RASD

Federico Valentino, Nicola Zarbo

October 2023

Contents

Chapter 1

Introduction

1.1 Purpose

CodeKataBattle (CKB) is a new platform that helps students improve their software development skills by training with peers on code katas . Educators use the platform to challenge students by creating code kata battles in which teams of students can compete against each other, thus proving (and improving) their skills.
A code kata battle is essentially a programming exercise in a programming language of choice (e.g., Java, Python). The exercise includes a brief textual description and a software project with build automation scripts (e.g., a Gradle project in case of Java sources) that contains a set of test cases that the program must pass, but without the program implementation. Students are asked to complete the project with their code. In particular, groups of students participating in a battle are expected to follow a test-first approach and develop a solution that passes the required tests. Groups deliver their solution to the platform (by the end of the battle). At the end of the battle, the platform assigns scores to groups to create a competition rank.

1.1.1 Goals

The CKB system is thought, designed and proposed to two types of users: Students and Educators.
The firsts, will be able to join the platform to test their coding skills in Tournaments, a series of code battles, where students will be able to participate in groups.
Educators instead will be able to create and manage tournaments and decide whether or not a manual score in a battle is needed or not.
Below the list of goals of the CKB platform.

ID	Description
G1	An Educator can manage a tournament.
G2	An educator can create battles inside of a tournament in which he is involved.
G3	Students can participate in tournaments created by an educator
G4	Students can participate and compete in battles created by an educator, alone or in groups.
G5	Students are scored based on their performance in battles.
G6	The platform allows students and educators to compare the performance of students.

Table 1.1: The goals.

1.2 Scope

1.2.1 World phenomena

ID	Description
W1	An educator wants to create a tournament to evaluate students performance.
W2	Students fork the created repository for a battle on Github.
W3	An educator creates the battle assignment.
W4	Students write source code for the code kata battle.
W5	Students create commits on Github.
W6	Students decide to join a battle.
W7	Students create groups for battles.
W8	Students setup an automated workflow for the forked repository on Github.
W9	Students decide to join a tournament.
W10	Educator decides to close tournament.

Table 1.2: World Phenomenas.

1.2.2 Shared phenomena

ID	Description	Controller	Observer
SP1	An educator fills out a tournament creation form.	Educator	CKB

SP2	An educator uploads the details of a code kata battle(the assignment, the rules, the tests).	Educator	CKB
SP3	A group(maybe singleton) joins a battle respecting the rules regarding the min and max group size.	Student	CKB
SP4	An educator logs into the platform.	Educator	CKB
SP5	A student logs into the platform.	Student	CKB
SP6	The system requires additional manual evalution by an educator for a battle(if required by the rules).	Educator	CKB
SP7	The educator inserts an additional manual score for a battle.	Educator	CKB
SP8	A student invites other students to a group to participate to a battle.	Student	CKB
SP9	Github on commit notifies the code kata platform.	GitHub	CKB
SP10	An educator registers an account on the platform.	Educator	CKB
SP11	A student registers an account on the platform.	Student	CKB
SP12	A student subscribes to a tournament.	Student	CKB
SP13	Students and educators look at the rank for a battle they are involved in.	User	CKB
SP14	Students and educator look at the rank for a tournament.	User	CKB
SP15	Educator closes tournament.	Educator	CKB
SP16	User looks at list of available tournament.	User	CKB
SP17	A student accepts and invitet o a group to partecipate to a battle.	Student	CKB
SP18	The platform notifies all students when a tournament is created.	CKB	Student
SP19	The platfotm notifies all students subscribed to a tournament of new upcoming battles.	CKB	Student
SP20	The platform notifies the final score to all students subscribed to a battle, when that battle ends.	CKB	Student
SP21	When the platform is notifies about a commit, it pulls from the committed repository to start the mandatory analysis.	CKB	Github

SP22	The platform creates the github repository for a battle.	CKB	Github
SP23	The platform sends links to the created repository for the battle to all students who are subscribed to the battle.	CKB	Student

Table 1.3: Shared Phenomenas.

1.3 Definitions, Acronyms, Abbreviations

Acronym	**Definition**
CKB	CodeKataBattle
CK	Code Kata

Table 1.4: Acronyms used in the document

Educator involved in a tournament: These are the educator who created the tournament and the collaboratorating educators who were granted ability to create a battle.
Battles in tournament context: These are the battles created within a tournament by an involved educator, all students subscribed to said tournament can join them.

1.4 Document Structure

The document is divided in six sections.
The first section introduces the goals of the project and shared phenomena with also a list of definitions useful to understand the problem.
The second section provides a more accurate description of the problem, describing the details of domain and scenarios.
The third section focuses on specific requirements and provides an analysis on interface requirements, functional requirements and so on.
The fourth section provides a formal analysis using Alloy, crucial to prove the correctness of the model described.
The fifth one reports the effort spent by each group member in the redaction of this document and the last section is simply a list of bibliography references.

Chapter 2

Overall description

2.1 Product perspective

2.1.1 Class Diagrams

The diagram below represents and describes the classes involved in the system, their functionalities and how they interact.

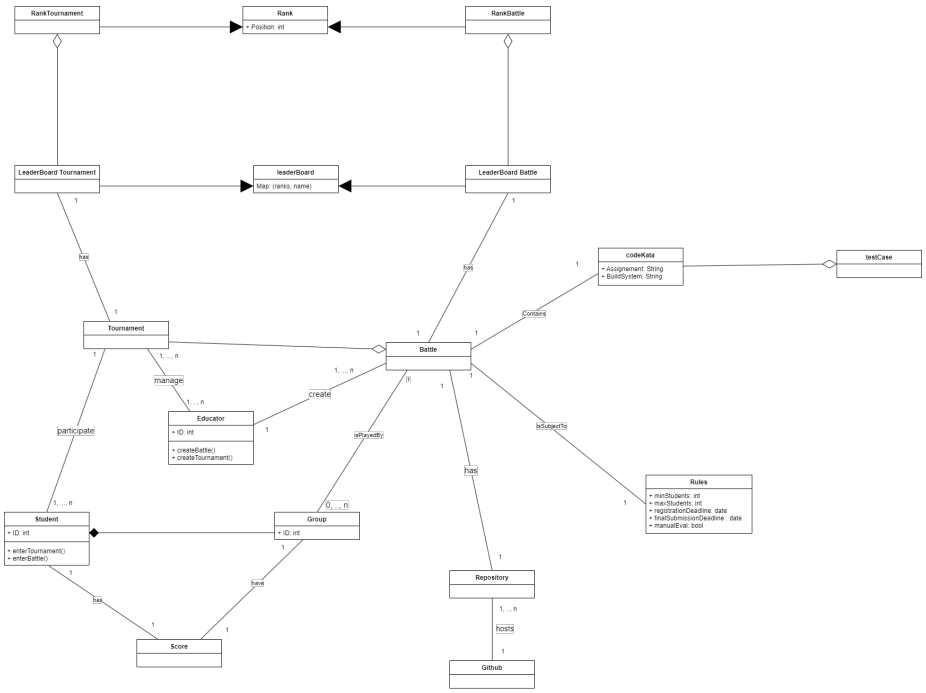

Figure 2.1: Class Diagram

2.1.2 Scenarios

1. **User signs into platform**
   Professor Layton wants to evaluate the performance of his programming students but he has no easy way to do so. Fortunately he knows about CKB that can allow him to write a few coding challenges and automatically score his students submissions. He reaches the platform and is presented with a Sign In page where he has to full fill his details and set his profile to educator.
   Luke is a student of Professor Layton and when he heard that the professor is using CKB he immediately reached it and signed up by full filling his details and setting his profile to student.

2. **User logs into platform**
   A student that want to access the functionalities of the platform can log in , an educator does the same.

3. **Educator creates tournament**
   Professor Layton, now signed into CKB, can create a new tournament and add collaborators to it. Once the tournament has been created every user subscribed to the CKB platform is notified and can enter it.

4. **Educator creates battle**
   An educator wants to add another battle in the context of a tournament he is partecipating in. He starts by creating a CodeKata (CK) assignment (containing description, software project, test cases and build automation scripts) and ultimately creates the battle using the CKB platform. On the CKB platform the educator then uploads the CK assignment and sets:

   - minimum and maximum number of students per group,
   - set a registration deadline,
   - set a final submission deadline,
   - set additional configurations for optional manual scoring

   Once the battle in is created all the students subscribed to the tournament of said battle receive a notification of its creation.

