POLITECNICO
MILANO 1863

SOFTWARE ENGINEERING II

CKB – CodeKataBattle
Design Document

Version 1.0

Polito Attilio

Rigione Pisone Raimondo

Soricelli Francesco

January 7, 2024

CONTENTS

LIST OF TABLE

LIST OF FIGURE

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to describe in a more specific and precise way, with respect to the RASD document, the behaviors and the characteristics of the system. In particular the choices regarding the architecture will be justified and a description of the specific components, needed for the implementation of the system, run-time processes and all other design choices will be analyzed. Furthermore, some additional and more specific user interfaces regarding the system both from the side of the student and from the side of the educator will be presented. Finally, the implementation and integration plan and the testing plan will be showed in full details. All these elements, exposed in the document, will be used by different stakeholders and by the software engineers that oversee the project's implementation and testing, in order to be able to understand the detailed specifications without any ambiguity. The aim of the CodeKataBattle (CKB) platform is to provide a dynamic and engaging environment for students to improve their software development skills through collaborative learning. The project aims to foster a culture of continuous improvement using coding challenges and competitions in a structured educational environment. The platform not only improves technical skills, but also provides continuous learning and improvement for the students in the field of software development.

1.2 Scope

The CodeKataBattle (CKB) project demands innovative solutions to effectively train students, preparing them for the challenges of the professional world, with the increasing importance of skills in software development and the increasing complexity of the industry providing students with an interactive and collaborative environment to improve their abilities. CKB provides a platform that allows students to refine their skills through practical and exciting programming challenges.

The platform has 2 main actors - students and educators - who are actively involved, each playing a unique role. Students who make up the active components of the system participate in the Battle by registering individually or forming teams based on specific participation requirements. When a student joins the battle, after the battle starts, he will receive a notification with a link to the assigned code kata GitHub repository for his Team. Here, following the test-driven development (TDD) approach, he'll complete the exercise and submit the solution by the specified deadline.

Educators, on the other hand, act as architects of competitions. They have a role in creating tournaments and defining combat specifications, including details such as code type descriptions, programming languages, tests to pass, registration and submission deadlines. An Educator can also set mandatory evaluation parameters, such as test code quality, and optional evaluation parameters, such as the manual evaluation feature. After an Educator has created a battle, the platform manages the student enrollment process, distributes GitHub repositories, and automates evaluation through GitHub Actions.

During the competition, the system dynamically assigns scores to the team based on automatic evaluation (test success, timeliness of submission, quality of code, etc.) and manual evaluation by educators. The scores accumulated in each battle contribute to each student's overall tournament score. To further enrich the experience, the platform has introduced a gamification system through the awarding of badges. Educators define rules and variables associated with badges that are assigned to students based on their performance during the tournament. These gamification badges will be displayed in the student's profile.

Joining CodeKataBattle not only improves technical skills, but also promotes collaboration, problem solving, and compliance with best practices in software development. This platform prepares students for the challenges they face in their professional careers. In addition, the gamification element adds an exciting layer to the learning process and motivates students to strive for excellence.

Going forward, the CKB team expects continuous improvement and expansion of the platform. Future plans include considering additional features, improving the evaluation process, and incorporating feedback from both students and educators to ensure an optimal learning experience. The goal is a major platform to hone the skills of the next generation of software developers evolving with the ever-changing landscape of software development education.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Definition	Description
CodeKataBattle (CKB)	The online platform designed to hold coding competitions and challenges to improve students' software development abilities.
Code Kata	The exercise that includes a brief textual description and a software project with build automation scripts that contains a set of test cases.
Test-first or test-driven development (TDD) approach	A software development model that necessitates writing automatic tests before writing the software that needs to be tested. The development of the application software is solely focused on passing the previously written automatic tests.
Tournament	Organization on CKB created by an educator, consisting of a series of challenges.
Battle	Challenge or competition that is developed by an educator and held during a tournament.
Team	A group of students coming together to take part in a particular CKB battle. Students create teams on the platform before starting a battle.
GitHub repository	A distributed version control system and digital storage area powered by Git. In this instance, the "code kata" refers to the CKB platform programming project linked to a particular battle, and it can be found in the GitHub repository. For the battle, every team has a dedicated GitHub repository where students can work together to track changes, solve problems, and version code.
GitHub Action	GitHub provides an automation service that lets developers automate software workflows. In this case, GitHub Actions are useful to set up an automated workflow between GitHub and the Platform. This workflow is used in the competition to send an API notification to the CKB platform after every commit, enabling the team's score to be automatically updated and computed based on the most recent changes made to the code.
Score	A natural numerical value between 0 and 100 that is established by accounting for some required factors that are assessed entirely automatically and other optional factors that are assessed manually by educators.
Manual evaluation	A subjective evaluation component made by educators who look over and assess students' work during the consolidation phase. This stage

	provides a chance to assess factors that might not be picked up by an automated evaluation and that might be more specialized or unique.
Notification system	The automated system that alerts teachers and students to impending fights, deadlines, and tournament outcomes.
Gamification badges	Virtual awards that educators define as a representation of each student's achievements during a competition.
Valid credentials	The term valid credentials is referred to the email and the password inserted by a user during the login process; they are said to be valid if they correspond to the one inserted by such user during the successful registration process.
Conflicting data	The email address inserted during the registration process by a user is defined as a conflicting data if there are already others users of the CKB application that have registered with the same address
Tournament Management Permissions	An educator can manage a tournament (creating battles, closing the tournament…) if he has created the tournament or another educator has given him the permissions though the "granting permission" function.

Table 1: Table of definitions

1.3.2 Acronyms

Acronym	Description
DD	Design Document
RASD	Requirement Analysis and Specificatio Document
API	Application Programming Interface
CKB	CodeKataBattle
TDD	Test-Driven Development
DBMS	Database Management System
HTTP	Hypertext Transfer Protocol
TCP-IP	Transmission control protocol – Internet protocol

Table 2: Table of acronyms

1.3.3 Abbreviations

Abbreviation	Description
Rn	n-th functional requirement

Table 3: Table of abbrevations

1.4 Revision history

Version 1.0 - 07/01/2024

1.5 Reference documents

1- Specification document: R & DD Assignment A.Y. 2022-2023;
2- Course slides;
3- DDs of the previous academic years;
4- CODEWARS: https://www.codewars.com/;
5- CodeKata: http://codekata.com/;
6- TDD: https://en.wikipedia.org/wiki/Test-driven_development.

1.6 Document structure

• **Section 1:** In this section, it is described the purpose of the document and, more in general, the purpose of the CKB project. Furthermore, in the scope section it is analyzed the problem that the system aims to solve, providing a description of the environment in which the system will be used and the goals that it has to reach in order to satisfy the users' needs. In addition, it is reported a list of definitions, acronyms and abbreviations, with their respective description, that will be used throughout the document. Finally, it contains other sub-sections dedicated to the revision history, the reference documents and the document structure.

• **Section 2:** This section deals with the architectural design of the system. It contains: an overview on the most important choices made regarding the architecture, a class diagram, a component diagram and a deployment diagram, each of them paired with the correspondent analysis and description. In addition it contains the runtime view, which includes the updated sequence diagrams with respect to the software components defined in the component diagram. Finally, it contains a sub-section that highlights the utilized interfaces and their dependencies, one that describes the architectural styles and design patterns that have been chosen and one that presents other design decisions.

• **Section 3:** This section contains the graphical user interface mockups for the different views of the system.
• **Section 4:** This section contains a mapping of the requirements, which have been defined in the RASD document, to the software components (described in this document) that will ensure that they are satisfied.

• **Section 5:** This section presents the implementation, integration and testing plan that it has been decided to adopt for the developing of the system.

• **Section 6:** This section indicates the effort spent by each group member while working on this document.
• **Section 7:** This section contains the references.

2 ARCHITECTURAL DESIGN

2.1 Overview

The CodeKataBattle (CKB) web application is designed with three key components to deliver its functionalities effectively. These components include a client, a server, and a database. The client, tailored for compatibility with web browsers on PCs, serves the needs of both students and educators engaging in coding challenges. The server acts as the central hub for communication among clients, manages interactions with the internal database, and interfaces with external data sources through APIs. To ensure load distribution and scalability, a load balancer is implemented. The internal database stores diverse information contributed by users and periodically integrates data obtained from external sources.

The CKB web application adopts a three-tier architecture, dividing the system into three distinct groups of components, each serving a specific purpose:

1. **Client Tier:**
    - This tier houses components responsible for views and controllers, defining the user interfaces for students and educators accessing the web application via their PCs.
    - User interfaces obtain necessary data consistency through communication with the application logic tier, minimizing data allocation on the client side.
    - Following a thin client model, this component remains connected to the server to deliver the system's functionalities.
2. **Application Logic Tier:**
    - This tier encapsulates all application logic necessary for processing data collected from the client tier and retrieved from the data tier.
    - It can perform operations such as adding, deleting, or modifying data in the data tier, utilizing API calls for communication.
    - Interacts with external services through API calls to acquire relevant information for the system, subsequently copying it into the internal database.
3. **Data Tier:**
    - Corresponding to a database, this tier houses all data relevant to the CKB web application's scope.
    - Data includes information from external services, user-provided information, and user credentials.
    - The application tier accesses the data tier whenever information retrieval or modification is required.

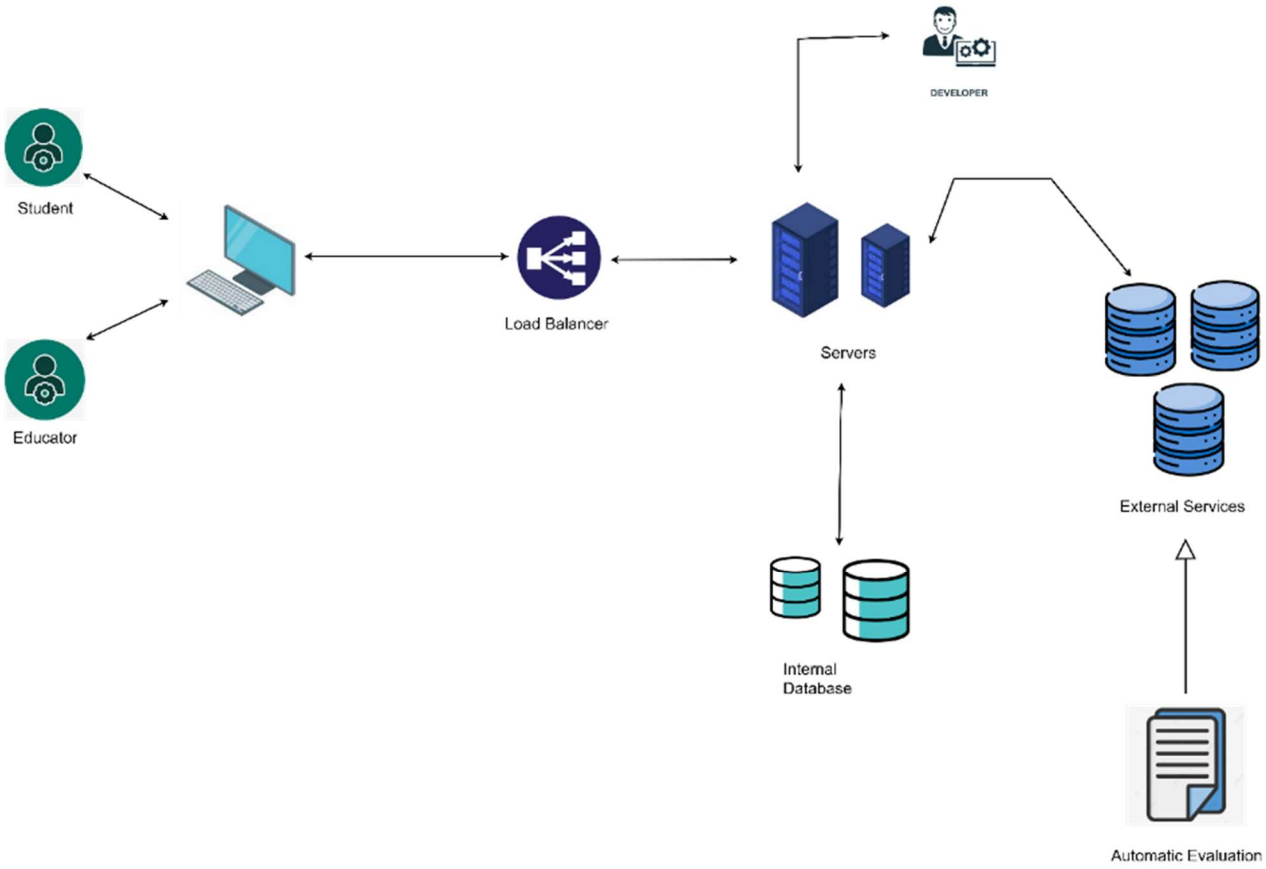

Figure 1: High level architecture

This three-tier architecture ensures a structured and modular design for the CodeKataBattle web application, promoting scalability, maintainability, and effective separation of concerns across different layers of the system.

2.2 Component View

2.2.1 Class Diagram

In this section, it is reported the class diagram, which has been already analyzed in the RASD document. In this version of the diagram, there have been added the data types for each entity and some attributes that were considered to be important for the implementation of the application, such as the “role” attribute for the user.

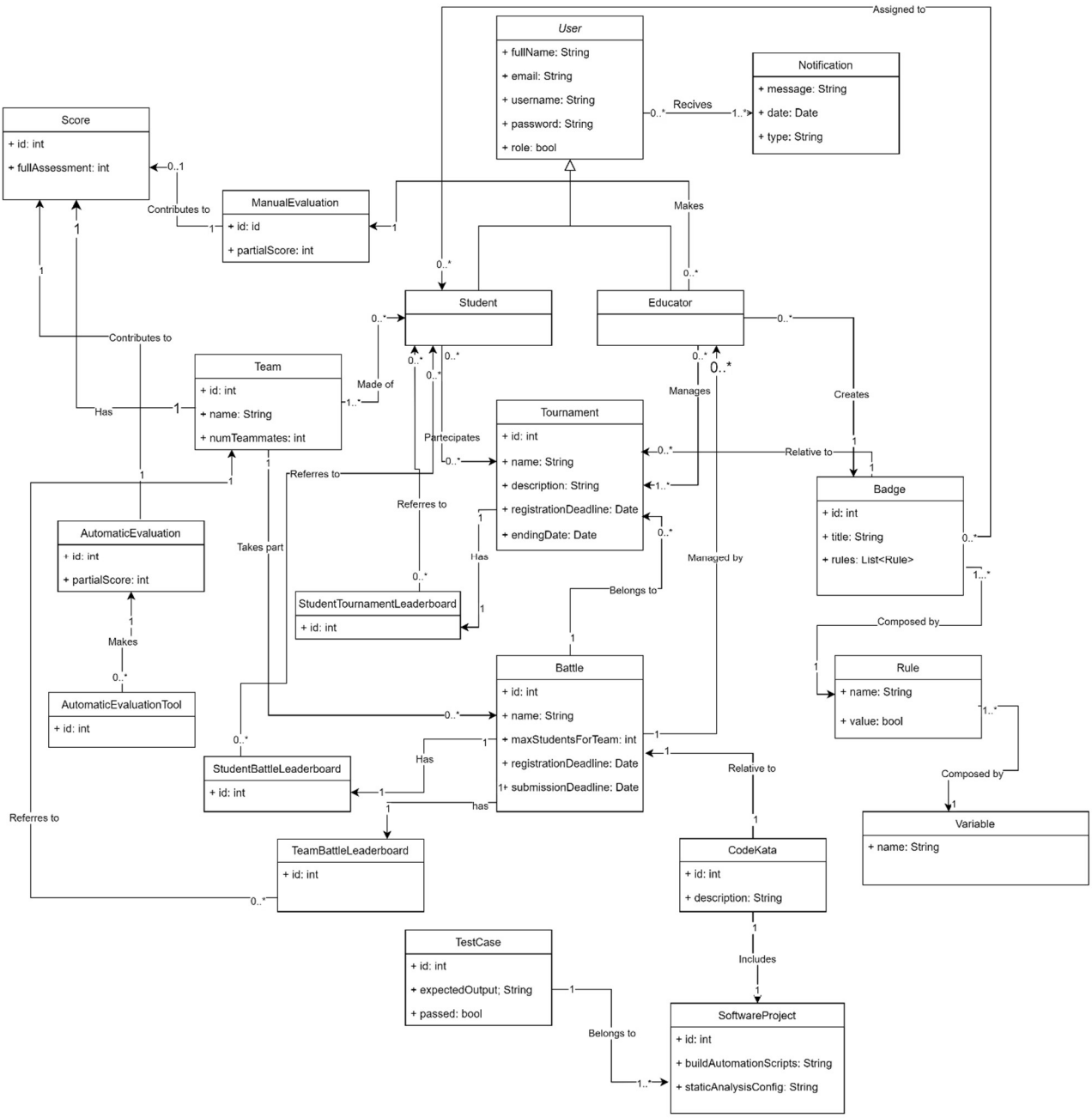

Figure 2: Class Diagram

The "role" attribute has been introduced into the diagram for a User; it is set to true if the user is an educator and false if the user is a student.

