
  maxTeamSize: one Int,               --Max Number of Students allowed in a team
  minTeamSize: one Int,                --Min Number of Students allowed in a team
  registrationDeadline: one Time,  -- Deadline for team registration
  submissionDeadline: one Time    -- Deadline for code submission
}

-- Signature representing a team participating in a battle
sig Team {
  teamId: one Int,                    -- Unique identifier for a team
  members: set Student,           -- Set of users in the team
  submissions: set Submission -- Set of submissions of the team

}

-- Signature representing a code submission by a team
sig Submission {
  submissionId: one Int,      -- Unique identifier for a submission
  team: one Team,              -- Team that submitted the code
  code: lone String,            -- Code submitted by the team
  timestamp: one Time        -- Timestamp of the submission
}

sig Badge {
  badgeId: one Int,  -- Unique identifier for a badge
  description: one String  -- Description of the badge
}
```

```
// FACTS

-- Fact spcifying time constraint
fact TimeContraints{
  all t: Time |
    t.value > 0
   and
  (all disj t1, t2: Time |
    t1.value ≠ t2.value)
}

-- Fact specifying constraints on user attributes
fact UserConstraints {
  (all u: User |
    u.userId > 0 and
    u.email ≠ none and
    u.password ≠ none)
  (all disj u1, u2: User |
     u1.email ≠ u2.email and
     u1.username ≠ u2.username and
     u1.userId ≠ u2.userId)
  (all s: Student |
     #s.badges ≥ 0)
}

-- Fact specifying constraints on code kata attributes
fact CodeKataConstraints {
  (all ck: CodeKata |
    ck.kataId > 0 and
    ck.description ≠ none and
    ck.programmingLanguage ≠ none)
  and
  (all disj ck1, ck2: CodeKata |
    ck1.kataId ≠ ck2.kataId)
}

-- Fact specifying constraints on tournament attributes
fact TournamentConstraints {
  (all t: Tournament |
    t.tournamentId > 0 and
    t.educators ≠ none and
    #t.battles ≥ 0)
  and
  (all disj t1, t2: Tournament |
    t1.tournamentId ≠ t2.tournamentId)
}

-- Fact specifying constraints on battle attributes
fact BattleConstraints {
  all b: Battle |
    b.battleId > 0 and
    b.codeKata ≠ none and
    b.teams ≠ none and
    b.registrationDeadline ≠ none and
    b.submissionDeadline ≠ none
  and
  (all disj b1, b2: Battle |
    b1.battleId ≠ b2.battleId)
}

-- Fact specifying constraints on team attributes
fact TeamConstraints {
  all t: Team |
    t.teamId > 0 and
    t.members ≠ none and
    t in Battle.teams
  and
  (all disj t1, t2: Team |
    t1.teamId ≠ t2.teamId)
}

-- Fact specifying constraints on submission attributes
fact SubmissionConstraints {
  (all s: Submission |
    s.submissionId > 0 and
    s.team ≠ none and
    s.code ≠ none and
    s.timestamp ≠ none and
    (all t: Team |
    s in t.submissions iff s.team = t))
  (all disj s1, s2: Submission |
    s1.submissionId ≠ s2.submissionId
    and
    (s1.team = s2.team implies s1.timestamp ≠ s2.timestamp))
}
```

```
-- Fact specifying constraints on submission attributes
fact BadgeConstraints {
  all b: Badge |
    b.badgeId > 0 and
    b.description ≠ none
  and
  (all disj b1, b2: Badge |
    b1.badgeId ≠ b2.badgeId)
}

-- Fact specifying the contraint  of the team size in a battle
fact TeamInBattleConstraints {
  all b: Battle |
    all t: b.teams |
      #t.members ≤ b.maxTeamSize and
      #t.members ≥ b.minTeamSize
}

-- Fact specifying the constraint of the size of teams
fact SizeOfTeamsContraints {
  all b: Battle |
    b.minTeamSize ≥ 1 and
    b.maxTeamSize ≥ 1
}

-- Fact specifying the membership of the student to a single team in the scope of a battle
fact MembershipOfStudentConstraints {
  (all b: Battle |
    all disj t1, t2: b.teams |
      all s: t1.members |
        s not in t2.members)
}

-- Fact specifying the time constraint between registration and submission deadlines
fact RegistrationSubmissionContraints {
  (all b: Battle |
    b.registrationDeadline.value < b.submissionDeadline.value)
}




// ASSERTIONS

-- Assertion: All users have unique user IDs
assert UniqueUserIDs {
  all u1, u2: User | u1 ≠ u2 implies u1.userId ≠ u2.userId
}

-- Assertion: All code katas have unique IDs
assert UniqueKataIDs {
  all ck1, ck2: CodeKata | ck1 ≠ ck2 implies ck1.kataId ≠ ck2.kataId
}

-- Assertion: All tournaments have unique IDs
assert UniqueTournamentIDs {
  all t1, t2: Tournament | t1 ≠ t2 implies t1.tournamentId ≠ t2.tournamentId
}

-- Assertion: All battles have unique IDs
assert UniqueBattleIDs {
  all b1, b2: Battle | b1 ≠ b2 implies b1.battleId ≠ b2.battleId
}

-- Assertion: All teams have unique IDs
assert UniqueTeamIDs {
  all t1, t2: Team | t1 ≠ t2 implies t1.teamId ≠ t2.teamId
}

-- Assertion: All submissions have unique IDs
assert UniqueSubmissionIDs {
  all s1, s2: Submission | s1 ≠ s2 implies s1.submissionId ≠ s2.submissionId
}

-- Assertion: All team members are distinct
assert DistinctTeamMembers {
  all t: Team | no disj u1, u2: t.members | u1 ≠ u2
}

-- Assertion: CodeKata associated with a battle must be unique
assert UniqueCodeKataPerBattle {
  all b: Battle | all disj ck1, ck2: b.codeKata | ck1 ≠ ck2
}
```

```
-- Assertion: Submission timestamp is within the battle deadlines
assert SubmissionWithinDeadlines {
  all b: Battle, s: Submission | s in b.teams.submissions implies
    b.registrationDeadline.value ≤ s.timestamp.value and
    s.timestamp.value ≤ b.submissionDeadline.value
}


-- Assertion: Each team in a battle has a distinct set of members
assert DistinctTeamMembersInBattle {
  all b: Battle | all t1, t2: b.teams | t1 ≠ t2 implies no t1.members & t2.members
}

-- Assertion: All battles within a tournament have distinct IDs
assert UniqueBattlesPerTournament {
  all t: Tournament | all disj b1, b2: t.battles | b1 ≠ b2
}



// PREDICATES

-- Predicate representing the action of a user registering for a battle
pred RegisterForBattle[student: Student, battle: Battle] {
  student in battle.teams.members
}

-- Predicate representing the evaluation of a code submission in a battle
pred EvaluateSubmission[battle: Battle, submission: Submission] {
  submission in battle.teams.submissions
}

run show{} for 3 but exactly 20 String
```



Alloy Model

4 Effort Spent

Emanuele Pocelli

Chapter	**Effort (in hours)**
1	2
2	10
3	16
4	2

Fabrizio Sordetti

Chapter	**Effort (in hours)**
1	4
2	8
3	5
4	13

Andrea Varesi

Chapter	**Effort (in hours)**
1	9
2	5
3	12
4	4

5 References

- Diagrams made with: draw.io
- Mockups made with: moqups.com
- Alloy models made with: Alloy Analyzer 6.1.0
- Alloy models runned and checked with: Alloy Analyzer 6.1.0