5. **Student subscribes to a tournament**
   Once a student is logged into the website he is presented with a list of available tournaments he can join. By clicking on one of them the tournament description will be available for him to read and if he wants he can click on join to take part in that tournament.

6. **Student joins a team for a battle**
   The students are notified of a new battle in the context of a tournament. A student that decides to join the battle may be asked to join as a group (in case the battle rules ask for it), and so he invites other students using

the CKB platform to create a group that respects the group dimension rule. Once every student accepts the new group joins the battle.

Once the subscription deadline expires the CKB platform creates a GitHub repository containing the CK and sends every student a link for it. At this point for every group the students fork the repository and set up the automated workflow with Github Actions. At this point the group is ready to start working on the CK.

7. **Application scores a commit from a student**
   Every time a student commits some work to the forked repository the platform is notified and starts to run its automated evaluation by pulling the sources and running tests for:

   - functional aspects, measured in terms of number of test cases that pass out of all test cases;
   - timeliness, measured in terms of time passed between the registration deadline and the last commit;
   - quality level of sources, extracted through static analysis tools.

8. **Users look at current battle ranks**
   At any point the students that joined a battle and the educators that are involved with it are able to look at the ranking for said battle. Every group position is determined by the score given by the platform to their solution of the CK.

9. **Battle submission deadline expires**
   When the submission deadline expires the battle goes into a consolidation stage where, if decided at the creation, educators can manually assign scores to each team by inspecting sources. Once this stage has ended all students participating in the battle are notified about their final battle rank and the tournament score is updated.

10. **Educator closes tournament**
    At some point an educator decides that it is time to close a tournament, which he does through the platform. Then, as soon as the final tournament rank is available, CKB platform notifies all the students that were subscribed to the tournament that it has ended.

11. **Users look at tournament ranks**
    Between battles and after the end of a tournament users can look at the tournament leader board. The position of every student is determined by the sum of all battle scores.

2.2 Product functions

An Educator Creates a Tournament
The main functionality of CKB is to offer Educators the possibility of creating tournaments and battles inside of them. An Educator can decide to also add collaborators to the tournament, in order to ease the managing of it all. **A Student joins a Tournament**
CKB offers the possibility to Students to join any tournament they like and to partecipate to as many as they want. Inside a tournament students can then join battles in a group and climb its leader-boards.

2.3 User characteristics

The actors listed below are considered in the CKB system:

- Student: User that participates in tournaments and battles. He/She can subscribe to new tournaments and join battles alone or in team;
- Educator: User that managed tournaments and battles. He/She can create and end tournaments and add new battles to the created tournaments;
- GitHub: Another platform used by the CKB system to create and manage code repositories.

2.4 Assumptions, dependencies and constraints

ID	Description
A1	The students can correctly setup the Github actions workflow.
A2	An educator will complete the manual evaluation
A3	Github will always notifies the CKB platform after every student commit
A4	Students are always able to create a group to join a battle
A5	Educators will only close a tournament when no battle is still ongoing
A6	Only the Educator who created the tournament will close it

Chapter 3

Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

There are no user interface requirements.

3.1.2 Hardware Interfaces

There are no hardware interface requirements.

3.1.3 Software Interfaces

The CKB system relies on the use of the GitHub system in order to manage the battle's repositories. With every new battle created a new repository is generated on Github. Every group of student will then fork this repository and setup a workflow in order to notify the CKB system of every new commit through Github.

3.1.4 Communication Interfacess

There are no communication interface requirements.

3.2 Functional Requirements

3.2.1 Requirements

The CKB system offers several functionalities. In the following table we describe all the requirements that the system should respect in order to achieve its goals.

ID	Description
R1	The platform allows a signed in educator to create tournaments
R2	Educators can create battles in the context of a specific tournament they are involved in (Either by creation or by invitation)
R2.1	The platform allows an educator that created a tournament, to invite other educator and to grant them permission to create battles in the tournament context
R2.2	The platform allows an educator to upload the codekata (description and software project, including test cases and build automation scripts) when creating a battle
R2.3	The platform allows an educator to set subscribtion and submission deadlines when creating a battle
R2.4	The platform allows an educator to set minimum and maximum group size when creating a battle
R2.5	The platform allows an educator creating a battle to include a manual evaluation stage
R3	The platform allows students to subscribe to a tournament
R4	The platform allows a student subscribed to a tournament to join a battle in that tournament context
R4.1	The platform allows students to create a group by inviting other students when joinin a battle
R5	The platform allows Students to login
R6	The platform allows Educators to login
R7	If manual evaluation is required during consolidation stage the platform allows an educator to go through sources and add a score to the group score
R8	The platform allows all users to view student ranks from previous and current tournaments
R8.1	For every tournament the platform maintains a ranking for students based on the sum of their battle scores
R9	The platform pulls group sources from Github when it receives a notification within the submission deadline
R9.1	The platform analyzes, runs testcases and scores the source of a group solution for a codekata battle and updates the group score accordingly
R10	The platform allows all user to sign in
R11	The platform allows educators to close tournaments

R12	When a battle deadline expires the platform sets the battle status to consolidation stage
R13	The platform allows all users who are involved in a battle to look at group ranks for that battle
R13.1	For every battle the platform maintains a ranking of the groups based on their battle score
R14	The platform will notify signed students of a newly created tournament
R15	The platform shall notify students who are subscribed into a tournament that a new battle is available in that tournament context
R16	The plaftorm shall notify students who are participating in a battle that the final rank for that battle is available
R17	The plaftorm shall notify students who are subscribed in a tournament that the final rank for that tournament is available
R18	The platform shall create a new repository with Github for every new battle in any tournament after the submission deadline expires
R19	The platform shall send the battle repository link to students who joined that battle after the submission deadline expires

3.2.2 Mapping on goals

Goal	Domain Assumption	Requirements
G1	A5, A6	R1, R11, R6, R10
G2		R2, R2.1, R2.2, R2.3, R2.4, R2.5, R6, R10
G3		R3, R14, R5, R6, R10
G4	A1, A4	R4, R4.1, R15, R5, R6, R10, R18, R19
G5	A2, A3	R7, R9, R9.1, R12, R5, R10
G6		R8, R8.1, R13, R13.1, R16, R17, R5, R6, R10

An educator can manage a tournament

- R1: The platform allows a signed in educator to create tournaments
- R11: The platform allows an educator to close a tournament
- R6: The platform allows Educators to login
- R10: The platform allows all user to sign in

- A5: Educators will only close a tournament when no battle is still ongoing
- A6: Only the Educator who created the tournament will close it

An educator can create battles inside of a tournament in which he is involved

- R2: Educators can create battles in the context of a specific tournament they are involved in (Either by creation or by invitation)
- R2.1: The platform allows an educator that created a tournament, to invite other educator and to grant them permission to create battles in the tournament context
- R2.2: The platform allows an educator to upload the codekata (description and software project, including test cases and build automation scripts) when creating a battle
- R2.3: The platform allows an educator to set subscribtion and submission deadlines when creating a battle
- R2.4: The platform allows an educator to set minimum and maximum group size when creating a battle
- R2.5: The platform allows an educator creating a battle to include a manual evaluation stage
- R6: The platform allows Educators to login
- R10: The platform allows all user to sign in

Students can partecipate in tournaments created by an educator

- R3: The platform allows students to subscribe to a tournament
- R14: The platform will notify signed students of a newly created tournament
- R5: The platform allows Students to login
- R6: The platform allows Educators to login
- R10: The platform allows all user to sign in

Students can partecipate in battles created by an educator, alone or in groups

- R4: The platform allows a student subscribed to a tournament to join a battle in that tournament context
- R4.1: The platform allows students to create a group by inviting other students when joinin a battle

- R15: The platform shall notify students who are subscribed into a tournament that a new battle is available in that tournament context
- R5: The platform allows Students to login
- R6: The platform allows Educators to login
- R10: The platform allows all user to sign in
- R18: The platform shall create a new repository with Github for every new battle in any tournament after the submission deadline expires
- R19: The platform shall send the battle repository link to students who joined that battle after the submission deadline expires
- A1: The students can correctly setup the Github actions workflow
- A4: Students are always able to create a group to join a battle