Another important change is relative to the badge creation process: rule and variable classes were made in order to precisely specify methods useful to provide this functionality of the system.

These updates have been made also in the RASD document by creating a second version of it.

The "staticAnalysisConfig" attribute of the SoftwareProject corresponds to a string (code fragment) containing all parameters for the “Static Evaluation Tool” in order to make the automatic evaluation process possible.

Finally, the methods that will implement the several functionalities of the application are not represented; however, they will be analyzed in a diagram that will combine the class diagram and the component diagram, which represents all the possible services.

2.2.2 Component Diagram

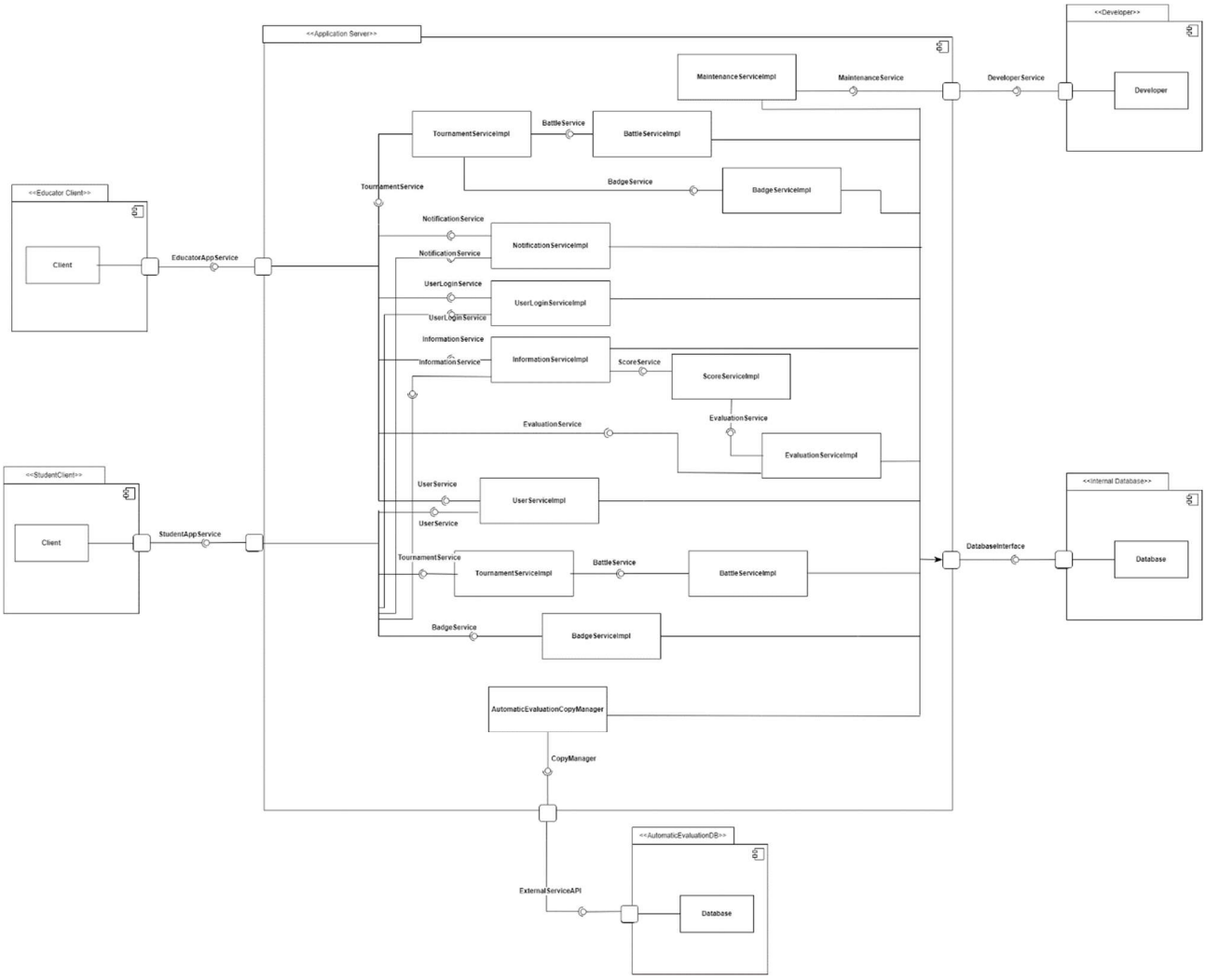

Figure 3: Component Diagram

The diagram above represents the main components of the architecture previously described. The server includes several services, which are necessary to provide all the functionality to users. These services implement more general interfaces, which indicates that there may be several ways to implement them.

Regarding BadgeService, BattleService, TournamentService, they are duplicated to provide a more visible diagram and to better understand how they interact with each other based on the type of user using them; All the application logic is contained in the server component, as mentioned above, while the two different types of clients (the student's and the educator's) and the database reside on separate software components and interact with the server through specific dedicated interfaces. Finally, the communication of the system with external service, represented by an external database related to automatic assessment, is handled by the server through the implementation of a specific interface that will take care of accessing the external database and copying its contents into the internal database of the application.

Below is provided another more specific UML diagram (which has been split into three images to make it more visible) that includes the signatures of the main methods of the services specified in the component diagram, and which model classes are used in each service. The additional attributes (compared to those of the entities in the class diagram) are written in blue and represent the associations between the different entities needed to implement the methods of the services. Interfaces representing specific services are connected to classes through "use" arrows, meaning that such an interface makes use of the respective classes to implement its methods. For example, the "UserLoginService" interface uses the User class to provide a method that verifies the credential entered on the login page. Also, if a "use" arrow connects an implementation of one service to the interface of another service, it means that that implementation makes use of some methods of the other service to provide its functionality. Finally, implementations of the services are connected through an arrow to the respective interface of the service they implement.

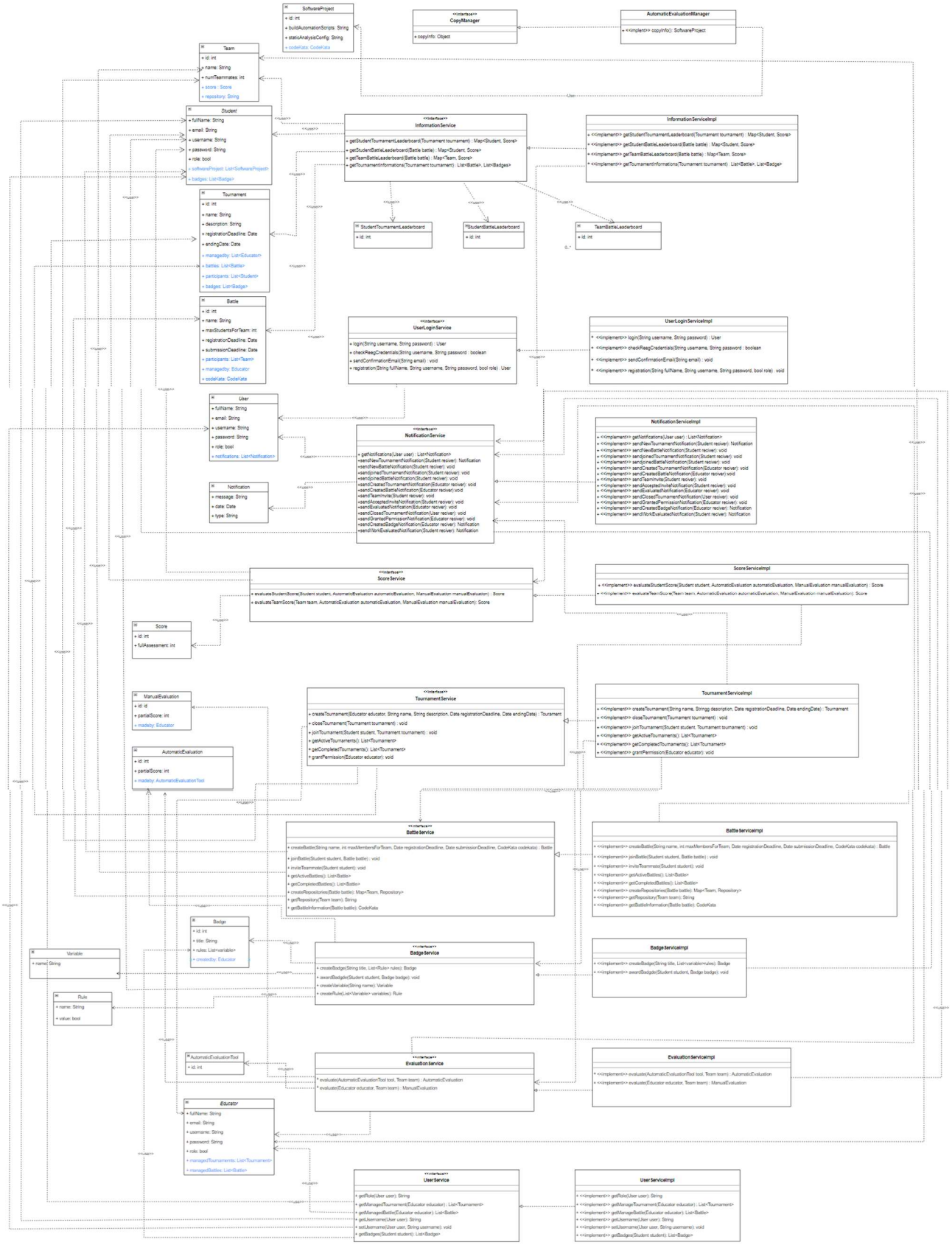

Figure 4: UML Diagram

CopyManager Interface:

The CopyManager Interface orchestrates the copying of data into the internal database. The primary method, **copyInfo()**, universally returns a generic **Object**. To diversify implementations, the interface introduces the **AutomaticEvaluationCopyManager** that specializes in copying information pertaining to automatic evaluations. The **copyInfo()** method in this implementation specifically returns an instance of **AutomaticEvaluation**.

InformationService Interface:

The InformationService Interface is tailored to retrieving and presenting diverse information types for user visualization. Methods such as **getStudentTournamentLeaderboard** and **getStudentBattleLeaderboard** offer insights into tournament and battle leaderboards for individual students. Additionally, **getTeamBattleLeaderboard** provides team-based leaderboard information. The method **getTournamentInformations** delivers a list of battles and a list of badges associated with a particular tournament.

UserLoginService Interface:

UserLoginService Interface serves the purpose of facilitating user authentication and registration processes. The **login** method assesses the validity of user credentials, while **checkRegCredentials** ensures the integrity of parameters during registration. The interface further supports processes like sending a confirmation email (**sendConfirmationEmail**) and user registration (**registration**).

NotificationService Interface:

Dedicated to communication with users, NotificationService enables the platform to send Notifications and retrieve them. The **getNotifications** retrieves a list of notifications for a given user, all the other methods are responsible for notifications' creation.

ScoreService Interface:

ScoreService Interface handles the evaluation of student and team scores. The methods **evaluateStudentScore** and **evaluateTeamScore** are central to assessing individual and team performances, respectively. These evaluations involve scores, automatic evaluations, and manual evaluations.

TournamentService Interface:

TournamentService Interface focuses on managing tournaments. Methods like **createTournament** initiate the creation of tournaments, specifying details such as name, description, registration deadline, and ending date. The **joinTournament** method allows students to participate, and **getActiveTournaments** and **getCompletedTournaments** retrieve lists of active and completed tournaments.

BattleService Interface:

BattleService Interface is designed for handling battles within the system. The **createBattle** method initiates the creation of battles, specifying details like name, maximum members for a team, registration deadline, and submission deadline and the CodeKata for that Battle. Other methods, such as **joinBattle** and **inviteTeammate**, facilitate student participation and teammate invitations.

BadgeService Interface:

BadgeService Interface is responsible for managing badges within the system. The **createBadge** method initiates the creation of badges, requiring a title and a list rules. Badge Creation is possible thanks to createVariable and createRule methods, wich allow educators to create variables and rules for a Badge. The interface further supports awarding badges to students through the **awardBadge** method.

EvaluationService Interface:

EvaluationService Interface focuses on evaluating students and teams, both automatically and manually. The **evaluate** methods take parameters such as the evaluation tool, student or team, and educator. The interface plays a crucial role in assessing performances and assigning appropriate evaluations.

UserService interface:

UserService Interface foucses on retrieving users informations. The **getRole method** is used to understand the role of an user (student or educator); **getManagedTournaments** and **getManagedBattles** are instead used for retrieving informations regarding tournaments and battles managed by an educator, while **getUsername** and **setUsername** are used for username management and visualization. At the end, we have the **getBadges** method that retieves all the badges achieved by a student.

These interfaces collectively form the core functionality of the CodeKata system, covering user management, communication, evaluation, and tournament/battle orchestration.

2.3 Deployement view

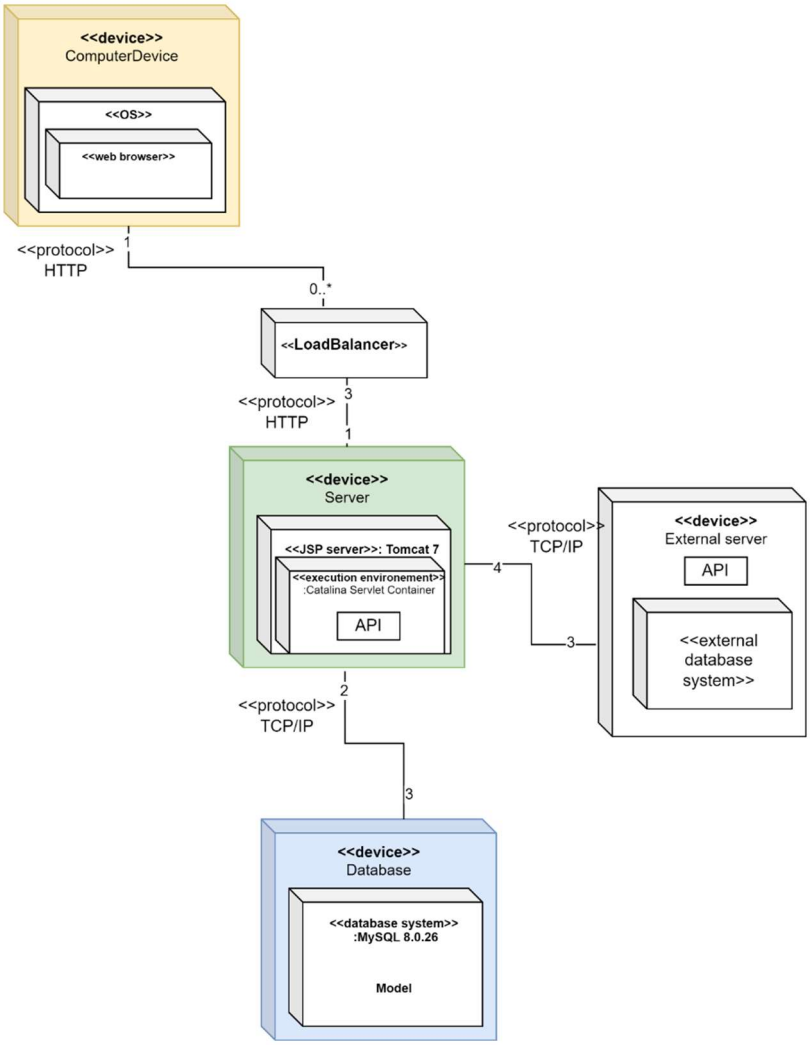

Figure 5: Deployment Diagram

The above deployment diagram shows the architecture of the system in its distributed nature. The nodes represent hardware or software execution environments and the middleware connecting them. The colors used in the diagram reflect the fact that the system is developed as a three-tier architecture (as described in the section 2.1). In fact, the nodes represented in yellow are part of the client tier, the one in green composes the application logic tier and the one in blue the data tier. Furthermore, it is shown a node representing the external services, which reside on external servers and are equipped with APIs that allow the communication with the system. In addition, it is reported the load balancer, which is important to manage the concurrent accesses of several clients to CKB web application. Finally, it can be observed that the cardinalities of the associations between the nodes are chosen accordingly to the fact that the functionalities of the application logic and data tiers have to be replicated on multiple devices in order to face possible failures and prevent information losses; in particular, it was deemed sufficient to consider three servers and two databases that compose the system, but in future implementations of the CKB system it could be considered necessary to integrate some additional devices.

