POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Software Engineering 2 Design Document

Author(s): **Federico Zanca**

Federico Costantini

Michele Zhenghao Zhuge

Academic Year: 2023-2024

Contents

1 | Introduction

1.1. Purpose

This document provides a comprehensive overview of the CodeKataBattle (`CKB`) platform, elucidating the strategies to meet the project requirements delineated in the Requirements Analysis and Specification Document (RASD). Tailored for developers and testers, this document serves as a guide, offering a functional delineation of the system's components, their interactions, interfaces, and the planned implementation strategies. For developers, the document acts as a guide, providing a closer look at the details of CKB's architecture and design choices. Testers will find valuable information on the expected functionalities and the interaction patterns within the system. Furthermore, this document serves as a bridge between the abstraction of the concepts in the RASD and the concrete implementation detailed here. It addresses each requirement outlined in the RASD, explaining how the various system components, described herein, collectively fulfill these requirements. This document aims to enhance comprehension of how the `CKB` platform meets the specified project objectives by detailing its internal workings and external interfaces.

1.2. Scope

The Design Document for the CodeKataBattle (CKB) platform provides a comprehensive overview of the system's architecture, design, and functionality. It details the interfaces of various components such as the UserManager and TournamentManager, which are integral to the operation of the platform.
The document also elaborates on the structural design and pattern choices, including the use of a firewall and a reverse proxy for security, and the deployment of two databases (one SQL and one NoSQL) for efficient data management.
The interface design is meticulously outlined to ensure a user-friendly experience, facilitating seamless interaction between users and the platform.
The document also includes a section on requirement traceability, ensuring that every requirement defined in the RASD is accounted for in the design and implementation phases.

This helps in maintaining the coherence and completeness of the project.
The implementation, integration, and testing sections provide a roadmap for the development process, outlining the strategies for integrating the various components and the approaches for testing the system's functionality and performance.
As described in the RASD, CKB is a platform designed to enhance software development skills through competitive coding challenges. It allows users to participate in code kata battles, where they compete against each other in tournaments by solving programming exercises. The platform collects the solutions, allows educators to create battles and evaluate the solutions, and updates the scores and rankings of the teams based on the evaluation.
Incorporating gamification elements like badges, CKB motivates users and enhances their learning experience. Educators can create badges with custom rules and assign them to users based on their performance. The platform also supports the creation of tournaments, where multiple battles are grouped together, and users compete for the highest cumulative score.
In essence, CKB is a comprehensive platform for competitive coding challenges, providing a dynamic, engaging, and educational environment for users to improve their coding skills. The Design Document serves as a blueprint for realizing this vision, detailing the system's design and implementation strategies to ensure the successful execution of the project.

1.3. Definition, Acronyms, Abbreviations

Acronyms	Definition
CKB	Code Kata Battle
RASD	Requirements Analysis and Specification Document
DD	Design Document
WPX	World Phenomena X
SPX	Shared Phenomena X
GX	Goal Number X
DAX	Domain Assumptions X
UCX	Use Case X
DBMS	Database Management System
UI	User Interface
API	Application Programming Interface
OS	Operating System
HTTP	Hypertext Transfer Protocol
UX	User Experience

Table 1.1: Acronyms used in the document.

1.4. Revision History

This is the first version of this document.

1.5. Reference Documents

- The specification document Assignment RDD AY 2023-2024.pdf
- RASD document

1.6. Document Structure

The structure of this Design Document is organized into seven distinct sections, each contributing to a comprehensive understanding of the CodeKataBattle (CKB) platform.

The opening section (Introduction) serves as a preamble, introducing the purposes and objectives of the document. It also includes a compilation of abbreviations and definitions used to enhance the readability of the document.

The second section (Architectural Design) delves into the chosen architectural design for CKB. It describes the identified system components, their interrelations, communication interfaces, behavioral aspects within the system, and the utilization of architectural styles.

The third section (User Interface Design) focuses on user interfaces, presenting and elucidating the mockups designed to enhance the user experience on the CKB platform.

Section four (Requirements Traceability) systematically demonstrates how the CKB system aligns with the defined requirements. It begins by establishing a mapping between identified components and the functional requirements outlined in the third section of the RASD. Subsequently, a detailed account of the fulfillment of performance requirements and system attributes is provided.

The fifth section (Implementation, Integration, and Test Plan) outlines the plan for implementing, integrating, and testing various subcomponents of the CKB system. It details the sequence in which subcomponents will be implemented and integrated, setting the groundwork for the practical realization of the platform.

Section six (Effort Spent) provides transparency regarding the effort invested by each group member in the creation of this document.

The concluding section serves as a repository of bibliography references and additional resources utilized in the creation of this Design Document.

2 | Architectural Design

2.1. Overview

This section provides a comprehensive overview of the design and architecture of CodeKataBattle (CKB). The component view subsection illustrates the logical structure of the system, including the components, their relationships, and responsibilities. The runtime view, on the other hand, depicts the dynamic behavior of the system, including the scenarios, interactions, and state changes of the components. Here, we also delve into the specifics of component interfaces, detailing the methods, parameters, and structure of each system component, such as the user manager, battle manager, tournament manager, badge manager, notification manager, among others. The document also sheds light on the selected architectural styles and patterns, explaining the rationale behind the design choices, including the use of 3-tier architecture, WebSocket with Pub/Sub pattern, worker pooling, microservices, NoSQL with SQL databases, firewall, reverse proxy, and more. The integration, implementation, and testing section outlines the development, combination, and verification processes of the system components, along with the tools and technologies employed. The deployment view provides a glimpse into the physical configuration of the system, showcasing the hardware and software components, their distribution, and interconnections. This comprehensive section, therefore, serves as a detailed guide to the intricate workings of the CodeKataBattle platform.

2.2. Component View

The UML component diagram shows all the identified components in the `CKB` system. It also describes the relations between the modules, representing the verse of the communication flow and the actors participating. It is divided into a WebApplication, which is the front-end for users, and a back-end, composed of the `CKB Core Business Logic` and the `CKB Notification System`. External entities to which the system holds a dependency are also shown.

2.2.1. Component diagram

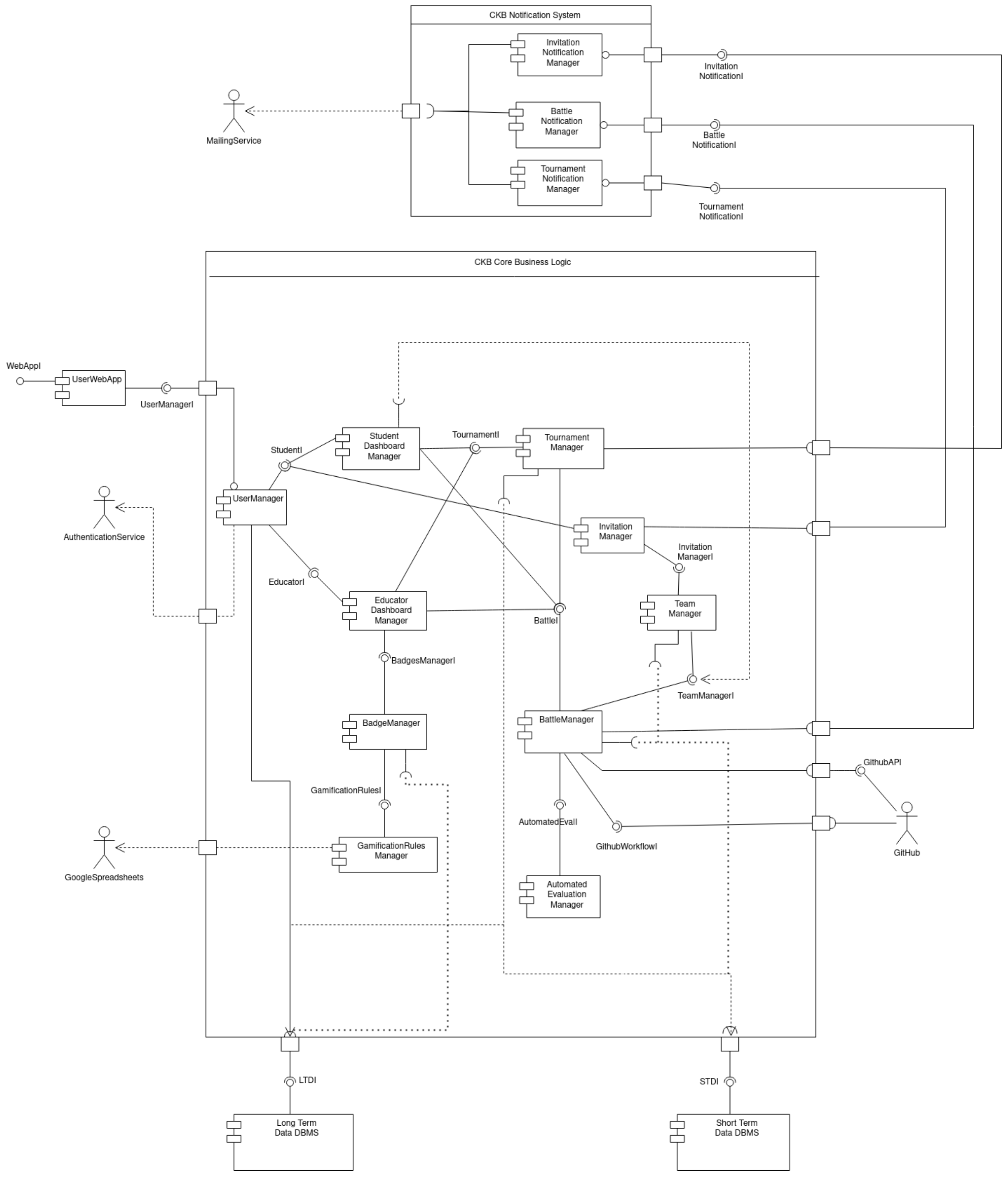


Figure 2.1: Component diagram of the CKB platform.

2.2.2. Components description

The components are:

- **User WebApp.** `User WebApp` is the front-end for users.
- **UserManager.** `UserManager` component offers, through the interface `UserManagerI`, the basic function for handling users:
  - **Register**
  - **Login**

  It also provides user information to the `EducatorDashboardManager` component and the `StudentDashboardManager` component depending on the role of the logged user.
- **Educator Dashboard Manager.** `Educator Dashboard Manager` handles the main functionalities accessible by an Educator. It allows creating and managing both CodeKataBattles and Tournaments, defining new badges and rules and performing all the operations that an Educator should be able to perform according to the RASD.
- **Badge Manager.** `Badge Manager` is used to manage badges, allowing to perform operations such as assigning a badge to a Student, creating new badges and defining new rules and variables to obtain them.
- **Gamification Rules Manager.** `Gamification Rules Manager` is used to manage rules and variables used to determine winners of badges. This component relies on an external actor, Google Spreadsheet, to allow defining new rules in a simple but effective way (Spreadsheet formulas).
- **Student Dashboard Manager.** `Student Dashboard Manager` handles the main functionalities accessible by a Student. It allows joining and participating in CodeKataBattles and Tournaments, checking the status of the ongoing ones and the results of the past ones.
- **Battle Manager.** `Battle Manager` is used to manage CodeKataBattles, allowing to perform operations such as creating a new CodeKataBattle, joining an existing one and checking the status of the ongoing ones through its interface `BattleI`.
- **Tournament Manager.** `Tournament Manager` is used to manage Tournaments, allowing to perform operations such as creating a new Tournament, joining an existing

one and checking the status of the ongoing ones through its interface `TournamentI`. Educators can also add other Educators as admin to a tournament they created.

- **Automated Evaluation Manager.** `Automated Evaluation Manager` is used to manage the automated evaluation of CodeKataBattles assigning scores to teams.
- **Team Manager.** `Team Manager` is used to manage teams, allowing to perform operations such as creating a new team, joining an existing one and inviting a student to join a team.
- **Invitation Manager.** `Invitation Manager` is used to manage team invitations, it is responsible for keeping track of invitations sent by Students to other Students and their status (whether they have been accepted or not). It communicates with `TeamManager` and `NotificationManager` to update teams and deliver notifications to Students.
- **CKB Notification System.** `CKB Notification System` allows Students to be notified via email when a new CodeKataBattle or Tournament is created or ends. It relies on an external Mailing Service. This component handles notifications to be sent relying on three modules:
  - **Tournament Notification Manager.** `Tournament Notification Manager` is used to handle notifications elated to Tournament events (such as Tournament creation, registration deadline expiration and closing) through its interface `InvitationNotificationI`.
  - **Battle Notification Manager.** `Battle Notification Manager` handles notifications related to CodeKataBattles within Tournaments. It exposes an interface called `BattleNotificationI` to allow the `Battle Manager` to deliver notifications about Battle creation, start and end to Students.
  - **Invitation Notification Manager.** `Invitation Notification Manager` is used to handle notifications regarding invitations to join a team. Its interface is called `InvitationNotificationI`.