Students are scored based on their performance in battles

- R7: If manual evaluation is required during consolidation stage the platform allows an educator to go through sources and add a score to the group score
- R9: The platform pulls group sources from Github when it receive a notification within the submission deadline
- R9.1: The platform analyzes, runs testcases and scores the source of a group solution for a codekata battle and updates the group score accordingly
- R12: When a battle deadline expires the platform sets the battle status to consolidation stage
- R5: The platform allows Students to login
- R10: The platform allows all user to sign in
- A2: An educator will complete the manual evaluation
- A3: Github will always notifies the CKB platform after every student commit

The platform allows students and educators to compare the performance of students

- R8: The platform allows all users to view student ranks from previous and current tournaments
- R8.1: For every tournament the platform maintains a ranking for students based on the sum of their battle scores

- R13: The platform allows all users who are involved in a battle to look at group ranks for that battle
- R13.1: For every battle the platform maintains a ranking of the groups based on their battle score
- R16: The plaftorm shall notify students who are participating in a battle that the final rank for that battle is available
- R17: The plaftorm shall notify students who are subscribed in a tournament that the final rank for that tournament is available
- R5: The platform allows Students to login
- R6: The platform allows Educators to login
- R10: The platform allows all user to sign in

3.2.3 Use Case Diagrams

Here the Use Case diagrams are divided by primary actor. There is no diagram for Github, as it is just a supporting actor.

Student Use Case Diagram

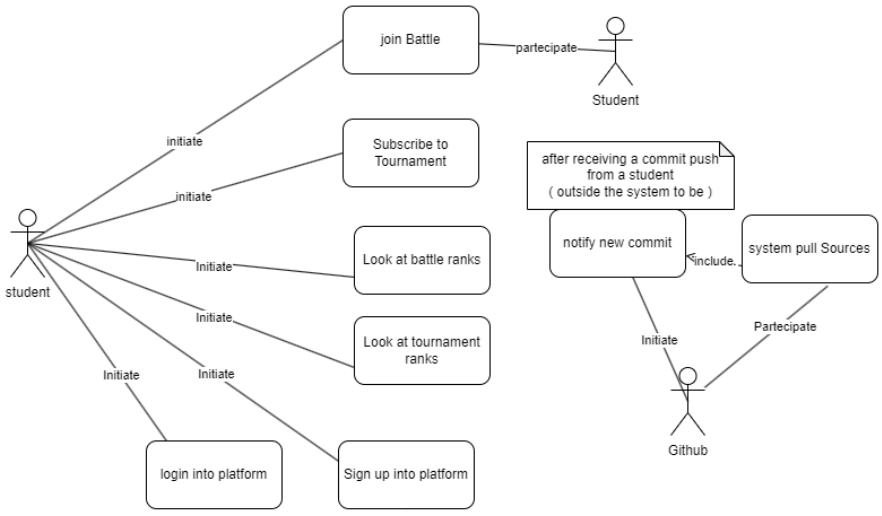

Figure 3.1: Student Use Case Diagram

Educator Use Case Diagram

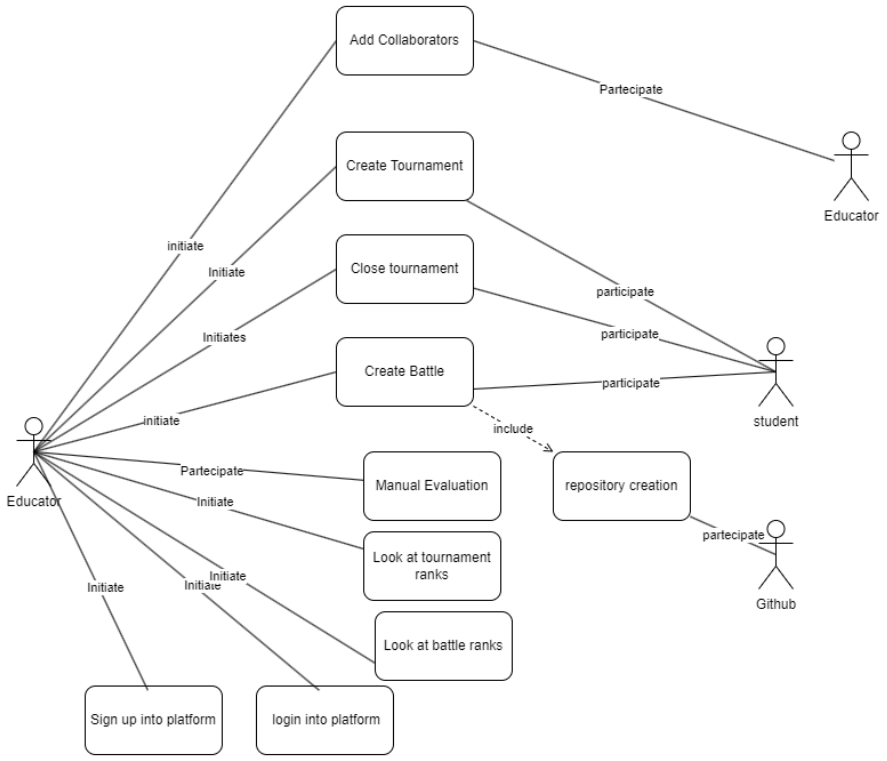

Figure 3.2: Educator Use Case Diagram

3.2.4 Use Cases

In this section we analyze the various use cases of the system by describing the actors, entry conditions, event flow, exit conditions and exceptions for all of them.

UC1: Sign Into Platform

Actor	Users(Educators, Students)
Entry condition	A user accesses the platform for the very first time
Event Flow	1. User reaches the platform 2. User clicks on "sign in" button 3. User fills out own info and decides if they want to be an Educator or student 4. User concludes the registration by clicking "finish" button
Exit	User is successfully registered on the platform as either educator or student
Exception	Missing or wrong input by user during registration form

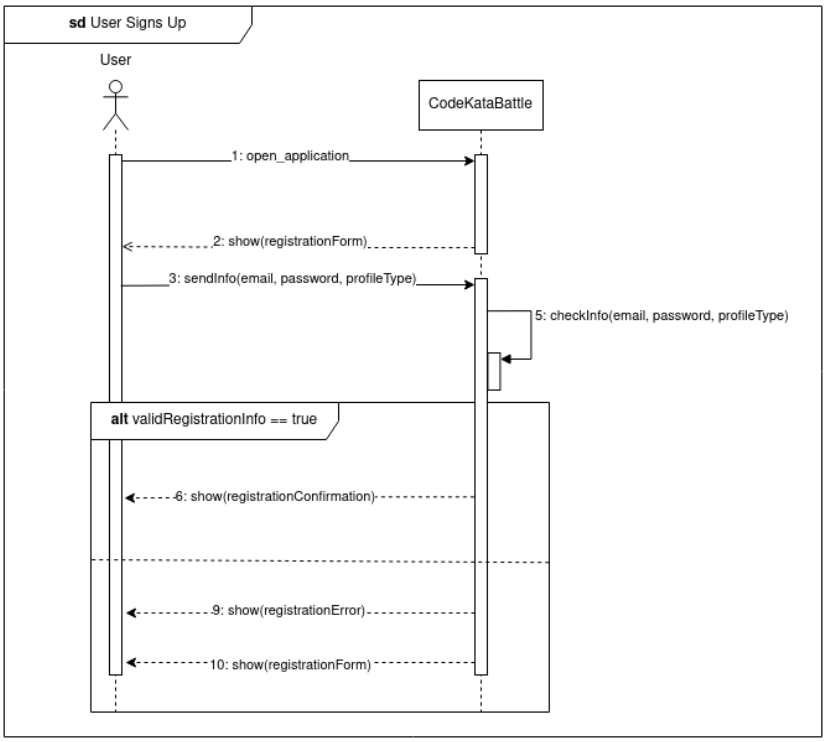

Figure 3.3: User Signs Up sequence diagram

UC2: Log Into Platform

Actor	User
Entry condition	User is signed into platform
Event Flow	1. User reaches the platform 2. User clicks on "log in" button 3. User fills out email and password used during registration 4. User completes login by clicking "confirm" button
Exit	User is successfully logged into platform
Exception	Missing or wrong input by user during log in form