2.4 Runtime view

In this section all the sequence diagrams that have been made in the RASD document will be expanded specifying which services of the application server operate to provide such functionalities. The entire communication between the CKB application and the services is deployed through the services' methods specified in the UML diagram in 2.3. Whilst, for what concerns methods about the communication between user and CKB application, they are generic and will be provided through specific elements in the user interfaces. In particular, from the side of the client: in order to upload some data, it is used a method upload(some parameters), in order to request a specific view, it is used a method get(specific view), in order to choose some parameters for filtering, it is used a method choose (some parameters) and so on. While, from the side of the application: in order to show a view it is used a method show(specific view), in order to show a success or error message it is used a method show(Message) and so on. In some situations the order of actions of these sequence diagrams differs from the original one in the RASD document. This choice has been made in order to be coherent with the specific methods (explicated only in the previous chapter) that have to be used.

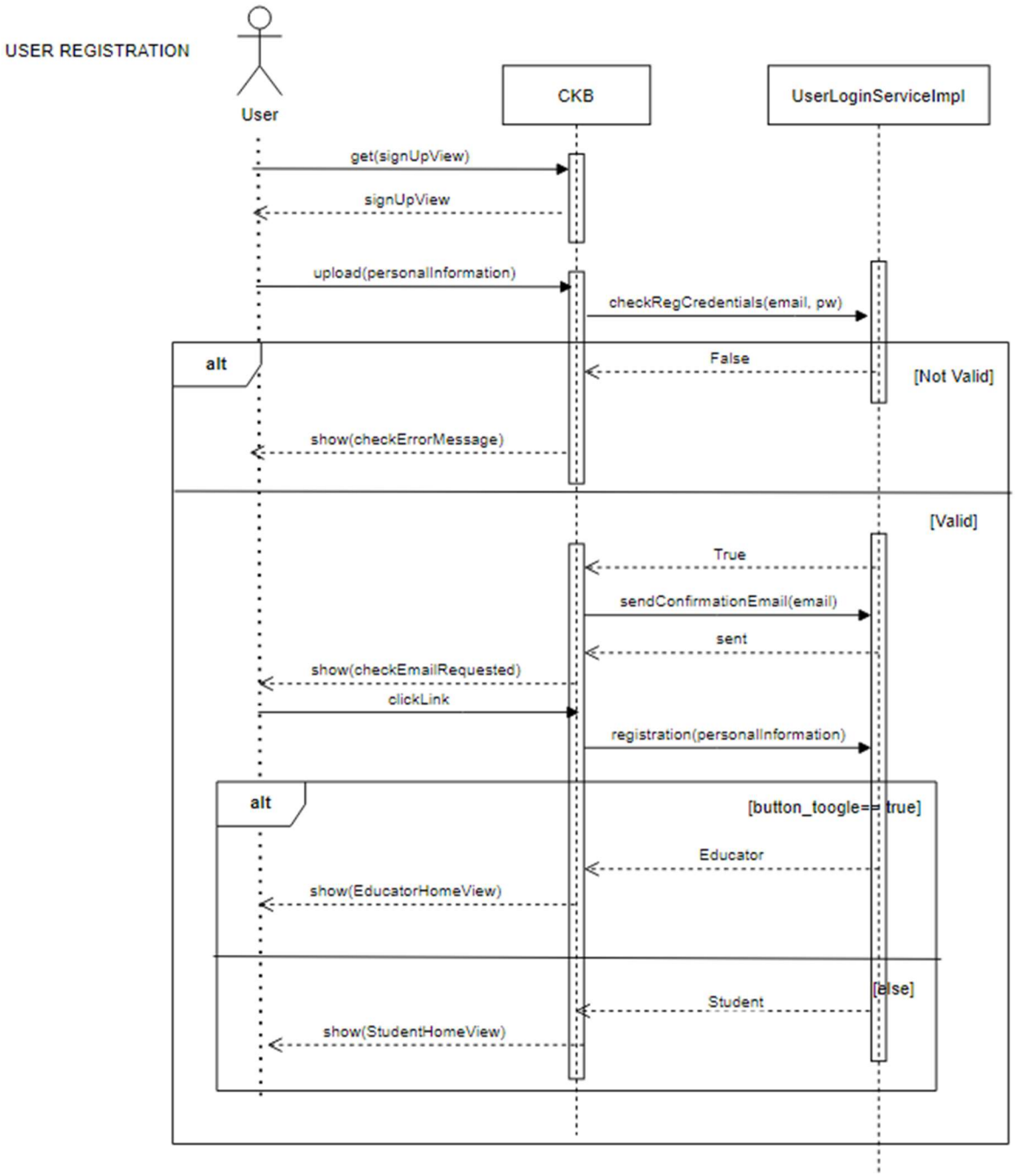

Figure 6: 1st sequence diagram

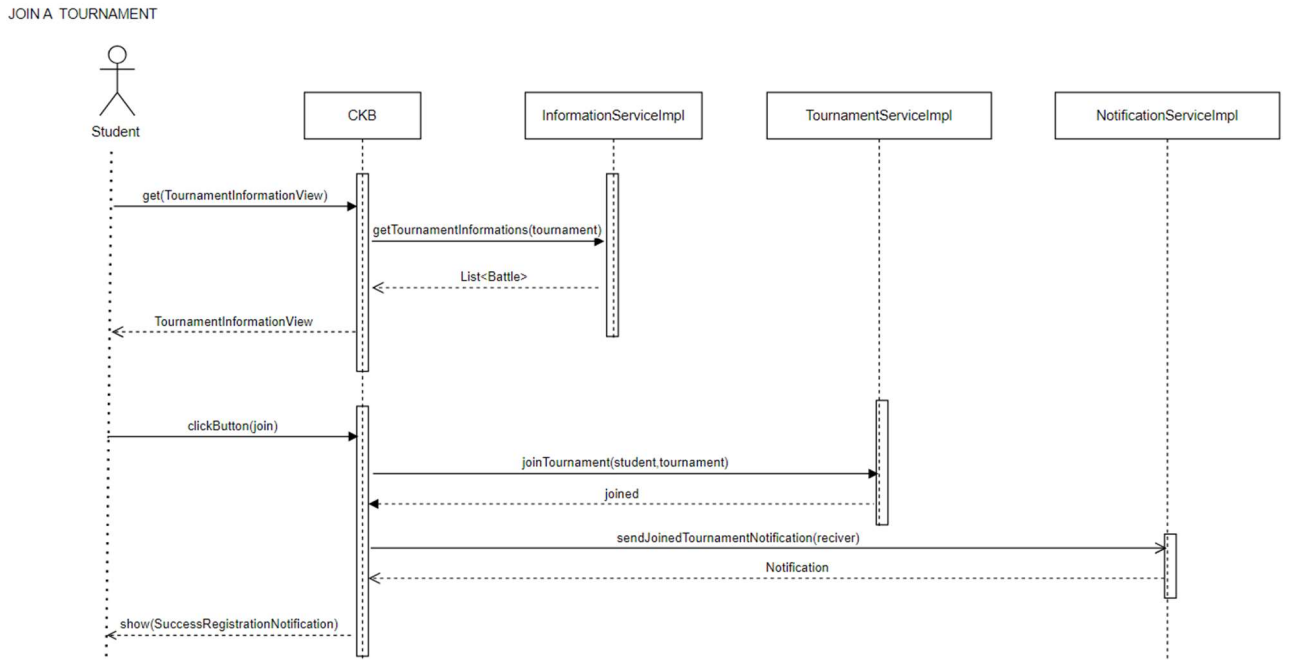

Figure 7: 2nd sequence diagram

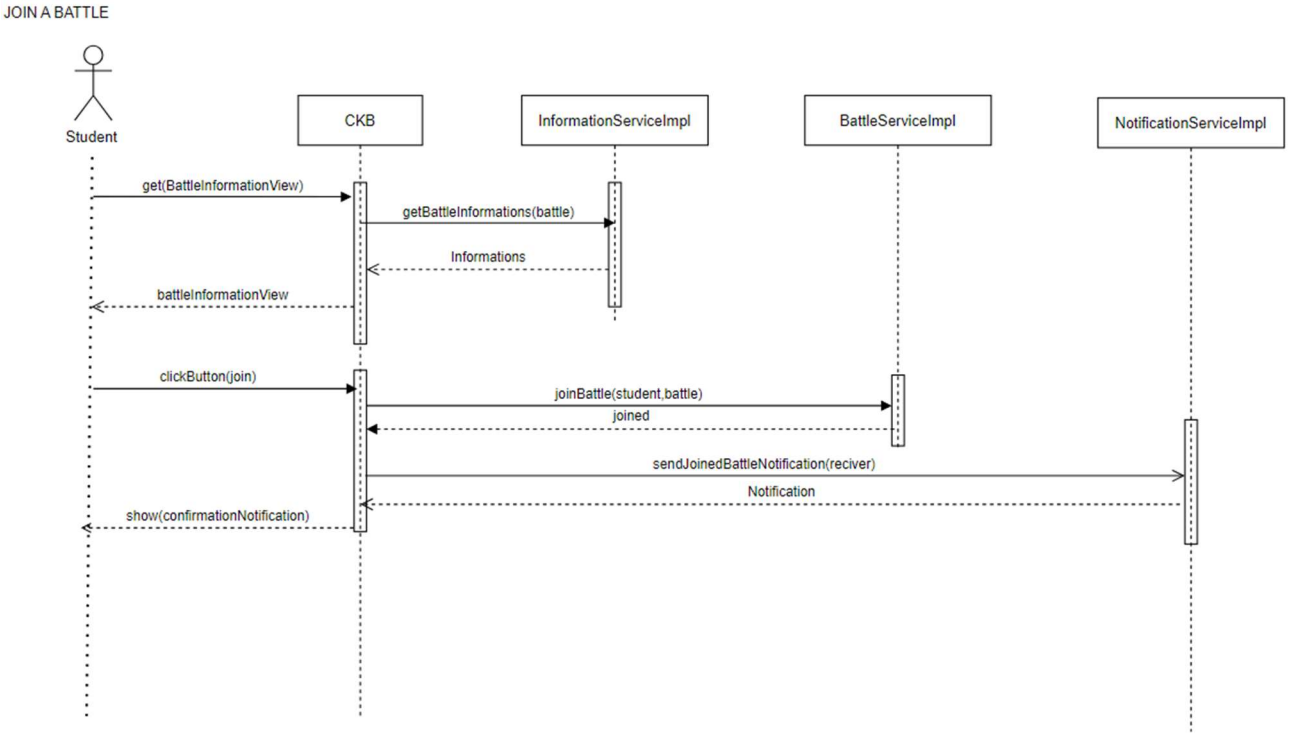

Figure 8: 3rd sequence diagram

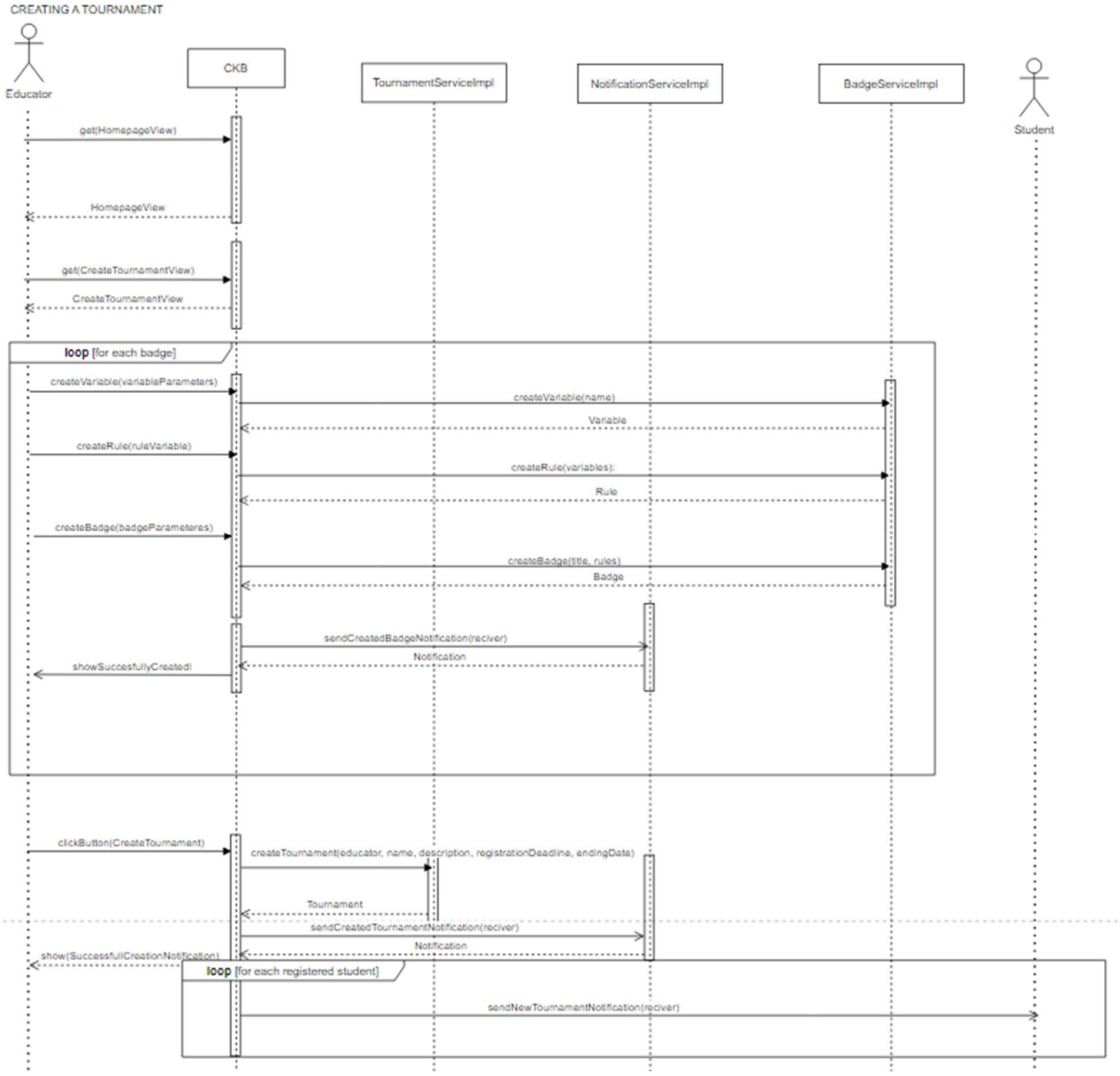

Figure 9: 4th sequence diagram

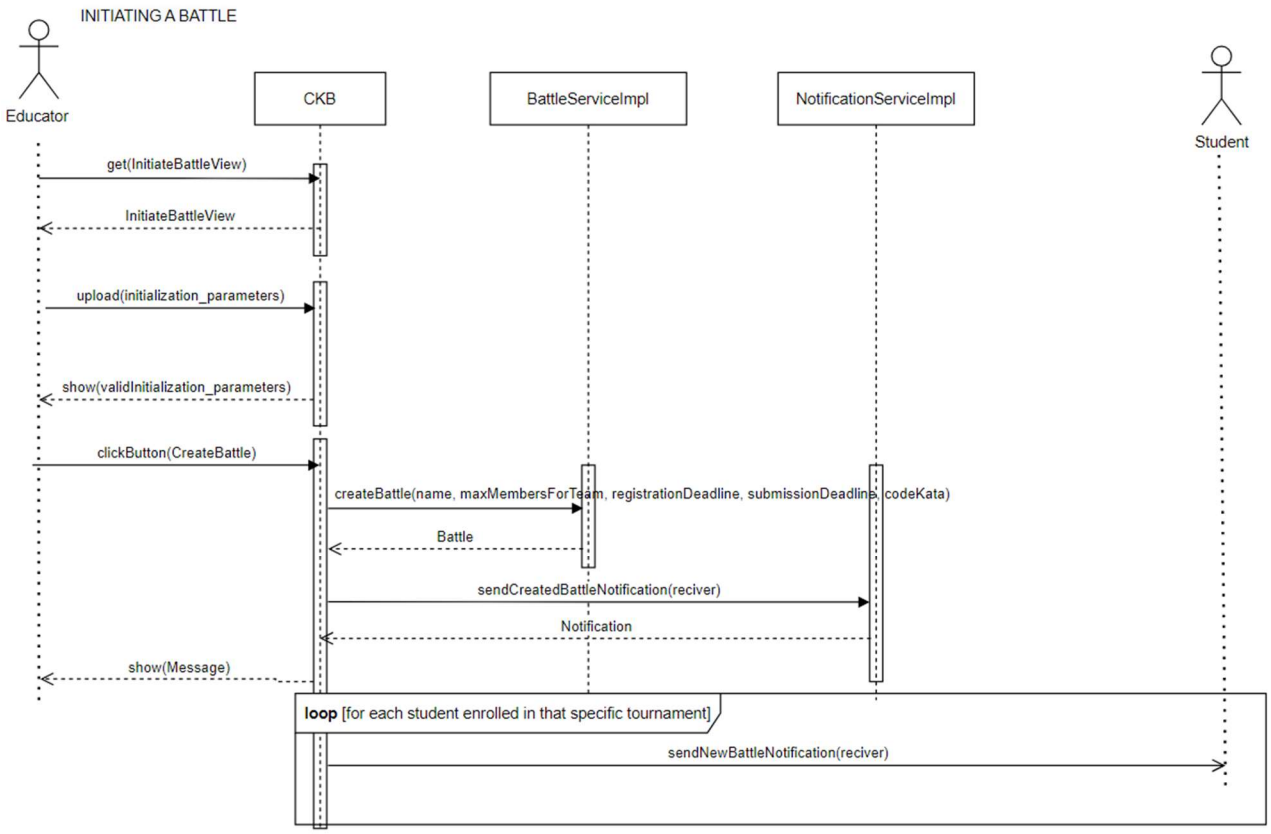

Figure 10: 5th sequence diagram

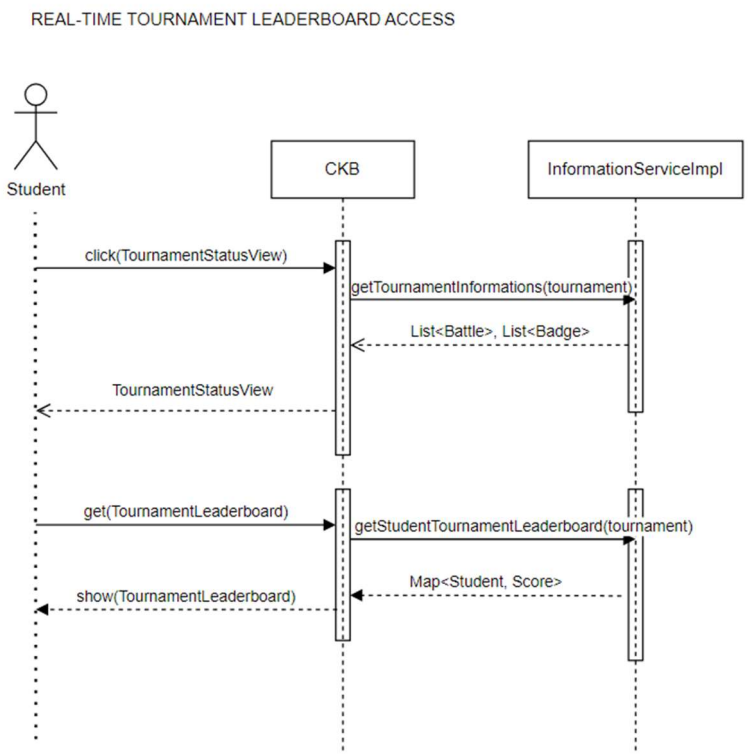

Figure 11: 6th sequence diagram

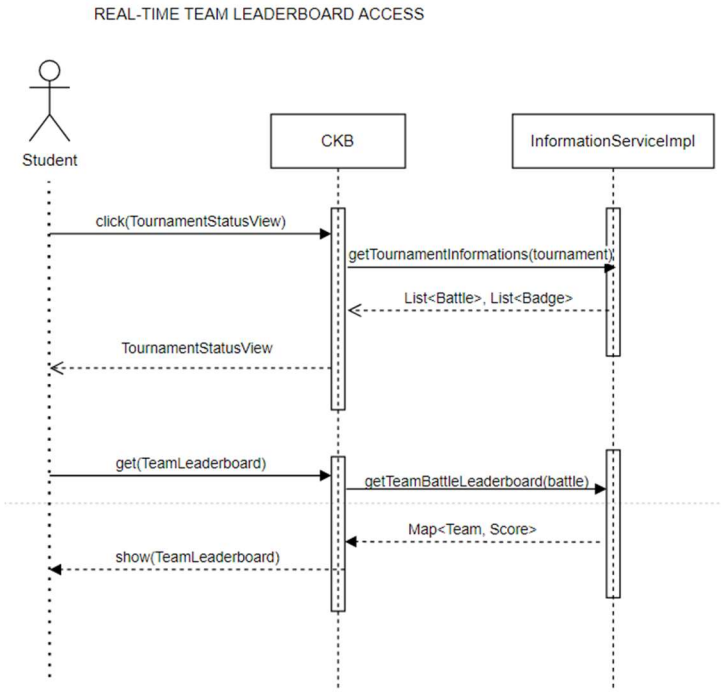

Figure 12: 7th sequence diagram

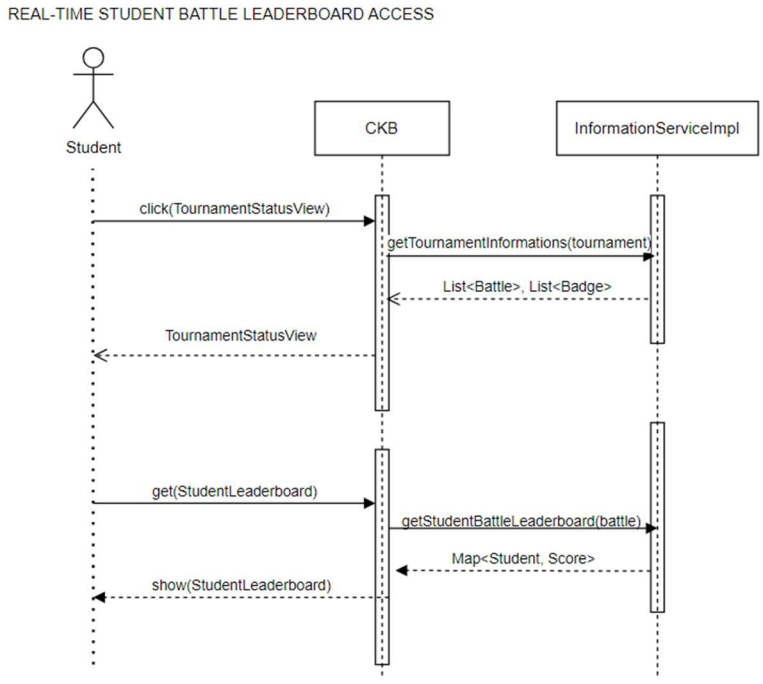

Figure 13: 8th sequence diagram

FORMING A CODE TEAM

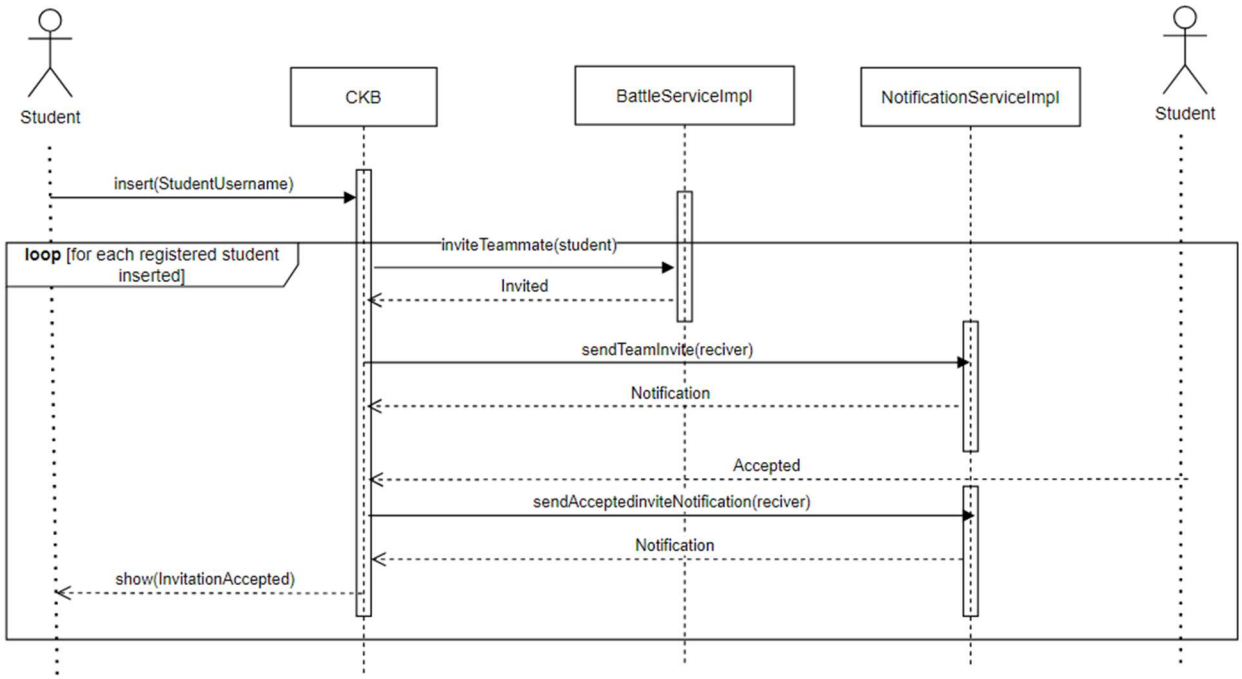

Figure 14: 9th sequence diagram

MANUAL ASSESSMENT BY AN EDUCATOR

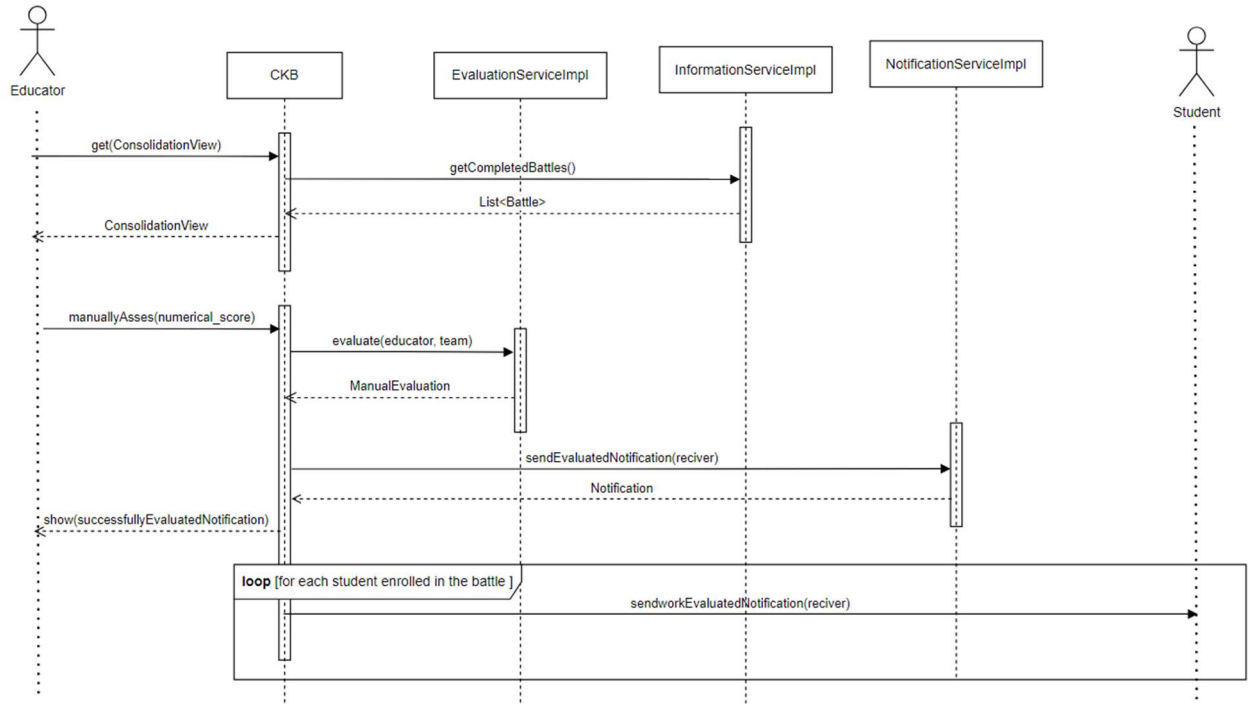

Figure 15: 10th sequence diagram

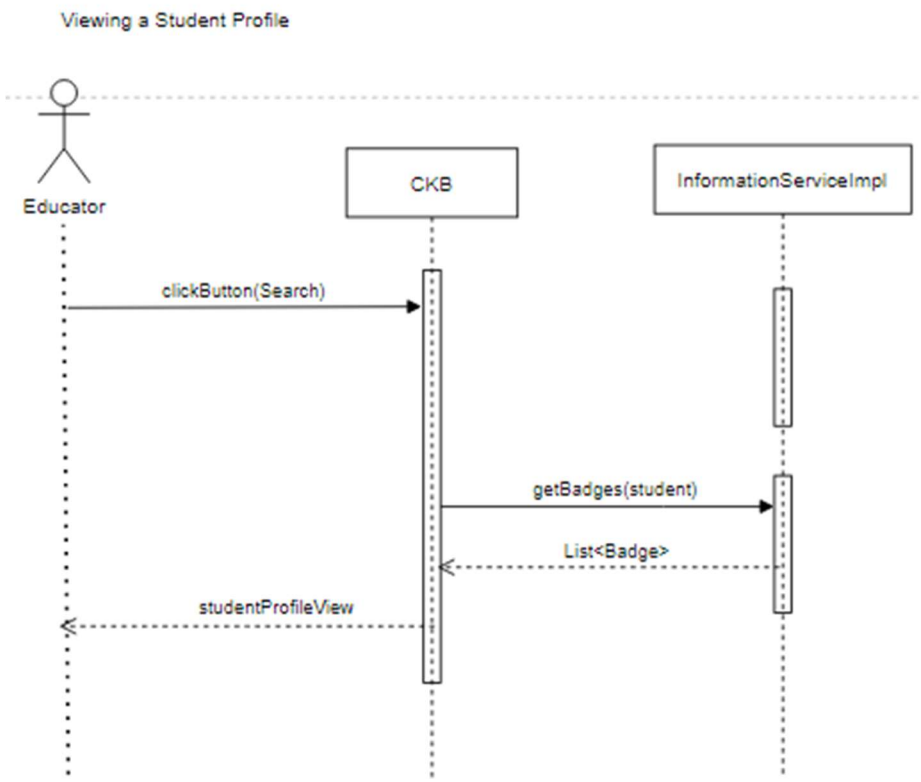

Figure 16: 11th sequence diagram

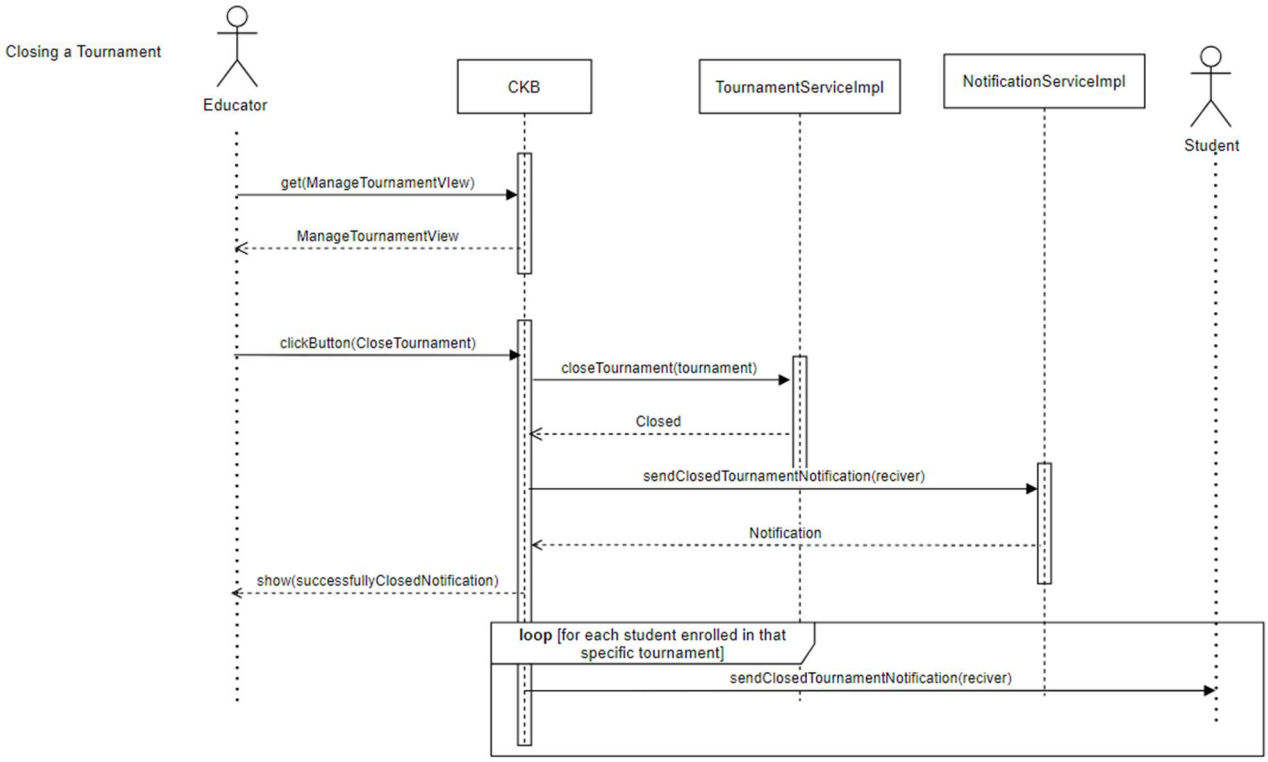

Figure 17: 12th sequence diagram

GrantingPermission

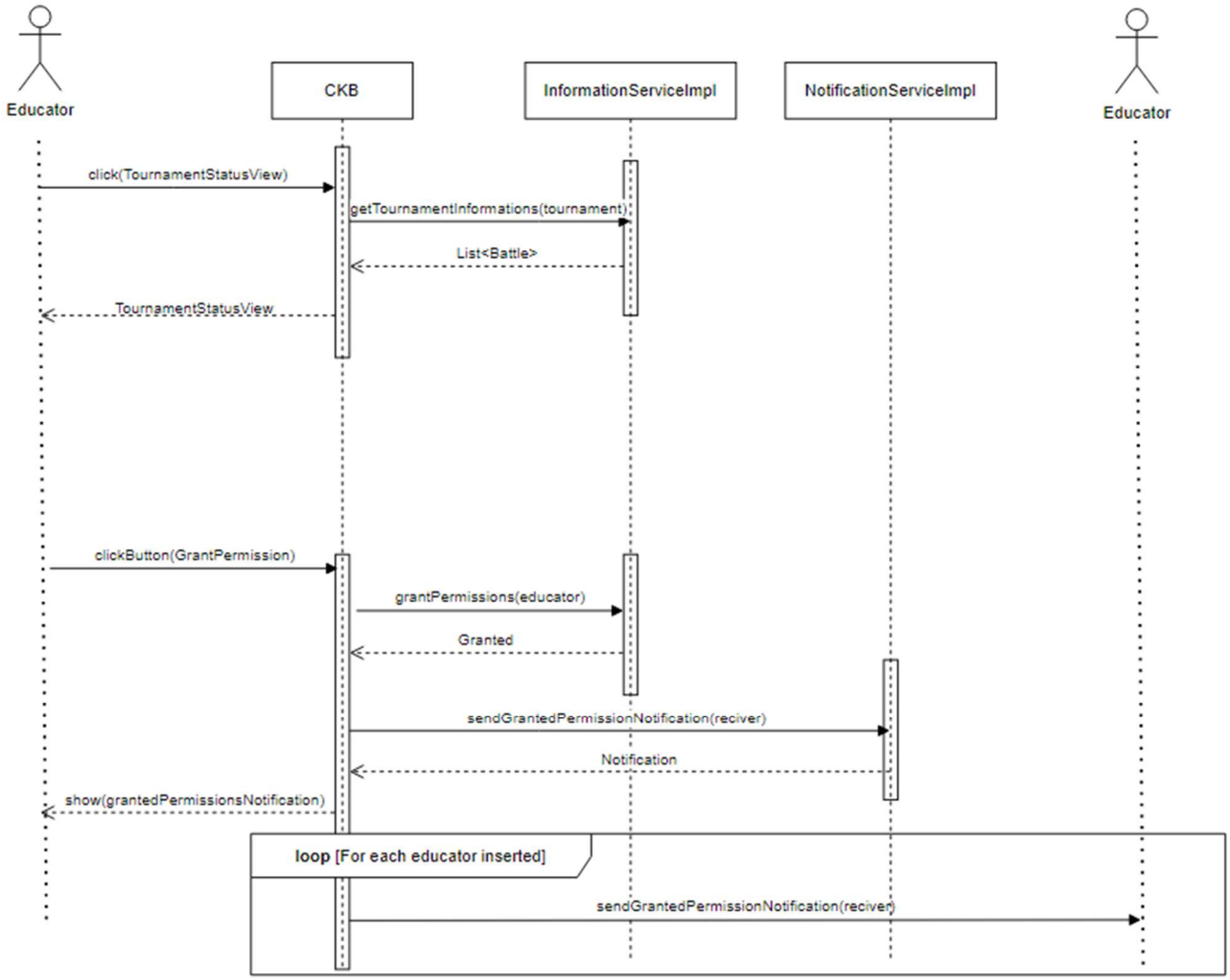

Figure 18: 13th sequence diagram

2.5 Component Interfaces

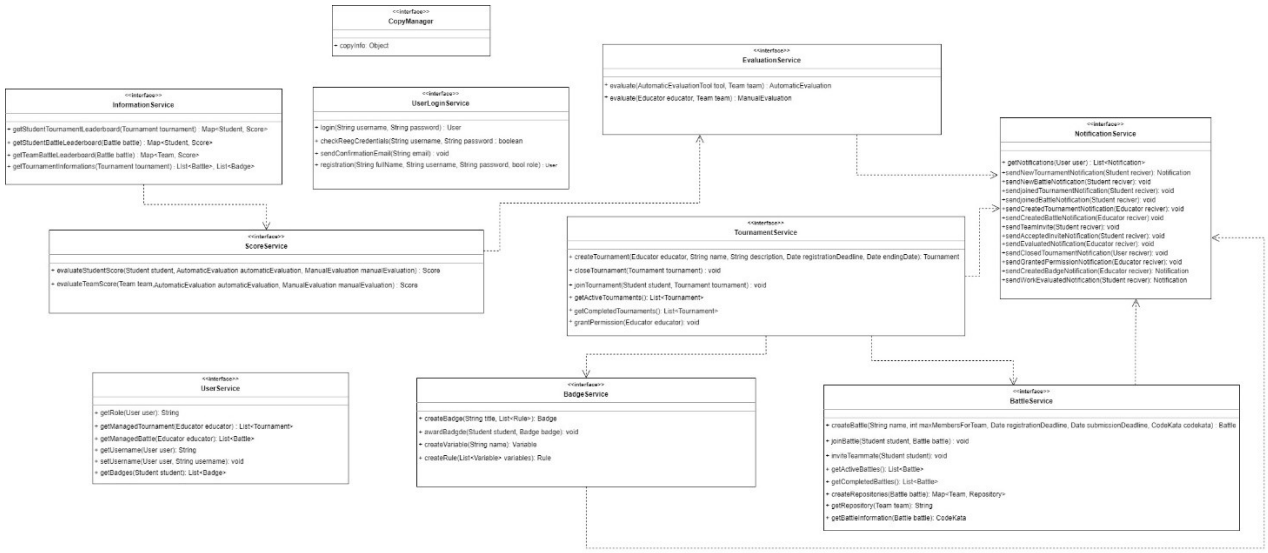

Figura 19: Interfaces' dependencies diagram

The following diagram represents the dependencies among the interfaces of the system. These dependencies are justified by the usage relations defined in the previous UML diagram and are reported here to make them clearer.

2.6 Selected architectural styles and patterns

Three tier architecture:

As already mentioned in the section 2.1, the type of architecture chosen to implement the CKB system is a three-tier architecture. This architecture requires the division of the system into three logical and physical tiers: the presentation tier, the application tier and the data tier. The first has the scope of delivering information to the users of the application and, at the same time, retrieving from them information that have to be processed to provide the functionalities of the system. The second correspond to the part of the system in which reside the main logic of the application; it has the scope of processing all the data retrieved both from the users and the data tier. The application tier is also responsible of managing (i.e. inserting, removing and updating) the data present in the data tier, communicating with it by means of API calls. Finally, the data tier is where all the necessary data are stored. In a three-tier architecture, it is not possible for the presentation tier to interact directly with the data tier, in fact, all communication has to go through the application tier. The main benefit of such type of architecture, which is also the reason why it has been chosen for the CKB system, is that the three tiers can be developed separately and run on different infrastructures. In such way, it will be also easier to update and extend the system whenever it is needed, modifying only the appropriate tier without any undesirable side effect on the other two tiers.

Component based development:

As it is highlighted in the previous sections, the development of the system is structured in components and sub-components. Therefore, it has been put into practice a component-based development, which brings a great number of benefits both in short and long terms. In a system with a great number of functionalities, a

component-based approach can be useful in order to develop a reusable software system, defining, implementing and composing loosely coupled independent components.

Relational DBMS:

A DBMS is a software system that allows the storage and management of data in a database. In the development of the CKB system it has been chosen a relational DBMS, which is a type of management system that stores data in a row-based table structure, in which the data of different tables are related. Furthermore, the DBMS implements some functions that deal with the security, accuracy, integrity and consistency of the data.

Model View Controller (MVC):

It is a software design pattern that requires the presence of three main logical components: the model, the view and the controller. The model has the scope of managing the data of the database and it corresponds, in the CKB system, to the classes defined in the class diagram (section 2.2.1). The View deals with the presentation of the information to the users and resides in the presentation tier. The Controller allows the communication between view and model and contains the services (defined in section 2.2.2) that provide all the needed functionalities. The main reason why it has been chosen this pattern is that it leads the software to be easily modifiable and its components widely reusable.

2.7 Other Design Decisions

2.7.1 Used Algotithms

