RASD:

Requirement Analysis and Specification Document

POLITECNICO
MILANO 1863

AY 2023/2024

Alessandro Fornara Simone Colecchia

Prof. Matteo Rossi

Version 2.0

February 3rd, 2024

INDICE

1. INTRODUCTION

A. Purposes and goals

A.1. Purpose

If you ever looked for a very intuitive and interactive platform to learn or to teach software development and programming, you are in the right place: CodeKataBattle might help you!

The main purpose of the platform is to facilitate the improvement of software development skills through competitive coding exercises known as "code kata". The primary objectives of the platform are as follows:

- Skill enhancement:

  All the subscribed students will enhance their software development skills through practical coding challenges. These challenges are presented in the form of code katas, which are exercises designed to encourage participants to make incremental improvements in their coding abilities.

- Assisted Teaching:

  Teachers can use CKB for many purposes such as giving their pupils homework, which are fully or partially automatically evaluated; create constructive competitions among the class or the institute, where the taught theory becomes practice; test the students with some code exercises of different levels of difficulty.

- Collaborative Learning:

  Students may have to form teams to participate in the tournaments and cooperate to solve the puzzles or gain points. This promotes teamwork and the sharing of knowledge and skills among peers.

A.2. GOALS

[G1]: All unregistered students must be able to subscribe and login to the CodeKataBattle platform.

[G2]: Educators must be able to create and close tournaments.

[G3]: Educators must be able to create battles, or grant permission to other colleagues to do so, and close their consolidation stage.

[G4]: All users subscribed to CKB must be able to see the information about public ongoing tournaments as well as the current rank for each one.

[G5]: All students must be able to subscribe to a tournament and form/join teams to participate in its battles.

[G6]: GitHub must be able to send and receive information to/from the system.

[G7]: All users involved in an ongoing battle must be able to see its current rank.

[G8]: Educators must be able to evaluate students through an automated evaluation and an additional personal evaluation in each battle.

[G9]: All students subscribed to CKB must be able to receive a notification by the system about the creation of a tournament.

[G10]: All students subscribed to a tournament must be able to receive a notification by the system about the creation of a battle for that tournament.

[G11]: All students who have subscribed to a battle must be able to receive notifications by the system about its start.

[G12]: Students who are involved in tournaments must be able to receive a notification by the system regarding battle final ranks and tournament final ranks.

B. Scope

B.1. Scope

CodeKataBattle allows users to organize and participate in code kata battles through tournaments, with the possibility of getting both an automated evaluation and an evaluation from an educator.

Each educator can create a tournament in which students can participate. Once the tournament has been created, a notification will be sent by the system to each student subscribed to CKB. After that the educator can start creating code kata battles.

The educator who created the tournament can also grant permission to other educators to create battles.

The students will be able to see public tournaments and will have the possibility of subscribing to them. Students can also access private tournaments through, for example, a specific keyword. After subscribing to a tournament, the student will receive a notification showing battles that have been created for that tournament and will also receive notifications for new battles once they are created.

To participate in a battle, each student will have to form teams with peers before the registration deadline, while adhering to the constraints about that specific battle. After the registration deadline has been reached, the system will create a GitHub repository containing the exercise, and each team will have to fork it and set up an automated workflow through GitHub Actions that informs the CKB platform as soon as students push a new commit into the main branch of their repository. For each push in the repository, the system will compute and update the team score (a natural number between 0 and 100) until the submission deadline of that battle is reached. After that, an educator can decide to add a manual evaluation to each student of the team.

After all battles of a tournament have ended, the creator of the tournament can close it. This will trigger the finalization of rankings: for each student, the system will sum the scores obtained in each battle and create the tournament final rank. Finally, the students that have participated in the tournament will be notified that the tournament final rank is available and will be able to see it.

B.2. Phenomena

The following table describes the world, shared and the machine phenomena, including the reference to which part controls the phenomena, according to the paper "The World and the Machine" by M.Jackson and P.Zave.

Phenomenon	Who controls it?	Is shared?
Teacher decides to use CodeKataBattle with his class	W	N
User registration	W	Y
User login	W	Y
Check username and password	M	N
Create a tournament	W	Y
Notify all users	M	Y
Click on a tournament	W	Y
Visualize tournament information	W	Y
Create team	W	Y
Register team in the database	M	N
Notify upcoming battles	M	Y
Join team	W	Y
Create battle	W	Y
Create GitHub repository	M	Y
Fork GitHub repository and set up automated workflows	W	Y
Automated evaluation	M	N
Additional evaluation	W	Y
Battle rank computation	M	N
Battle rank publication	W	Y
Tournament final rank computation	M	N
Tournament final rank publication	W	Y
Close tournament	W	Y

C. Definitions, Acronyms, Abbreviations

C.1. Definitions

- User:
  It is a generic user of the platform, who wants to teach (Educator), learn, or improve coding (Student). *(See 2.C. for further details)*

- Battle:
  It is an event with fixed starting and ending time, consisting of a fight among multiple teams, which have to solve a Code Kata and provide their solution.

- Tournament:
  It is a championship in which Students compete and it is composed of several sequential Battles.

- Public Tournament:
  It is a tournament that is visible and open for all the Students subscribed to CKB.

- Private Tournament:
  It is a tournament that is open only to a group of Students, who have been given the permission to visualize it and subscribe.

- Code Kata:
  A kata is an exercise in karate where you repeat a form many, many times, making little improvements in each. Code Kata is an attempt to bring this element of practice to software development. See more at https://codekata.com/.

- Tournament and Battle Final Rank:
  The final rank for a tournament or battle is the rank that is computed and finalized at the end of them. It will be definitive, and no changes will be applied to it.

- Team:
  It is a group of Students created to participate to a single Battle. Each team is composed by a certain number of Students between a minimum and a maximum value (fixed by the creator of the Battle).

- Keyword:
  It is a unique alphanumeric string which is automatically generated by the system to preserve privacy and security. It is used to protect access to Private Tournaments or Teams.

- Automated Workflow:
  A set of actions performed by an actor to automatically retrieve and push/send data whenever some changes are observed.

- Automatic Evaluation:
  It is an automatic computation of a team score performed by the system, based on the provided solution for a Code Kata and considering some criteria (set by the creator of the Battle).

- Manual Evaluation:
  It is a personal additional evaluation for a team score, which is performed manually by an Educator with respect to each member of the team and according to his discretion.

- GitHub:
  Hosting at https://github.com/, it is the most famous platform for software development and version control, allowing users to store, manage and share their code.

C.2. Acronyms

- CKB: CodeKataBattle, the name of the platform

- GDPR: General Data Protection Regulation is EU regulation for personal data and privacy.

- TCP/IP: Transmission Control Protocol and Internet Protocol.

- TLS: Transport Layer Security is a cryptographic protocol designed to guarantee communications security, providing authentication, data integrity and confidentiality.

- UTC: Universal Time Coordinated is the most used time standard.

C.3. Abbreviations

- **[Gn]** – Used to list all the goals, it stands for the n-th one.
- **[An]** – Used to list all the domain assumptions, it stands for the n-th one.
- **[RRLn]** – It points to the n-th requirement concerning registration and login functionalities.
- **[RTn]** – It points to the n-th requirement concerning tournament functionalities.
- **[RBn]** – It points to the n-th requirement concerning battles aspects.
- **[RGn]** – It points to the n-th requirement concerning GitHub.
- **[RNn]** – It points to the n-th requirement concerning notifications.
- **[UCn]** – It refers to the n-th use case provided.

D. Revision history

- ✓ **V1.0 –** First version of the document. Revised on 2nd December 2023.
- ✓ **V2.0 –** Second version of the document. Revised on 2nd February 2024
  - Changed assumption [A3]
  - Fixed some errors and inconsistencies in Use Cases

E. Reference Documents

- "Systems and software engineering —Life cycle processes — Requirements engineering"- ISO/IEC/IEEE 29148:2018(E).
- "The World and the Machine" by M. Jackson and P. Zave.
- "Assignment RDD AY 2023-2024".
- Unified Modeling Language specifications - https://www.omg.org/spec/UML/
- Alloy documentation - https://alloy.readthedocs.io/en/latest/

F. Document Structure

- **Section 1**: Introduction
  This section offers a brief introduction of the purposes and scope of the platform. It also contains the list of definitions, acronyms and abbreviations that could be found in this document. Finally, there are changelog of the document, containing the revisions list and their content, and document structure, which describes the main purposes of the sections of this document.

- **Section 2**: Overall Description
  This section offers a summary description about the overall organization of the system, providing some scenarios and a description of the main features offered by the application, and of the actors who use it. It also contains the considered Assumptions.

- **Section 3**: Specific Requirements
  This section contains a description of functional requirements through use cases and diagrams, along with the Hardware and Software constraints and the interfaces needed to get it work.

- **Section 4**: Formal Analysis through Alloy
  This section contains a formal analysis of the model presented in the previous sections.

2. OVERALL DESCRIPTION

A. Product perspective

A.1. Scenarios

- *Mr. Thompson's new class*

Mr. Thompson has just started teaching Computer Science at a school lab. He would like to make coding more interesting for all his students, even those who are not (yet) very enthusiastic about it. He decides to register on the CodeKataBattle platform as an Educator and then he creates a private tournament, sharing the entry code with the whole class.
His idea is to organize a battle for each weekend, in which he challenges the students upon the topics he has explained during the last week.
In this way he intends to light up a constructive competition among the pupils.

- *Sam and his team*

Sam goes to High School, and he decides to enroll in Mr. Thompson's class about Computer Science and Software Development. He has just got registered to CodeKataBattle platform, since Mr. Thompson asked him to do so, and now he can subscribe to the private tournament of his class entering the code the teacher sent him by e-mail.
At this point, Sam receives a notification from CKB that informs him about the battle starting the next Saturday. The notification also reports the principal information about the battle:

"DataTypes&Casting Battle"
Starting: Saturday 24th, October – 9:30 A.M.
Ending: Sunday 25th, October – 10:30 P.M.
Register before: Friday 23rd, October – 11:59 P.M.
Language: C++
Team members: between 2 and 4

Now Sam has to contact at least one friend to form up his team to participate in the following weekend battle.

- *Mr. Thompson and consolidation stage*

It's Monday and the first battle of Mr. Thompson's tournament has just ended. Now he can have a look at his students' work and revise the automatic evaluation the platform has provided on it. Through the interaction between the platform and GitHub, he can see some statistics that are relevant for the actual evaluation of his pupils.
Inspecting the results and the insights of a team called "CodeSenseis" (3 members), he notices that the score they have achieved is *85* and that Student1 has worked more than Student2 and Student3. Then, he decides to add 5 additional points to Student1 and to subtract 1 point from Student2 and Student3. Now the final score achieved by each member of the team is:

Student1: 90
Student2: 84
Student3: 84

- *Co-managed Tournament*

Mr. Smith, Mr. Johnson and Mrs. Brown are three Computer Science professors at the University of Milan. They give the same lectures on Java coding and Object-Oriented Programming to different students. At the end of April, they will have explained the basics of the OOP paradigm and the most important features of programming in Java, but they want to find a way to create a competition among the three classes to see if all the students have got to the same level. So, they are looking for a platform which allows them to create a coding challenge that is spread across all the students, automatically evaluated and that can be managed by the three of them.
CKB offers them the possibility to create a private tournament (sharing a keyword to all the students) which is fully automatically evaluated with the possibility to intervene in them.
So, Mr. Johnson creates the tournament, becoming the 'Admin', and makes Mr. Smith and Mrs. Brown 'Moderators' of the tournament such to help him with the additional evaluations.

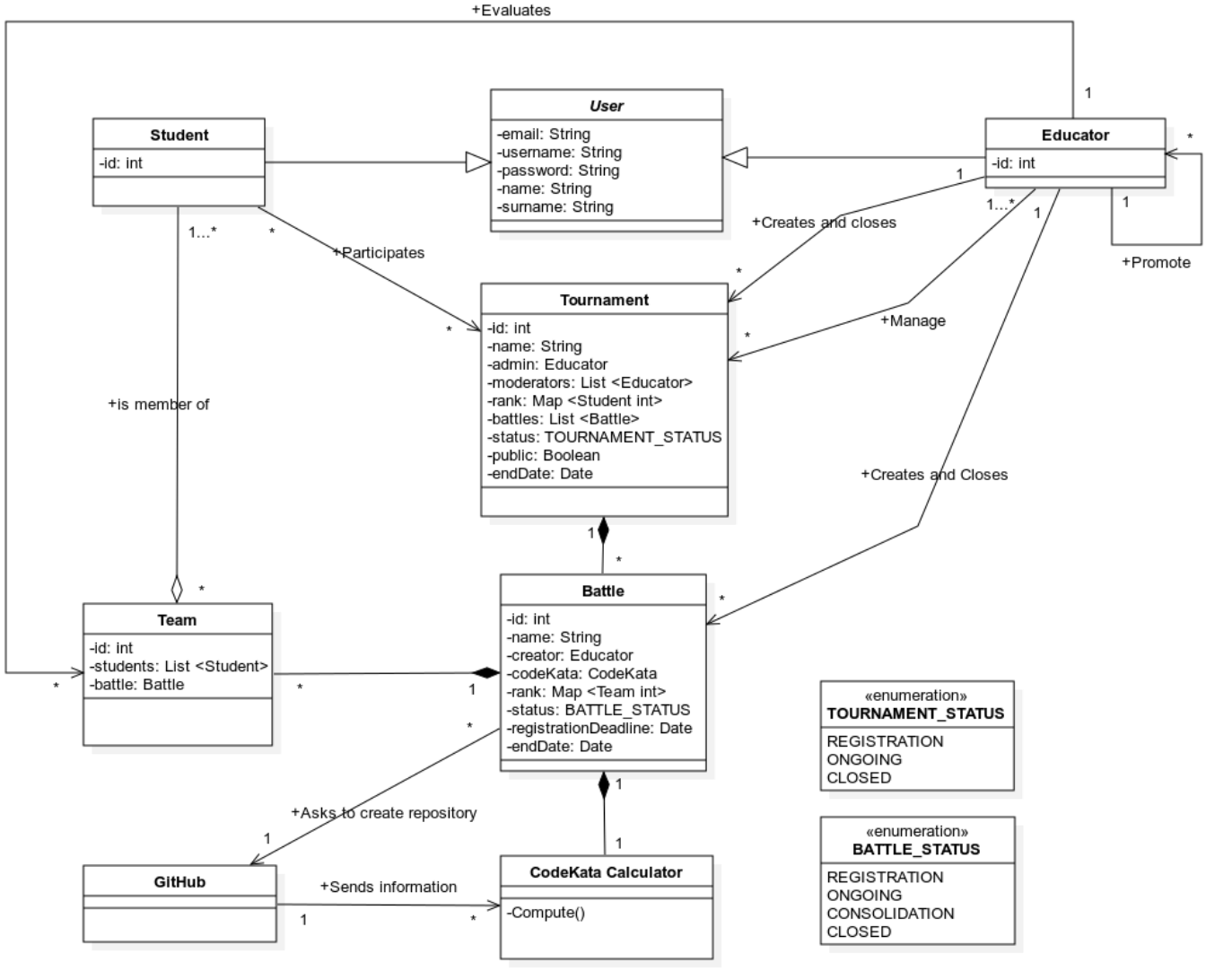
- *CKB for fun and improvement!*

  Philip is a bachelor student in Mathematics. He is very interested in coding but the courses he is attending now are not practical enough. So, he is looking for some online platform to improve his coding skills and compete against other students. After hearing about CKB platform from a friend of him, he decides to register.
  Then, Philip receives a notification about a public tournament starting next week, discovering that the main programming language to be used is the one he is most experienced in. So, he decides to subscribe and give it a try!