External entities are:

- **Google Spreadsheet.** `Google Spreadsheets` is the engine used by Educators when creating new rules for badges.
- **Mailing Service.** `Mailing Service` is used to send emails to Students when a new CodeKataBattle or Tournament is created or ends.

- **Github.** `Github` interacts with the `Battle Manager` component to handle the code that represents the solution of a CodeKataBattle. CKB platform offers APIs to allow Github Actions Workflow to trigger a pull request to the repository of the solution.

DBMS components are:

- **LongTermData DBMS.** `LongTermData DBMS` is the DBMS used to store and manage data that is meant to be accessed less frequently and that will probably be stored for more time, like Users, Badges, Tournament results.
- **ShortTermData DBMS.** `ShortTermData DBMS` is the DBMS used to store and manage structured data that will probably be accessed more frequently by a higher number of users, like CodeKataBattles, Tournaments or Teams. This kind of data can be stored for a shorter period of time because after the end of a Tournament or a CodeKataBattle most of it will not be useful anymore.

2.3. Deployment View

The distribution of components capturing the topology of the system is illustrated below by using a deployment diagram. The system is structured in a multitier architecture.

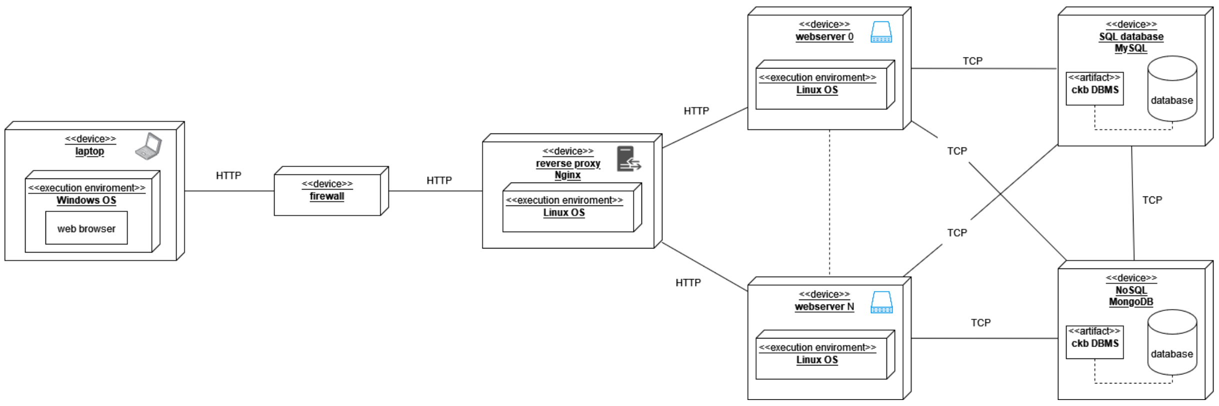


Figure 2.2: Deployment diagram of the CKB platform.

Laptop devices

This node represents a user's laptop, which is the client-side hardware. The "Windows OS" indicates that the laptop is running on the Windows operating system. The "web browser" is the software application used to access the web interface of the platform.

Firewall

This is a security device that monitors and filters incoming and outgoing network traffic based on an organization's previously established security policies. Here, it acts as a barrier between secure internal networks and untrusted external networks, such as the internet.

Reverse Proxy

This server is running Nginx, which is a web server that can also be used as a reverse proxy. This means it can distribute traffic to various servers, thereby acting as an additional layer of abstraction and control to smooth the flow of network traffic between clients and services.

Webserver 0

This is one of potentially multiple web servers that handle the incoming HTTP requests from the client's web browser, process those requests, and serve the appropriate web pages. It runs on a Linux operating system, which suggests a preference for open-source solutions.

Webserver N

This indicates there are multiple webservers in this deployment, following a similar configuration to "Webserver 0." The "N" represents an indefinite number, showing that the architecture is scalable and can include as many webservers as needed.

SQL database (MySQL)

This database node uses MySQL, which is a relational database management system (RDBMS) based on SQL (Structured Query Language). It's used to store and manage the platform's structured data efficiently.

NoSQL database (MongoDB)

This is a NoSQL database, specifically MongoDB, which is designed for storing unstructured data. It offers high performance, high availability, and easy scalability.

CKB DBMS

This artifact within both database nodes represents the database management software that's part of the CodeKataBattle platform. It is likely the collection of schemas, tables, queries, reports, views, and other objects associated with the platform's database management.

2.4. Component Interfaces

Invitation Notification Manager

```
sendMail(n: Notification): JSONObject
listenEvent(e: InvitationEvent): boolean
updateInvitationNotification(n: set<Notification>): boolean
getNotifications(userId: Long): set<Notification>
```

Battle Notification Manager

```
sendMail(n: Notification): JSONObject
listenEvent(e: battleEvent): boolean
updateBattleNotification(n: set<Notification>): boolean
getNotifications(userId: Long): set<Notification>
```

Tournament Notification Manager

```
sendMail(n: Notification): JSONObject
listenEvent(e: tournamentEvent): boolean
updateTournamentNotification(n: set<Notification>): boolean
getNotifications(userId: Long): set<Notification>
```

Tournament Manager

```
AddBadges(tId: Long, b: set<Badges>): boolean
AddBattle(tId: Long,e: Educator, b: Battle): boolean
AddStudent(tId: Long, s: Student): boolean
notifyStudents(tId: Long, e: Event): JSONObject
getBattles(tId: Long,): set<Battle>
getLeaderBoard(tId: Long,): map<int, Student>
updateLeaderBoard(tId: Long,): boolean
createTournament(e: Educator, b: set<Badges>): JSONObject
closeTournament(tId: Long,): boolean
notifyStudents(tId: Long, event: JSONObject): boolean
addEducator(tId: Long, e: Educator): boolean
getBadges(tId: Long): set<Badge>
```

Student Dashboard Manager

```
getBadges(id: Long): set<Badge>
getTournaments(id: Long): set <Tournament>
answerInvitation(id: Long, in: InvitationNotification): boolean
getTeams(id: Long): set<Team>
```

User Manager

```
getNotifications(id: Long): set<Notification>
deleteNotification(id: Long, n: Notification): boolean
createUser(username: String, role: userType): boolean
deleteUser(userId: Long): boolean
```

Invitation Manager

```
sendInvitation(userId: Long, info: JSONObject): boolean
answerInvitation(userId: Long, info: JSONObject): boolean
```

Team Manager

```
createTeam(name: String, s: set<Student>, b: Battle): boolean
addMember(tId: Long, s: Student): boolean
getMembers(tId: Long): set<Student>
getScore(tId: Long): int
setGitHubRepo(tId: Long, url: String): boolean
```

Educator Dashboard Manager

```
getTorunaments(eId: Long): set<Tournaments>
grantPermissionToEducatorForTournament(eId: Long, tId: Long,
                                        receiverId: Long): boolean
```

Badge Manager

```
createBadge(rules: JSONObject): boolean
getBadge(bId: Long): Badge
getBadges(): set<Badge>
```

Battle Manager

```
addTeam(bId: Long, t: Team): boolean
getLeaderBoard(bId: Long): map<int, Team>
updateLeaderboard(bId: Long): boolean
createBattle(e: Educator, battleInfo: JSONObject): boolean
notifyTeams(bId: Long, event: JSONObject): boolean
```

Automated Evaluation Manager

```
evaluateTeam(b: Battle, t: Team): int
```

Gamification Rules Manager

```
createRule(rule: JSONObject): boolean
```

```
createVariable(var: JSONObject): boolean
getRules(): JSONObject
getVariables(): JSONObject
```

2.5. Runtime View

Here we present the dynamic of our system through sequence diagrams. We have found the components that communicate with each other to form our system, so now we explain their behaviors.

First, we will present CKB platform actions from the point of view of a general user, i.e., logging in, booking a charge, managing his profile, etc. Then, we will present CKB platform actions from the point of view of the student and educator, actions that are related to their specific role. In this section, we hadn't presented all the RASD document use cases because we have decided to focus on the critical part of the system functionalities.

Registered User Login When a user WebApp wants to log in to the CKB platform, he calls the "`logIn`" function from the `UserManager` component through it's interface. The `UserManager` component then requests the UserWebApp to display the login form from which, after the user inserted the credentials it sends the information to the `AuthenticationService` to validate it's correctness. Based on the response from the `AuthenticationService` the `UserManager` component will send a confirmation or an error to the client.

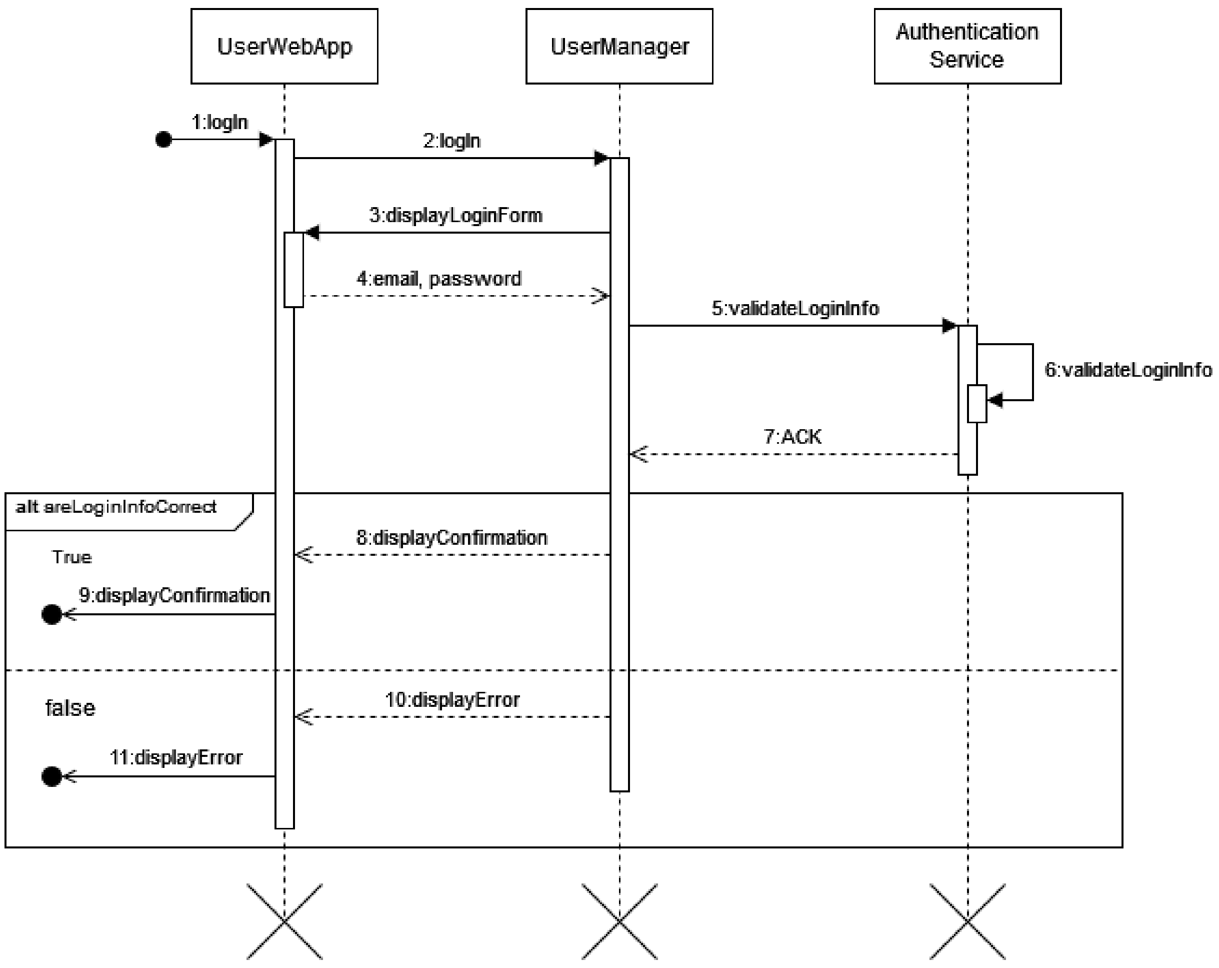


Figure 2.3: Registered User logs in sequence diagram

Student subscribe to a tournament When a student wants to subscribe to a open tournament, the User WebApp after login knows that the user is a student so it shows the student the corresponding StudentDashboard. In the StudentDashboard the student can select a tournament through the list of tournaments and clicks the subscribe button near the selected tournament. With the interface `TournamentI` the `StudentDashboardManager` calls the subscribe function, the `TournamentManager` will check if in the CKB data base the student has already subscribed. Only then the `TournamentManager` will sends a corresponding confirmation or an error to the client.