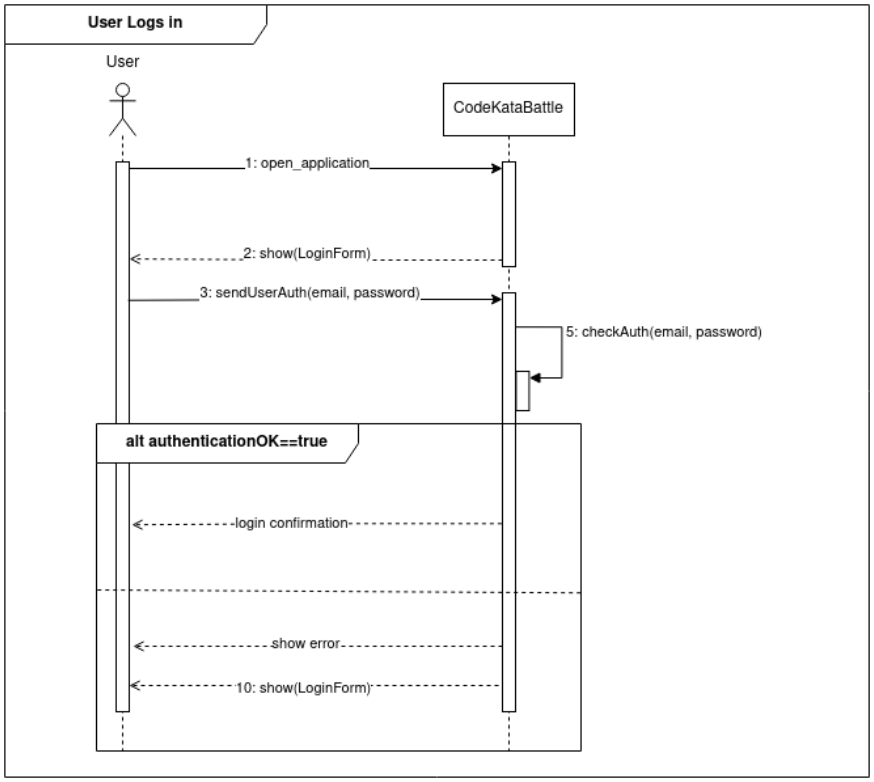

Figure 3.4: Create tournament sequence diagram

UC3: Create Tournament

Actor	Educator, Students
Entry condition	Educator is logged into the platform
Event Flow	1. Educator presses "Create Tournament" button 2. Educator fills out details of tournament 3. Educator adds other collaborators to tournament(other Educators) 4. Educator completes tournament creation 5. Platform notifies all Students of new tournament
Exit	Tournament is created successfully
Exception	Missing or wrong input by Educator during tournament creation form

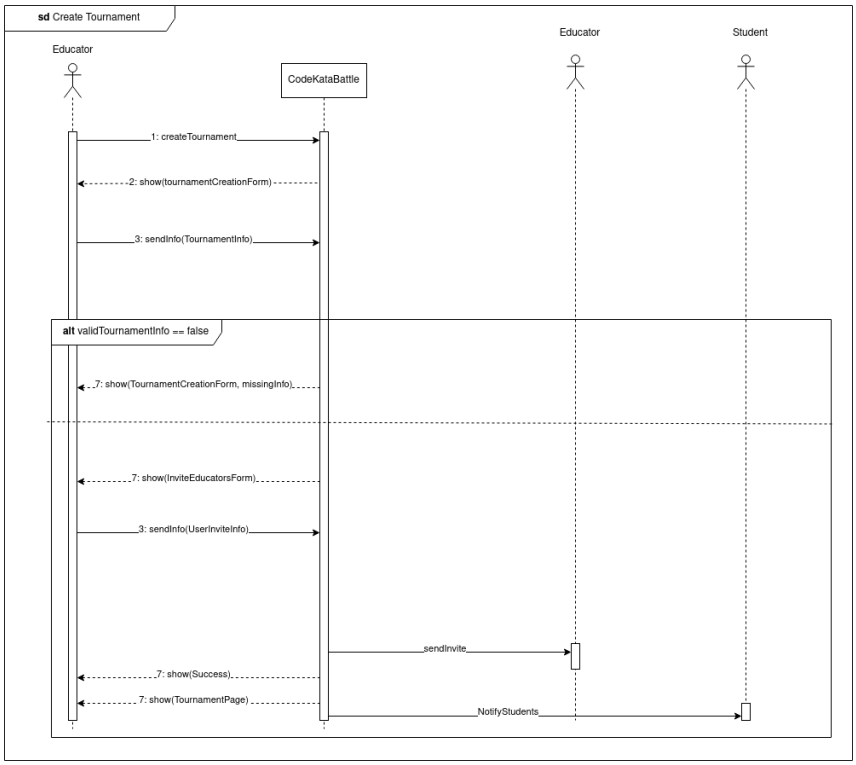

Figure 3.5: Create tournament sequence diagram

UC4: Subscribe to Tournament

Actor	Students
Entry condition	Student is logged into the platform
Event Flow	1. Student is notified of new tournament 2. Student presses "Join tournament" button
Exit	Student is subscribed to tournament
Exception	

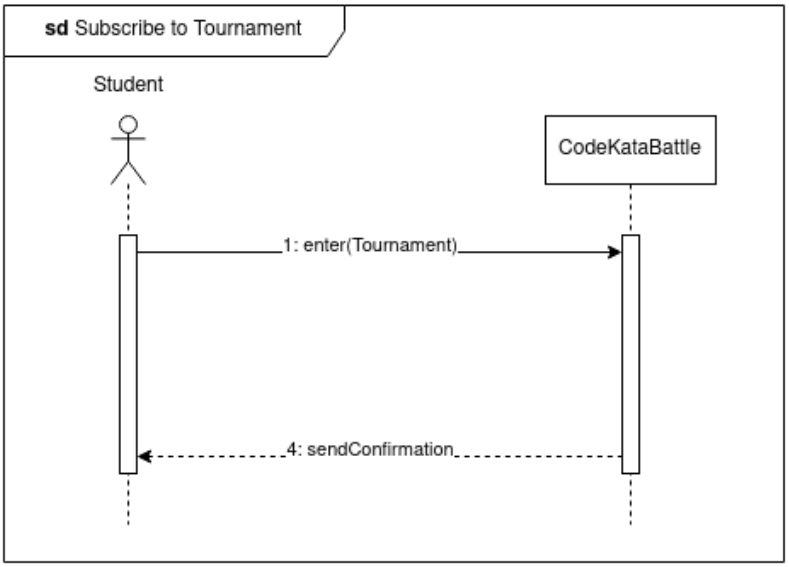

Figure 3.6: Subscribe to Tournament sequence diagram

UC5: Create Battle

Actor	Educator, Student
Entry condition	Educator has logged in and is inside the context of a tournament
Event Flow	1. Educator press "create new battle" button 2. Educator upload the CK assignment, tests and project build 3. Educator sets subscription and submission deadlines and group size rules 4. Educator complete battle creation 5. The platform sends notification to all the students subscribed to the tournament
Exit	Battle created correctly
Exception	Missing or wrong input by Educator

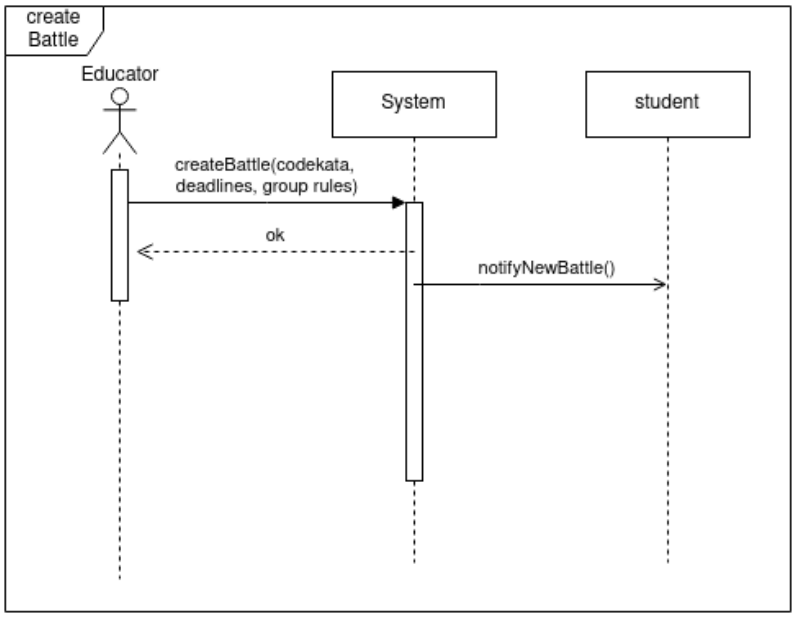

Figure 3.7: Create Battle sequence diagram

UC6: Join Battle

Actor	Student
Entry condition	Student subscribed to a tournament, student logged in, battle available for subscription in tournament
Event Flow	1. Student press "join battle" button in tournament context 2. platform shows group size rules 3. student uses "invite" button 4. Student inserts other students identifiers 5. other students accepts invite 6. Group of student is created 7. Group of students joins battle
Exit	Student group joins battle
Exception	