- *Coding for everyone*

  Mr. Miller got graduated in Computer Science two years ago and this year he is teaching a course of coding for all ages at the local library.
  During the first lecture, he asks his students to register to CodeKataBattle platform and to enter the code for his private tournament. He wants to make things practical and less boring, so for each lesson he asks his students to subscribe to the battle he creates on the spot to see if all the attendants have got the point of the lesson.
  The platform allows him to set some parameters such as the minimum and maximum number of students per each team, the programming language of the battle and some configuration parameters for the automated evaluation. Then he can start uploading the Code Kata for his students while they subscribe.

A.2. Domain-level diagram



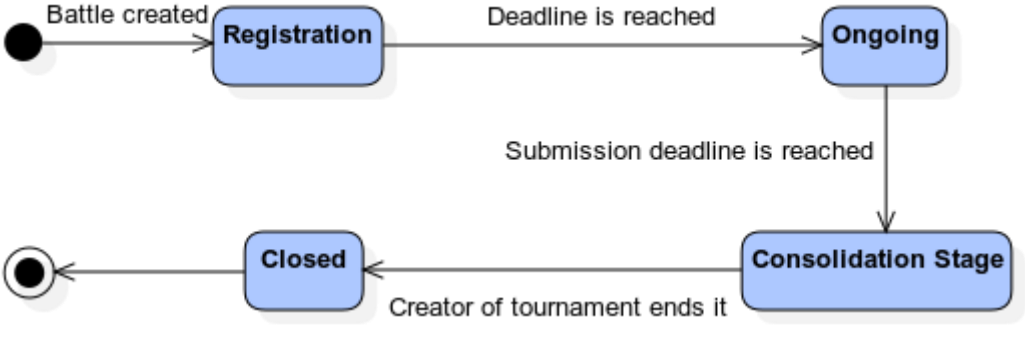
A.3. Statecharts

The statecharts discussed in this section are necessary to have a better understanding of the main processes of the platform.

a) Tournament

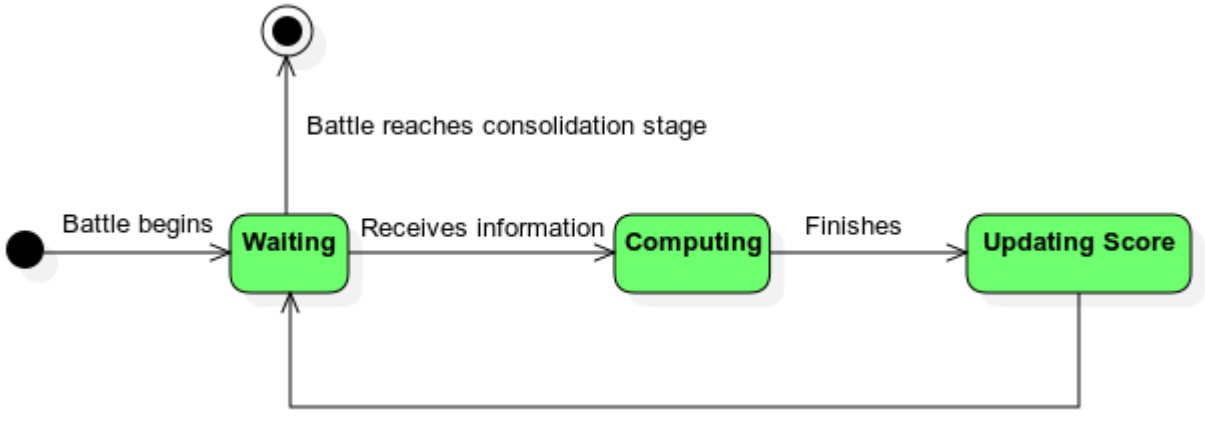
The initial state after a tournament is created is the Registration state. In this phase the system is waiting for students to subscribe to the tournament. As soon as the registration deadline is reached, the tournament begins and the process goes into the Ongoing state, during which the students will be able to compete, subscribe to battles etc. After all battles have ended, the creator of the tournament can decide to close it, making the process reach the Closed state, during which the final rank will be visible to all students who are notified.

b) Battle



The initial state after a battle is created is the Registration state. In this phase the system is waiting for teams of students to subscribe to the battle. After the registration deadline is reached, the battle begins and the process goes into the Ongoing state, during which the students will have to work on the provided problem. After the submission deadline is reached, the process goes into the Consolidation Stage state, during which the creator of the battle, and other educators that were given permission, can decide to add their own additional evaluation to any team. After they're done evaluating, the creator of the tournament will end the consolidation stage and the process will reach the Closed state. Both the battle and tournament ranking of each student will be updated and published, and each student will be notified.

c) CodeKata Calculator



After the battle begins the calculator goes into the Waiting state. Whenever the calculator receives from GitHub information about a new push from a team, it goes into the Computing state during which it will assign a score to the received code. After the calculator is done, it goes into the Updating Score state during which it updates the rank of the battle and after that it returns to the Waiting state.
When the battle goes into the Consolidation stage, the calculator will reach the final state.

B. Product functions

- *Sign up and Login*

These functions will be available to all the users. The sign-up functionality allows users to create an account to register themselves to the platform. Each user will be asked to select if he/she is an educator or a student and provide personal data such as name, surname, email, username and password. The login functionality allows users to access an existing account using the credentials chosen at registration.

- *Create tournament*

This function allows educators to create a tournament for students. The user will be asked to provide a name, a registration deadline and to select if the tournament should be private or public. If the private option is selected, then the user will be provided a keyword

that students can use to join the tournament. If the public option is selected the system will automatically notify all students subscribed to the CKB platform.

- *Join tournament*

  This function allows students to subscribe to tournaments. If it's a private tournament, the students can access it by providing a keyword. The keyword is generated by the system and must be provided to the user by the creator of the tournament.

- *Create battle*

  This function allows the creator of a tournament, and those who have received permission, to create a code kata battle. The user will be asked to set some preferences (for example: programming language, registration deadline etc.). Then, the system will automatically notify all students subscribed to the tournament.

- *Join battle*

  This function allows students to subscribe to a battle. The user will be asked to choose if he/she wants to create a team or join a team. If the first option is selected, then the user will be asked to provide a name for the team. If the second option is selected, then the user will be asked to provide the keyword corresponding to the team he wants to join. The keyword is generated by the system when the team is created and must be provided to the user by the creator of the team.

- *Evaluate*

  This function allows the creator of a battle, and those who have received permission, to add their own evaluation on the code of any team. The user will be asked to provide a number and select a student from a team. The number will be added to the personal score of the student.

- *Close tournament*

  This function allows the creator of a tournament to close it. This event is going to trigger the finalization of the tournament rank and the system will automatically notify the students subscribed to the tournament.

- *Promote to moderator*

  This function allows the creator of a tournament to promote other educators to ”Moderators” of the tournament. The user will be asked to provide the email of the educator he/she wishes to promote. The system will automatically notify the new “Moderator” of the tournament.

C. User characteristics

Here it is provided a more detailed characterization of the two different types of users of the platform, also specifying the roles they can assume in different contexts.

A. Educator

An Educator can be whoever wants to teach software development to a group of students in general terms. After his/her registration, he/she gets the functionalities which are reserved for Educators such as Creating, Managing and Closing a Tournament or a Battle, Evaluate the students’ result etc.

More precisely, within the context of a Tournament an Educator can cover two different roles, each of which has the ability to perform different actions.

- ***Admin***:
  When an Educator creates a tournament, he/she becomes the Admin of that tournament. He/she can:
  - Promote another Educator to “Moderator” of the tournament, granting him/her some special permissions on that tournament (see below).
  - Create Battles by setting some parameters like starting and finishing time, deadlines, number of members per team etc.
  - Evaluate a Team, which means adjusting the automatic evaluation performed by the system by optionally adding or subtracting extra points to each member of the team.
  - Close the Consolidation Stage, thus interrupting the possibility for educators to further evaluate the teams.
  - Close the Tournament, when all the Battles are completed and he doesn’t want to create any other.

- ***Moderator***:
  When promoted by an Admin of a certain tournament, he/she can perform only these actions:
  - Create Battles
  - Evaluate a Team

B. Student

A Student is an individual whose aim is to improve him or her programming skills through Code Kata battles. He can be either a pupil who has complied with the tasks assigned by his teacher/professor, or even an enthusiast who wants to improve by challenging himself. He/she must register to the platform to start enjoying it!

D. Assumptions, dependencies, and constraints

[A1]: The user always selects the correct option about his/her position (student/educator).

[A2]: All users have an active email address.

[A3]: Educators involved in the same tournament coordinate well on the evaluation of teams, ensuring that they do not overwrite each other's rating.

[A4]: All users have a GitHub account.

[A5]: All Students are capable of forking a repository on GitHub and set up an automated workflow through GitHub Actions that informs the CKB platform for each push. The name of the repository will be a combination of the surnames of the students (example: FornaraColecchia).

[A6]: GitHub is always available and responds to all the request performed by the API, always providing for correct information. All the requests are satisfied within a reasonable time.

3. SPECIFIC REQUIREMENTS

A. External Interface Requirements

A.1. *User Interfaces*

The platform is featured with several Graphic User Interfaces such to allow all the different kinds of users to interact with all its functionalities. The most important ones are presented below:

- Login/Registration Interface
  This interface provides for two different forms to fill in personal data, either to register to the platform or to access the personalized Home Page.

- Home Page:
  In this page the user visualizes key information about all the upcoming and ongoing public Tournaments. If he/she is a Student, he/she has the possibility to insert keywords to join private ones. If he/she is an Educator instead, he/she can create new tournaments.

- My Tournaments / My Battles Interface:
  Here the user can have a direct look at the Tournaments or Battles he/she is subscribed to, and directly navigate to the corresponding Info Page.

- Tournament / Battle Info Page:
  This interface supplies the main information regarding a specific Tournament or a Battle (such as creator, status, deadlines, rank and so on) and provides the access to some functionalities to interact with them (such as subscribe, create team, create battle, promote to moderator etc.).

- Profile Page:
  With this interface, the user is able to review his/her personal data set during Registration.

A.2. *Hardware Interfaces*

The platform does not provide any hardware interface since it is primarily a coding platform: it does not require any external component or device besides the one it's running on.

A.3. *Software Interfaces*

The software, through an appropriate API, communicates with GitHub to manage some essential aspects of battles. As previously mentioned in section 1.B., the CKB platform will use the API to create a repository on GitHub. Furthermore, the platform will keep track of students' progresses in a battle through the API, thanks to the automated workflow set up by them autonomously.

A.4. *Communication Interfaces*

The platform exploits the internet connection for the communication to the main server, whose role is to manage all back-end functions such as storing data, responding to deadlines, computing scores and so on.

B. Functional Requirements

B.1. Functional requirements

- REGISTRATION and LOGIN:

  [RRL1] - The system should provide a "Registration" functionality.
  [RRL2] - The system should provide a "Login" functionality.

- TOURNAMENTS:

[RT0] - The platform should allow users to see information regarding all ongoing and upcoming public tournaments, including the current rank.

- Educators:

[RT1] - The platform should provide the “Create Tournament” functionality.
[RT2] - The platform should provide the “Close Tournament” functionality.
[RT3] - The platform should provide the “Promote to Moderator” functionality.

- Students:

[RT4] - For each tournament, the platform should provide the principal information and the “Subscribe” functionality.

- BATTLES:

[RB0] - The platform should allow users involved in a battle to see its current rank.
[RB1] - The platform should be able to perform an automated evaluation of provided code based on functional aspects, timeliness and quality level of sources.

- Educators:

[RB2] - The platform should provide the “Create Battle” functionality.
[RB3] - During the consolidation stage, the platform should provide the “Evaluate” functionality.
[RB4] - For each battle, the platform should provide the “Close Consolidation Stage” functionality.

- Students:

[RB5] - For each battle, the platform should provide the “Create Team” functionality.
[RB6] - For each battle, the platform should provide the “Join Team” functionality.

- GITHUB:

  [RG1] - The system should be able to request GitHub to create a repository and receive back the corresponding link.
  [RG2] - The system should be able to get information regarding new pushes of code on GitHub.

- NOTIFICATIONS:

  [RN1] - The platform should notify all students when a new tournament is created.
  [RN2] - When a student subscribes to a tournament, the platform should send a notification about all upcoming battles.
  [RN3] - When a battle within a tournament is created, the platform should notify all students subscribed to that tournament.
  [RN4] - When a battle starts, the platform should notify all members of each team by sending the link to the repository on GitHub.
  [RN5] - When the consolidation period of a battle is completed, the platform should notify all the participating students about the battle final rank.
  [RN6] - When a tournament is closed, the platform should notify all participating students about the tournament final rank.
  [RN7] - When an Educator gets promoted to Moderator, the platform should notify him/her.

B.2. Requirements mapping

[G1] - All unregistered students must be able to subscribe and login to the CodeKataBattle platform.	
• **[RRL1] -** The system should provide a “Registration” functionality. • **[RRL2] -** The system should provide a “Login” functionality.	• **[A1]** • **[A2]**

[G2] - Educators must be able to create and close tournaments.	
• **[RT1]** - The platform should provide the “Create Tournament” functionality. • **[RT2]** - The platform should provide the “Close Tournament” functionality.	

[G3] - Educators must be able to create battles, or grant permission to other colleagues to do so, and close their consolidation stage.	
• **[RT3]** - The platform should provide the “Promote to Moderator” functionality. • **[RB2]** - The platform should provide the “Create Battle” functionality. • **[RB4]** - For each battle, the platform should provide the “Close Consolidation Stage” functionality.	• **[A3]**

[G4] - All users subscribed to CKB must be able to see the information about public ongoing tournaments as well as the current rank for each one.	
• **[RT0] -** The platform should allow users to see information regarding all ongoing and upcoming public tournaments, including the current rank.	

[G5] - All students must be able to subscribe to a tournament and form/join teams to participate in its battles.	
• **[RT4] -** For each tournament, the platform should provide the principal information and the "Subscribe" functionality. • **[RB5]** - For each battle, the platform should provide the "Create Team" functionality. • **[RB6]** - For each battle, the platform should provide the "Join Team" functionality.	

[G6] - GitHub must be able to communicate with the system	
• **[RG1]** - The system should be able to request GitHub to create a repository and receive back the corresponding link. • **[RG2] -** The system should be able to get information regarding new pushes of code on GitHub**.**	• **[A4]** • **[A5]** • **[A6]**

[G7] - All users involved in an ongoing battle must be able to see its current rank.	
• **[RB0] -** The platform should allow users involved in a battle to see its current rank. • **[RB1] -** The platform should be able to perform an automated evaluation of provided code based on functional aspects, timeliness and quality level of sources.	

[G8] - Educators must be able to evaluate students through an automated evaluation and an additional personal evaluation in each battle.	
• **[RB1] -** The platform should be able to perform an automated evaluation of provided code based on functional aspects, timeliness and quality level of sources. • **[RB3] -** During the consolidation stage, the platform should provide the “Evaluate” functionality. • **[RB4]** - For each battle, the platform should provide the “Close Consolidation Stage” functionality.	

[G9] - All students subscribed to CKB must be able to receive a notification by the system about the creation of a tournament.	
• **[RT1] -** The platform should provide the “Create Tournament” functionality. • **[RN1] -** The platform should notify all students when a new tournament is created.	• **[A2]**

[G10] - All students subscribed to a tournament must be able to receive a notification by the system about the creation of a battle for that tournament.	
• **[RT4] -** For each tournament, the platform should provide the principal information and the “Subscribe” functionality. • **[RB2] -** The platform should provide the “Create Battle” functionality. • **[RN2] -** When a student subscribes to a tournament, the platform should send a	• **[A2]**

notification about all upcoming battles. • **[RN3]** - When a battle within a tournament is created, the platform should notify all students subscribed to that tournament.	