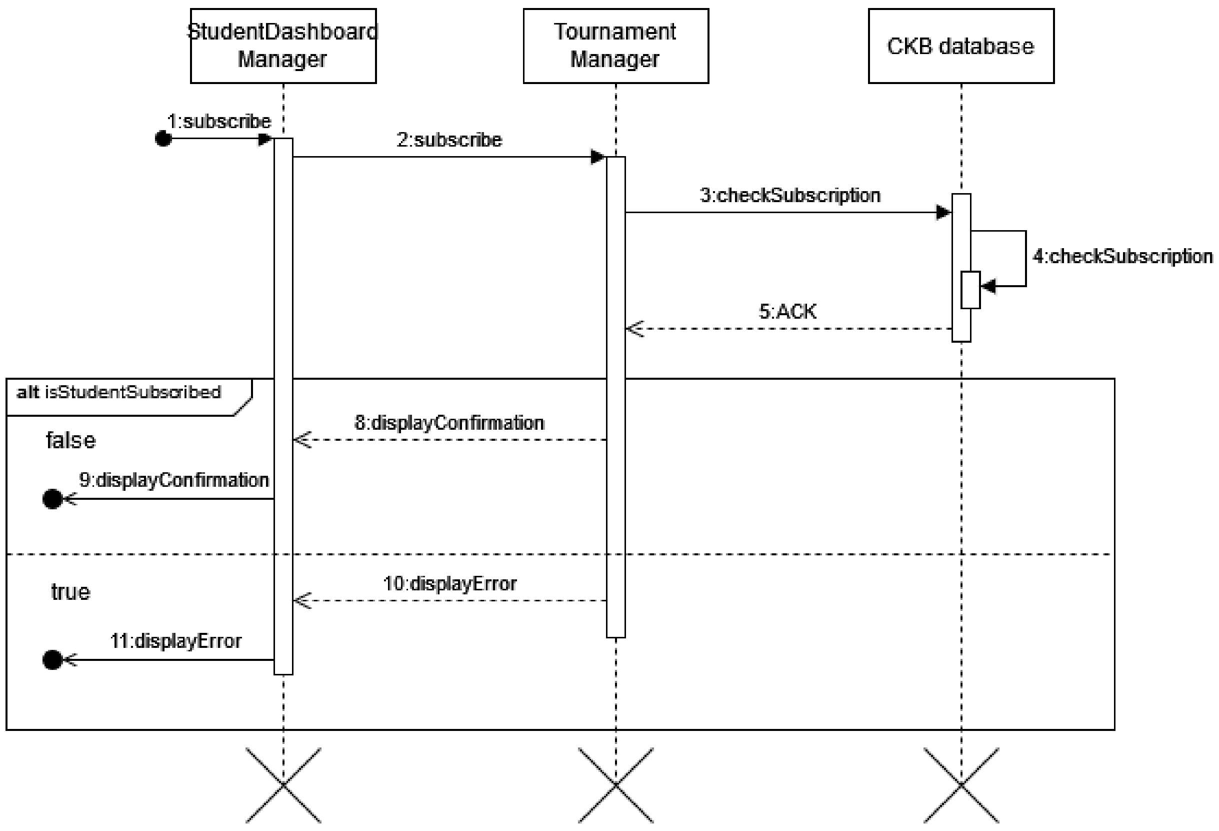


Figure 2.4: Student subscribe sequence diagram

Student creates a team For students already subscribed in a tournament, they can call the `createTeam` function through the `TeamManagerI` interface. `TeamManager` component will send a request to the `StudentDashboardManager` for the size and name of the group. After the student has inputed the information, the `TeamManager` will check the CKB database to ensure there are no teams with the same name already inside the DB. Then it will create the Team and norificate the student of the success or send an error.

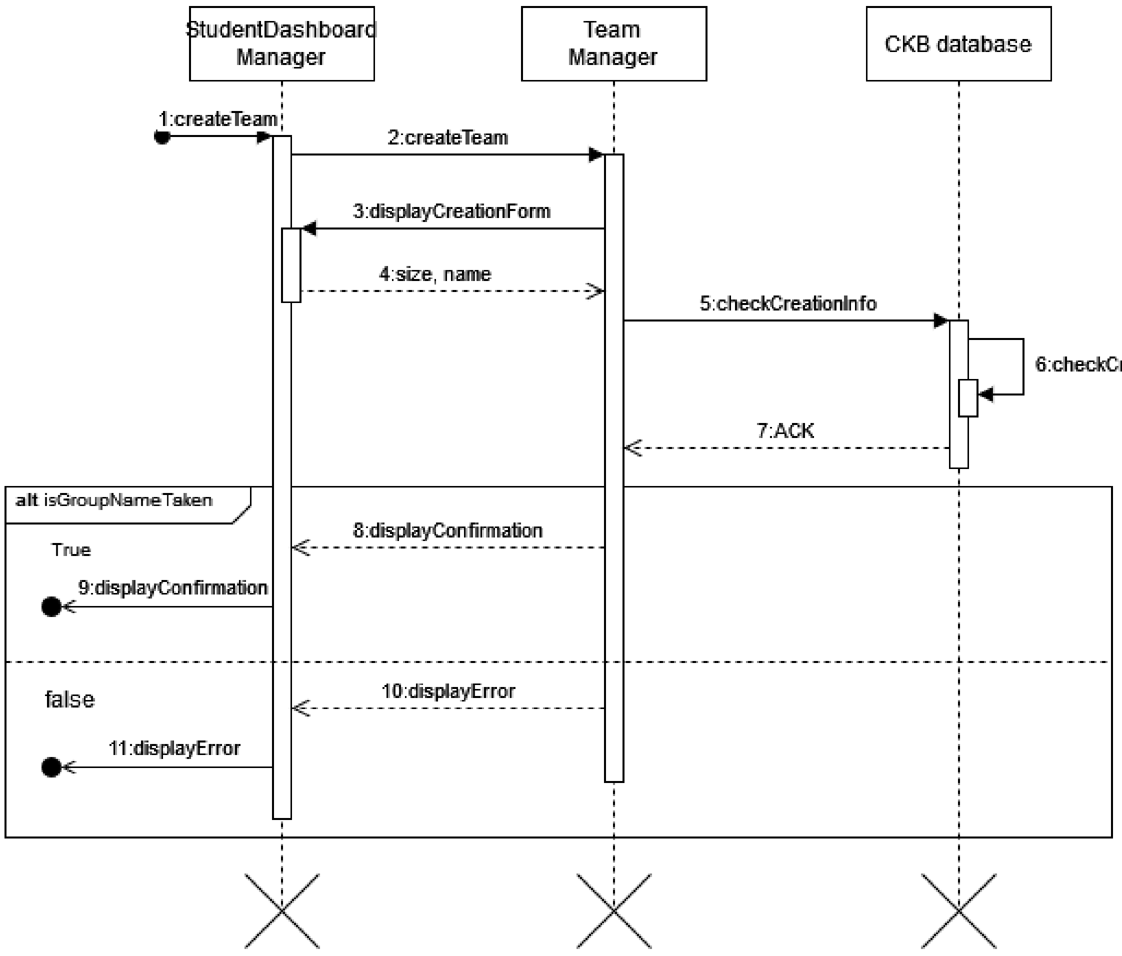


Figure 2.5: Student create team sequence diagram

Student invites another student to the team After the team has been created, the student can invite other students to join the team. The student can call the `inviteStudent` function through the `TeamManagerI` interface. `TeamManager` component will send a request to the `StudentDashboardManager` for the email of the student to invite. After the student has inputed the information, the `TeamManager` will check the CKB database to ensure the student is not already in a team. Then it will send an email to the student to invite him to the team and notificate the student of the success or send an error.

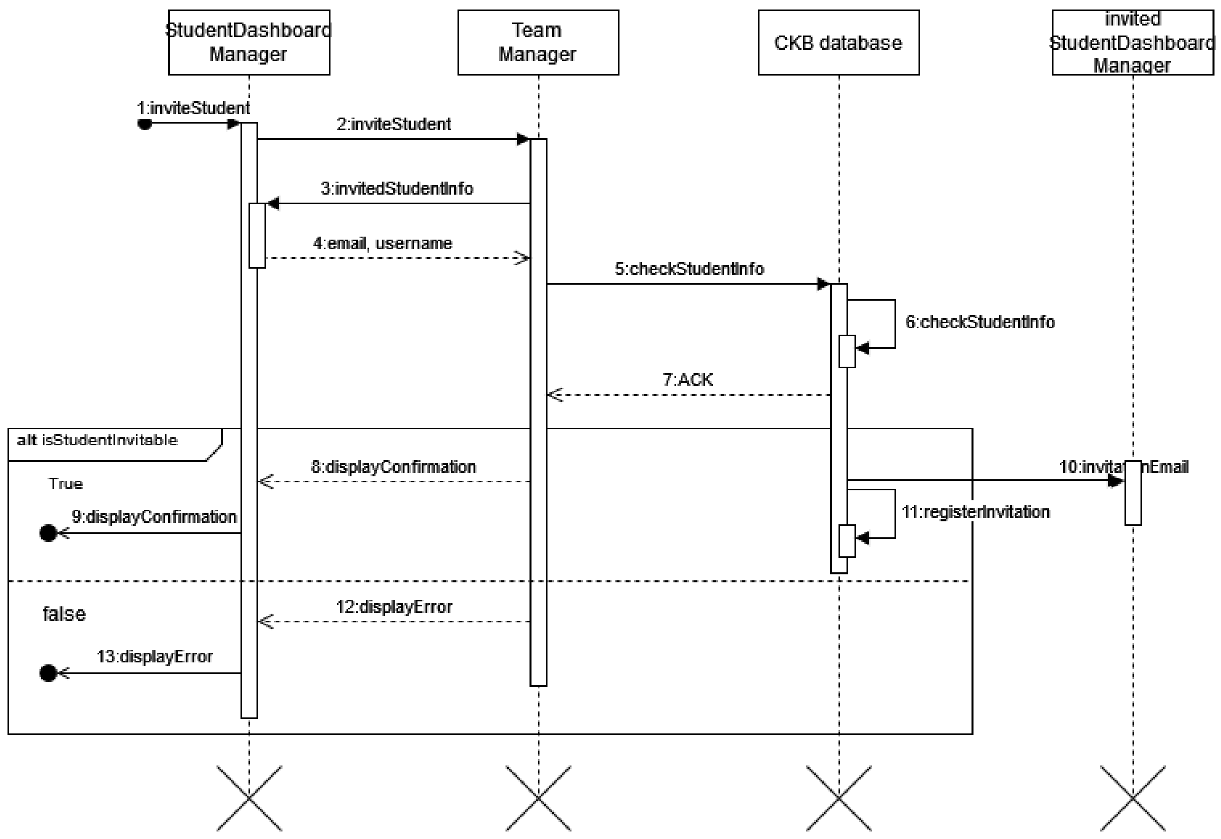


Figure 2.6: Student invitation sequence diagram

Student accepts or rejects an invitation to join a team After receiving an invitation to join a team, a student can accept or reject this invitation by invoking the `acceptInvitation` or `rejectInvitation` function through the `TeamManagerI` interface. The `TeamManager` component processes the request, either adding the student to the team or removing the invitation from the CKB database, depending on the student's decision. The `TeamManager` then notifies the student of the outcome. If the operation is successful, a confirmation is sent. If an error occurs, an error message is returned, allowing the student to handle the error appropriately.