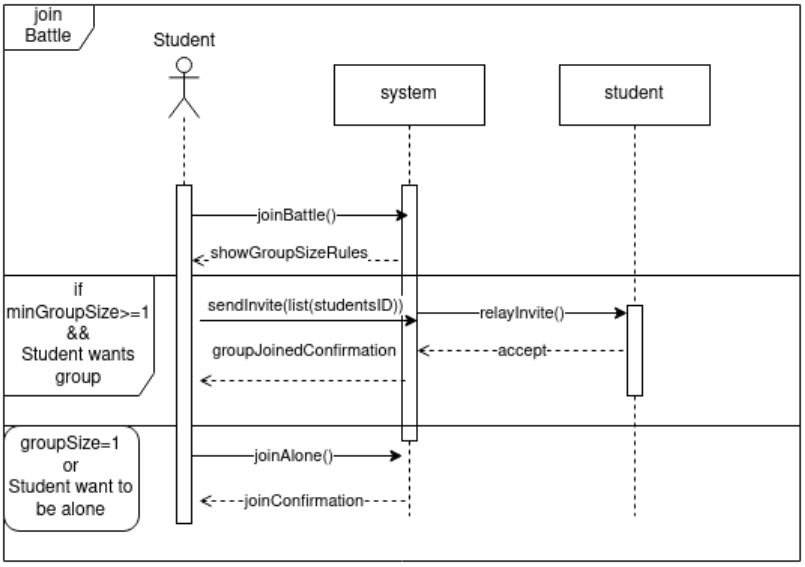

Figure 3.8: Join Battle sequence diagram

UC7: Create Repository

Actor	Github, Student
Entry condition	Subscription deadline of battle expired
Event Flow	1. The platform creates a repository for the codekata battle in github 2. The platform sends to all students subscribed to the battle the link to the repository
Exit	Student recieve repository link
Exception	

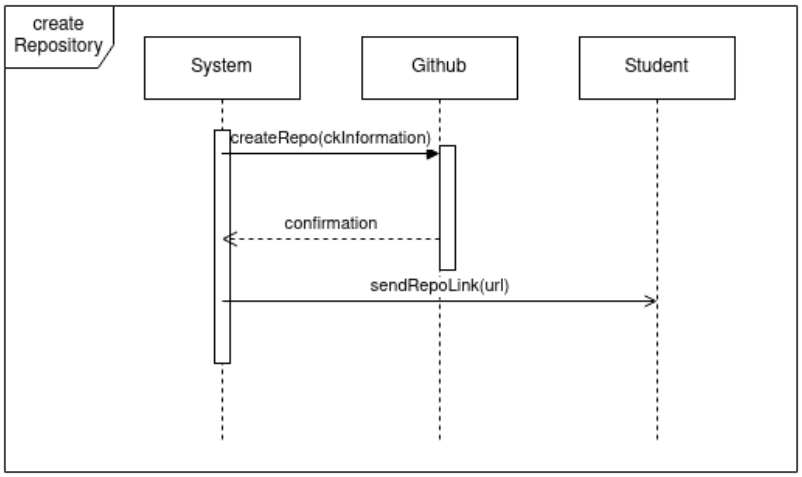

Figure 3.9: Create Repository sequence diagram

UC8: Score Commit

Actor	GitHub
Entry condition	Github notified platform of student commit
Event Flow	1. Platform receives notification of commit 2. Platform pulls sources and compiles them 3. Platform starts evaluation(functional, timeliness, quality) 4. Platform updates score for battle
Exit	Battle leader board is updated
Exception	Compilation of sources fails

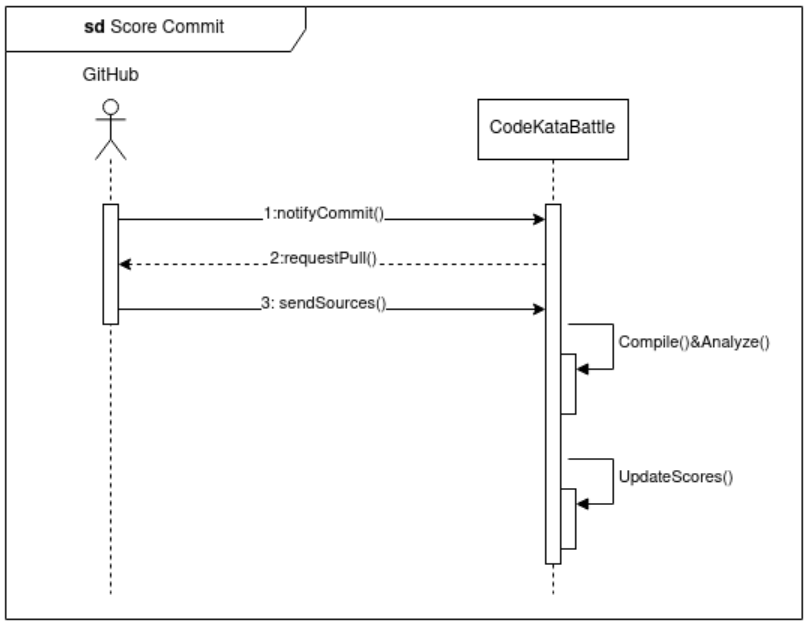

Figure 3.10: Score Commit sequence diagram

UC9: View Battle Ranking

Actor	User(Educators, Students)
Entry condition	Educator is involved with tournament of the battle, student has joined battle
Event Flow	1. User press "view ranking" button in battle context 2.platform shows ranking table
Exit	User sees ranking table
Exception	

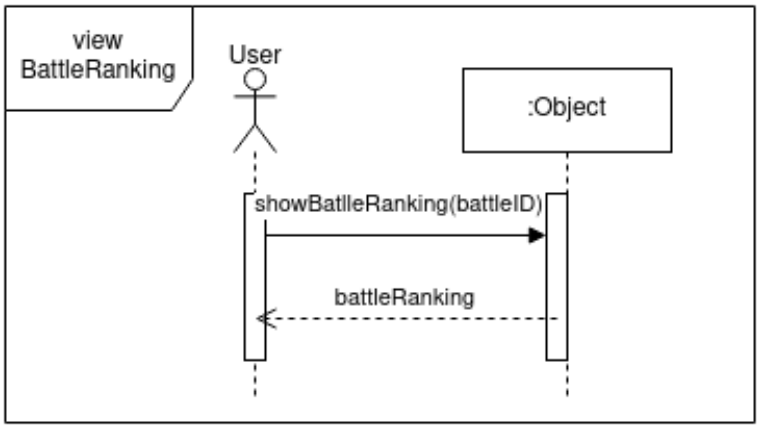

Figure 3.11: View Battle Ranking sequence diagram

UC10: Manual Evaluation

Actor	Educators
Entry condition	Battle deadline Expires
Event Flow	1. Platform notifies educators that battle ha ended and manual evaluation is required(as decided during battle creation) 2. Educators use platform to analyze sources
Exit	Educator add a manual score
Exception	Educator adds illegal score