[G11] - All students who have subscribed to a battle should receive notifications about its start.	
• **[RB5]** - For each battle, the platform should provide the “Create Team” functionality. • **[RB6]** - For each battle, the platform should provide the “Join Team” functionality. • **[RN4]** - When a battle starts, the platform should notify all members of each team by sending the link to the repository on GitHub.	• **[A2]**

[G12] - Students who are involved in tournaments must be able to receive a notification by the system regarding battle final ranks and tournament final ranks.	
• **[RB1] -** The platform should be able to perform an automated evaluation of provided code based on functional aspects, timeliness and quality level of sources. • **[RB4] -** For each battle, the platform should provide the “Close Consolidation Stage” functionality.	• **[A2]**

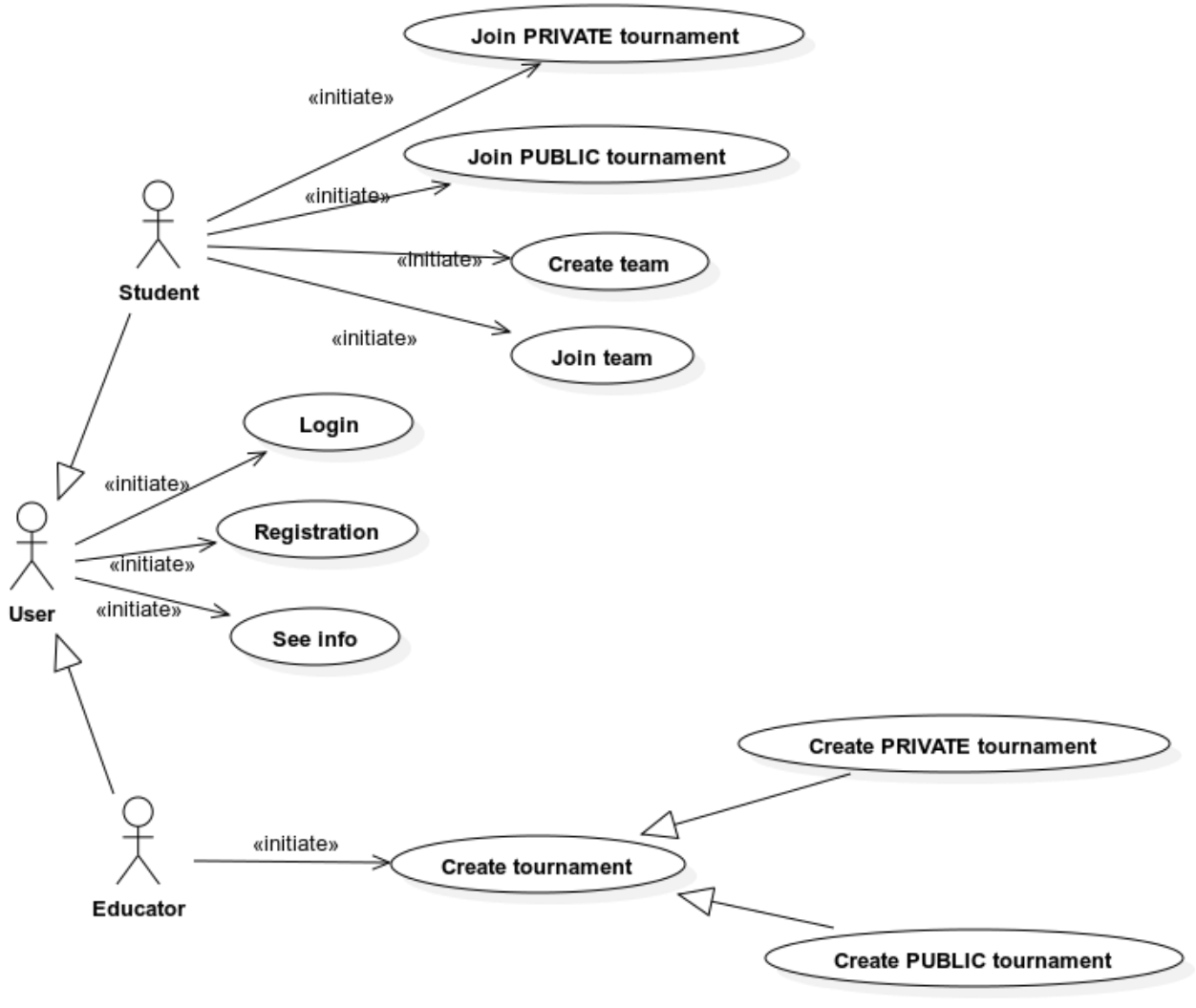
• **[RN5] -** When the consolidation period of a battle is completed, the platform should notify all the participating students about the battle final rank. • **[RT2] -** The platform should provide the “Close Tournament” functionality. • **[RN6] -** When a tournament is closed, the platform should notify all participating students about the tournament final rank.	

B.3. Use cases and diagrams

In this section, some of the most significant Use Cases for CKB platform have been represented, dividing them into two groups, one for each category of user (as defined in the section 2.C.).

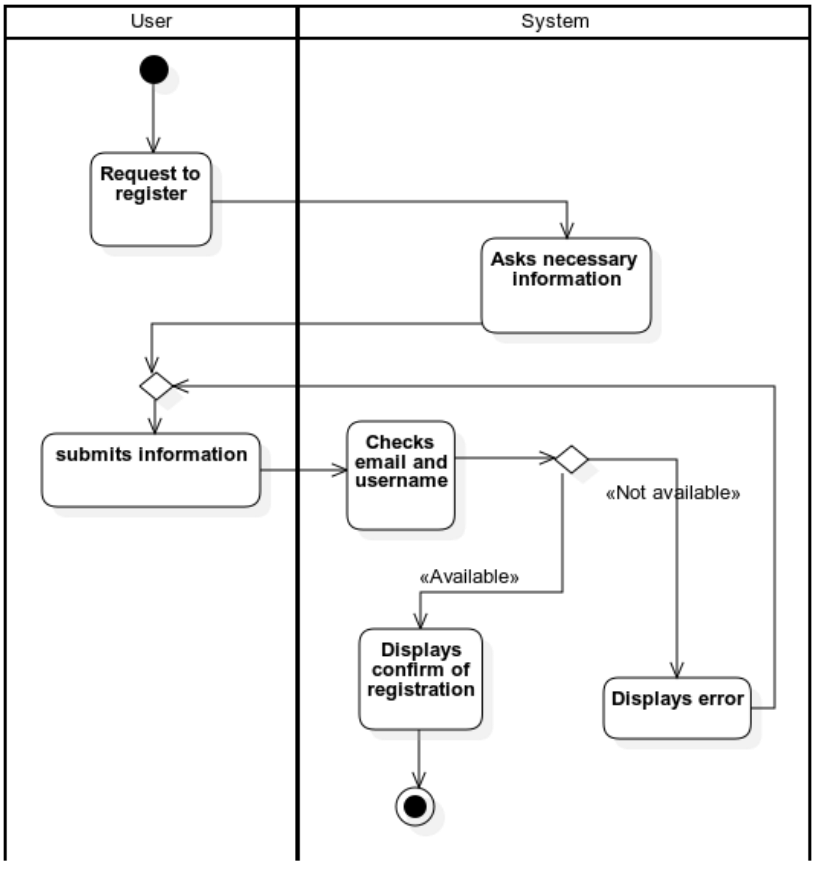
N.B.:
- All the actors which are denoted in **blue**, are involved in a use-case only by mean of some kind of notification.
- All the exceptions included in use-cases are represented in the related activity/sequence diagram only if they don't lead to the termination of the process.

STUDENT and EDUCATOR:



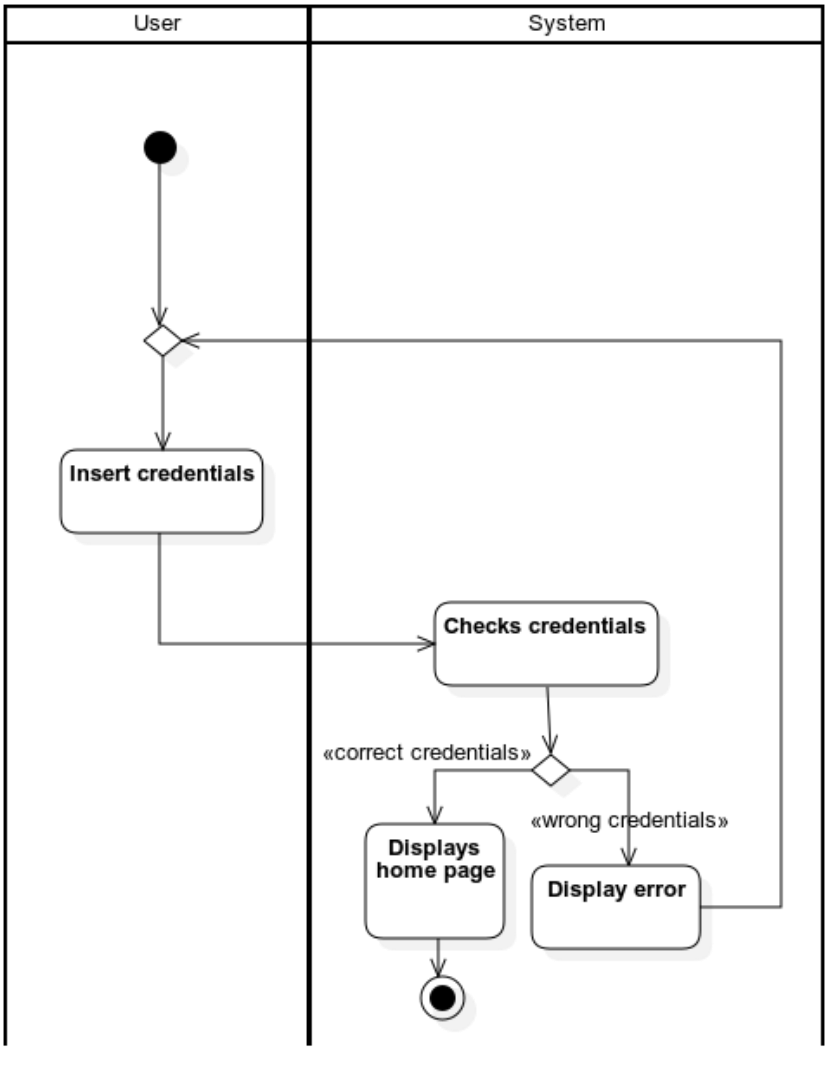
[UC1]

Name	Register user
Actors	User
Entry conditions	User has opened the platform
Events flow	1. The User requests to register 2. The system asks the user to choose if he/she is a Student or Educator and to provide: email, username, name, surname, password 3. User submits all necessary information 4. The system checks if email or username have been already used by another user to register 5. System updates the database with the User's information and displays a message of confirmed registration
Exit Condition	User has successfully registered
Exception 1	User provides an email or username already registered in the database. The system displays an error



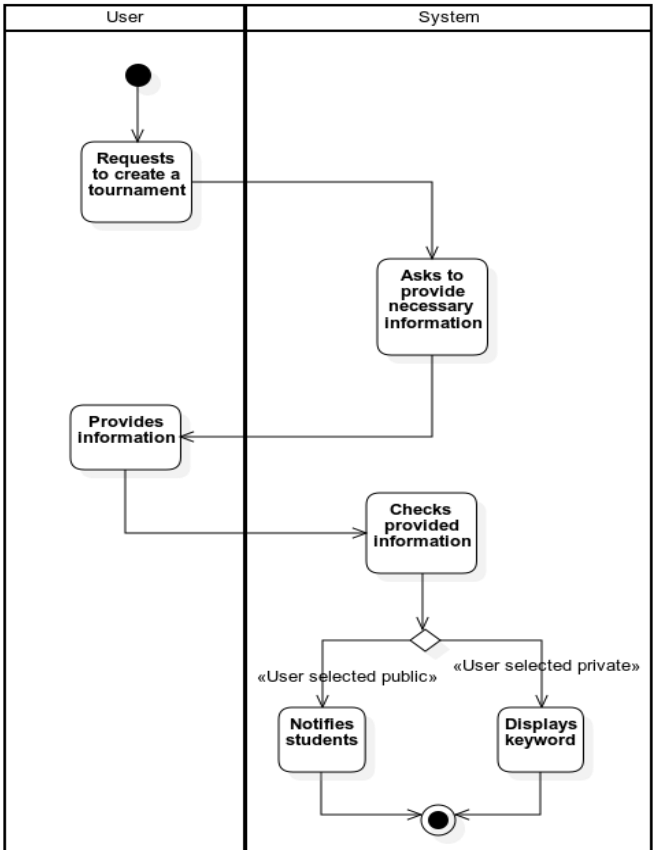
[UC2]

Name	Login user
Actors	User
Entry conditions	• User has registered to the platform
Events flow	1. User inserts in apposite fields the credentials for logging in (username and password) and presses the “Login” button 2. The system checks the correctness of the credentials inserted 3. The system displays the home page
Exit Condition	User is logged in
Exception	User inserts wrong combination of credentials and presses Login button. In this case, the application displays the Login page with an error



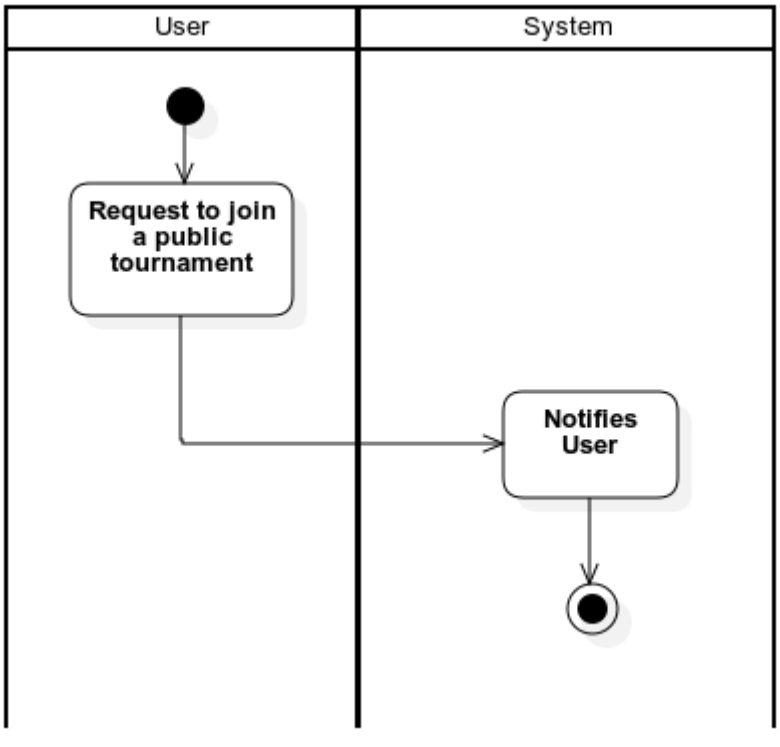
[UC3]

Name	Create tournament
Actors	User (Educator), Student
Entry conditions	User has logged in as Educator
Events flow	1. User requests to create a tournament 2. The system asks the User to choose if the tournament should be private or public, provide a name and a registration deadline. 3. User submits the requested information 4. The system checks the information provided by the User 5. The system updates the database, and notifies all Students subscribed to CKB
Alternate 1	This flow is taken after 5 if the User selected “private” 6. The system shows to the User the needed keyword to join the tournament
Exit Condition	User has successfully created a tournament and Students subscribed to CKB have been notified if “public”, the system showed the keyword if “private”
Exception 1	User provides a name which is not available. The system displays an error