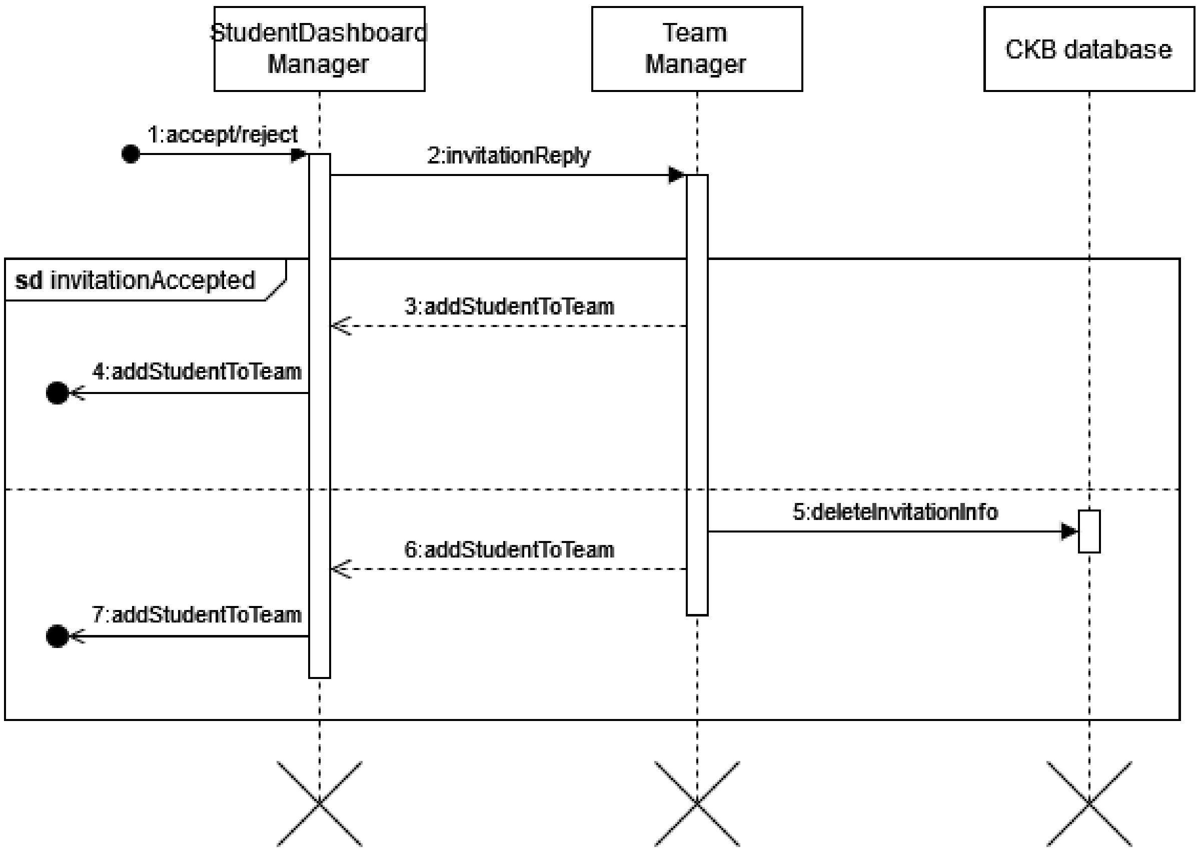


Figure 2.7: Student handle invite sequence diagram

Student join a battle When a student, either individually or as part of a team, wishes to join a battle, they invoke the `joinBattle` function from the `BattleI` interface. This function requires specific parameters such as the battle ID and student or team ID. The `BattleManager` receives the request and checks the registration deadline. If the deadline has not passed, the `BattleManager` admits the student or team to the battle and updates the CKB database accordingly. A confirmation is then sent to the client. If the deadline has passed or if any other error occurs, an error message is returned to the client, allowing them to handle the error and provide appropriate feedback to the student.

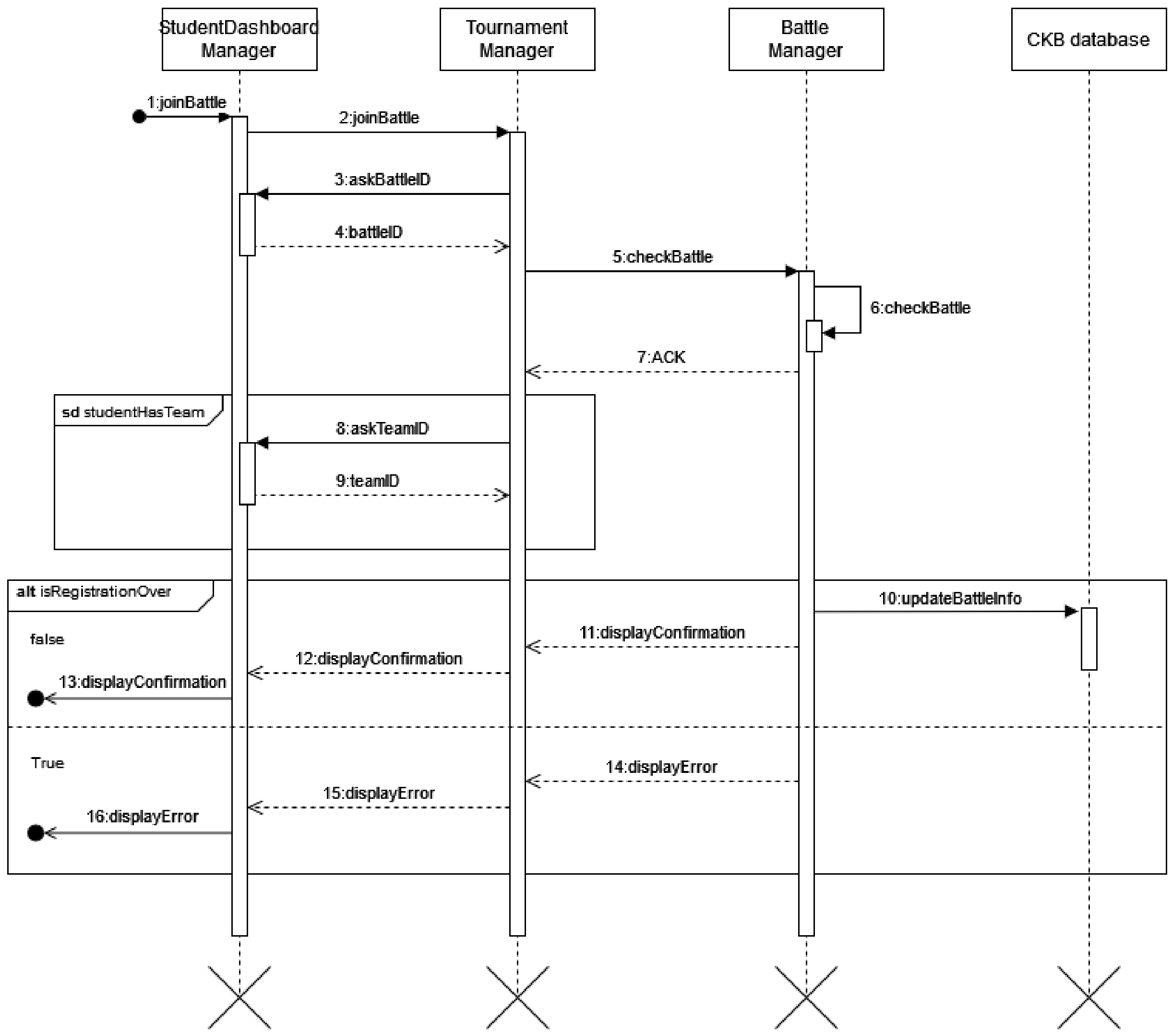


Figure 2.8: Student join battle sequence diagram

Educator creates a tournament To create a tournament, the `EducatorDashboardManager` invokes the `createTournament` function from the `TournamentI` interface. This function requires specific parameters such as the tournament name, number of participants, and other relevant details. The `createTournament` function then passes the request to the `TournamentManager` component. The `TournamentManager` handles the creation of the tournament, including any necessary error checking, and inserts the tournament details into the database. If the tournament creation is successful, a confirmation is returned to the `EducatorDashboardManager`. If the creation fails, an error message is returned instead, allowing the `EducatorDashboardManager` to handle the error and provide appropriate feedback to the educator.

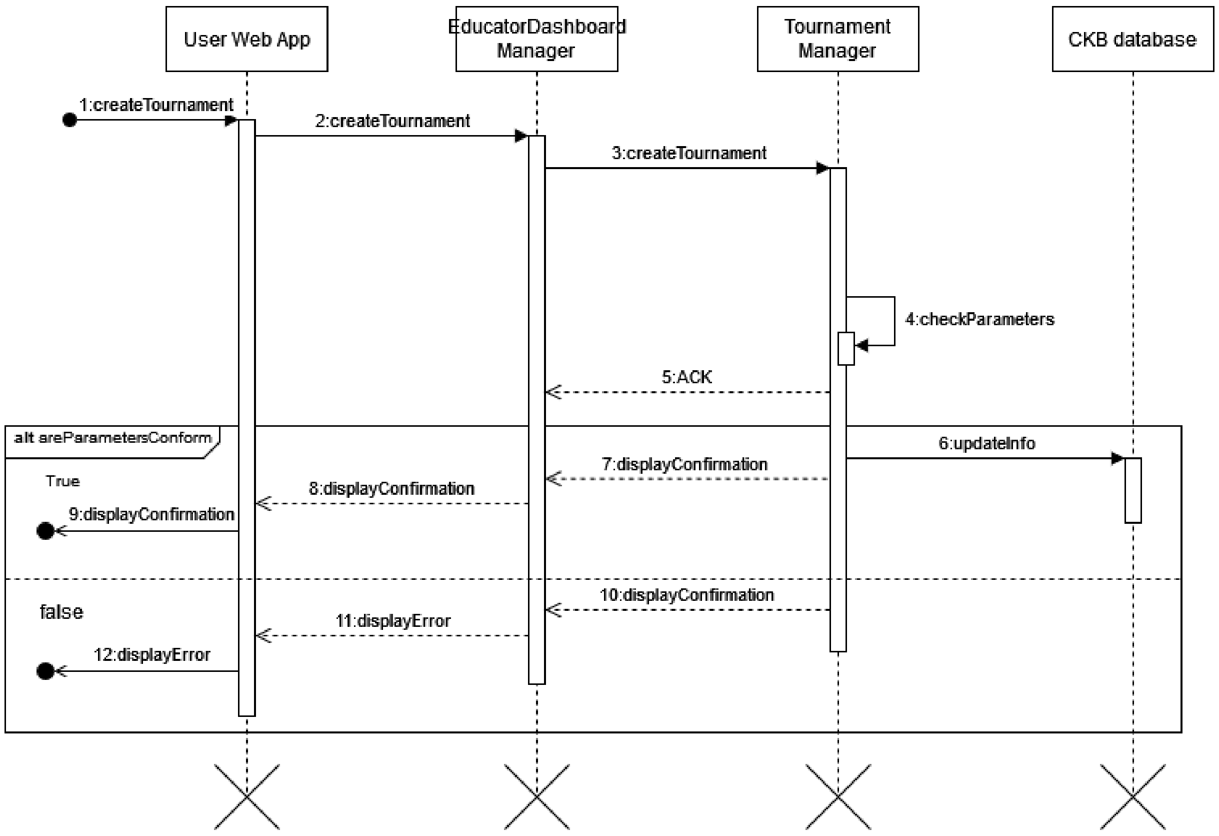


Figure 2.9: Educator create tournament sequence diagram

Educator uploads a CKB When an educator needs to upload a CKB, they can invoke the `uploadCKB` function from the `BattleI` interface. This function requires specific parameters such as the CKB file and its associated metadata. The `BattleManager` component receives the request and processes it by validating the CKB file and its metadata, and then storing them in the CKB database. During this process, the `BattleManager` performs necessary checks, such as verifying the file format and integrity. If the operation is successful, a confirmation is sent back to the `EducatorDashboardManager`. In case of an error, such as the file being in an incorrect format or corrupted, an error message is returned, allowing the `EducatorDashboardManager` to handle the error and provide appropriate feedback to the educator.

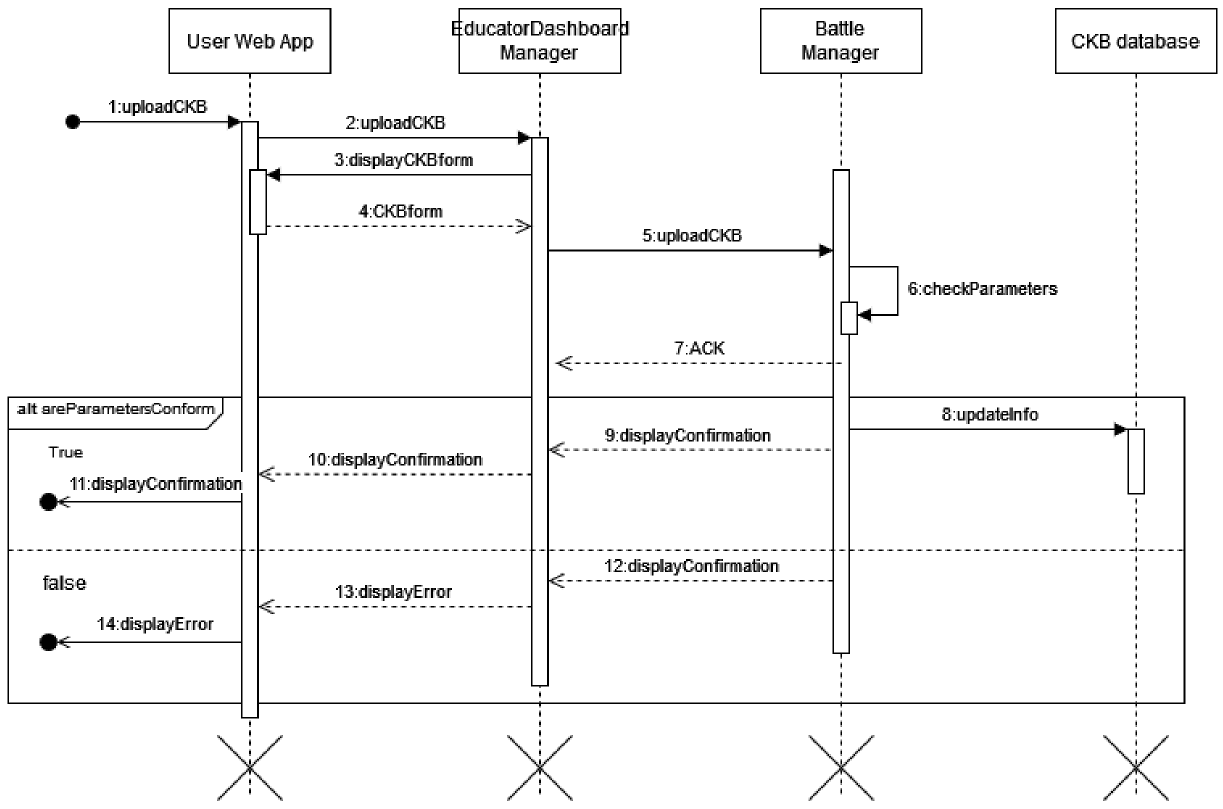


Figure 2.10: Educator upload ckb sequence diagram

Educator grants permission to another educator When an educator needs to grant permission to another educator, they can invoke the `grantPermission` function from the `TournamentI` interface. This function requires specific parameters such as the tournament ID and the educator ID. The `TournamentManager` component receives the request and processes it by granting the specified educator permission to manage the specified tournament.