The automated evaluation process is designed to objectively assess the performance of student teams based on functional aspects, timeliness, and the quality level of their source code. The following algorithm outlines the steps involved in the automated evaluation specifying how the aspects should be evaluated:

1. **Functional Aspects Evaluation (Test Cases):** To appraise the functional facets of the implemented solution, we initiate by categorizing test cases into two groups: "Passed" and "Not Passed." The algorithm then computes the Functional Aspects Score by determining the percentage of passed test cases relative to the total number of test cases. Functional Aspects Score = (Number of Passed Test Cases / Total Number of Test Cases) * 100
2. **Timeliness Evaluation:** Encouraging punctual submissions and penalizing delays, the algorithm calculates the time interval between the submission deadline and the last commit. It assigns a score proportional to the time remaining before the deadline. If the last commit occurs at the deadline, the score is null. The computation involves:
    - Total time window: TotalTime = Deadline - StartTime.
    - Time passed since the start: TimePassed = CurrentTime - StartTime.
    - Time left: TimeLeft = TotalTime - TimePassed. Timeliness Score = (TimeLeft / TotalTime) * 100
3. **Quality Level of Sources Evaluation:** Assessing source code quality in terms of security, reliability, and maintainability, the algorithm utilizes static analysis tools chosen by educators. These tools evaluate various aspects, leading to the calculation of the Quality Level Score on a scale from 0 to 100.

4. **Computation of the Overall Automated Score:** The final Automated Evaluation is determined integrating individual scores from functional aspects, timeliness, and source code quality,. The algorithm assigns weights to each score according to educator preferences and sums them up. Overall Automated Score = x1 * Functional Aspects Score + x2 * Timeliness Score + x3 * Quality Level. Score Weights (x1, x2, x3) are normalized to ensure a natural number outcome between 0 and 100, reflecting the relative importance of each aspect. The normalization condition is set as: x1 + x2 + x3 = 1.

3 USER INTERFACE DESIGN

In order to provide a clear representation, the links between the various user interfaces will be explicated after each of them. The software is projected for web devices, in fact in this document are presented the user interfaces related to the web application.

3.1 Login and Registration Interfaces for User

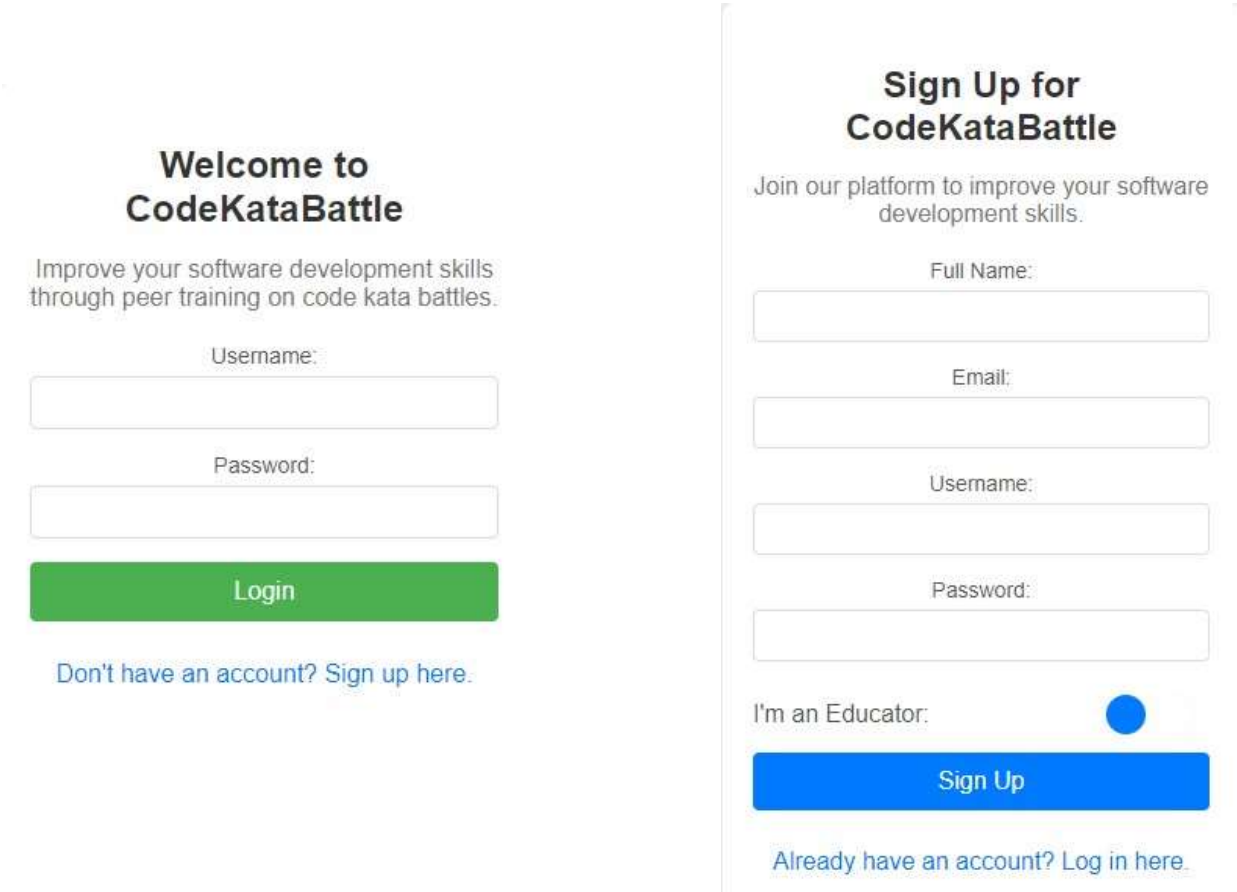

Figure 20: Log in interface (on the left) and Sign Up interface (on the right) for a User of CodeKata Battle

These interfaces show the very first screens of the application where a student or educator can access the application by entering their credentials. If a student or educator is not yet registered on the platform, they

can register by following the instructions on the web page. It is possible to register both as a new student and as an educator by filling in the fields with the required data, taking care to set the toggle button to “on” if you are an Educator.