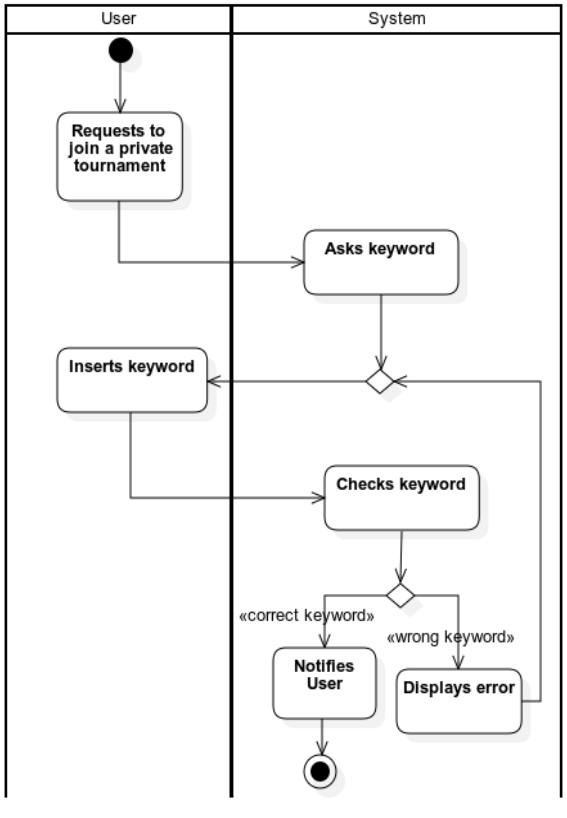
[UC4]

Name	Join PUBLIC tournament
Actors	User (Student)
Entry conditions	• User has logged in as Student • The tournament is in the *Registration* state
Events flow	1. User selects a tournament 2. The system displays the tournament information page 3. Student requests to subscribe to the tournament 4. The system updates the database and notifies the User about upcoming battles
Exit Condition	User has correctly subscribed to the tournament and has been notified about upcoming battles
Exception 1	The user is already subscribed to the tournament, the system displays an error
Exception 2	The tournament has moved from the *Registration* state to the *Ongoing* state before the user managed to complete the subscription. The system will display an error



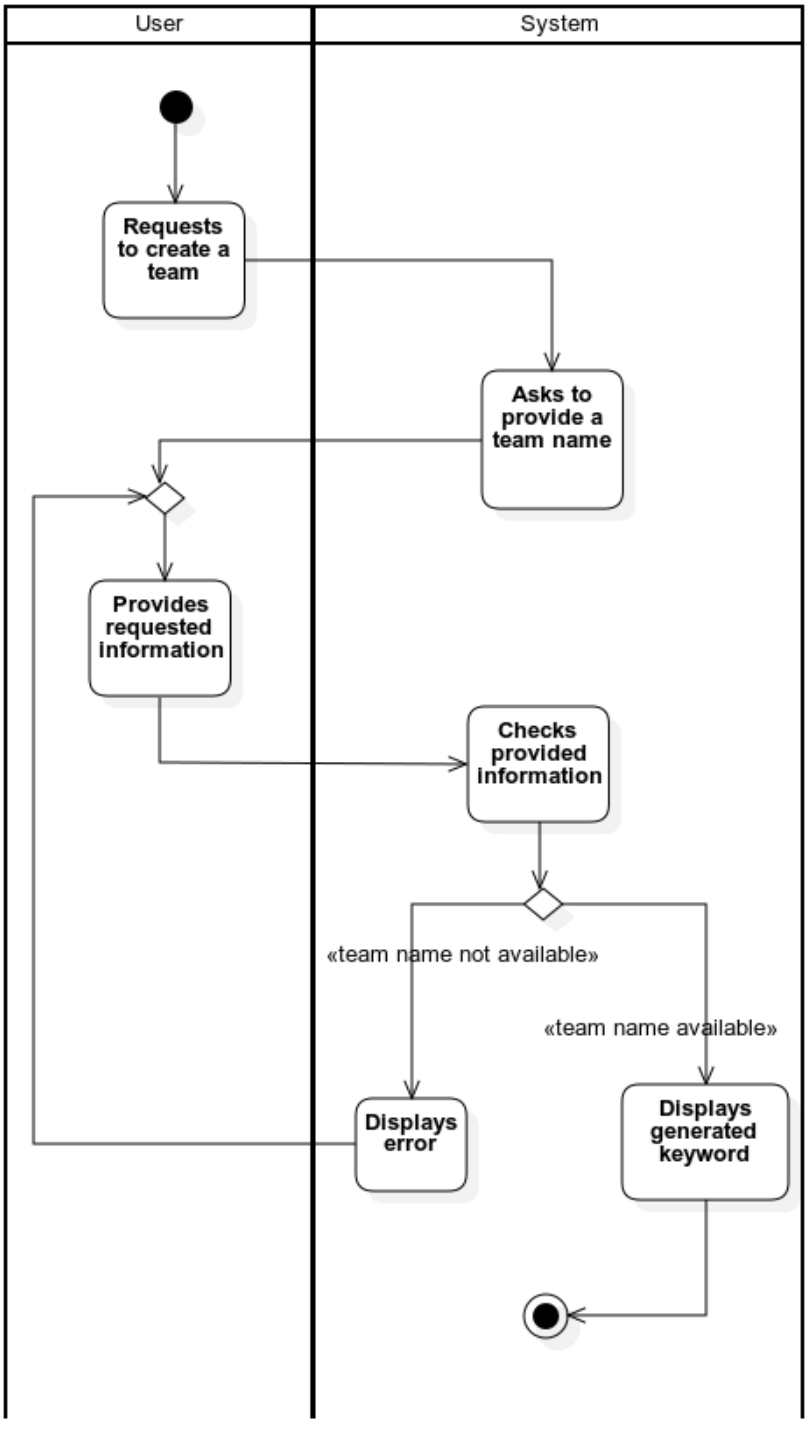
[UC5]

Name	Join PRIVATE tournament
Actors	User (Student)
Entry conditions	• User has logged in as Student • The tournament is in the *Registration* state
Events flow	1. User requests to join a private tournament 2. The system asks to provide a keyword 3. User inserts the keyword 4. The system checks keyword and searches for the corresponding private tournament 5. The system updates the database and notifies the User about upcoming battles
Exit Condition	User has successfully subscribed to the tournament and has been notified about upcoming battles
Exception 1	User inserts a keyword that does not correspond to any tournament. The system displays an error
Exception 2	The user is already subscribed to the tournament, the system displays an error
Exception 3	The tournament has moved from the *Registration* state to the *Ongoing* state before the user managed to complete the subscription. The system will display an error



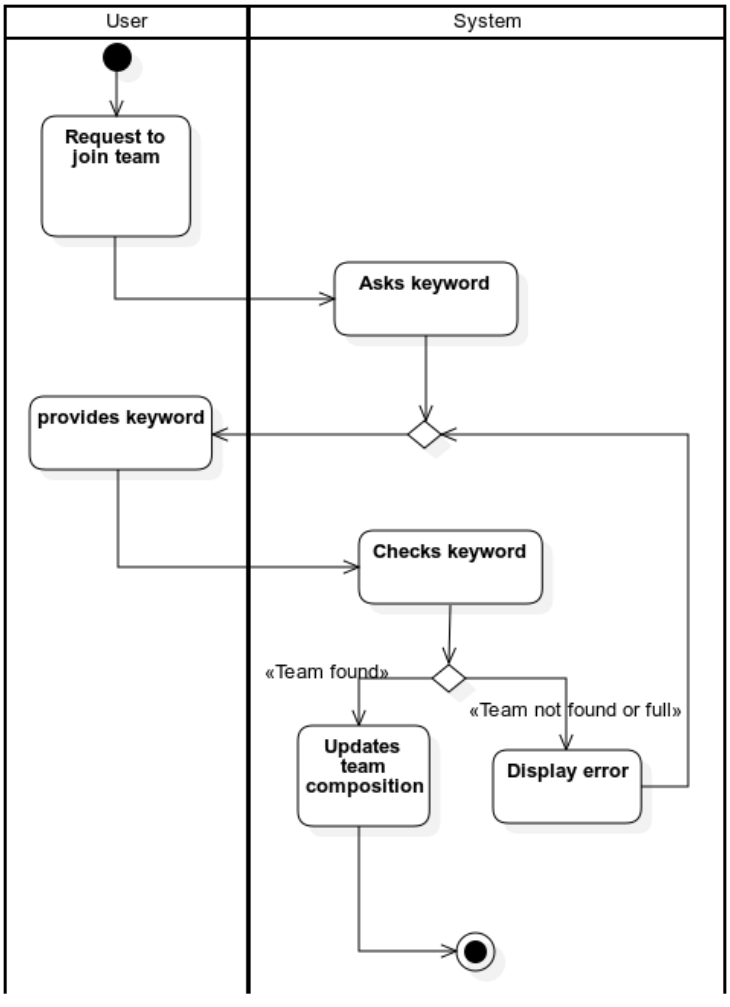
[UC6]

Name	Create team
Actors	User (Student)
Entry conditions	• User has logged in as Student • User is subscribed to a tournament • The tournament is in the *Registration* state or *Ongoing* state and has at least one battle to which the User is not subscribed in the *Registration* state
Events flow	1. User selects a battle 2. The system displays the battle page 3. User requests to create a team 4. The system asks to provide a team name 5. User provides the requested information 6. The system checks the provided information 7. The system displays a generated keyword that can be used to join the team
Exit Condition	User successfully created a team for a battle
Exception 1	User inserts a name for the team that is already in use for that battle. The system will display an error
Exception 2	The battle has moved from the *Registration* state to the *Ongoing* state before the user managed to complete the creation of the team. The system will display an error


User
System
Requests to create a team
Asks to provide a team name
Provides requested information
Checks provided information
«team name not available»
«team name available»
Displays error
Displays generated keyword

[UC7]

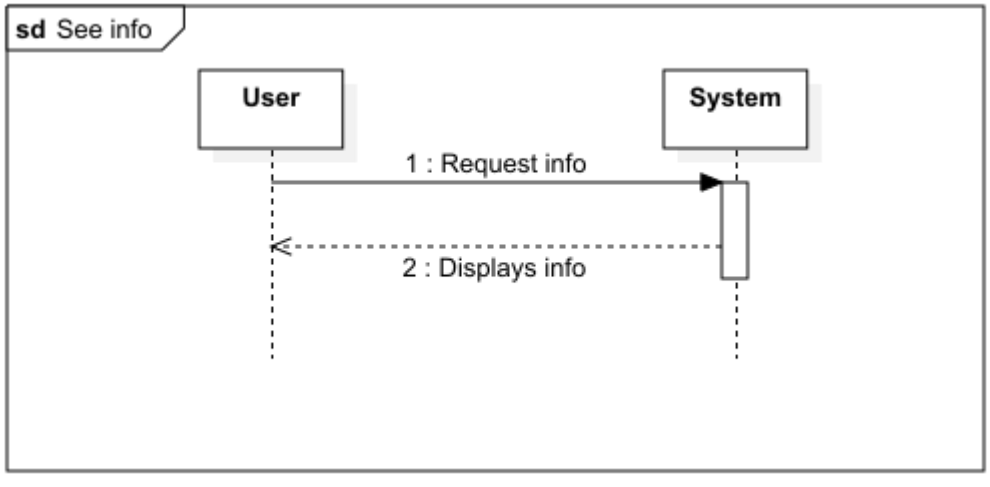
Name	Join team
Actors	User (Student)
Entry conditions	• User has logged in as Student • User is subscribed to a tournament • The tournament is in the *Registration* state or *Ongoing* state and has at least one battle to which the User is not subscribed
Events flow	1. User selects a battle 2. The system displays the battle page 3. User requests to join a team 4. The system asks to provide a keyword 5. User provides a keyword 6. The system checks if there is a team that corresponds to the provided keyword 7. The system updates the team composition
Exit Condition	User successfully joins a team
Exception 1	User inserts a keyword that does not correspond to any team subscribed to the battle. The system displays an error
Exception 2	User inserts a keyword that corresponds to a team, but the team is full according to the constraints of the battle. The system displays an error
Exception 3	The battle has moved from the *Registration* state to the *Ongoing* state before the system managed to update the team composition. The system will display an error



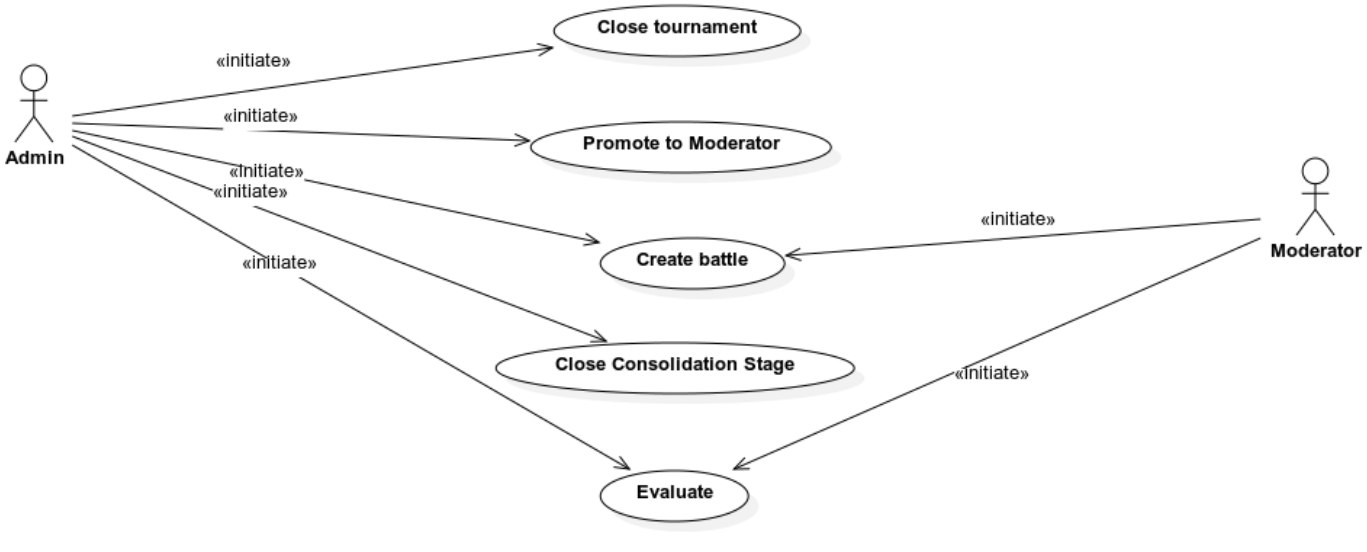
[UC8]

Name	See info
Actors	User
Entry conditions	• User has logged in Tournament: • If the tournament is private, User is an Admin/Moderator or a subscribed Student Battle: • If User is a Student, he/she is subscribed to the battle • If User is an Educator, he/she is an Admin/Moderator of the tournament the battle belongs to

Events flow	1. User requests to see all information about a tournament/battle 2. The system display the requested information
Exit Condition	The system successfully displayed the tournament/battle info

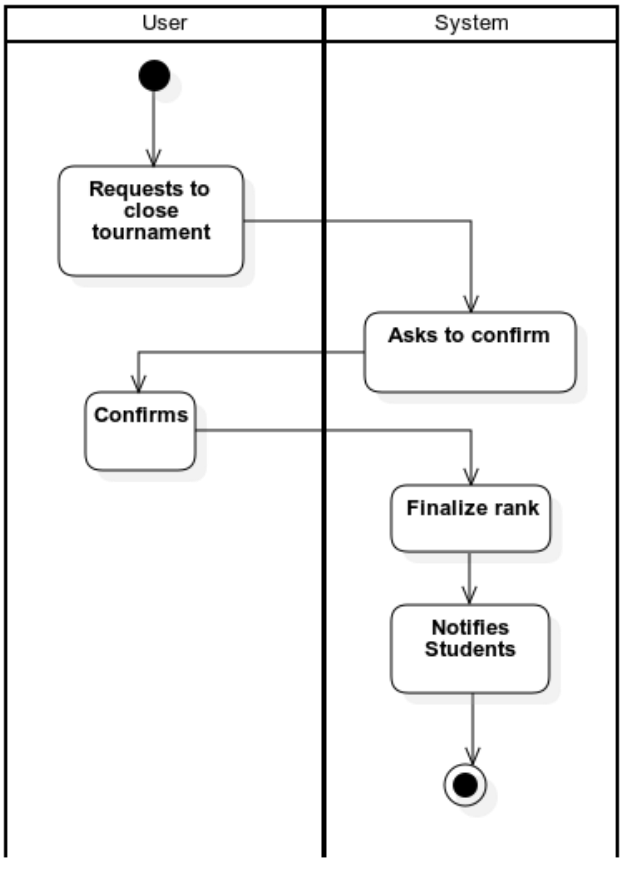


ADMIN and MODERATOR:



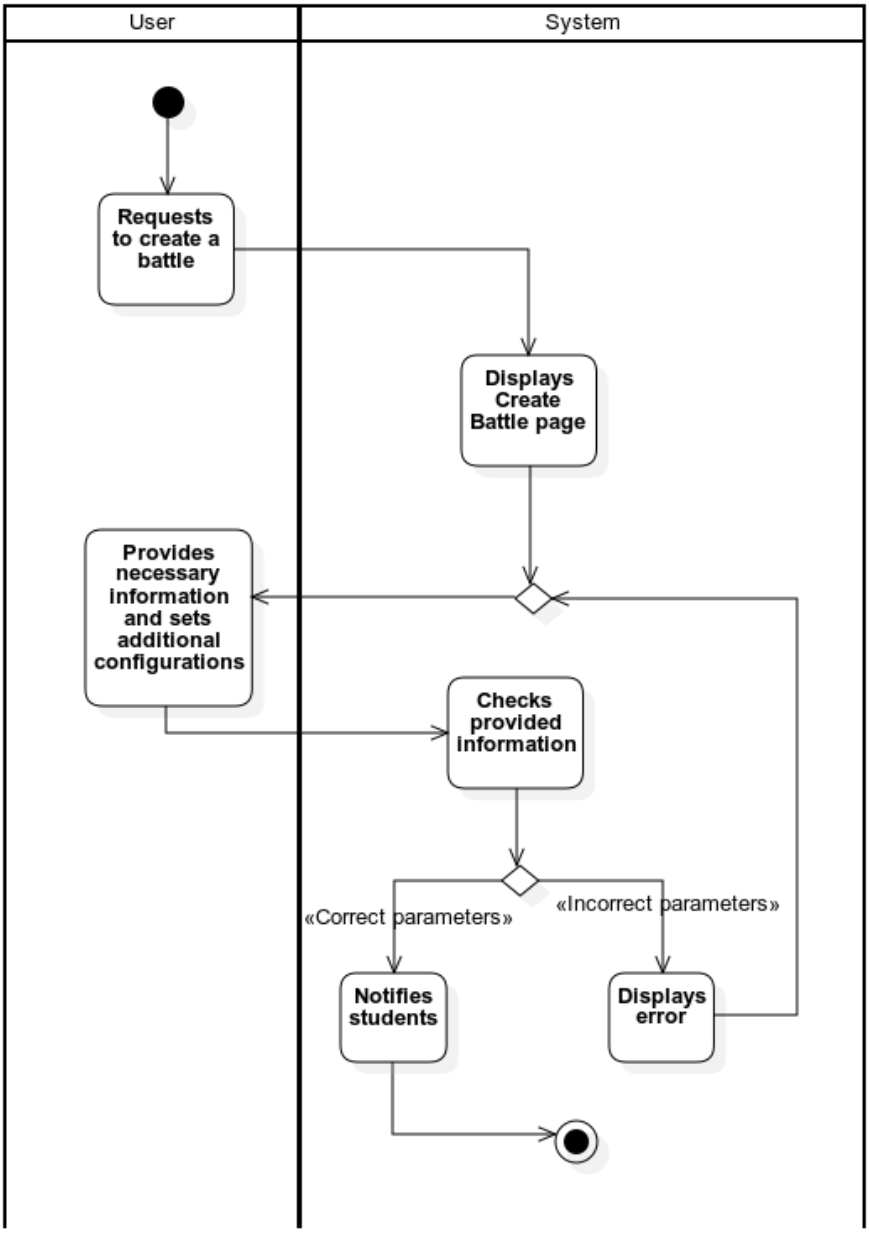
[UC9]

Name	Close tournament
Actors	User (Admin), Student
Entry conditions	• User has logged in as Educator • User has created a tournament • Tournament is in the *Registration* state or *Ongoing* state • All battles within the tournament are in the *Closed* state
Events flow	1. User requests to close a tournament 2. The system asks to confirm 3. User confirms 4. The system finalizes the tournament final rank and updates the database 5. The system notifies all subscribed Students that the tournament final rank is available
Exit Condition	User has successfully closed the tournament and Students have been notified
Exception 1	User clicked on the "Close tournament" button but changed his mind. User cancels instead of confirming



[UC10]

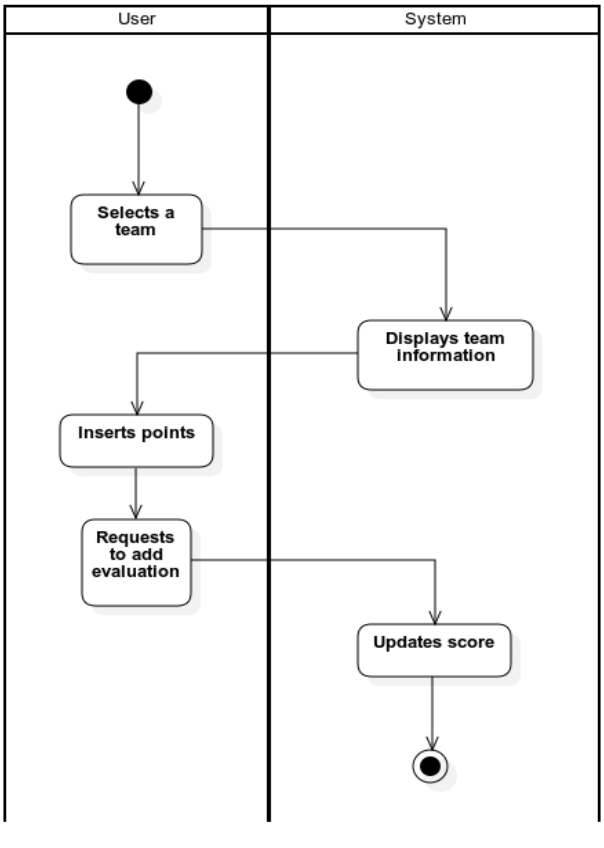
Name	Create battle
Actors	User (Admin/Moderator), Student
Entry conditions	• User has logged in as Educator • User has created a tournament or has been granted permission to create battles withing a tournament (promoted to Moderator) • Tournament is in the *Registration* state or *Ongoing* state
Events flow	1. User requests to create a new battle 2. The system displays the Create Battle page 3. User sets a name for the battle 4. User uploads the code kata (description and software project, including test cases and build automation scripts) 5. User sets minimum and maximum number of students per group 6. User sets a registration deadline (which must be after the tournament registration deadline) 7. User sets a final submission deadline 8. User sets additional configurations for scoring to measure the quality level of the sources (security, reliability and maintainability) 9. User submits all information 10. The system checks the correctness of all the submitted information 11. The system updates the database and notifies all Students subscribed to the tournament
Exit Condition	User has successfully created a new battle, all Students subscribed to the tournament have been notified
Exception 1	User doesn't set parameters correctly. The system displays an error
Exception 2	The tournament has moved from the *Ongoing* state to the *Closed* state before the User could submit the information. The system displays an error


User
System
Requests to create a battle
Displays Create Battle page
Provides necessary information and sets additional configurations
Checks provided information
«Correct parameters»
«Incorrect parameters»
Notifies students
Displays error

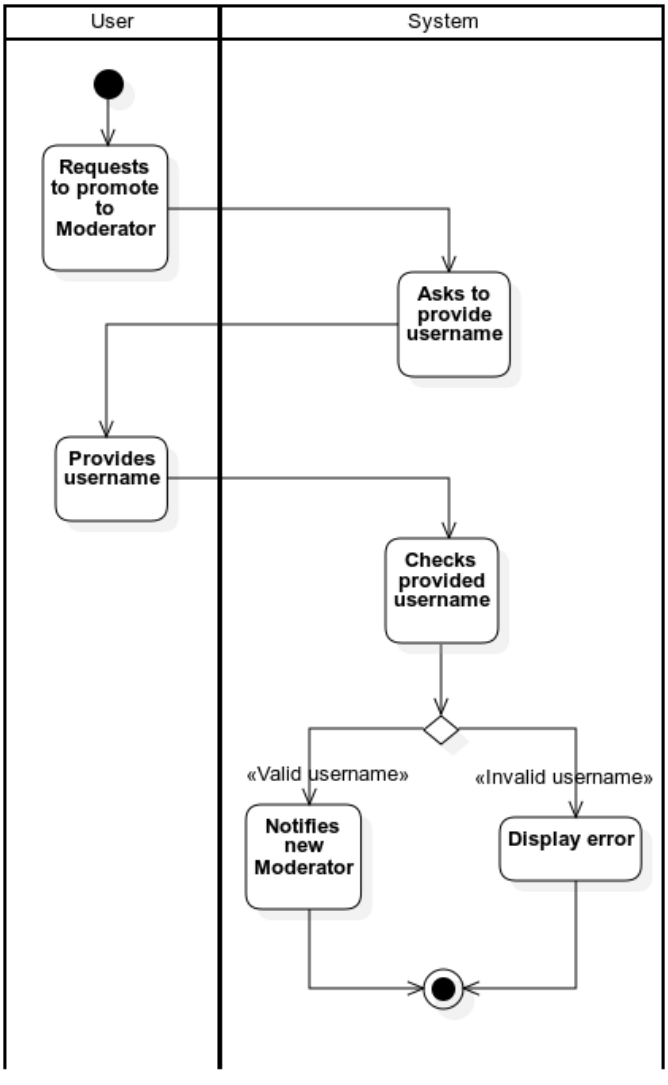
[UC11]

Name	Evaluate
Actors	User (Admin/Moderator)
Entry conditions	• User has logged in as Educator • Tournament is in the *Ongoing* state • Battle is in the *Consolidation Stage* state • User is an Admin or Moderator of the tournament
Events flow	1. User selects a team 2. System displays the information about that team (members and points obtained with the automated evaluation) 3. User inserts a number he/she wants to add to one or more Students' score 4. User requests to add the evaluation 5. The system updates the score
Exit Condition	User has successfully evaluated one team
Exception 1	Battle goes into the *Closed* state before the score has been updated. The system displays and error

[UC12]



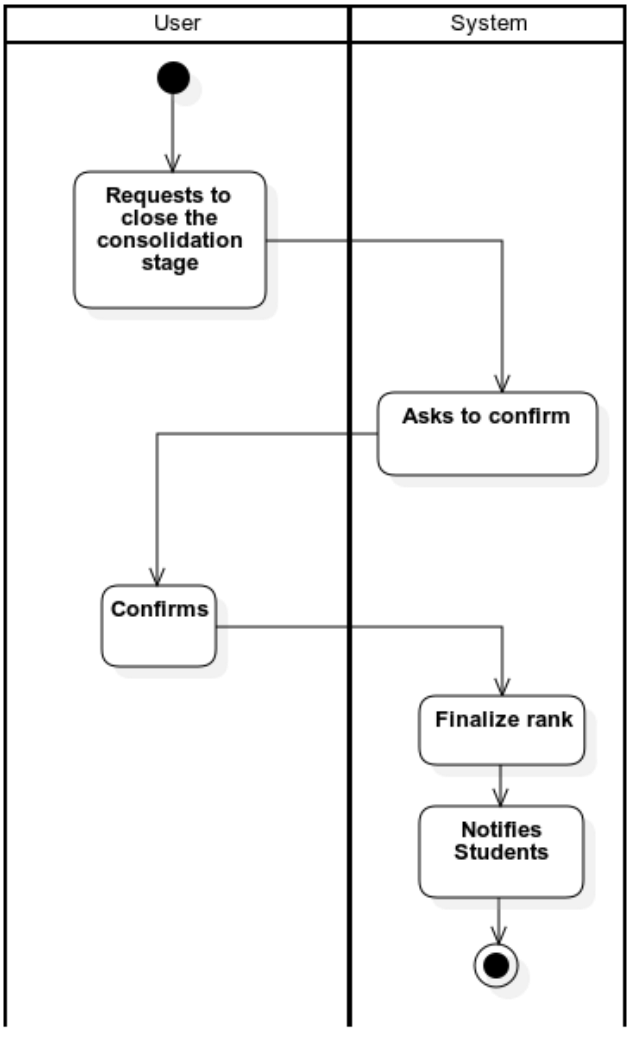
Name	Promote to moderator
Actors	User (Admin), Moderator
Entry conditions	• User has logged in as Educator • User is Admin of the tournament • Tournament is in the *Registration* state or *Ongoing* state
Events flow	1. User requests to promote and Educator to Moderator 2. The system asks to provide a username 3. User inserts the username related to the account of the Educator he wishes to promote 4. The system checks the username 5. System updates the database and notifies the new Moderator
Exit Condition	User has successfully promoted another Educator to Moderator and the system has notified him/her
Exception 1	User inserts his own username. The system displays an error
Exception 2	User inserts a username linked to a Student account. The system displays an error
Exception 3	User inserts a username not registered in the database. The system displays an error
Exception 4	User inserts a username of an Educator that is already a Moderator for that tournament. The system displays an error



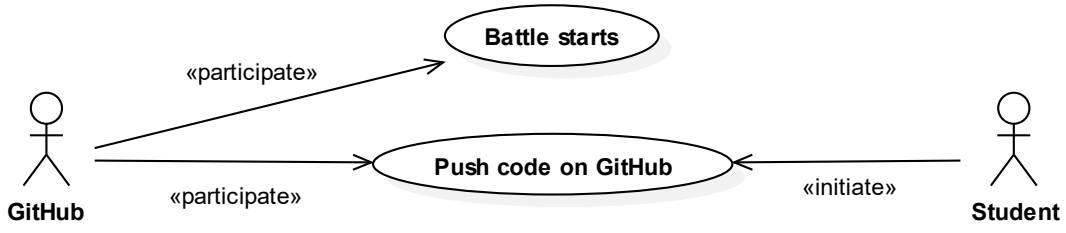
[UC13]

Name	Close consolidation stage
Actors	User (Admin), Student
Entry conditions	• User has logged in as Educator • User is Admin of the tournament • Tournament is in the *Ongoing* state • Battle is in the *Consolidation Stage* state
Events flow	1. User requests to close the consolidation stage 2. The system asks to confirm

	3. User confirms 4. The system finalizes the battle final rank and updates the tournament rank by adding to each student his/her score obtained in that battle 5. The system notifies all Students subscribed to the battle that the battle final rank is available
Exit Condition	User has successfully closed the consolidation stage and all Students subscribed to the battle have been notified