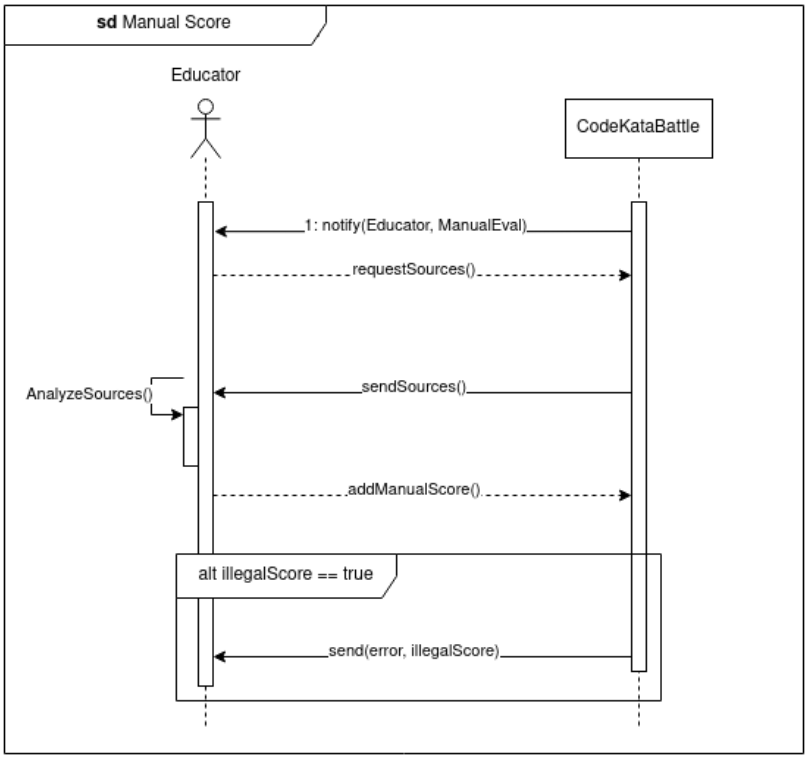

Figure 3.12: Manual Evaluation sequence diagram

UC11: Look at tournament ranks

Actor	Students, Educators
Entry condition	Student or Educator are on the main tournament page and are involved
Event Flow	Student or educator click on tournament leader-board
Exit	Platform shows Tournament leader board
Exception	Tournament hasn't yet started

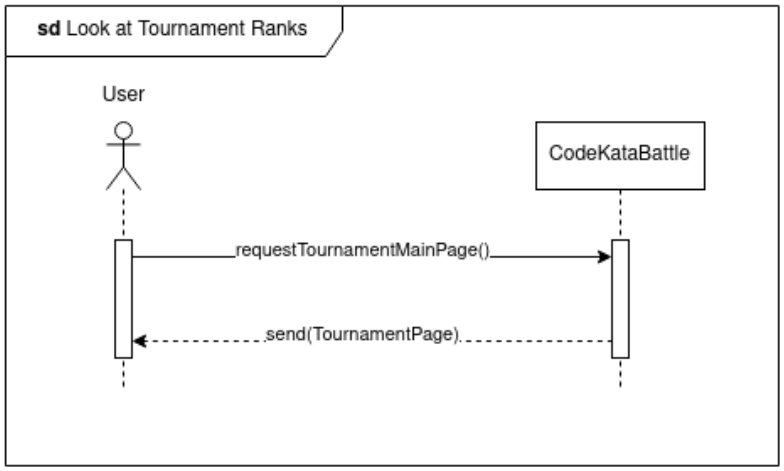

Figure 3.13: Look at tournament ranks sequence diagram

UC12: Close Tournament

Actor	Educator, Student
Entry condition	Educator logged in, Tournament still ongoing, no ongoing battle
Event Flow	1. Edu press "close tournament" in the tournament context 2. The platform closes the tournament 3. The platform notifies all the students subscribed to the tournament
Exit	Tournament closed, platform notifies students
Exception	

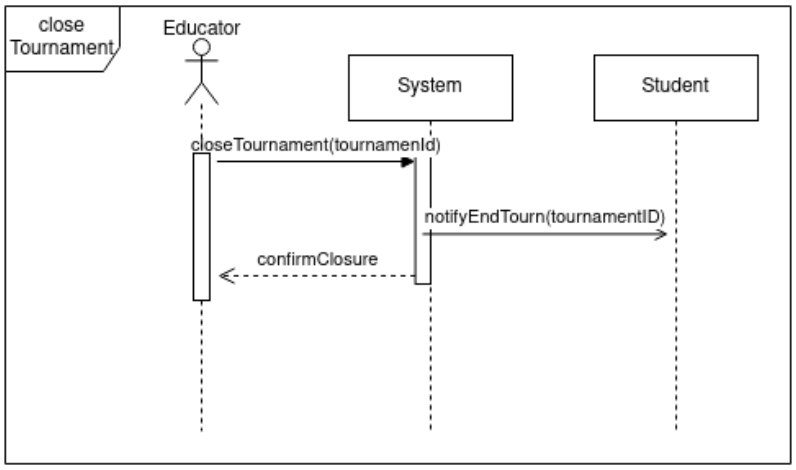

Figure 3.14: Close Tournament sequence diagram

3.2.5 Mapping on requirements

Use Case	Requirements
UC1	R10
UC2	R5, R6
UC3	R6, R2.1, R1, R14
UC4	R5, R3
UC5	R6, R2, R2.1, R2.2, R2.3, R2.4, R2.5, R15
UC6	R5, R4, R4.1, R15
UC7	R18, R19
UC8	R9, R9.1, R13, R13.1
UC9	R5, R6, R13, R13.1
UC10	R6, R7, R2.5, R16
UC11	R5, R6, R8, R8.1
UC12	R6, R17, R11

3.3 Design Constraints

3.3.1 Standards compliance

First of all, the system should respect all the laws regarding privacy and data treatment and exchange with third parties (i.e. CPOs); to work in Europe, the system should respect the EU GDPR. In particular, a general description of the main principles that data should have in order to guarantee their privacy is given in Art. 5 of the GDPR document.

3.3.2 Hardware limitations

There are no hardware contraints.

3.3.3 Any other constraint

There are no other constraints.

3.4 Software System Attributes

3.4.1 Reliability

The system should be able to restart automaticly shortly after an interruption and recover the previous backupped state.

3.4.2 Availability

The system should be operative at least 99% of the time to avoid the impairment of activities with deadlines and in case of downtime the system should recover any missed notification received from GitHub on group commit in battles when it restarts.

3.4.3 Security

The system should ensure safety of the user credentials and it should keep log of access and of sensible operations, such as in the case of educator closing a tournament or manually evaluation of sources, to avoid malicious use of the educator authority.

3.4.4 Maintainability

The development should follow good practice for maintainability, such as dividing the system in enough modules depending on the functionalities. This is necessary in order to facilitate maintenance and substitution of modules. Every functionality needs to be well documented.

3.4.5 Portability

The platform as a website should behave as expected on at least the most used browsers, without requiring additional effort from the end user. The back end should be written in a language that ensures portability and its functionalities should not be reliant on a particular compiler of system.