3.2 Student Interfaces

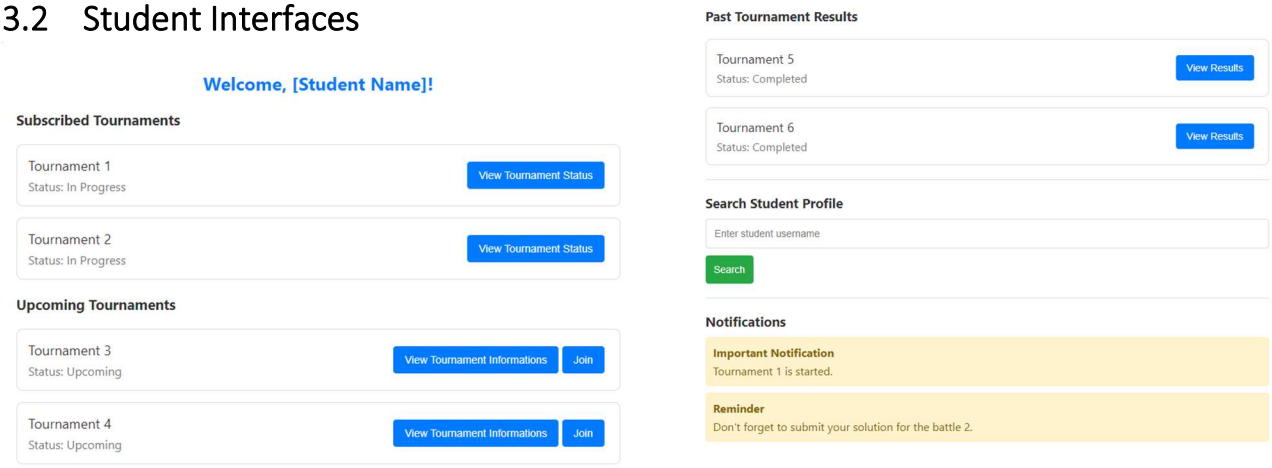

Figure 21: Home interface for a Student

The "CodeKataBattle" student dashboard provides an overview of activities and opportunities in the learning environment. The "Subscribed Tournaments" section allows students to view the status of the tournaments they are enrolled in. In the "Upcoming Tournaments" section, students can find details about upcoming events and they can quickly participate by clicking the "Join" button. Additionally, the "Past Tournament Results" section gives access to results from previous tournaments, allowing students to review their past performances. The "Search Student Profile" section enables students to connect with each other students by entering a colleague's username, accessing their profile. Lastly, the "Notifications" section serves as a central hub for important communications. Students receive notifications regarding tournament status, final ranking updates, and other alerts.

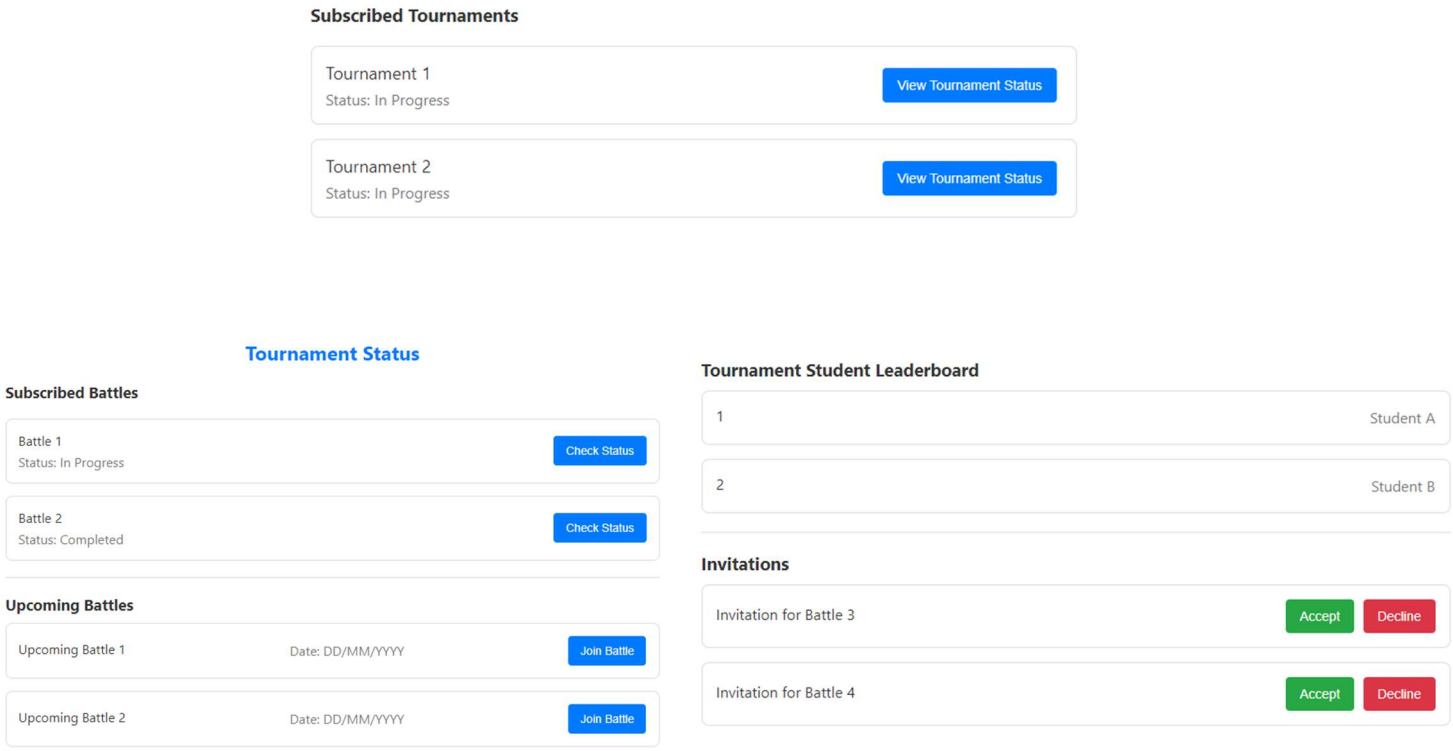

Figure 22: Tournament Status interface for a Student

From the home interface, clicking on the "View Tournament Status" button, the student is directed to the Tournament Status interface. Here, the student can observe his subscribed battles and review their status on a separate web page. They also have the option to subscribe to an upcoming battle by clicking the "Join Battle" button. The interface also allows them to view the leaderboard, indicating their ranking within the tournament. At the bottom of the page, the student can find invitations from specific teams for a battle. Through a button, they can decide whether to accept or decline the invitation and choose whether to join the battle.

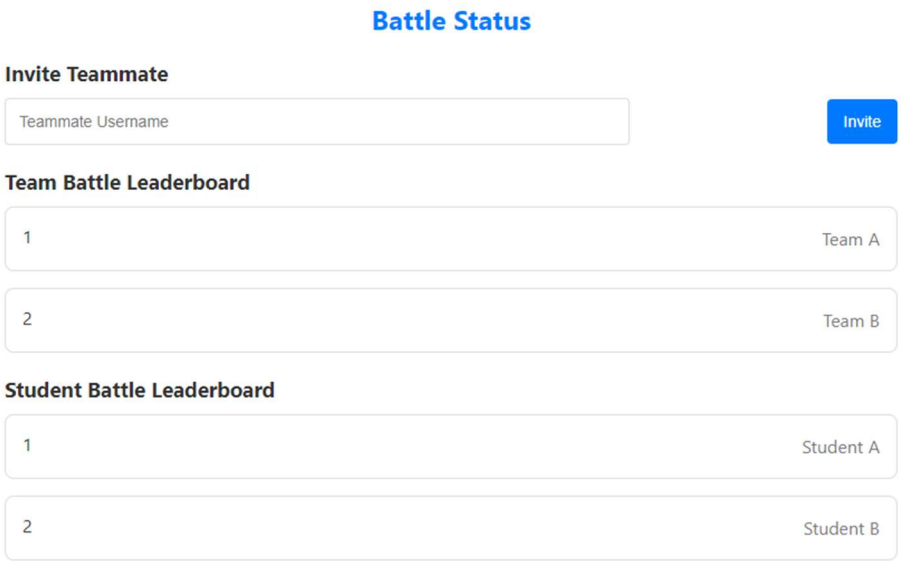

Figure 23: Battle Status interface for a Student

From the Tournament Status interface, by clicking on the 'Check Status' button, the student goes to the 'Battle Status' interface. Here he can invite other students to join his team, view the ranking of a team he is part of, and his ranking for individual battles. The student who clicks on 'join battle' button will log into the battle and be redirected to the ‘Battle Status’ interface for that battle.

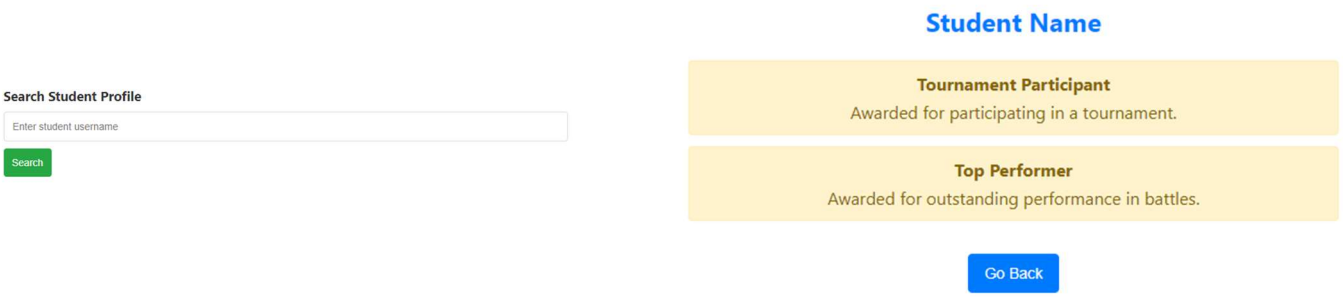

Figure 24: Search Student Profile section within the home page (on the left) and Student Profile page (on the right)

By entering a student's name on the home interface and subsequently clicking the “Search” button, each student can access the profiles of other students through another interface known as "Student Profile page."

3.2 Educator Interfaces

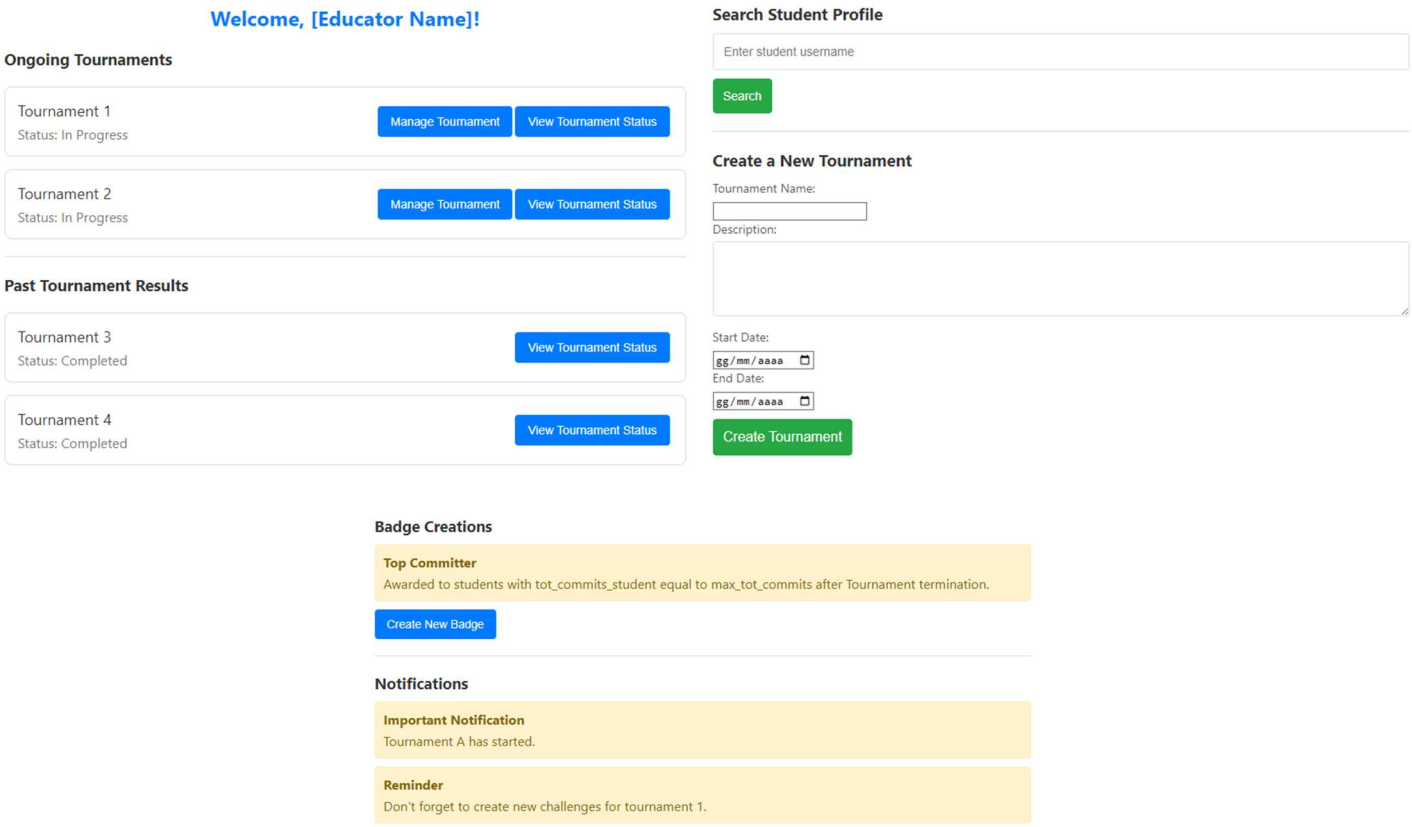

Figure 25: Home interface for an Educator

The "CodeKataBattle" educator dashboard provides a comprehensive overview of activities and responsibilities related to tournament management and student interaction. In the "Created Tournaments" section, educators can view a list of tournaments they have created, displaying each tournament's name and current status. The "Manage Tournament" button allows educators to access the management interface for tournament activities. The "Past Tournament Results" section grants educators access to past tournaments that require manual evaluation for conclusion. The "Evaluate Results" button enables educators to assess participants' performance. In the "Search Student Profile" section, educators can utilize a student search function by entering their username; searching a student in this section of the page will allow the educator to view his profile, visualizing students' achieved badges. The "Create a New Tournament" section features an interactive module enabling educators to initiate new coding competitions. The form prompts educators to input details such as tournament name, description, starting date, and ending date. Upon completion, educators can submit the form to create a new coding challenge. In the "Badge Creation" section, badges created by the educator are showcased, each with a title and a brief description. The "Create new badge" button empower educators to create a badge for a tournament. Criteria for each badge are provided, specifying that it is awarded to students with a certain level of effort after the conclusion of a specific tournament.