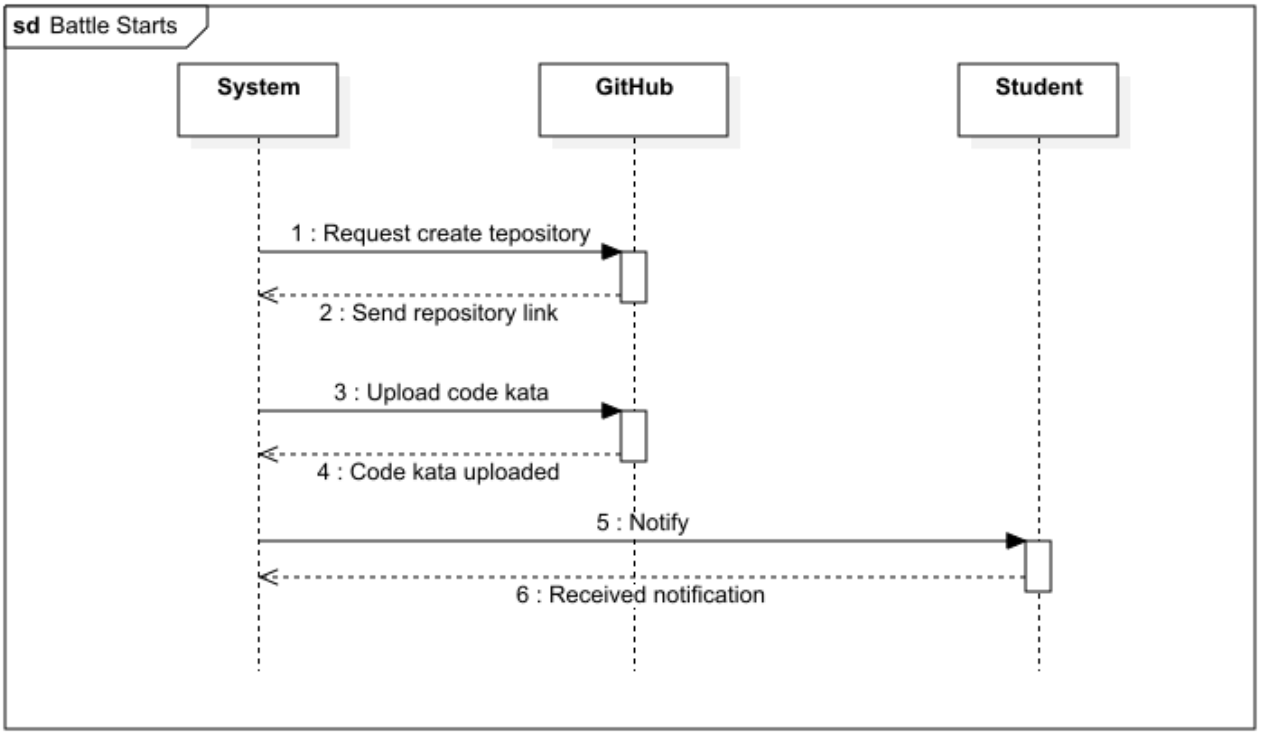


GITHUB:



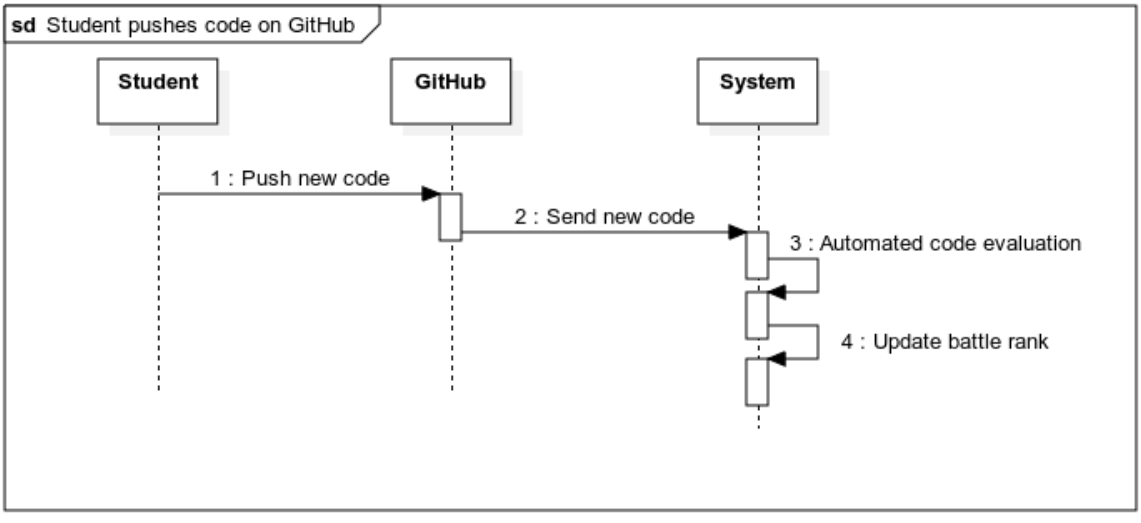
[UC14]

Name	Battle starts
Actors	GitHub, Student
Entry conditions	• The battle has transitioned from the *Registration* state to the *Ongoing* state • The tournament, to which this battle belongs, is in the *Ongoing* state.
Events flow	1. The system communicates with GitHub to create a repository for the new battle. 2. GitHub successfully creates the repository. 3. GitHub sends the link of the repository back to the system that saves it in the database. 4. The system communicates with GitHub to upload the code kata for the battle. 5. GitHub uploads the code kata to the repository. 6. The system notifies all Students subscribed to the battle, providing them with the link to access the code kata.
Exit Condition	The code kata has been uploaded in the new repository and the Students have been notified with the link to access it



[UC15]

Name	Push code on GitHub
Actors	Student, GitHub
Entry conditions	• The battle is in the *Ongoing* state • The tournament, to which this battle belongs, is in the *Ongoing* state. • The team the student belongs to has successfully forked the repository and created the automated workflow with the system through GitHub
Events flow	1. The student pushes new code to the GitHub repository of his team. 2. GitHub sends the output of the pushed code to the system after running it. 3. The system performs an automated evaluation of the code based on predefined criteria (selected at battle creation) 4. The system updates the battle rank of the Student's team
Exit Condition	The system successfully updated the battle rank of the Student's team



Requirements and use-cases about some other functions have not been included in this document, but they can be easily added should they ever become necessary or requested by the stakeholders. Here are some possible examples of additional features to be added:

- Delete Battle
- Demote Moderator
- Delete Team
- Kick member from a Team

The last two elements on the list can be obtained by adding a new role for a team's creator such as "Team Leader" or "Team Captain", while the first two don't necessarily require any new role definition.

B.4. Traceability matrix

Requirement	Use Case
[RL1]	Register user
[RL2]	Login user
[RT0]	See info
[RT1]	Create tournament
[RT2]	Close tournament
[RT3]	Promote to moderator
[RT4]	Join PUBLIC/PRIVATE tournament
[RB0]	See info
[RB1]	Push code on GitHub
[RB2]	Create battle
[RB3]	Evaluate
[RB4]	Close consolidation stage
[RB5]	Create team
[RB6]	Join team
[RG1]	Battle starts
[RG2]	Push code on GitHub
[RN1]	Create tournament
[RN2]	Join PUBLIC/PRIVATE tournament
[RN3]	Create battle
[RN4]	Battle starts
[RN5]	Close consolidation stage
[RN6]	Close tournament
[RN7]	Promote to moderator

C. Performance Requirements

- CKB platform is thought to support *multiple* tournaments at the same time. This could result in the fact that there could be many Students working on it simultaneously, pushing new code and requiring the system to recompute the points. In this regard, the system must support a high throughput rate of messages from/to GitHub and include the parallel computation of scores.

- *Notifications* play a key role for most of the functionalities of the platform: through them students know for example when tournaments or battles are created, when a final rank is available and so on. For this reason, it is very important that the system ensures the delivery of notifications within few minutes (5 minutes at the peak time of use and 2 minutes the rest of the time).

D. Design Constraints

D.1. Standards compliance

- The platform includes the full adherence to **GDPR,** which stands as one of the most significant and internationally recognized standards for the protection of personal data and ensuring user privacy. The system is committed to handling user data in GDPR-compliant ways, ensuring transparency in data collection and processing, and adopting appropriate security measures to protect such data.

- To obtain data correctness and protection even at communication level, the platform should adopt the use of **TCP/IP** together with the application of the TLS security protocol.

- The purpose of CodeKataBattle is to gather Students and Educators spread all over the world. So, it is very important to use a time standard, such as **UTC**, to achieve the

synchronization of all the users, the correct unfolding of Battles, and the handling of deadlines.

D.2. Hardware limitations

In order to enable an effective use of the platform to as many users as possible, the platform should not require high level hardware and should work on almost all kinds of machines.

D.3. Any other constraint

CKB platform is intended to welcome students from all over the world. So, it should be necessarily designed completely in English, allowing every student to understand its pages, interfaces, commands etc.

E. Software System Attributes

E.1. *Reliability*

Sometimes, the results achieved during battles are used to give students graduation marks. Hence, it is important that the system guarantees a high level of reliability and always provides correct data regarding automatic evaluations, final ranks, etc.

E.2. *Availability*

As mentioned before, CKB should be accessible from all over the world. Together with the fact that every user should have the possibility to work on it when he/she prefers, the platform should be available 24/7 and prevent downtimes except for maintenance times. Therefore, the system should achieve a 2-nines availability, which guarantees a maximum downtime of 3.65 days per year.

E.3. *Security*

- To match the GDPR compliance, the platform should achieve the protection of personal data through an authentication system involving unique usernames and strong passwords.

- The platform should take into account the observance of different levels of hierarchy (f.e. student-educator, admin-moderator) and so the system must prevent “lower” users from performing actions which are reserved only for “higher” users and vice versa.

- There are some aspects of the platform that are *private* (such as Teams or Private Tournaments), i.e. they are reserved only for a specific group of users. For this purpose, the platform should provide a keyword-based protection system, capable of generating and managing unique keywords: users are asked to submit the correct keyword to access private contexts.

E.4. *Maintainability*

It’s very important to ensure that the system source code can be easily understood, modified, and improved over time. To achieve this goal, the code should be clear and well-documented, making it easily comprehensible for developers and facilitating maintenance.

E.5. *Portability*

Given CKB's scale and reach, it's crucial to ensure its compatibility with a wide range of operating systems, including Windows, MacOS, and Linux, for an effortless deployment.

4. FORMAL ANALYSIS USING ALLOY

In this chapter, it is presented a report of the most significant results of the formal analysis of the system performed through the specification language Alloy 6. For a better understanding of the structure and of the main behaviors of the platform, two different models are described, both involving all the entities and actors but focusing on different points of view of the interaction among them.

A. Static Model

This model is aimed at describing the relationships between the entities acting in the unfolding of tournaments and battles. It does not dwell on the evolutions of some situations but rather it gives pictures of them, depicting cardinalities and verses of the main relations.

A.1. *Signatures*

Here is the list of all the signatures acting in the model, each one corresponding to an entity implicated in the system.

```
abstract sig User {}

sig Student extends User {
      subscribed: set Tournament,
      isMemberOf: set Team
}

sig Educator extends User {
      admin: set Tournament,
      moderator: set Tournament,
      create: set Battle
}

sig Tournament {
      battles: set Battle,
      status: one TournamentSTATUS
}
```

```
sig Battle {
      kata: one CodeKata,
      status: one BattleSTATUS
}

sig Team {
      participate: one Battle
}

sig CodeKata {}
sig TournamentSTATUS {}
sig BattleSTATUS {}
```

A.2. *Facts*

With the `fact` keyword, Alloy allows to define constraints and properties that must be true in all the possible instances of the model. Here is the list of all the constraints of the designed model.

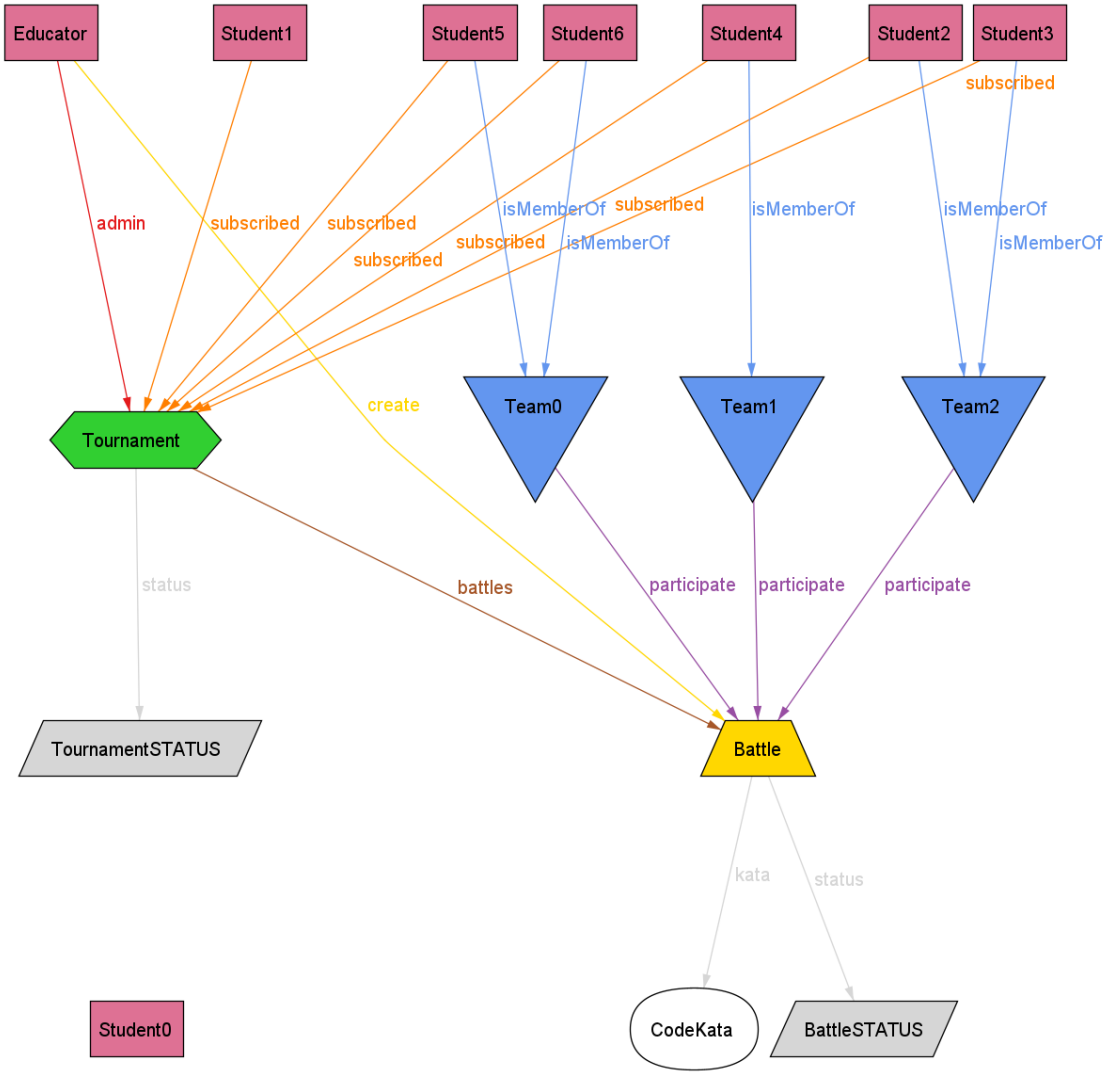
```
//All tournaments must have exactly one creator (Admin)
fact TournamentAdmin{
      all t: Tournament | one t.~admin
}

//An Admin can't be a Moderator and viceversa
fact Roles {
      all e: Educator | e.admin & e.moderator = none
}

//An Educator can't create a battle for a tournament if he/she is
not the Admin/Moderator for that tournament
fact EducatorRights{
      all b: Battle | b.~create in (b.~battles).~(admin+moderator)
}

//All battles must have exactly one creator
fact BattleCreator{
      all b: Battle | one b.~create
}

//All battles must be part of exactly one tournament
fact BattleTournamentRelationship{
      all b: Battle | one b.~battles
}

//Each team must have at least one student who is its creator
fact NoEmptyTeam{
      no t: Team | t.~isMemberOf = none
```

```
}

//A code kata must be unique and associated to exactly one battle.
fact CodeKata{
      all ck: CodeKata | one ck.~kata
}

//Each tournament status is associated with only one tournament
fact UniqueTournamentStatus{
      all ts: TournamentSTATUS | one ts.~status
}

//Each battle status is associated with only one battle
fact UniqueBattleStatus{
      all bs: BattleSTATUS | one bs.~status
}

//Each student participating in a battle must be subscribed to the
tournament associated to that battle
fact Subscription{
      all s: Student | all t: Team | t in s.isMemberOf
      implies (t.participate).~battles in s.subscribed
}

//Each student participating in a battle can't be part of more than
one team involved in that battle
fact StudentOnlyOneTeam{
      all s: Student | all disj t1, t2: Team | t1 in s.isMemberOf
and t2 in s.isMemberOf
      implies (no t1.participate & t2.participate)
}
```

A.3. *Scenario 1*

```
pred show1{
      one s: Student | s.(subscribed+isMemberOf) = none
      #Tournament = 1
      #Battle = 1
      #Team = 3
      one t: Team | #t.~isMemberOf = 1
}
run show1 for 9 but exactly 7 Student
```

By running this predicate, the aim is to show a scenario in which multiple *students'* conditions are represented.