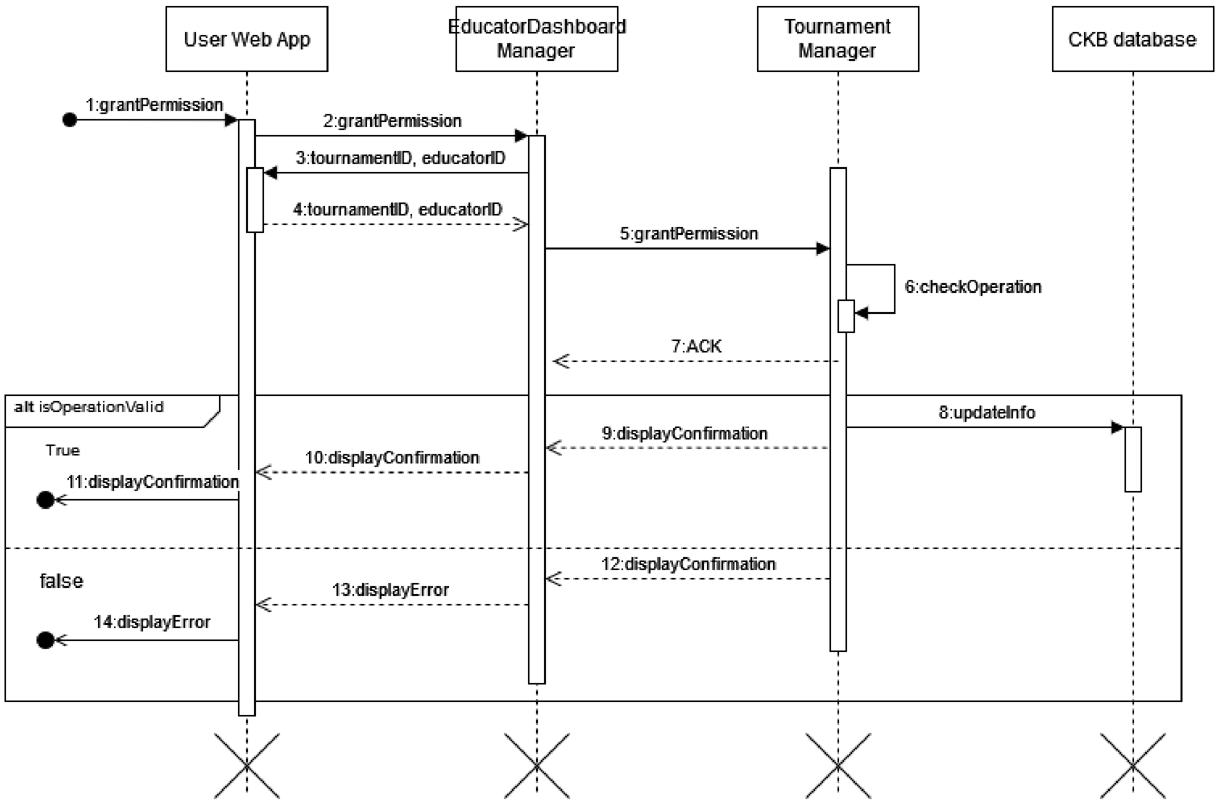


Figure 2.11: Educator grant permission sequence diagram

Educator creates a new badge When an educator needs to create a new bagde, they have to create new rules associated with the new badge. They can invoke the `createBadge` function from the `BadgeI` interface. This will required inputing the badge name, description and the rules associated with the badge. The `BadgeManager` should display a form with various pre-built rules to choose from, but the educator can also invoke the `createRule` function from the `GamificationRulesI` interface. This function requires specific parameters such as the rule name, the rule description and the rule formula. The `GamificationRulesManager` component receives the request and processes it by validating the rule formula and storing it in the CKB database. During this process, the `GamificationRulesManager` performs necessary checks, such as verifying the formula syntax. If the operation is successful, a confirmation is sent back to the `EducatorDashboardManager`. In case of an error, such as the formula being in an incorrect format or corrupted, an error message is returned, allowing the `EducatorDashboardManager` to handle the error and provide appropriate feedback to the educator. Once the rules are created, the `BadgeManager` can create the badge and store it in the CKB database. If the operation is successful, a confirmation is sent back to the `EducatorDashboardManager`. In case of an error, such as the badge name being already used, an error message is returned, allowing the `EducatorDashboardManager` to handle the error and provide appropriate feedback to the educator.

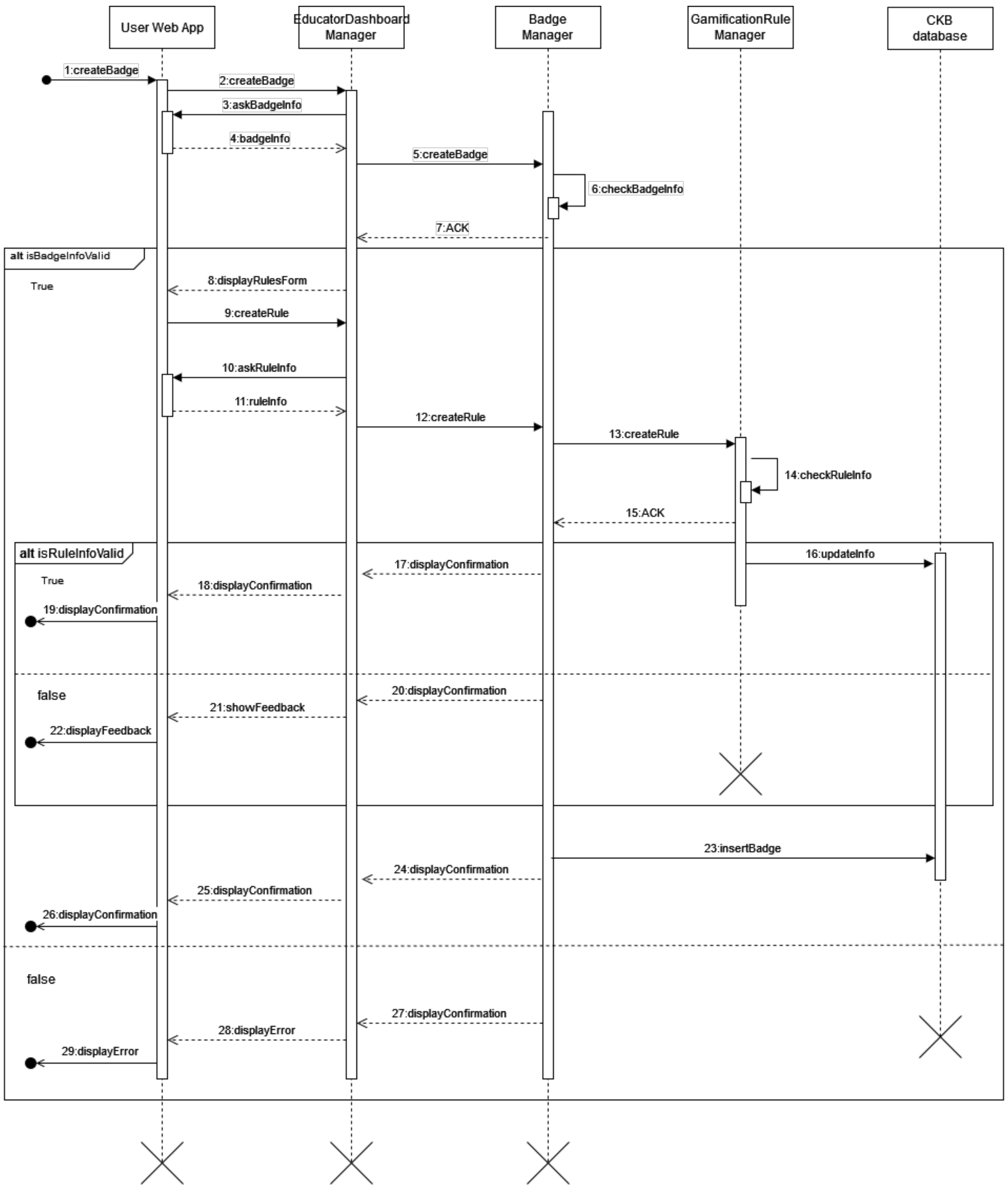


Figure 2.12: Educator create badge sequence diagram

Educator manually evaluates a team When an educator needs to manually evaluate a team, they can invoke the `evaluateTeam` function from the `BattleI` interface.

This function requires specific parameters such as the battle ID and the team ID. The `BattleManager` component receives the request and processes it by evaluating the specified team and updating the CKB database accordingly. If the operation is successful, a confirmation is sent back to the `EducatorDashboardManager`. In case of an error, such as the team not existing or the battle not being closed, an error message is returned, allowing the `EducatorDashboardManager` to handle the error and provide appropriate feedback to the educator.

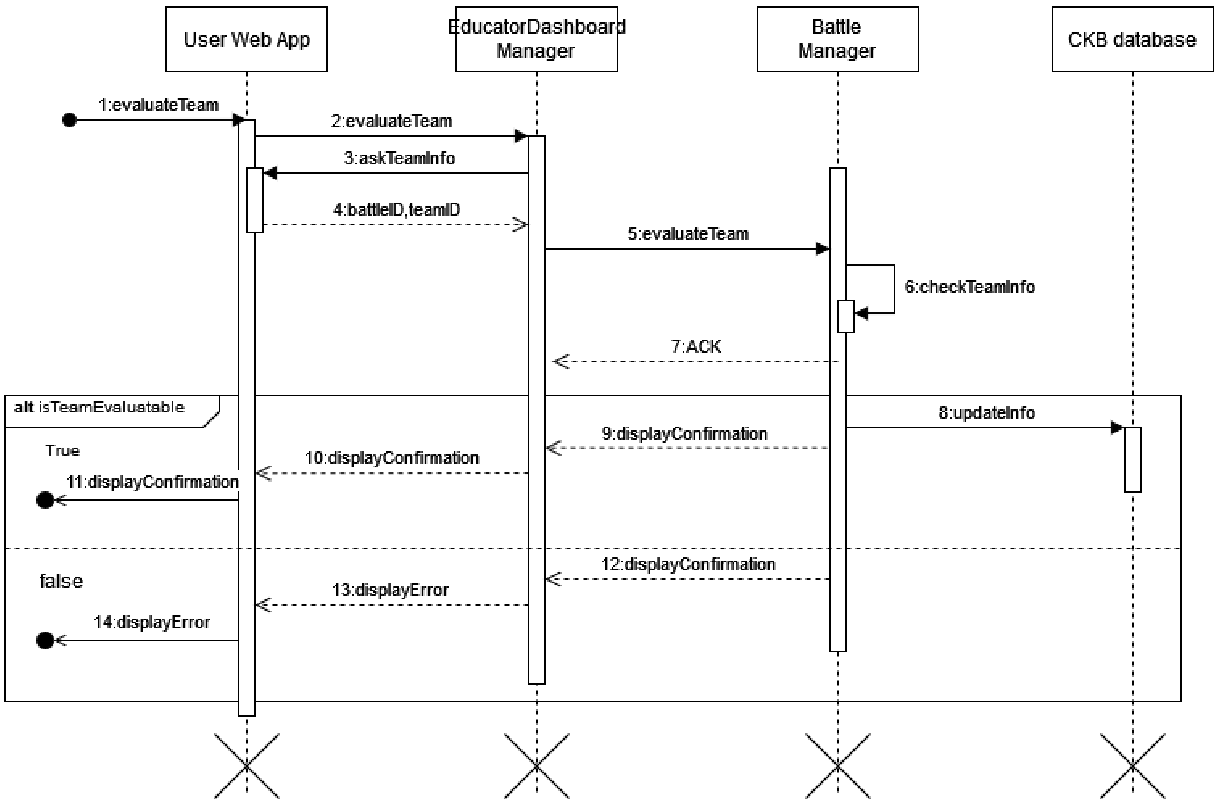


Figure 2.13: Educator manually evaluate sequence diagram

User visualize tournament rankings When a user wants to visualize the rankings of a tournament, they can invoke the `getTournamentRankings` function from the `TournamentI` interface. This function requires specific parameters such as the tournament ID. The `TournamentManager` component receives the request and processes it by retrieving the rankings of the specified tournament from the CKB database. If the operation is successful, the rankings are sent back to the client.