Finally, the "Notifications" section serves as a communication hub, delivering instant notifications to educators regarding events such as the need for grading student work, or other essential information.

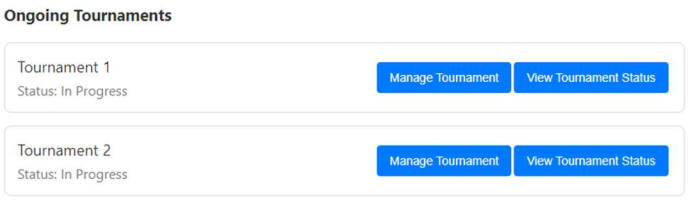

Figure 26: Ongoing Tournaments section within the homepage

From the home interface, clicking on the "View Tournament Status" button, the educator is directed to the “Tournament Status” interface of the selected tournament. If the educator clicks on “Manage Tournament” button he will be redirected to the “Tournament Management” interface.

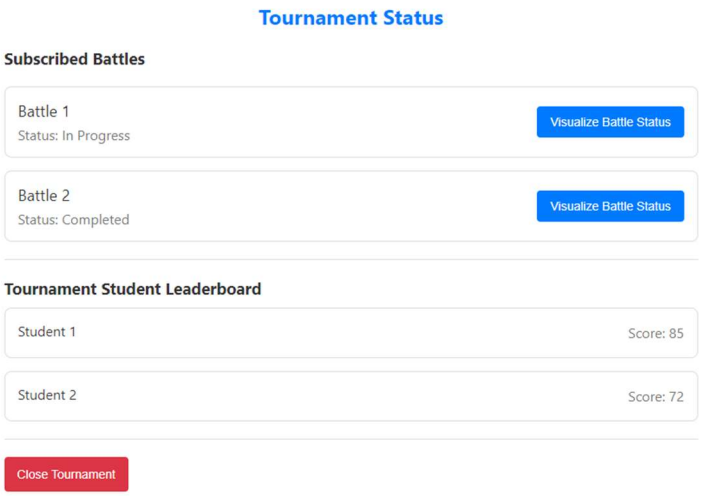

Figure 27: Tournament Status interface for an Educator

If the educator clicks on “View Tournament Status” button in the home interface, he will be redirected to the “Tournament Status” interface where he can view the status of individual battles on “Battle Status” interface and view individual student rankings for that tournament.

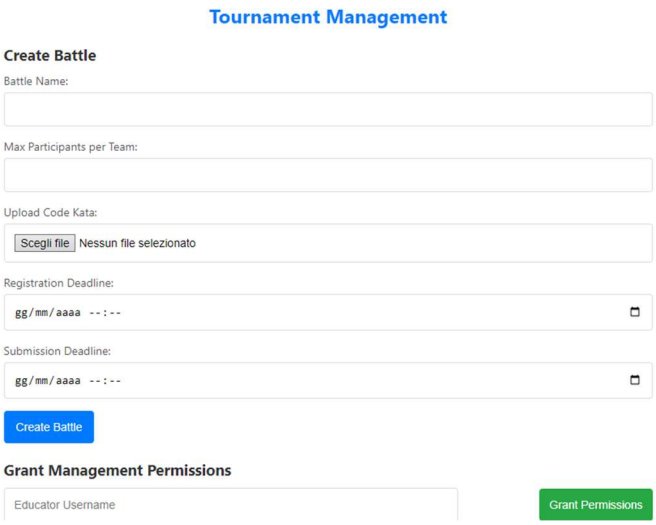

Figure 28: Tournament Managment interface for an Educator

If the educator clicks on “Manage Tournament” button in the home interface, he will be redirected to the “Tournament Management” interface where he can create a battle by providing a name, selecting the maximum number of participants, choosing the code for administering to the participants, setting expiration dates for registration and code submission. Additionally, he can specify permissions to allow other educators to manage the tournament.

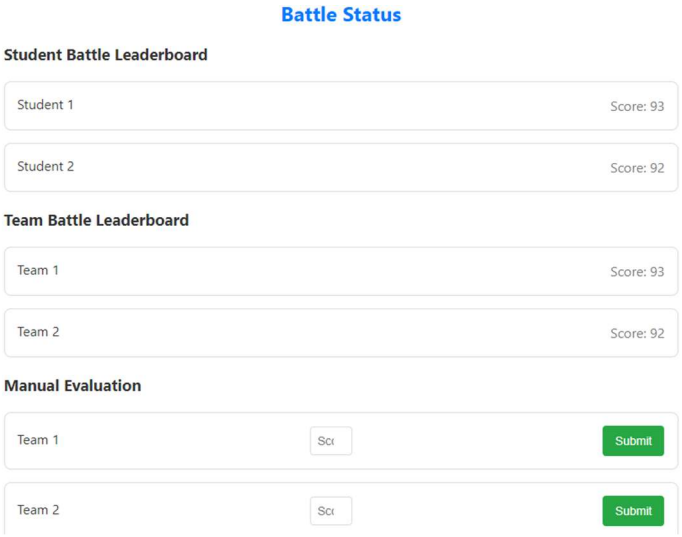

Figure 29: Battle Status interface for an Educator

If the educator clicks on “View Tournament Status” button in the home interface, he will be redirected to the “Tournament Status” interface where he can view the status of individual battles on “Battle Status” interface. In this interface he can view the status of the individual battles of each student and each team and assign a manual evaluation to the work carried out by the participants.

4 REQUIREMENTS TRACEABILITY

In this section, it is reported the table of requirements already discussed in the RASD document, with an additional column containing the software components (previously defined in this document) to which each requirement is mapped.

Requirement	Description	Mapped components
R1	The system shall allow an unregistered user to create an account.	User Client, UserLoginServiceImpl, Internal Database
R2	The system shall check if the personal data inserted by an unregistered user in the registration page are conflicting with respect to other existing accounts.	User Client, UserLoginServiceImpl
R3	The system shall send an email containing a link to the unregistered user once he has inserted his data.	UserLoginServiceImpl
R4	The system shall show a confirmation page to the user once he clicks on the link contained in the email.	User Client, UserLoginServiceImpl, Internal Database
R5	The system shall redirect the user, who is now registered, to the homepage.	User Client
R6	The system shall allow a registered user to login into the platform.	User Client, UserLoginServiceImpl
R7	The system shall check if username and password inserted by a user for login are valid.	UserLoginServiceImpl, Internal Database
R8	The system shall allow students to view the list of available tournaments.	Student Client, Internal Database
R9	The system shall allow students to join a tournament.	Student Client, TournamentServiceImpl, Internal Database
R10	The system shall allow students to select a tournament to view its status.	Student Client, InformationServiceImpl, Internal Database
R11	The system shall allow students to visualize information about upcoming tournaments.	Student Client, InformationServiceImpl, Internal Database
R12	The system shall allow students to view past tournament results.	Student Client, InformationServiceImpl, Internal Database
R13	The system shall allow students to participate in a coding battle if they are enrolled in the tournament of the battle.	Student Client, BattleServiceImpl, Internal Database
R14	The system shall allow educators to create a new coding battle if they have tournament management permission.	Educator Client, BattleServiceImpl, Internal Database
R15	The system shall allow educators to insert the code kata for a coding battle he's creating.	Educator Client, BattleServiceImpl, Internal Database
R16	The system shall allow educators to specify battle parameters such us minimum and	Educator Client, BattleServiceImpl, Internal Database

	maximum number of team member for the battle.	
R17	The system shall allow educators to specify the submission deadline of a coding battle he's creating.	Educator Client, BattleServiceImpl, Internal Database
R18	The system shall allow educators to specify the duration of a coding battle he's creating.	Educator Client, BattleServiceImpl, Internal Database
R19	The system shall allow educators to specify the evaluation criteria of a coding battle he's creating.	Educator Client, BattleServiceImpl, Internal Database
R20	The system shall allow users to monitor students' scores in real-time during a tournament.	User Client, InformationServiceImpl, Internal Database
R21	The system shall allow students to monitor their scores in real-time during a coding battle when they are enrolled in the battle.	Student Client, InformationServiceImpl, Internal Database
R22	The system shall allow students to monitor teams' scores in real-time during a coding battle, when they are enrolled in the battle.	Student Client, InformationServiceImpl, Internal Database
R23	The system shall allow educators to monitor students' scores in real-time during a coding battle, if they've created the battle.	Educator Client, InformationServiceImpl, Internal Database
R24	The system shall allow educators to monitor teams' scores in real-time during a coding battle, if they've created the battle.	Educator Client, InformationServiceImpl, Internal Database
R25	The system shall automatically update scores when students submit their solutions.	Evaluation Service, Internal Database
R26	The system shall send timely notifications to students when a new tournament is created by an educator.	NotificationServiceImpl, Internal Database
R27	The system shall send timely notifications to users when tournament results are available.	NotificationServiceImpl, Internal Database
R28	The system shall send timely notifications to users when battle results are available.	NotificationServiceImpl, Internal Database
R29	The system shall send timely notifications to students when there is a new available battle for a subscribed tournament.	NotificationServiceImpl, Internal Database
R30	The system shall create a GitHub repository for each team when a battle starts.	BattleServiceImpl, Internal Database
R31	The system shall send a notification to all the students with the link of the created GitHub repository.	NotificationServiceImpl, Internal Database
R32	The system shall allow students to form teams through invitations to collaborate on participation in battles.	Student Client, BattleServiceImpl, NotificationServiceImpl
R33	The system shall allow users to see the profile of every student subscribed to the platform.	User Client, InformationServiceImpl, Internal Database
R34	The system shall allow educators to create a new tournament.	Educator Client, TournamentServiceImpl, Internal Database

R35	The system shall allow educators to specify the duration of a tournament he's creating.	Educator Client, TournamentServiceImpl, Internal Database
R36	The system shall allow educators to specify the name of a tournament he's creating.	Educator Client, TournamentServiceImpl, Internal Database
R37	The system shall allow educators to make a description of a tournament he's creating.	Educator Client, TournamentServiceImpl, Internal Database
R38	The system shall allow educators to establish the starting and ending date of a tournament he's creating.	Educator Client, TournamentServiceImpl, Internal Database
R39	The system shall provide automatic evaluation of battle scores.	EvaluationServiceImpl, Internal Database
R40	The system shall allow to educators to manually evaluate students work relative to an ended battle that requires the consolidation phase.	Educator Client, EvaluationServiceImpl, Internal Database
R41	The system shall allow educators to view final results of a battle, including student and team battle leaderboard.	Educator Client, InformationServiceImpl, Internal Database
R42	The system shall allow educators to view final results of a tournament.	Educator Client, InformationServiceImpl, Internal Database
R43	The system shall allow students to view final results of a battle, including student and team battle leaderboard.	Student Client, InformationServiceImpl, Internal Database
R44	The system shall allow students to view final results of a tournament.	Student Client, InformationServiceImpl, Internal Database
R45	The system shall allow educators to define gamification rules for badge assignment during the tournament creation phase.	Educator Client, BadgeServiceImpl, Internal Database
R46	The system shall allow educators to create badges.	Educator Client, BadgeServiceImpl, Internal Database
R47	The system shall allow educators to delegate tournament management to other educators.	Educator Client, TournamentServiceImpl, Internal Database
R48	The system shall allow students to view other students' profiles.	Student Client, UserServiceImpl, InformationServiceImpl, Internal Database
R49	The system shall allow educators to view profiles of all the students subscribed to the platform.	Educator Client, UserServiceImpl, InformationServiceImpl, Internal Database
R50	The system shall display badges earned by a student in the student profile.	User Client, UserServiceImpl, Internal Database

Table 4: Table requirements' mapping to the software components

5 IMPLEMENTATION, INTEGRATION AND TEST PLAN

5.1 Implementation plan

The design and implementation of the various components of the CodeKataBattle (CKB) platform follow a detailed plan aimed at maximizing the efficiency and consistency of the system as a whole. The adoption of a modular architecture allows for the individual development of components, with the definition of specific testing procedures for each implementation phase.

The order of implementation has been carefully selected based on the dependencies between the various components and subcomponents, following a bottom-up approach. The development sequence is as follows:

Internal Database and Derived Data Model: The fundamental starting point, essential for the construction of the entire system. The internal database, together with its derived data model, constitutes the beating heart on which the other components are built.

Implementations of the Copy Manager Interface: This step is crucial as the Copy Manager is responsible for managing information from external services, acting as a bridge between the internal system and external resources.

Notification Services: Notification Service follows, as it provides an essential functionality and does not depend on other services. its implementation is a prerequisite for continued development.

Implementation of Tournament, Battle and Badge Services: they provide the core functionalities.

User Service: Given its importance in the CKB ecosystem the User Service is implemented next.

Other Services: With the main components now integrated, it is possible to proceed with the development of the other services, each contributing uniquely to the overall user experience.

During the application server development process, the simultaneous implementation of the client application modules, known as the "Student Client Module" and the "Educator Client Module," is also crucial. These modules enable incremental testing, ensuring proper integration between the components.

5.2 Integration and Testing

Integration and testing of the various components of the CodeKataBattle (CKB) platform are performed incrementally, that is, as the individual components are released. This approach aims to ensure a gradual and robust construction of the system, identifying and resolving potential problems in an early stage.

Components are integrated and tested in order of implementation, following a "bottom-up" strategy. This means that the first component to be integrated is the Internal Database and Derived Data Model, followed by the Implementations of the Copy Manager Interface, the Information and Notification Services, the User Service, and all other services.

Before the integration of components, developers perform unit tests on each of them. Unit tests are basic tests that verify the correct functioning of a single unit of code. For these tests, the white-box testing technique is used, which requires the tester to have a good understanding of the code to be tested.

Unit tests are performed randomly, with the generation of numerous test cases. This allows for observing the behavior of the component on a larger scale.

Integration tests, on the other hand, verify the correct functioning of multiple components that interact with each other. In this case, the black-box testing technique is used, which does not require the tester to have an understanding of the internal code of the components.

5.2.1 Sequence of components integration

The first things to be integrated and tested together will be the model of data associated with the internal database and the implementations of the Copy Manager service, which is needed to retrieve the information from the external services and store them into the internal database. Once integrated, these components will be tested together with a black-box approach and in a random way, inserting input data from the external services and verifying that such data are inserted in the database (trough automatic evaluation).

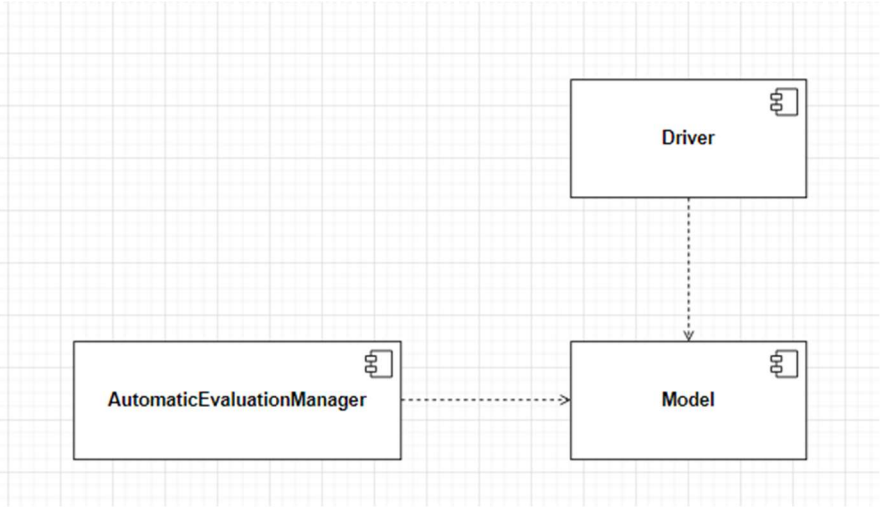

Figure 30: Copy Manager Service Implementation

Then, the Notification service is integrated with the model of data and its simple functionality regarding the creation of a message and its association with the receiver is tested. These tests are executed randomly, because at this level it is not necessary to consider a systematic approach, which is usually needed for searching possible bugs derived from the integration, due to the limited number of components involved.