It is relevant to notice that Student0 has just registered to the platform and so he/she is not yet subscribed to any tournament or battle.

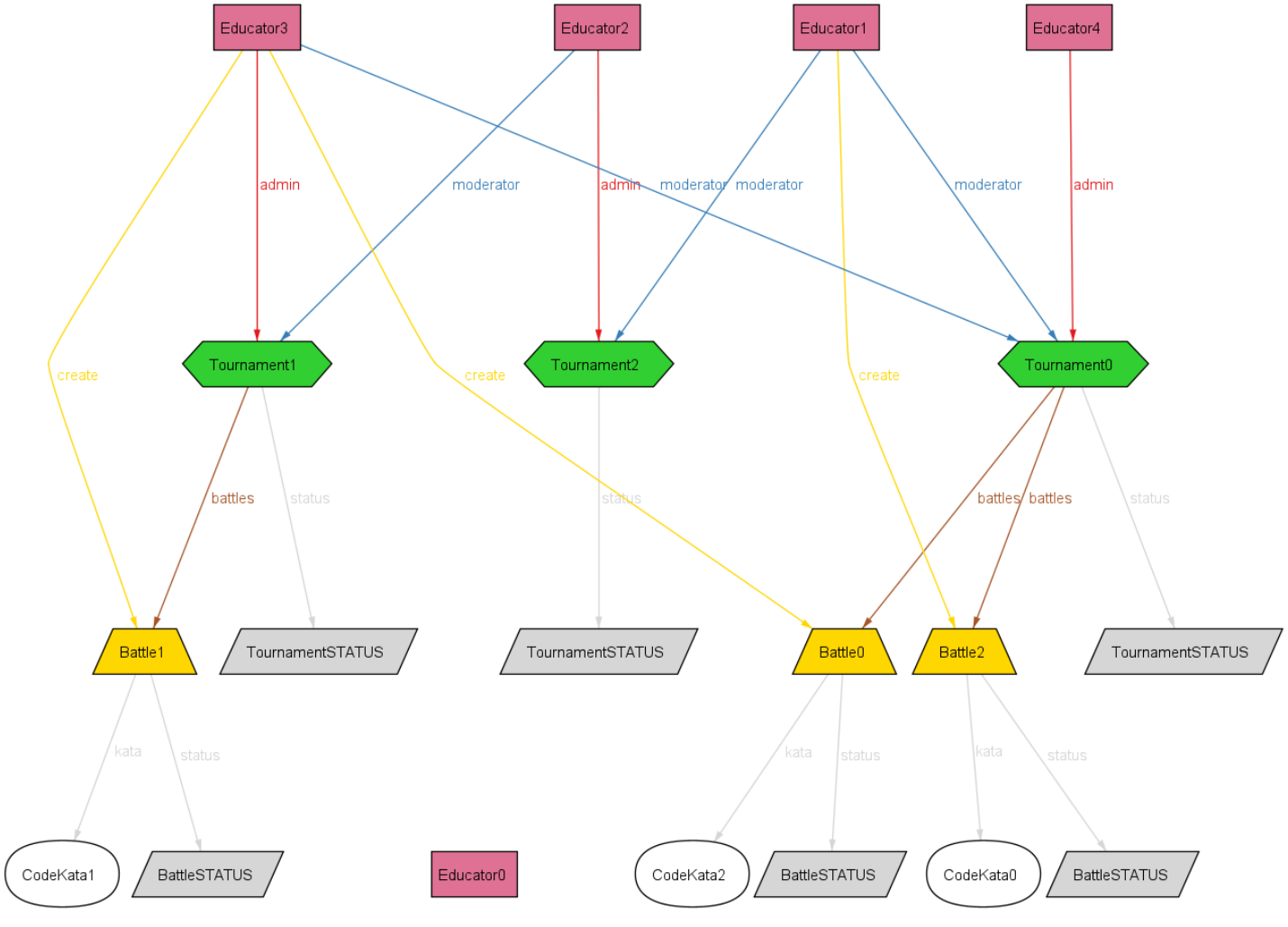
Moreover, Student1 has just subscribed to the Tournament, but he/she is not part of any team yet.

Finally, the other students are forming three different teams (Team0, Team1, Team2) participating in the only Battle that is taking place at the moment.

A.4. *Scenario 2*

```
pred show2{
      one e: Educator | e.(admin+moderator+create) = none
      one e: Educator | #e.admin = 1 and #e.moderator = 1 and (some
e.admin.battles & e.create) and (some e.moderator.battles &
e.create)
      #Tournament = 3
      #Battle = 3
}
run show2 for 5 but exactly 5 Educator
```

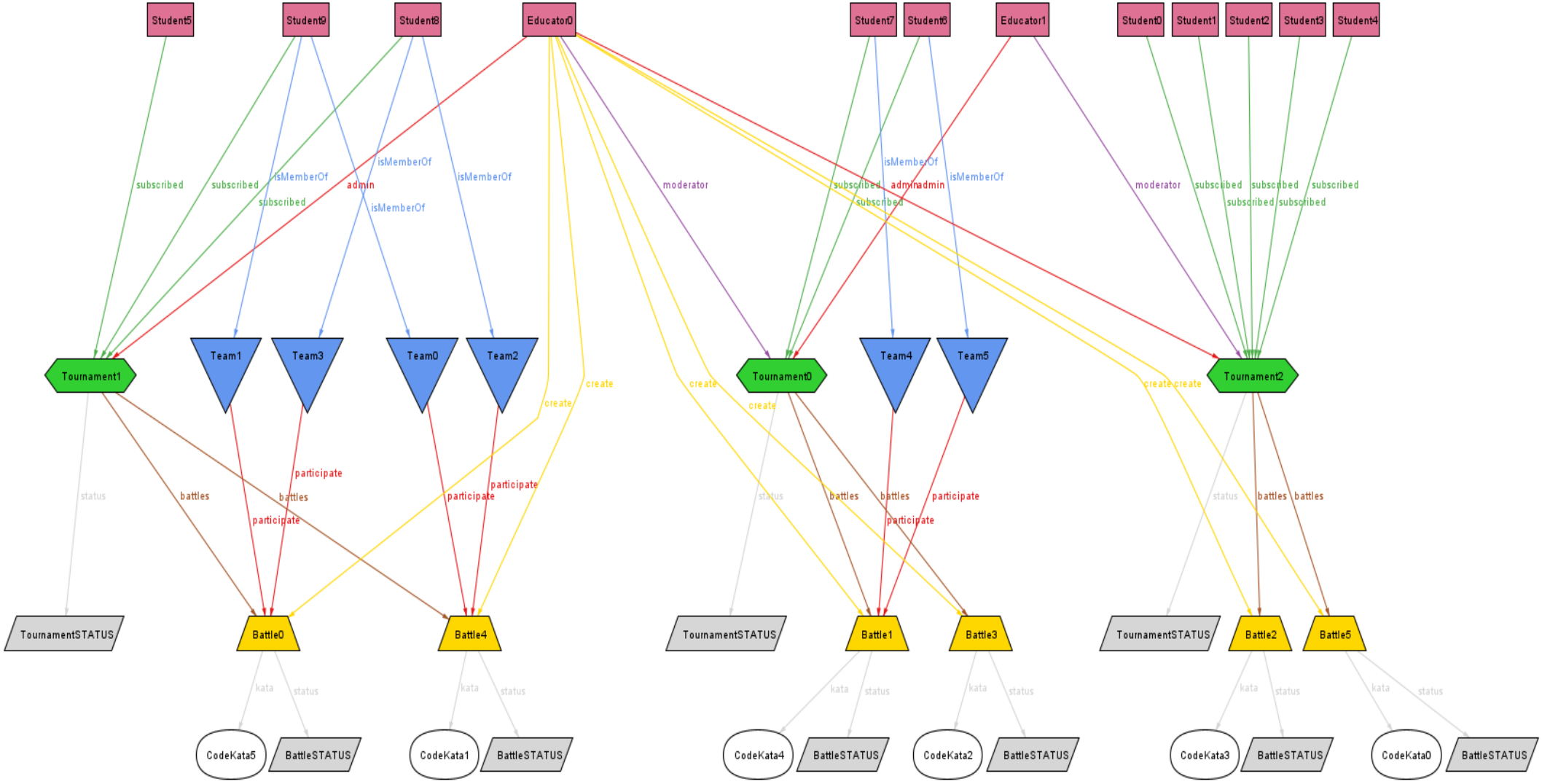
The goal of this predicate is to depict multiple situations related to *Educators*, such as being Admin and Moderator of tournaments and creating battles for them.



As previously shown for Students, Educator0 is an example of an Educator that has just registered to CKB.
Then, it may be of interest to observe that the model is showing that an Educator can be simultaneously Admin for a Tournament and Moderator for another, creating Battles for both. For example, Educator3 created Battle1 for Tournament1 and Battle0 for Tournament0.

A.5. *Scenario 3*

```
pred show3 {
      #Tournament = 3
      #Battle = 6
      all t: Tournament | #t.battles = 2
      #Team > 5
      all s: Student | #s.subscribed > 0
      all t: Tournament | #t.~subscribed < 6
}
run show3 for 15 but exactly 10 Student
```



This last scenario is global and shows a little bit bigger situation for the platform, involving 10 Students, 3 Tournaments, 6 Battles and at least 5 Teams. This representation shows one of the possible snapshots of the platform according to the predicate and the facts defined in the Alloy model. It is possible to deduce from this image that Tournament1 and Tournament0 are already developing, with teams fighting in different battles, while Tournament2 is still in his registration stage, where Students subscribe to the tournament but no Teams have been created.

B. Dynamic model

The purpose of the realization of this model is to show the evolution of entities and of relations between entities, focusing the attention on all the aspects of the system that change over time and cause variations in relationships with other objects.

Alloy 6 provides for the `var` keyword and for other time keywords (like `always`, `after`, `eventually` etc.) to build variable signatures and relationships and to define constraints and predicates that are not valid for every step of the evolution of the model.

B.1. Signatures

Here is the list of all the signatures of the dynamic model. All battle and tournament states are specified and some of the relations, already present in the static model, have been made variable.

```
abstract sig User {}

sig Student extends User {
      var subscribed: set Tournament,
      var isMemberOf: set Team
}

sig Educator extends User {
      admin: set Tournament,
      var moderator: set Tournament,
      create: set Battle
}

sig Tournament {
      battles: set Battle,
      var status: one TournamentSTATUS
}

sig Battle {
      var kata: one CodeKata,
      var status: one BattleSTATUS
}

var sig Team {
      var participate: one Battle
}

var sig CodeKata {}
```

```
var abstract sig TournamentSTATUS {}
var sig TRegistration extends TournamentSTATUS {}
var sig TOngoing extends TournamentSTATUS {}
var sig TClosed extends TournamentSTATUS {}

var abstract sig BattleSTATUS {}
var sig BRegistration extends BattleSTATUS {}
var sig BOngoing extends BattleSTATUS {}
var sig BConsolidation extends BattleSTATUS {}
var sig BClosed extends BattleSTATUS {}
```

B.2. Facts

Here all the `fact` statements of the dynamic model are listed, including both the previously defined facts (updated in order to work with variable relations) and the new constraints regarding the variable states of tournaments and battles.

```
//All tournaments must have exactly one creator (Admin)
fact TournamentAdmin{
      always all t: Tournament | one t.~admin
}

//An Admin can't be a Moderator and viceversa
fact Roles {
      always all e: Educator | e.admin & e.moderator = none
}

//An Educator can't create a battle for a tournament if he/she is not
the Admin/Moderator for that tournament
fact EducatorRights{
      always all b: Battle | b.~create in
(b.~battles).~(admin+moderator)
}

//All battles must have exactly one creator
fact BattleCreator{
      always all b: Battle | one b.~create
}

//All battles must be part of exactly one tournament
fact BattleTournamentRelationship{
      always all b: Battle | one b.~battles
}

//Each team must have at least one student who is its creator
fact NoEmptyTeam{
      always no t: Team | t.~isMemberOf = none
}
```

```
//A code kata must be unique and associated to exactly one battle.
fact CodeKata{
      always all ck: CodeKata | one ck.~kata
}

//Each tournament status is associated with only one tournament
fact UniqueTournamentStatus{
      always all ts: TournamentSTATUS | one ts.~status
}

//Each battle status is associated with only one battle
fact UniqueBattleStatus{
      always all bs: BattleSTATUS | one bs.~status
}

//Each student participating in a battle must be subscribed to the
tournament associated to that battle
fact Subscription{
      always all s: Student | all t: Team | t in s.isMemberOf
      implies (t.participate).~battles in s.subscribed
}

//Each student participating in a battle can't be part of more than
one team involved in that battle
fact StudentOnlyOneTeam{
      always all s: Student | all disj t1, t2: Team | t1 in
s.isMemberOf and t2 in s.isMemberOf
      implies (no t1.participate & t2.participate)
}
```

```
//With this fact "init", it is possible to specify the initial
conditions of the model
fact "init"{
      #Educator = 2
      #Student = 4
      #Tournament = 1
      #Battle = 1
      all t: Tournament, s: Student | t.status = TRegistration and
not t in s.subscribed
      all b: Battle | b.status = BRegistration
}

//This fact defines how a tournament must evolve
fact TournamentEvolutionRules{
      always no t: Tournament | t.status = TRegistration and after
t.status = TClosed
      always no t: Tournament | t.status = TOngoing and after
t.status = TRegistration
      always no t: Tournament | t.status = TClosed and
                                              after (t.status =
TRegistration or t.status = TOngoing)
}

//This fact defines how a battle must evolve
fact BattleEvolutionRules{
```

```
      always no b: Battle | b.status = BRegistration and after
(b.status = BConsolidation or b.status = BClosed)
      always no b: Battle | b.status = BOngoing and after (b.status =
BRegistration or b.status = BClosed)
      always no b: Battle | b.status = BConsolidation and after
(b.status = BRegistration or b.status = BOngoing)
      always no b: Battle | b.status = BClosed and
                                          after (b.status =
BRegistration or b.status = BOngoing or b.status = BConsolidation)
}

//There cannot be an ongoing battle within a tournament which is
still in the registration stage
fact {
      always all t: Tournament, b: Battle | (t.status = TRegistration
and b in t.battles)
      implies b.status = BRegistration
}

//Two battle of the same tournament can't be ongoing battles
simultaneously
fact {
      always all t: Tournament | no disj b1,b2: Battle | (b1 in
t.battles and b2 in t.battles)

                  and (b1.status = BOngoing and b2.status = BOngoing)
}

//To close a tournament, all its battle must be already closed
fact {
      always all t: Tournament, b: Battle | t.status = TClosed and b
in t.battles
            implies b.status = BClosed
}

//Students cannot subscribe if the tournament is not in the
registration stage
// and subscribed students cannot unsubscribe
fact {
      always all s: Student, t: Tournament | t.status = TRegistration
and t in s.subscribed
      implies always t in s.subscribed
      always all s: Student, t: Tournament | t.status !=
TRegistration and t not in s.subscribed
      implies always t not in s.subscribed
}

//All the moderator of a tournament keep their role
fact {
      always all e: Educator, t: Tournament | t in e.moderator
      implies always t in e.moderator
}

//The team that is formed for a battle remains till the end of the
battle
fact {
      always all t: Team, b: Battle | b.status = BRegistration and b
in t.participate
      implies always b in t.participate
```