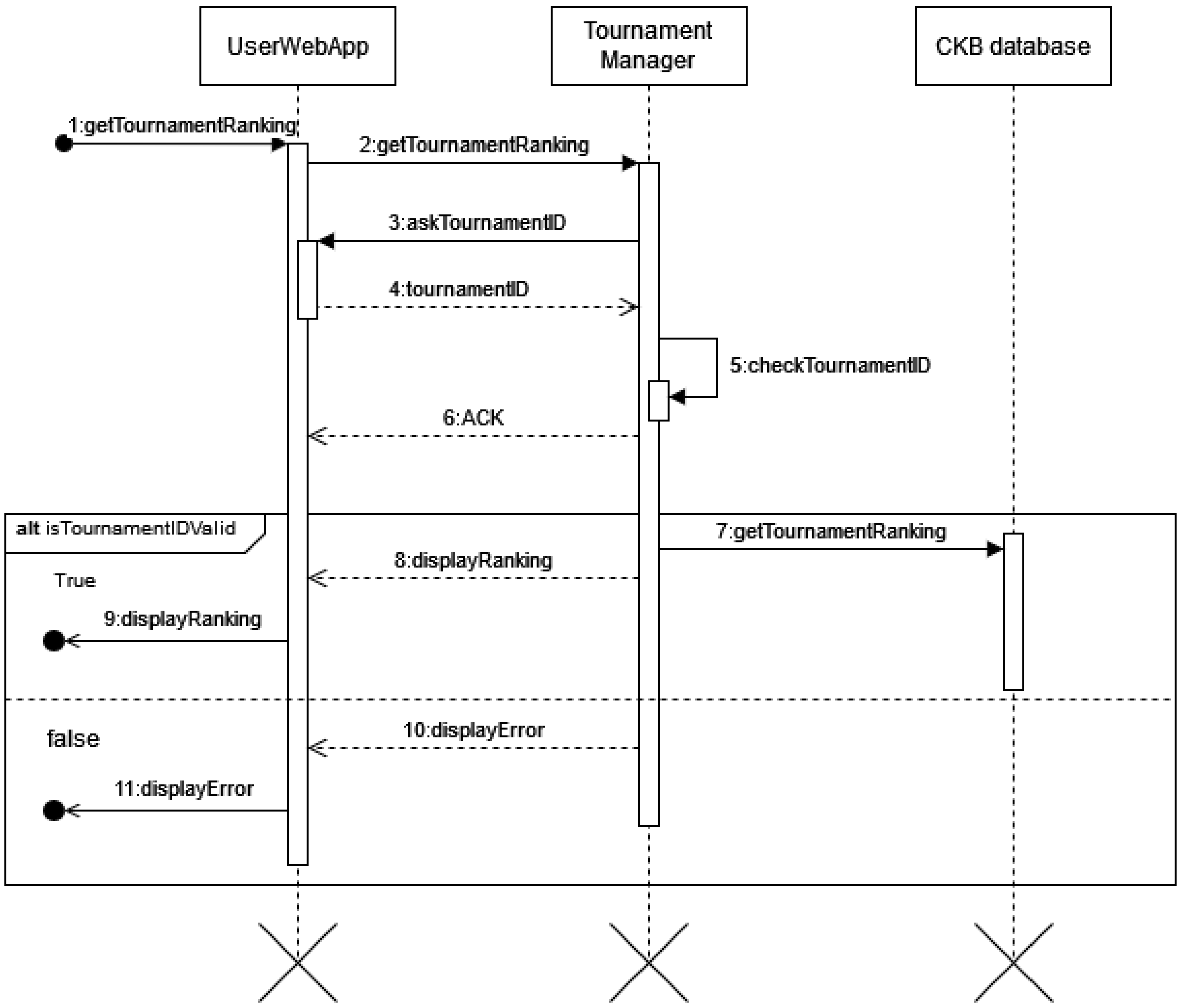


Figure 2.14: User see tournament ranking sequence diagram

User visualize other users' profiles When a user wants to visualize the profile of another user, they can invoke the `getUserProfile` function from the `UserManagerI` interface. This function requires specific parameters such as the user ID. The `UserManager` component receives the request and processes it by retrieving the profile of the specified user from the CKB database. If the operation is successful, the profile is sent back to the client.

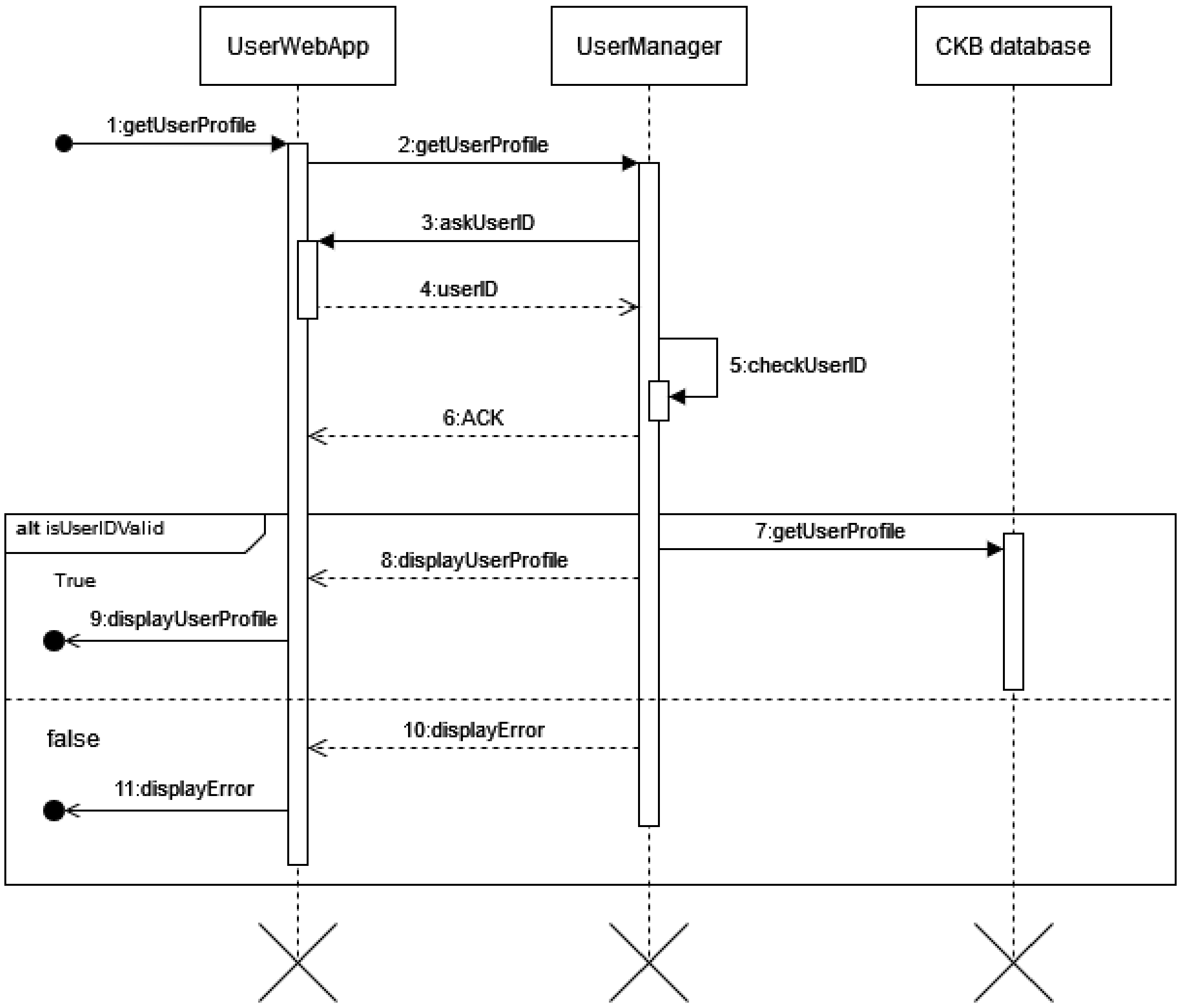


Figure 2.15: User see profile sequence diagram

2.6. Selected architectural styles and patterns

In this section are shown the most relevant architectural design choices, with also the explanation of their role and their advantages.

3 Tier Architecture

N-tier architecture refers to the structure of a software application divided into multiple tiers. A tier is a layer of the application that operates on its own infrastructure or server. In particular, we decided to implement a 3-tier architecture. The tiers are: presentation, logic, and data tier.
Presentation tier → It presents information to users and receive user inputs
Logic tier → It handles business logic and application processing logic. It's responsible for processing user requests, applying business rules, and managing the flow of data between the presentation and data tier
Data tier → It manages the storing and retrieving of the data. It handles data persistence and integrity too
The advantages of using this type of architecture revolve around its scalability, ease of management, and flexibility and security. If we need to add more resources to a specific tier or modify it, we can do so without affecting the other tiers. Moreover, we can secure each of the three tiers separately using different methods and techniques, based on the security needed for the specific tier

WebSocket with Pub/Sub Pattern

The WebSocket protocol provides a full-duplex communication channel over a single TCP connection. This means that data can be sent and received at the same time, allowing for real-time communication between the client and the server. The Publish/Subscribe pattern is a messaging pattern where senders (publishers) categorize messages into classes without knowing which subscribers (if any) there might be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowing which publishers there are When combined, WebSocket and Pub/Sub can be a powerful tool for real-time web applications. Regarding the CKB specifically:
1. Publishing → when an event occurs in CKB (e.g., a new battle is created, a score is updated,. . .), the server publishes a message about this event. The message is categorized based on the type of event (e.g., "New Battle", "Score Updated",. . .)
2. Subscribing → Clients (users of CKB) subscribe to the types of events they're interested in. For example, a student might subscribe to "New Battle" events to be notified when a new battle is created.
3. Real-Time Updates → Because the communication is happening over WebSocket, these

messages can be pushed from the server to the client in real-time. As soon as a new message is published, all clients subscribed to that type of event will receive the message.
4. Scalability → The Pub/Sub pattern allows for scalability. As the number of users (subscribers) and events (publishers) increases, the system can continue to function efficiently.
This combination of WebSocket and Pub/Sub can provide a dynamic, real-time user experience for CKB, making it more engaging and responsive for its users.

Worker Pooling
Worker pooling refers to a pattern where a set of initialized "workers" (threads, processes, or any kind of parallel-executing entities) are kept ready to perform a certain kind of task. In the context of CKB, worker pooling can help to improve the performance and responsiveness of your server by reusing workers instead of creating new ones for each request. This can be particularly helpful in a high-load scenario where new requests are coming in faster than the server can create new workers. Moreover, thanks to the presence of the reverse proxy, we can obtain a balanced load cross the network. Infact, a reverse proxy can distribute incoming requests to multiple server replicas. This is done to achieve evenly distributed load across the servers replica, preventing any single server from becoming a bottleneck. Each replica would have its own worker pool to handle the requests it receives

Microservices
The application uses a suite of small services, instead of being monolithic, each running in its own process and communicating with the other microservices. To communicate between each other, microservices can exploit an event-driven paradigm (like publisher and subscriber). Moreover, other design choice must be taken into consideration, for example to achieve resiliency. One suggestion could be to utilize a Circuit Breaker to inhibit all those microservices that are failing repeatedly. In the specific context of CKB, this could be momentarily debilitating some functions of the platform, such as the possibility to visualize the leaderboard of a Tournament or adding members to a team. This way, the platform itself can keep running, and maintenance can be executed on the faulty microservices to bring them back up as quickly as possible

Database: NoSQL with SQL
In the context of CodeKataBattle (CKB), the use of two different databases, a NoSQL database (MongoDB) and a SQL database (MySQL), is a strategic decision aimed at leveraging the strengths of both types of databases to efficiently manage and access data.
Let's look at the strength of using a NoSQL database.
Scalability → NoSQL databases are more scalable than traditional relational databases.