Chapter 4

Formal analysis using Alloy

4.1 Alloy Code

```
open util/integer

abstract sig User{}

sig Educator extends User{
   tournamentsInvolved : set Tournament
}

sig Student extends User{
   tournamentRank : disj set RankTournament,
   subscribedTournaments : set Tournament,
}

sig Group{
   students : some Student,
   battleRank: disj one RankBattle
}{this in Battle.currentGroups}

sig Tournament{
   battleList : disj some Battle,
   leaderBoard : disj one LeaderboardTournament
}{this in Educator.tournamentsInvolved }

sig Battle{
   currentGroups : disj set Group,
   assignment : disj one CodeKata,
   rules :disj one Rules,
   repo : disj lone Repository,
   leaderBoard: disj one LeaderboardBattle ,
    battleStatus : one BattleStatus
```

```
}{this in Tournament.battleList}

sig LeaderboardBattle{
   positions : disj set RankBattle
}{this in Battle.leaderBoard}

sig LeaderboardTournament{
   positions : disj set RankTournament
}{this in Tournament.leaderBoard}

sig RankTournament extends Rank{
    studentScore : disj one StudentScore
}{this in LeaderboardTournament.positions and this in
     Student.tournamentRank }

sig RankBattle extends Rank{
    battleScore : disj lone GroupScore
}{this in LeaderboardBattle.positions and this in Group.battleRank}

sig Repository{}{this in Battle.repo}

sig CodeKata{
   testCases : disj one TestCase
}{this in Battle.assignment}

sig TestCase{}{this in CodeKata.testCases}

sig Rules{
   minSize : Int,
   maxSize : Int,
   subscriptionDeadline :one DateTime,
   submissionDeadline:one DateTime
}{this in Battle.rules}

abstract sig Score{}
sig StudentScore extends Score{}{this in RankTournament.studentScore}
sig GroupScore extends Score{}{this in RankBattle.battleScore}

abstract sig Rank{
}

sig DateTime{}{this in Rules.submissionDeadline and this in
     Rules.subscriptionDeadline}

abstract sig TournamentStatus{}
one sig OpenTournament extends TournamentStatus{}
one sig ClosedTournament extends TournamentStatus{}

abstract sig BattleStatus{}
```

```
one sig SubscriptionOpen extends BattleStatus{} //no ranking no repo in
    battle,
one sig SubmissionOpen extends BattleStatus{}
one sig ConsolidationPhase extends BattleStatus{}
one sig BattleClosed extends BattleStatus{} //

///----------------------------------------------------FACTS----------------------

fact noRepoInSubscriptionPhase{
    all b : Battle | b.battleStatus=SubscriptionOpen iff (no b.repo)
}

fact minLessThanMaxSize{
   all r :Rules | (int r.minSize>=1 and int r.minSize<=int r.maxSize)
}
fact respectSizeRule{
   all b : Battle |
      (all g : Group |
         (g in b.currentGroups implies ( int #g.students>=int
             b.rules.minSize and int #g.students<=int b.rules.maxSize) )
      )
}

fact studentInOneGroupForBattle{
   all battle : Battle | (
      all disj g1,g2 : Group |( g1 in battle.currentGroups and g2 in
          battle.currentGroups implies (
         no s : Student | s in g1.students and s in g2.students)
      )
   )
}

fact rankAssociatedToBattleLeaderboard{
   all b : Battle | (
      all g : Group | g in b.currentGroups iff (g.battleRank in
          b.leaderBoard.positions)
   )
}
fact oneRankForTournament{//this implies that for every tournament the student
    is in there is only one rank and for every rank there is only one
    tournament subscribed
   all s: Student |(all t : Tournament | t in s.subscribedTournaments
       implies one r: Rank| r in s.tournamentRank and r in
       t.leaderBoard.positions)
              and (all r :RankTournament | r in s.tournamentRank implies
                  (one t : Tournament| t in s.subscribedTournaments and
                  r in t.leaderBoard.positions))

//without the 2 predicates there could be case were a student had 2 ranks for a
    tournament he wasnt in
```

```
}

fact ifInBattleThenInTournament{
   all t : Tournament | (
      all b : Battle | (
         all g : Group | (
            all st: Student | (g in b.currentGroups and st in g.students
                and b in t.battleList) implies t in
                st.subscribedTournaments
         )
      )
   )
}

fact mustBeGroupsInStartedBattle{
   all b :Battle| b.battleStatus!=SubscriptionOpen implies some g:Group
       |g in b.currentGroups

}

///-----------------------------------------------------PREDICATES---

pred noWrongRankLogic{//to test no student can have a score and rank within a
     tournament without being subscribed to said tournament
   (some s:Student| (some t: Tournament | t in s.subscribedTournaments
       and no r:Rank| r in s.tournamentRank) )
}

pred noStudentInTwoGroupBattle{//to test no student can join 2 groups for a
     battle
   some b : Battle| (some disj g1, g2 :Group| g1 in b.currentGroups
       and g2 in b.currentGroups and ( one s: Student | s in
       g1.students and s in g2.students)
   )
}


pred show{}

pred showGenericTournaments{
   some Tournament
   #Battle>1
}


pred showSomeRealistic{
   #Student > 4
   #Tournament > 1
   #Group > 1
```

```
   #Battle.currentGroups > 0
   Rules.maxSize<5
   one s : Student| no g:Group| s in g.students
}


pred showNoGroupBattle{//effectiviely a battle can have no subscribed groups
    only if it is within the subscription deadline
   one Battle
   lone s:Student | #s.subscribedTournaments>0
   no Group
}
pred joinNewBattle[s : Student]{//shows an instance where a student "s"
    joined with 2 other student a battle as a group
   one b : Battle| b.battleStatus=SubscriptionOpen and (one g:Group| s
       in g.students and g in b.currentGroups and #g.students=3)

}
pred joinTournament[s:Student]{//shows a student joining an existing
    tournament
   one t:Tournament| t in s.subscribedTournaments
   one Tournament
   one Student

}
run joinTournament for 6
```

4.2 Simulations

In this section we show some of the simulations of the built model.

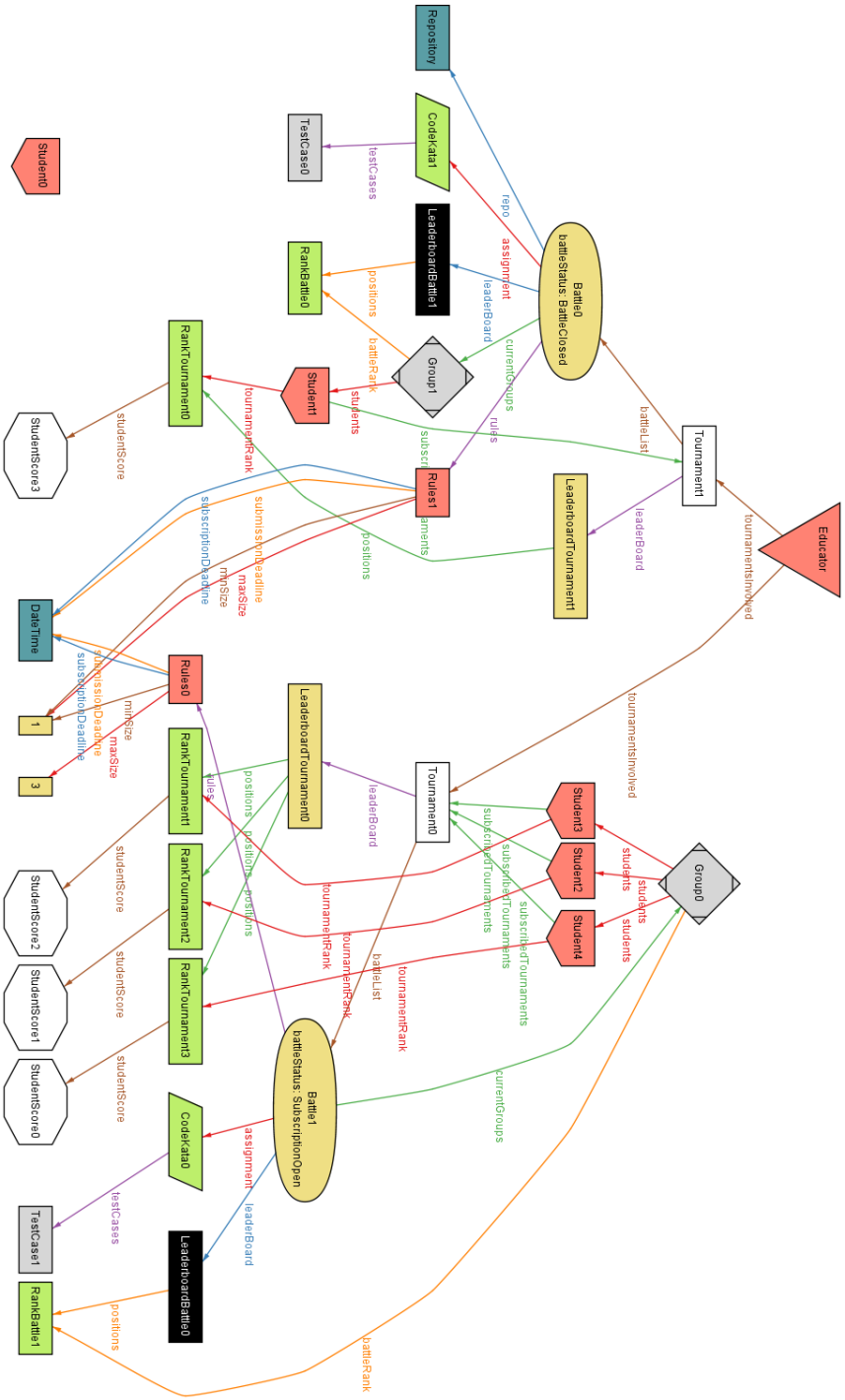

Figure 4.1: Simple world Alloy.