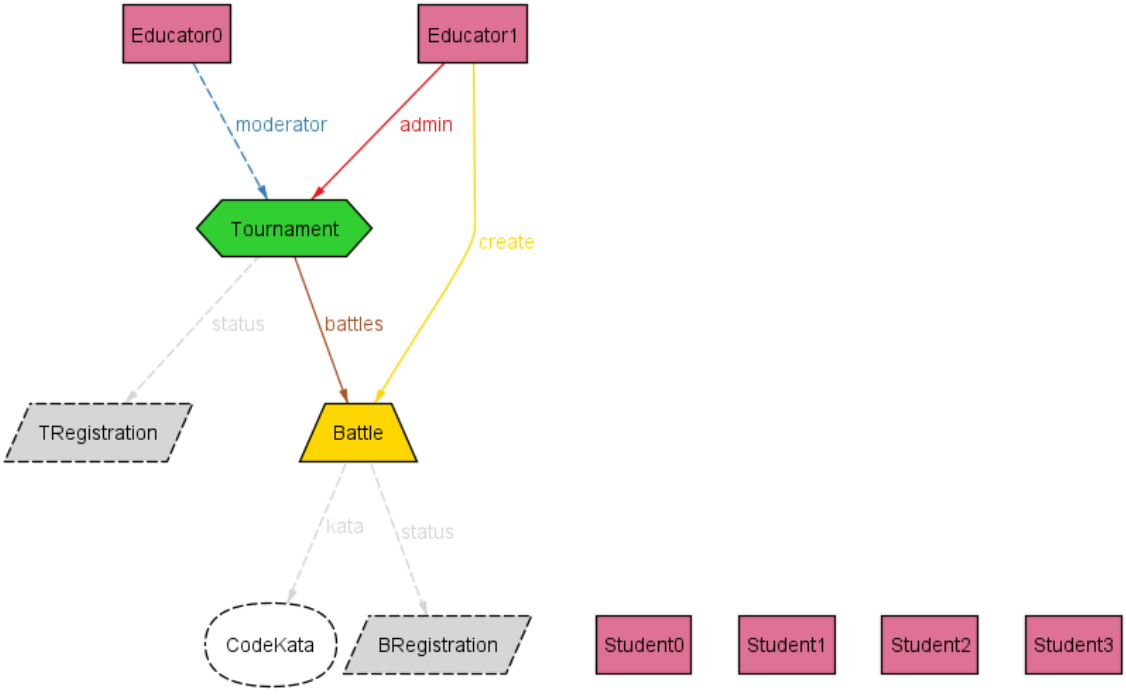
```
}

//The students componing a team remains in the team till the end of
the battle
fact {
      always all t: Team, s: Student | t in s.isMemberOf
      implies always t in s.isMemberOf
}
```

B.3. Dynamic Scenario

```
pred TournamentEvolution{
      eventually all t: Tournament | t.status = TOngoing implies
after t.status = TClosed
}

pred BattleEvolution{
      eventually all b:Battle | b.status = BRegistration
            implies b'.status = BOngoing and (b.~battles)'.status =
TOngoing
      eventually all b:Battle | b.status = BOngoing implies b'.status
= BConsolidation
      eventually all b:Battle | b.status = BConsolidation implies
b'.status = BClosed
}

pred TeamFormation {
      eventually some t: Team | all b: Battle |
      b.status = BRegistration and b in t.participate and after
(b.status = BRegistration and #Team = 2)
      //showing more numerous team (at most one singleton)
      always lone t: Team | #(t.~isMemberOf) = 1
}

pred TournamentSubscription {
      eventually all s: Student, t: Tournament | t.status =
TRegistration
      implies after t in s.subscribed and t.status = TRegistration
}

pred show{

      TournamentSubscription; TournamentEvolution;
      TeamFormation; BattleEvolution;
      BattleEvolution; TournamentEvolution

}
run show for 7
```

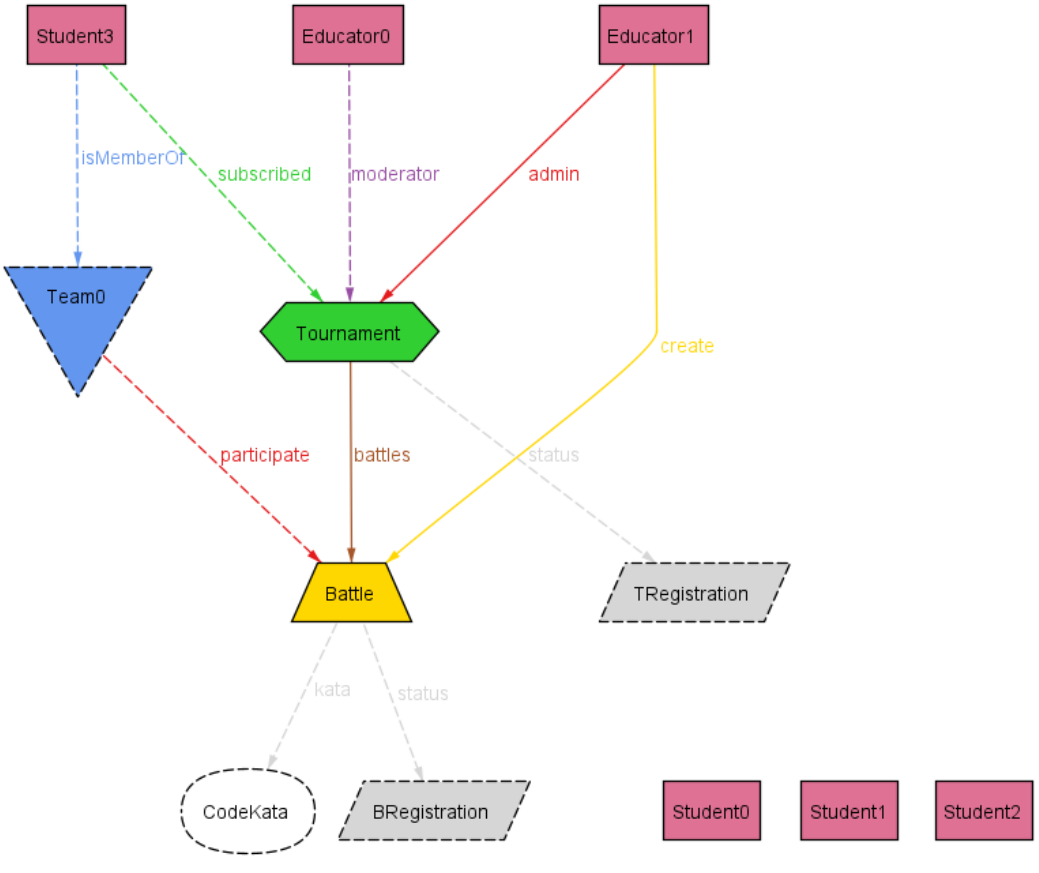
With the definition of the following predicates and the initial situation (see `fact "init"`), one of the instances of the evolution of the dynamic model is shown.

Here are all the steps proposed by the Visualizer.

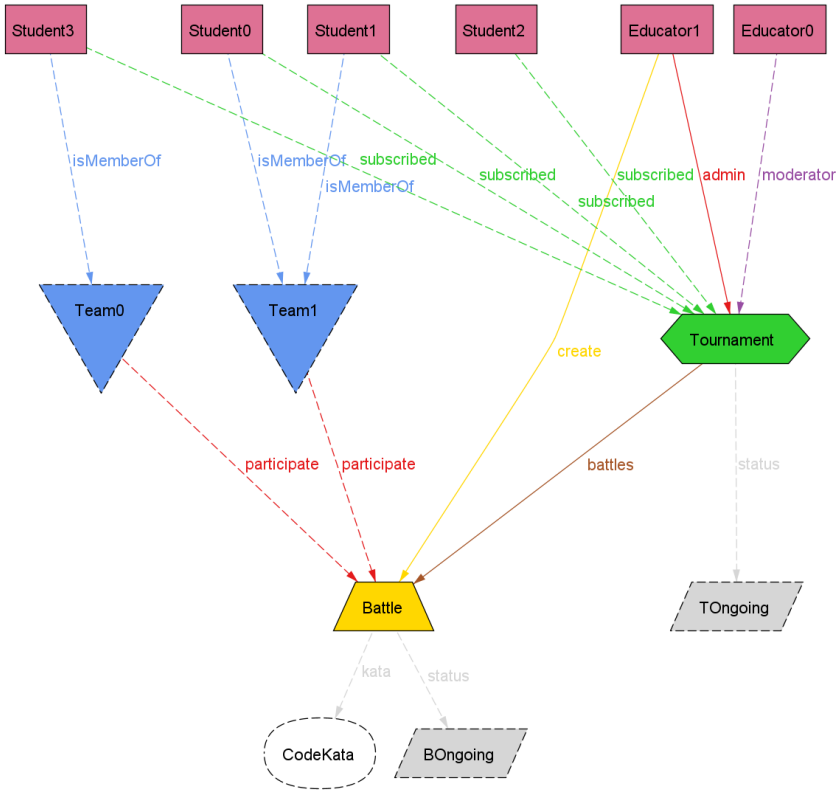


In this initial step, the tournament has just been created by Educator1, who has also created a Battle and given Educator0 the role of Moderator. Both the Tournament and the Battle are in the Registration stage.

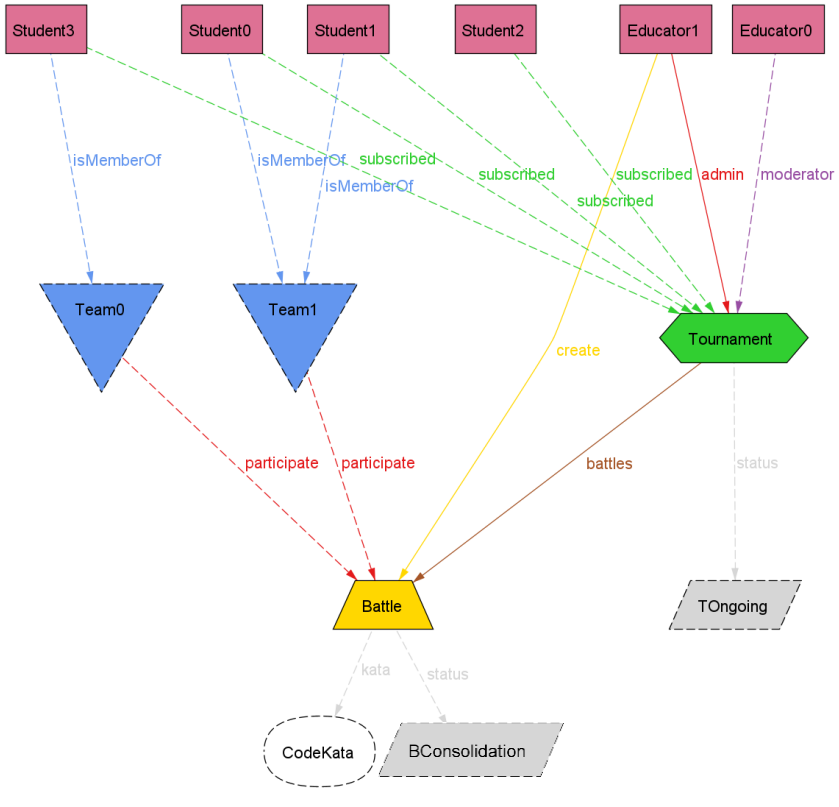
At this point, four Students are subscribed to CKB platform but none of them is subscribed to the tournament yet.



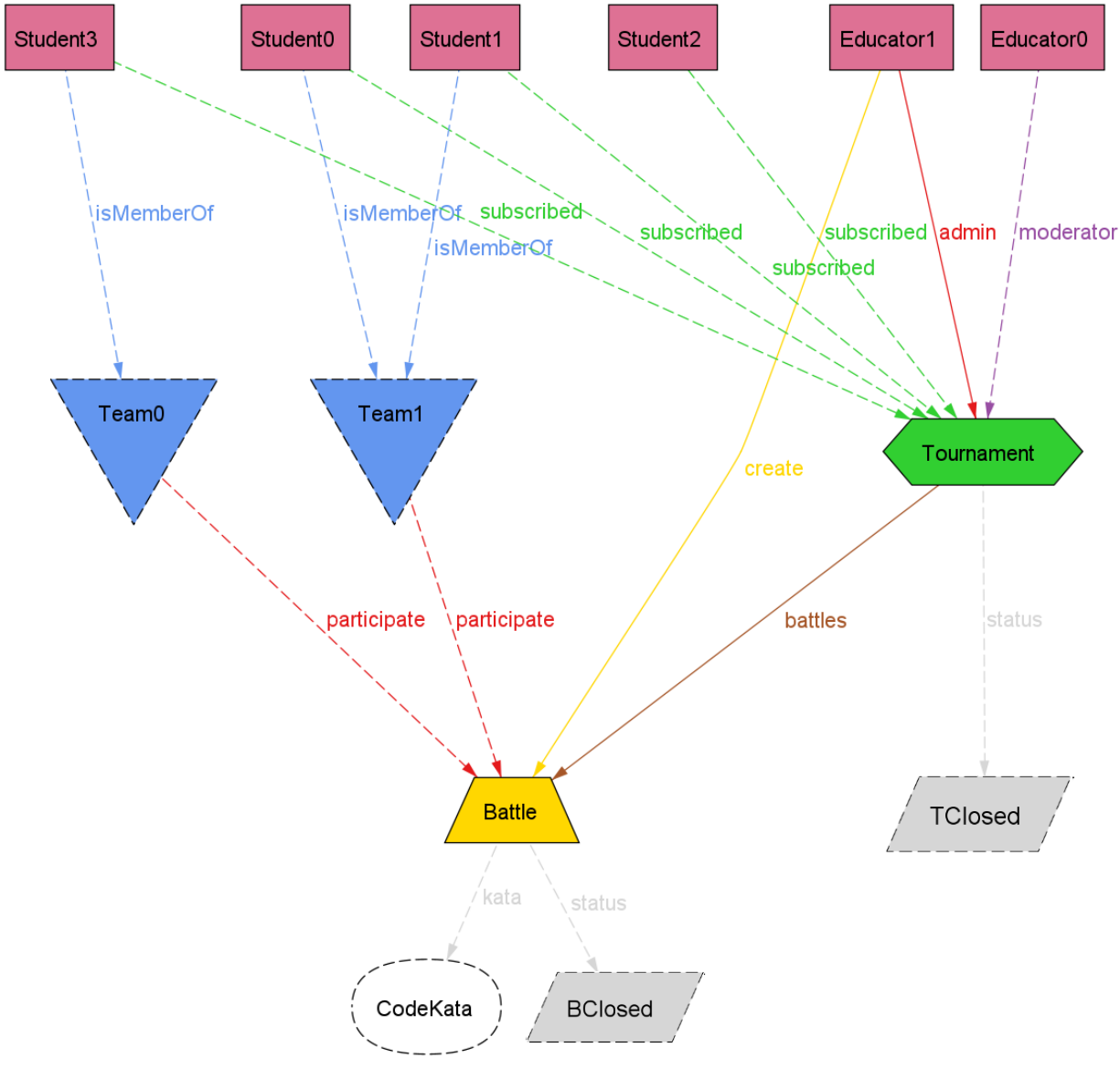
At this step, Student3 has subscribed both to the Tournament and to the Battle, creating a new team called Team0.



At this step, it is observable that Student0 and Student1 have subscribed to the Tournament and formed a team, called Team1, to participate in the Battle. In the meanwhile, the registration deadline of the Battle has expired: both the Battle and the Tournament move to the Ongoing state. From now on, students can't subscribe to the tournament and form team for the battle any longer.



At this step, the Battle has gone into the Consolidation stage and after it is ended by the Admin of the tournament both the tournament (involving no other battles) and the battle can go into Closed stage, where students can have a look at the results for a while. The final situation is shown just below.



5. EFFORT SPENT

Sections	Fornara	Colecchia	Together	TOTAL
Section 1	3	2	6	11
Section 2	3,5	3,5	4	11
Section 3	11	4,5	10	25,5
Section 4	1,5	7	9	17,5
Revision&Restyling	1	3	6	10
TOTAL	**20**	**20**	**35**	**75**