They can spread out the data storage and computing processes over a cluster of computers. This allows for almost unlimited growth as increasing capacity simply requires adding more computers to the cluster.
Flexible Data Model → Unlike relational databases, NoSQL databases can easily store and combine any type of data, both structured and unstructured. They can dynamically update the schema to evolve with changing requirements without any interruption or downtime to an application.
Resilience → With NoSQL, data is distributed across multiple servers and regions, so there is no single point of failure. As a result, NoSQL databases are more stable and resilient, with continuous availability and zero downtime.
Given these characteristics, a NoSQL database is suitable for storing most of the data from our CKB platform, like users, educators, tournaments' results, badges, notifications, etc. Infact, the platform should be able to deal with collecting a big number of data and storing them for later retrieval. We aspect that the system won't perform many queries on the data stored in the NoSQL database. Thus, it's main functionality is to store long time data to ensure that every user can check out past events without problem (so for example, a user that wants to see the leaderboard of a Tournament ended months before, or when visiting the profile of another user)
On the other hand, we have the SQL database. While the system exploits the NoSQL database mostly for storing tons of data over time, we are going to use the SQL database to store the data for which we expect a high number of queries will be performed. This is the case of an ongoing Tournament. In fact, lot's of user will access the battles of the tournament, push their solutions, form teams, and so on. This requires the system to perform a high number of data retrieval from the database, and using a SQL database it ensures smooth operation. Therefore, we will store data like Battles, teams, and Tournaments in the SQL database. Once a Tournament is closed, we could free the memory of the SQL databases after updating the NoSQL one. This will allow to avoid saturation of the database and faster retrieval of data.
The entity 'Tournament' is shared between both databases, necessitating synchronization to ensure data consistency. The decision between updating the NoSQL when a specific Tournament ends or while it is still on depends on the capacity of the system and the stress it can undergo. As a best practice, periodic synchronization between the SQL database and the NoSQL database while a Tournament is on are suggested.

2.7. Other Design decisions

Firewall

A firewall is a network security system that monitors and controls incoming and outgoing traffic based on predetermined security rules. The firewall sits between the CKB's network and the larger internet, examining all network traffic and blocking any that doesn't meet its configured security rules. This can help protect CKB from various types of attacks, such as SQL injection, cross-site scripting and other exploits that could take advantage of vulnerabilities in the web application. Moreover, a firewall can prevent unauthorized data from leaving the web application, adding an extra layer of data protection. This is particularly important for a platform like CKB, which might handle sensitive data such as coding challenge solutions and users' info. The firewall must be configured to allow WebSocket traffic, which is used for real-time updates in CKB.

Reverse Proxy

A reverse proxy is a server that sits between client devices and a web server, forwarding client requests to the web server and returning the server's responses back to the clients. In CKB, the reverse proxy plays a crucial role in managing and directing network traffic to multiple server replicas. The reverse proxy receives client requests and intelligently distributes them across multiple server replicas, each equipped with its own worker pool. This distribution is done in a manner that ensures an evenly balanced load across the network, preventing any single server from becoming a bottleneck. This is particularly beneficial in high-load scenarios where numerous requests are coming in simultaneously.
Moreover, the reverse proxy provides an additional layer of abstraction and control to ensure smooth flow of network traffic between clients and servers. It can provide additional features like SSL termination, HTTP/2 support, caching, and more. This not only enhances performance but also adds a layer of security, shielding the servers from direct exposure to the internet.
In addition, the reverse proxy can provide other benefits such as masking the internal structure of the backend network, providing a single point of entry into the system, and simplifying routing and network configuration. It can also provide resilience by detecting failure of a server replica and redirecting traffic to other healthy replicas

Model-View-Controller (MVC)

The Model-View-Controller (MVC) is a design pattern widely used in software development, particularly in object-oriented programming and web application. It separates an application into three main logical components: the Model, the View, and the Controller
Model $\rightarrow$ The Model contains the methods for accessing data useful to the application.

In CKB, this could include data related to users, battles, tournaments, scores, etc. The Model directly manages the data, logic, and rules of the application
View → The View is responsible for displaying data to the user and managing the interaction between the user and the underlying infrastructure. In CKB, this could include the user interfaces for viewing and joining battles, viewing scores, etc.
Controller → The Controller receives user commends through the View and reacts by performing operations that can affect the Model and generally lead to a change of state in the View. In CKB, this could include operations like creating a new battle, etc.
The MVC pattern simplifies the code of applications by logically separation concerns. This separation allows developers to focus on one aspect of the software at a time, making the code easier to understand, maintain, and expand. It also improves efficiency by allowing different parts of the code to be developed and tested independently.

Factory pattern

The factory pattern is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created. In the context of CodeKataBattle (CKB), the Factory pattern can be effectively used for badge creation, providing a robust and flexible solution for creating different types of badges. It simplifies the badge creation process, makes it easy to add new types of badges, and decouples the badge creation logic from the resto of the application

Decorator pattern

The decorator pattern is a design pattern that allows behavior to be added to an individual object, dynamically, without affecting the behavior of other instances of the same class. In the context of CKB, the decorator pattern can be effectively used for the score system. The base Score component defines the interface for objects that can have responsibilities added to them dynamically. Decorators wrap a Score component and define a new behavior for the methods that calculate the score.
The Decorator patten provides a flexible alternative to subclassing for extending functionality

3 | User Interface Design

In this section will be presented an overview of the user's interface of the CKB platform. The focus will be mainly on the user experience. `CKB` is accessible using any browser and does not have a client application. Images shown below depicting the user interface include both samples for the desktop browser version and the mobile version.

3.1. General Overview

The image below shows a map of the pages which can be accessed from the web application. All pages are available after the user has logged in, except for the registration page and the login page itself.

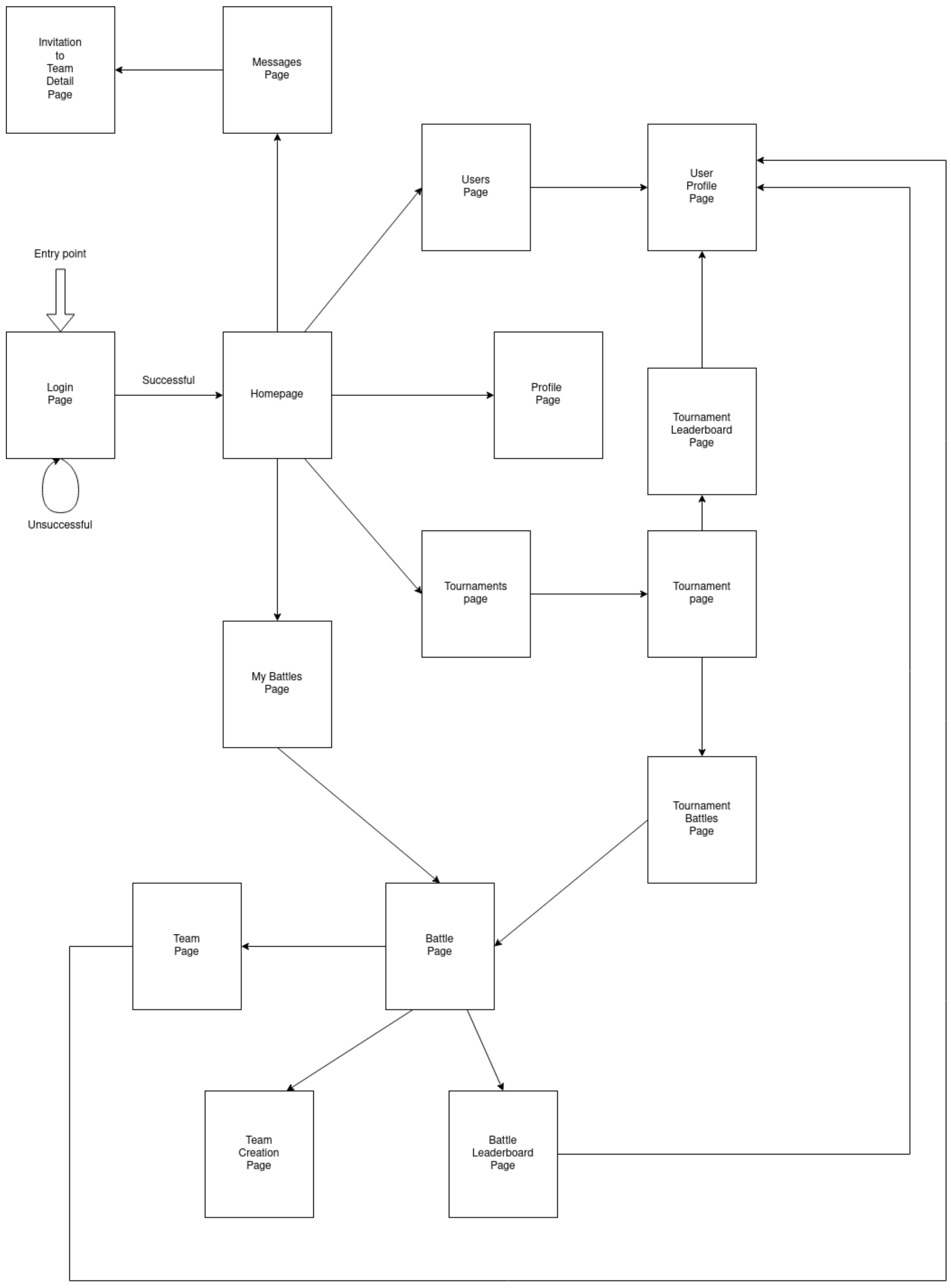


Figure 3.1: Pages of the CKB platform

The login page prompts the user to enter their institution email and redirects them to the login page of their institution, which is not part of the CKB platform so it is not shown here. A successful login (validated by the institution) will redirect the user to the homepage of the CKB platform, which will differ depending on the role of the user.

3.2. Registration, Login and Homepage

The following mockups show the SignUp page and the LogIn page. Depending on the type of user, the Homepage will be slightly different, as Educators will have a menu entry allowing them to create new Tournaments or CodeKataBattles directly from the Homepage.

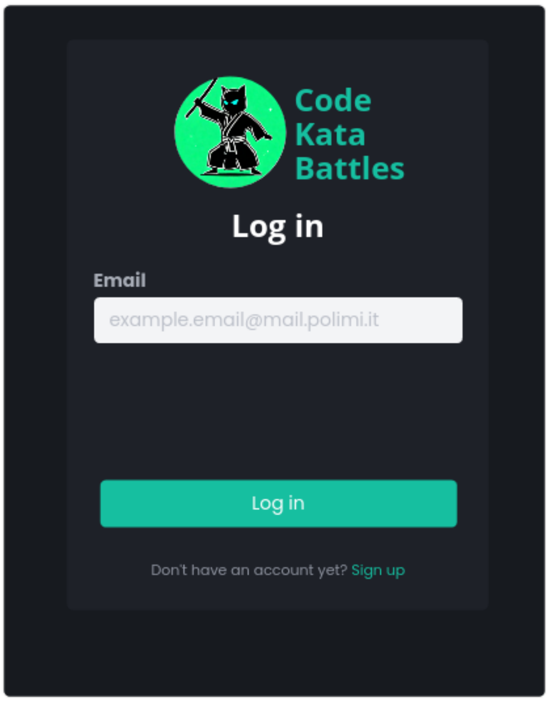

Code
Kata
Battles
Log in
Email
example.email@mail.polimi.it
Log in
Don't have an account yet? Sign up

Figure 3.2: CKB Log in page

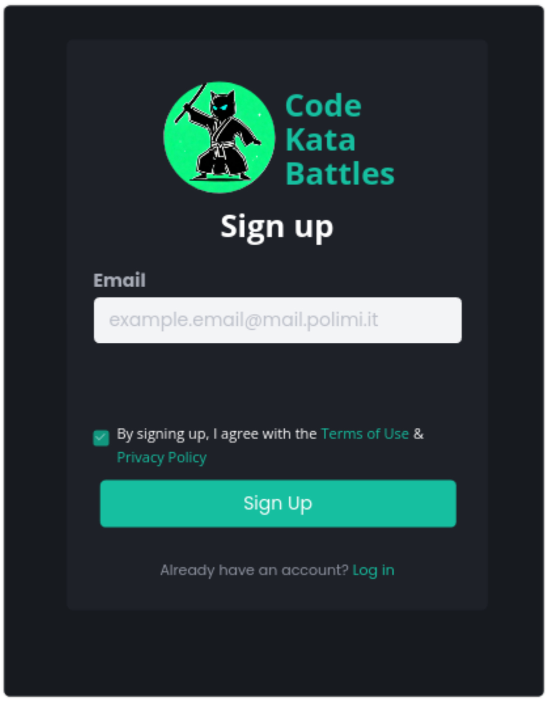

Code
Kata
Battles
Sign up
Email
example.email@mail.polimi.it
By signing up, I agree with the Terms of Use & Privacy Policy
Sign Up
Already have an account? Log in

Figure 3.3: CKB Sign Up page

Both the SignUp page and the LogIn page will redirect the user to the institution's login page as anticipated in Section 3.1.

3.3. Homepage

The Homepage is the first page the user will see after logging in.