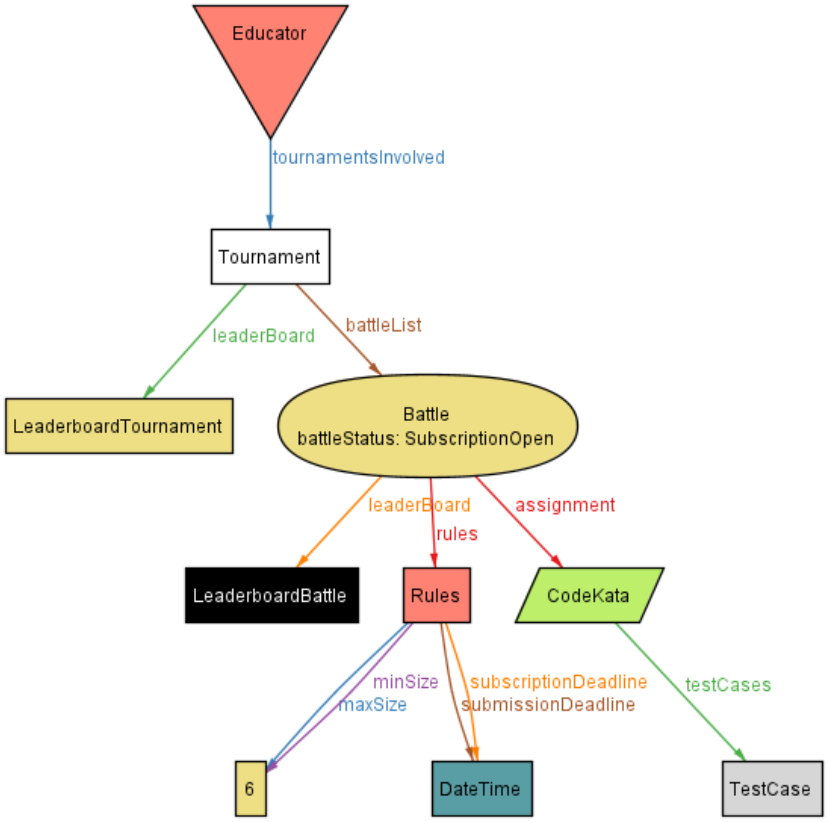

Figure 4.2: Battle with no groups

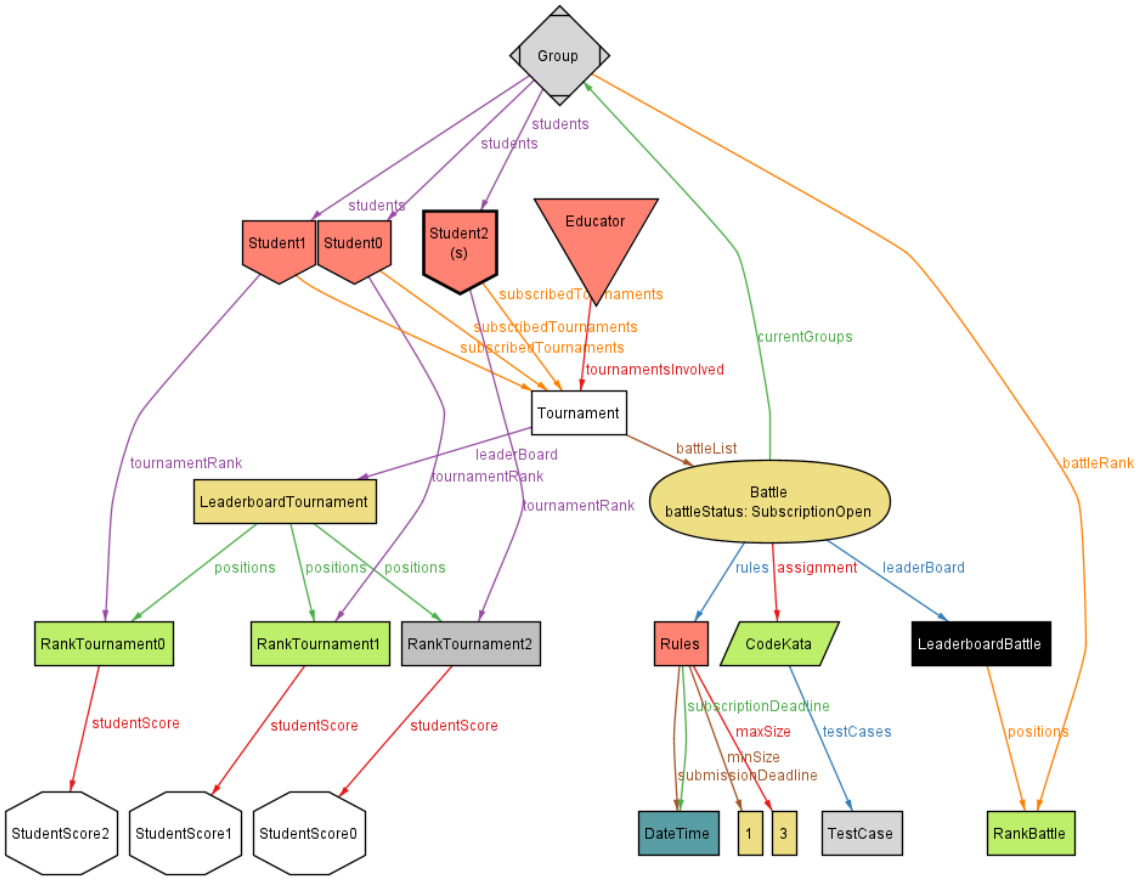

Figure 4.3: Student joins a battle with a group

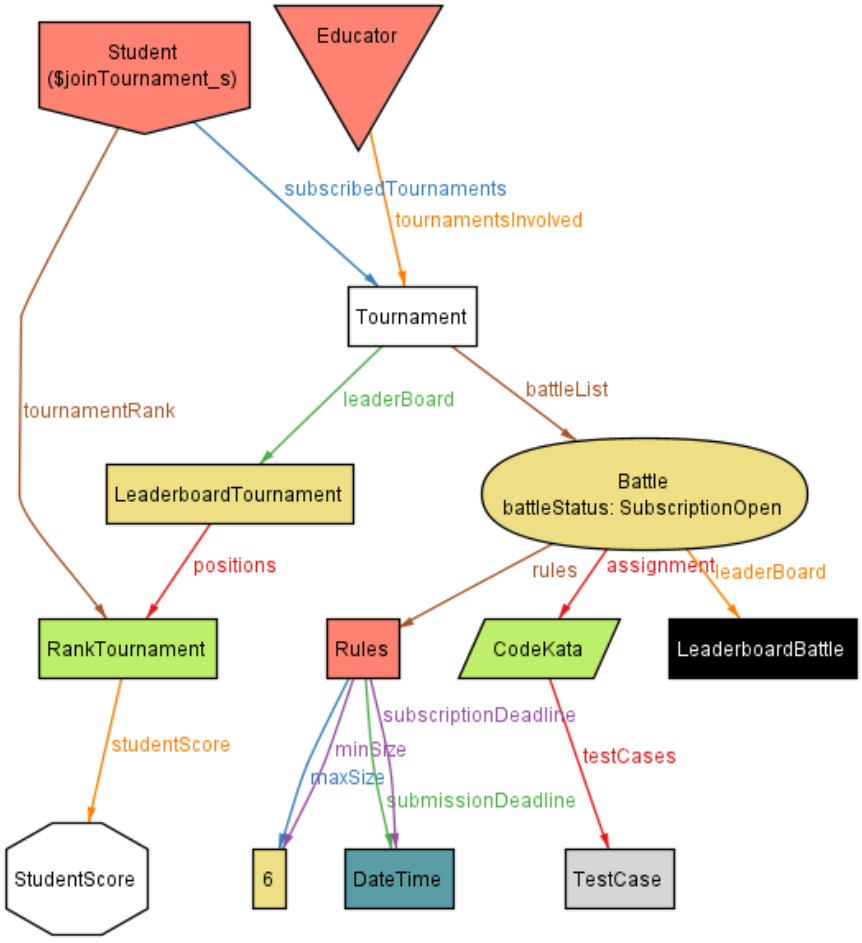

Figure 4.4: Student joins a tournament

Chapter 5

Effort spent

Chapter	Valentino	Zarbo
1	12	10
2	12	10
3	12	10
4	12	20

Table 5.1: Time Spent in hours for each chapter

Chapter 6

References

6.1 Used tools

- GitHub for project versioning
- Draw.io for UML diagrams
- Overleaf as LaTeX editor
- Alloy for formal analysis
- Visual Studio Code as Alloy editor