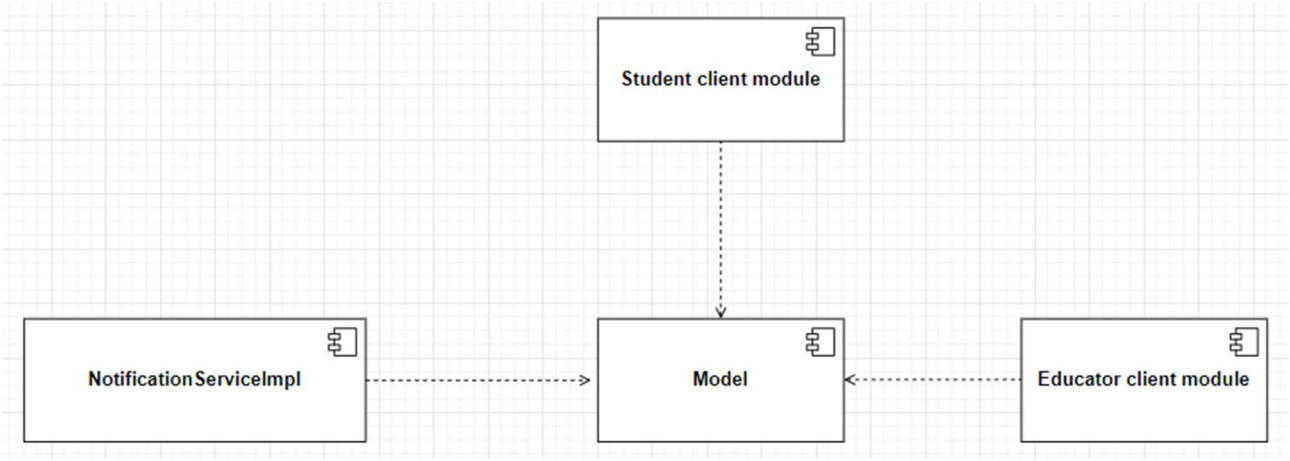

Figure 31: Notification Service Implementation

The Tournament, Badge and Battle Services must be integrated with the notification service. In such way it will be possible to test together the services with the model, testing their functionalities, which are: creating a tournament and its relative battles and badges (also variables and rules), make the manual evaluation for a battle, visualizing statuses, close a tournament, join a battle, invite a teammate for a battle.

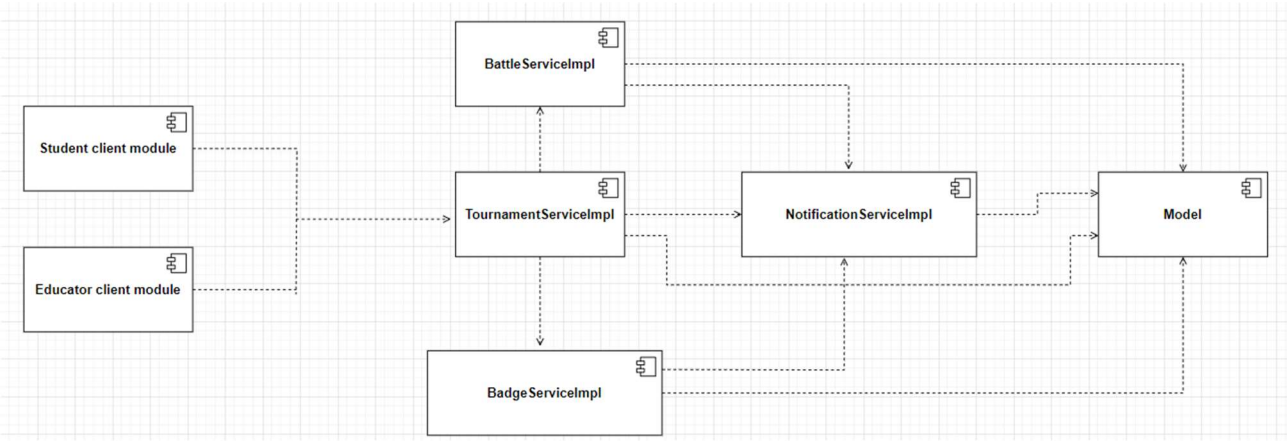

Figure 32: Tournament, Battle and Badge Service Implementation

In order to integrate most of the other services to the system, the functionalities of the User service is needed. For this reason, it will be the next component to be integrated. It can be noticed that it will interact only with the model of data. This service can be tested separately even if there are present already other services integrated to the system (that's why in the figure there are shown only the service and the model). To test the integration of this component will be executed a systematic testing, in order to see if the interaction between the service leads to any malfunction.

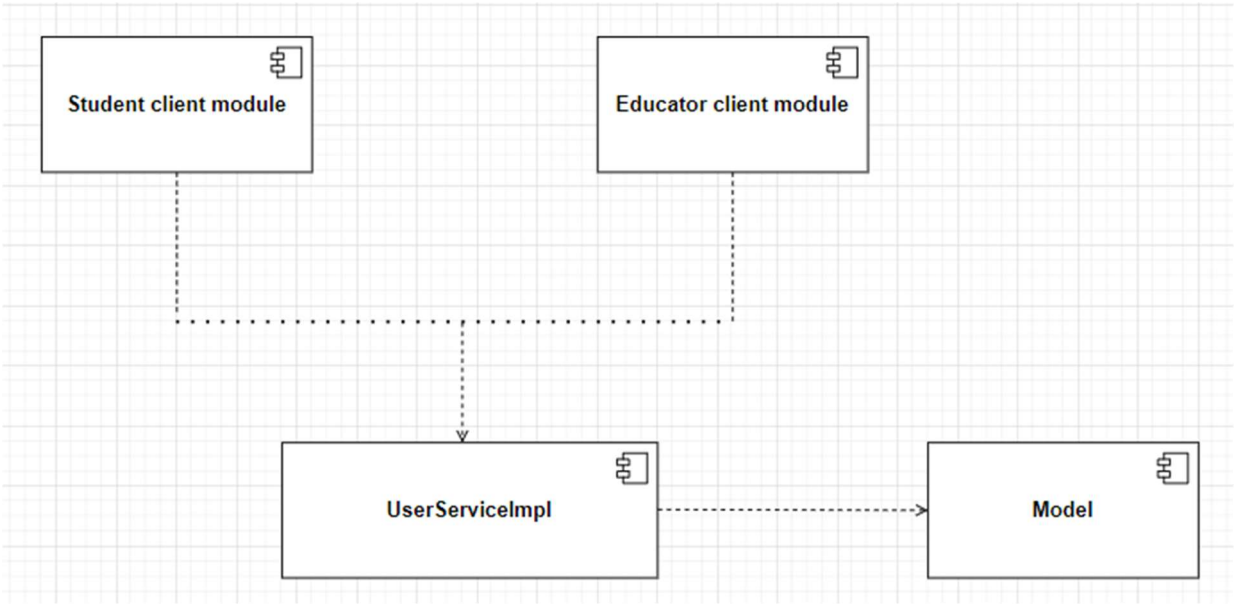

Figure 33: User Service Implementation

The last service that will interact with the student and educator clients is the Login service, which again, can be tested only with the model because it doesn't use other services in order to complete its functions. The

tests regarding this service will be of two types, first it will be execute random testing for the registration and login of multiple educators, in order to see if all works as desired; then, it will be executed systematic testing to see if inserting conflicting data or invalid credentials returns what was expected.

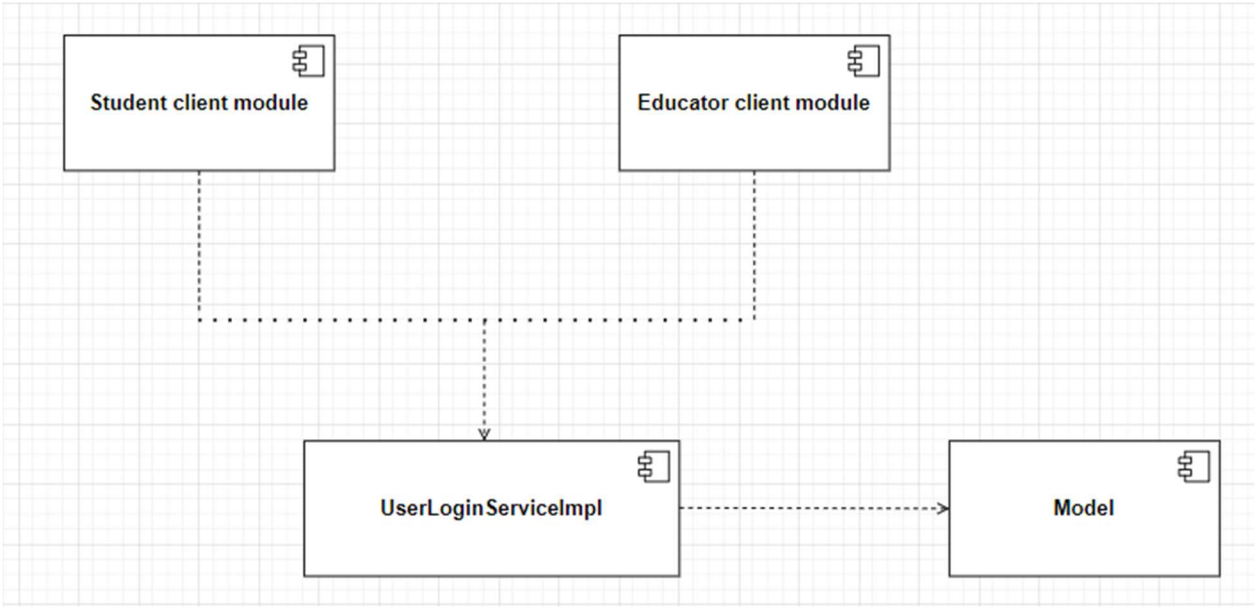

Figure 34: User Login Service Implementation

Each of the above-described tests will require the implementation of specific modules in the user client application; once the integration of those services in the application server is completed, it will be possible to put together also all the modules of the user interfaces relevant to the educator client and student client and test all the functionalities that the system must provide to a user. This test will be executed with a black-box approach, since it must verify that all the requirements are satisfied, and that the application is ready for the end-user (who knows nothing about how the code of the software components is written) to use it. The services that are still missing in the integration of the system are the ones related to the evaluation and information system. The first to be integrated will be the Information service which, to work properly, has to interact with the Score service, that needs the implementation of the Evaluation Service, witch needs to interact with the Notification Service. For this reason, they will be integrated and tested together with the model.

The implementation of the Information service will lead to test of its functionalities using a black-box approach, to see if its methods are able to retrieve data from the database and update data already present in it. In this way, it will be possible to test the visualization of leaderboards and other tournament and battles information.

The tests will deal with the verification of the service's functionalities, i.e. the generation of the various tournaments and battles and the user's interaction with them. These tests will require a systematic strategy because it is necessary to verify if the interaction between educators and students provides the expected results.

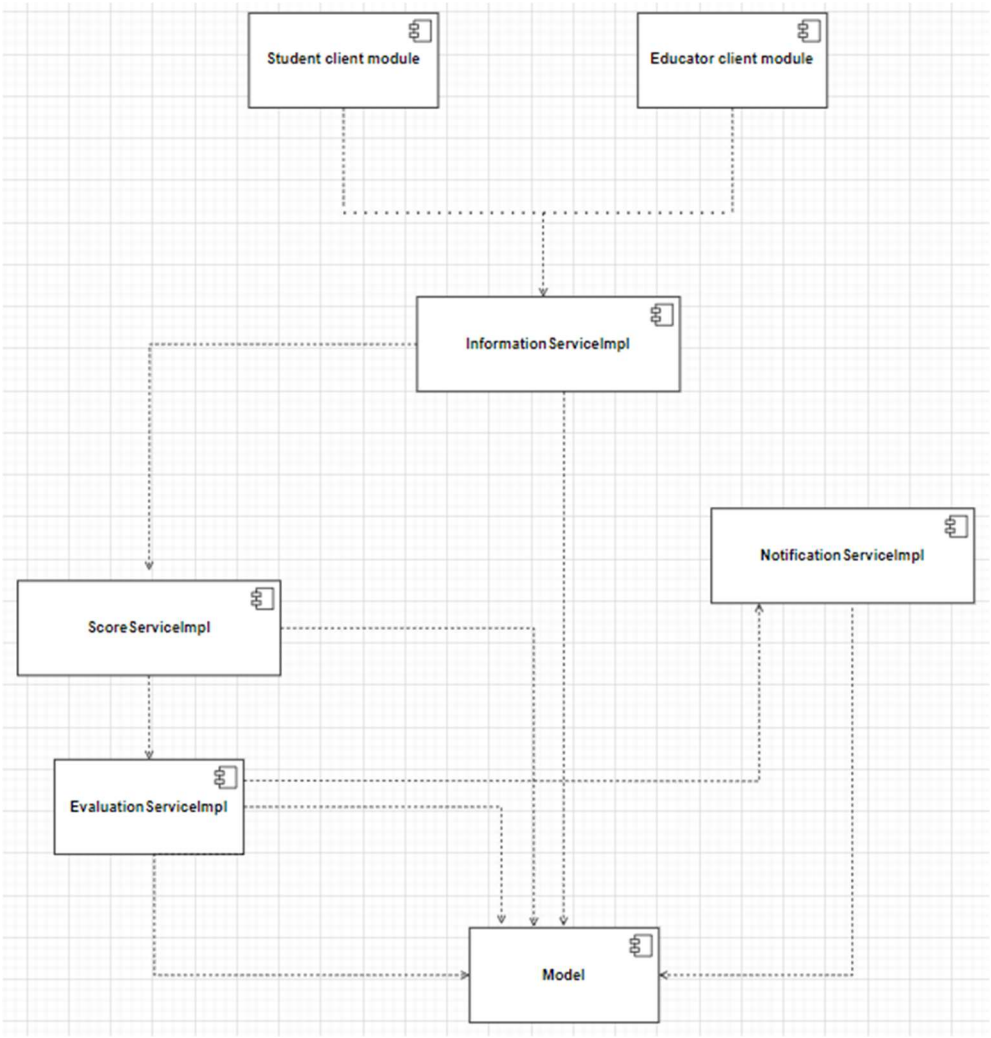

Figure 35: Information, Score and Evaluation Service Implementation

At this point, it will be possible to integrate together all the modules of the student and educator client applications and test their functionalities. The tests will follow a black-box approach and will verify that there are all the features that the system has to present to a student and an educator user, accordingly to the requirements.

6 EFFORT SPENT

Topic	Hours
Introduction	2:00h
Overview on the architecture	3:00h
Class Diagram	1:30h
Component Diagram	8:00h
Deployment view	3:00h
Runtime view	1:30h
Component Interfaces	2:30h
Architectural styles and patterns	3:30h
Other design decisions	1:00h
User interface design	2:00h
Requirements traceability	1:30h
Implementation plan	1:30h
Integration and testing plan	5:00h
Document organization	4:00h
Total effort spent	40:00h

Table 5: Effort spent by student 1

Topic	Hours
Introduction	2:00h
Overview on the architecture	3:00h
Class Diagram	1:30h
Component Diagram	8:00h
Deployment view	3:00h
Runtime view	1:30h
Component Interfaces	2:30h
Architectural styles and patterns	3:30h
Other design decisions	1:00h
User interface design	2:00h
Requirements traceability	1:30h
Implementation plan	1:30h
Integration and testing plan	5:00h
Document organization	4:00h
Total effort spent	40:00h

Table 6: Effort spent by student 2

Topic	Hours
Introduction	2:00h
Overview on the architecture	3:00h
Class Diagram	1:30h
Component Diagram	8:00h
Deployment view	3:00h
Runtime view	1:30h
Component Interfaces	2:30h
Architectural styles and patterns	3:30h
Other design decisions	1:00h
User interface design	2:00h
Requirements traceability	1:30h
Implementation plan	1:30h
Integration and testing plan	5:00h
Document organization	4:00h
Total effort spent	40:00h

Table 7: Effort spent by student 3

7 REFERENCES

- All the diagrams have been made with Draw.io
- All the user interfaces have been made with the VisualStudioCode using HTML.