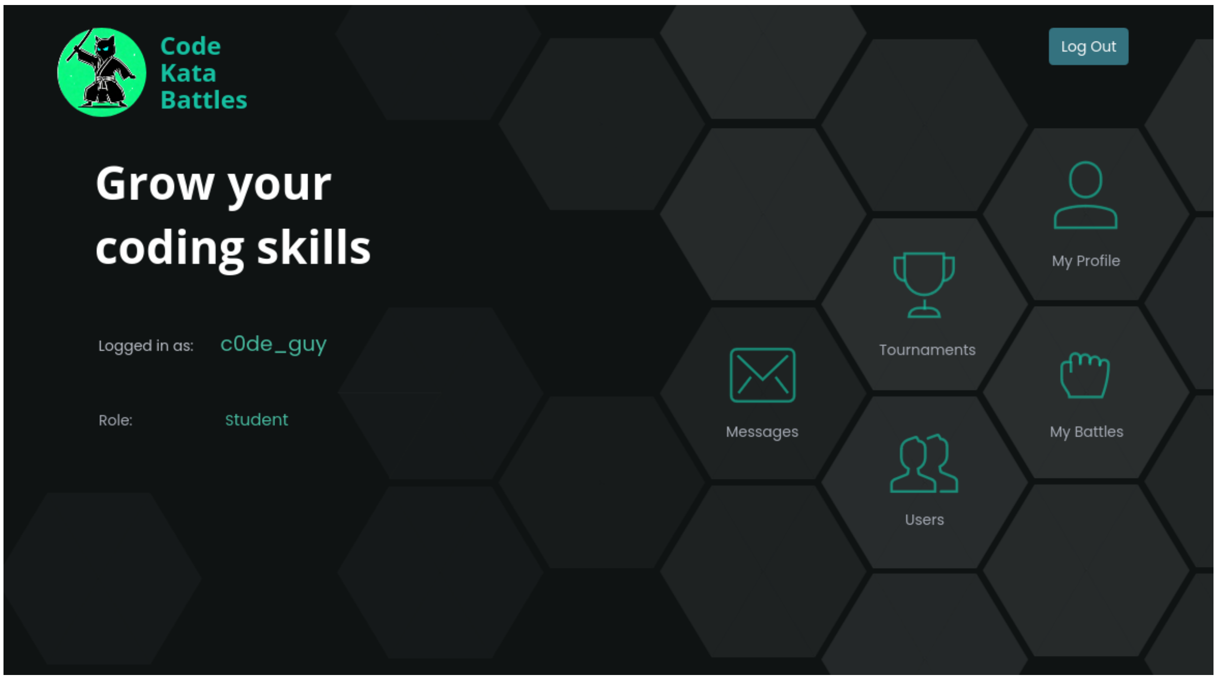


Figure 3.4: CKB Student Homepage

The Student Homepage (Figure 3.4) will show a LogOut button, and 5 menu options:

- **My Profile**: this will redirect the user to their profile page.
- **Tournaments**: this page will show the list of all the tournaments available and the ones the user is currently enrolled in.
- **My Battles**: this page will show the list of all the CodeKataBattles the user is currently enrolled in. This allows the student to jump directly to the page of a battle without having to go through the tournament page.
- **Users**: this page will show the list of all the Students enrolled in the platform. Selecting a user from the list in this page opens the profile page of the selected user, showing Badges earned.
- **Messages**: this page will show the list of all the messages received by the user, including Team Invitations and notifications about new Tournaments or Battles or the end of them.

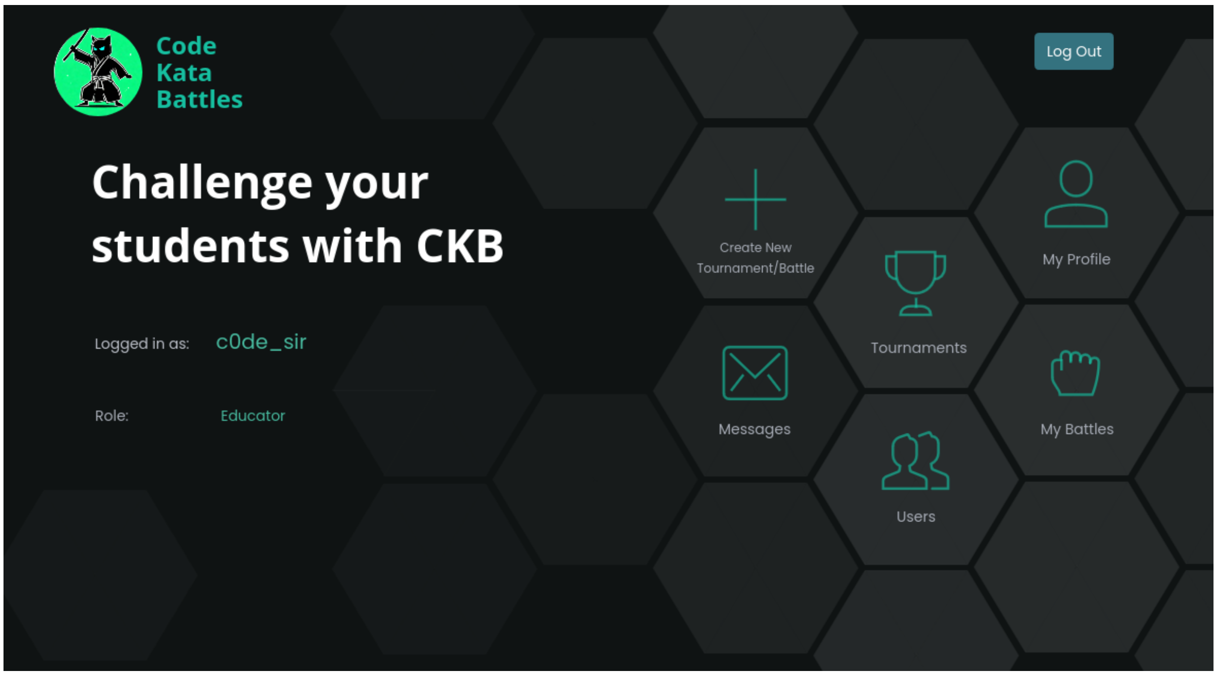


Figure 3.5: CKB Educator Homepage

The Educator Homepage (Figure 3.5) will show a LogOut button, and 6 menu options:

- **My Profile**: this will redirect the user to their profile page.
- **Tournaments**: this page will show the list of all the tournaments available.
- **My Battles**: this page will show the list of all the CodeKataBattles the Educator is currently administrating. This allows the educator to jump directly to the page of a battle without having to go through the tournament page.
- **Users**: this page will show the list of all the Students enrolled in the platform. Selecting a user from the list in this page opens the profile page of the selected user, showing Badges earned.
- **Messages**: this page will show the list of all the messages received by the user, including the start of a Battle administrated by the Educator.
- **Create new Tournament/Battle**: this page will allow the Educator to create a new Tournament/Battle.

3.4. Profile Page

After SignUp the user will be redirected to the Profile Creation page (Figure 3.6), where they will be asked to provide a unique personal username to be used in the CKB platform.

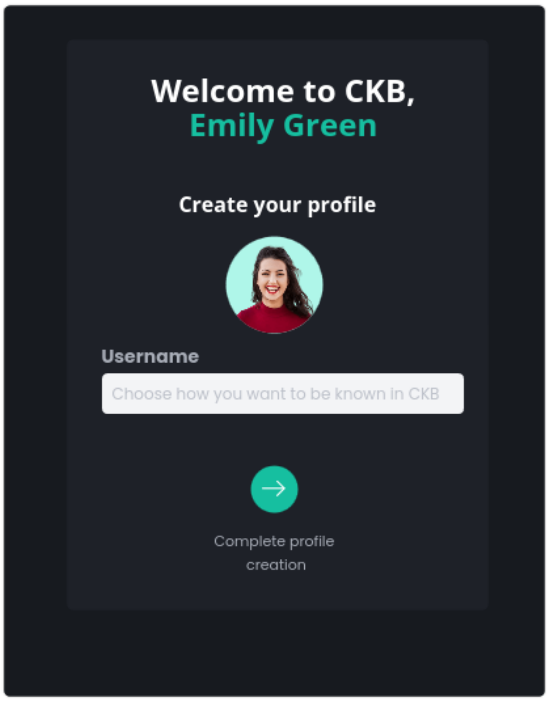

Welcome to CKB,
Emily Green
Create your profile
Username
Choose how you want to be known in CKB
Complete profile creation

Figure 3.6: Profile creation page

The Profile Page (Figure 3.7) will show the user's profile picture, the username, their real name and surname and the list of Badges earned. Clicking on a Badge will open a popup with a description of the Badge, describing how it was achieved.

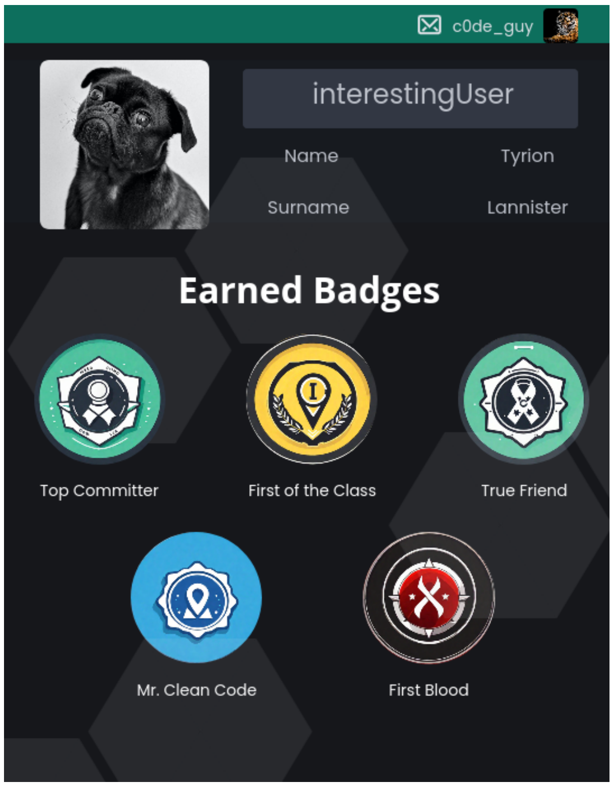
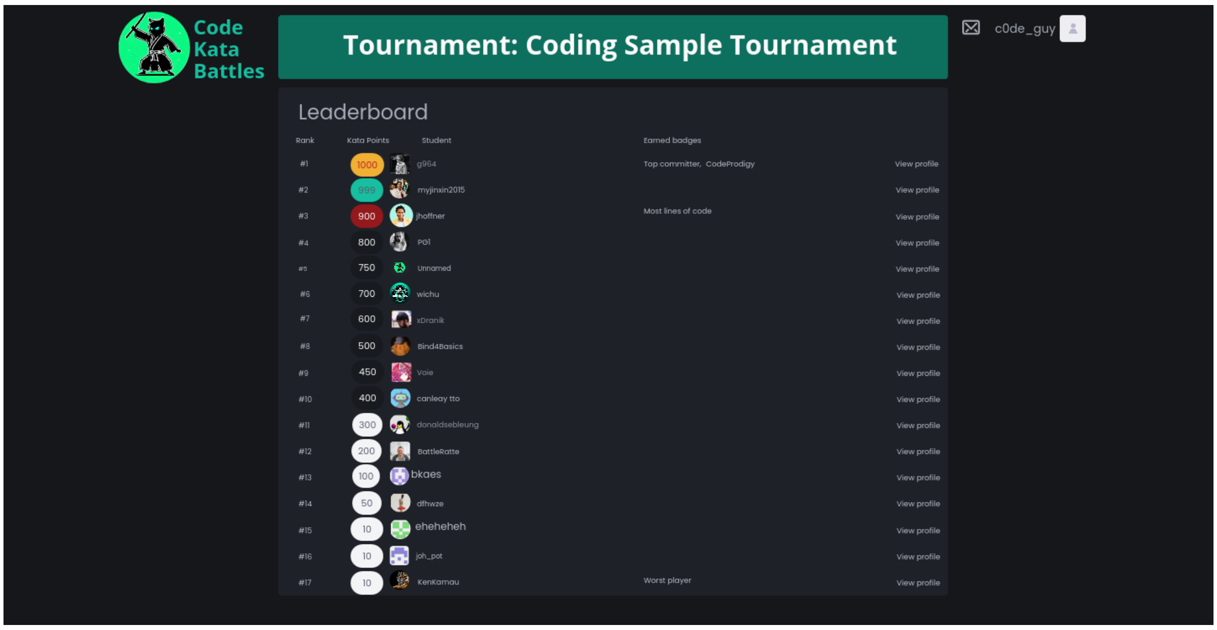


Figure 3.7: CKB profile page

3.5. Tournaments and Battles

Tournaments and Battles pages are simply a list of all the Tournaments/Battles available. Clicking on a Tournament/Battle will open the Tournament/Battle page. No mockups are provided for these pages as they are very simple.

3.5.1. Leaderboards



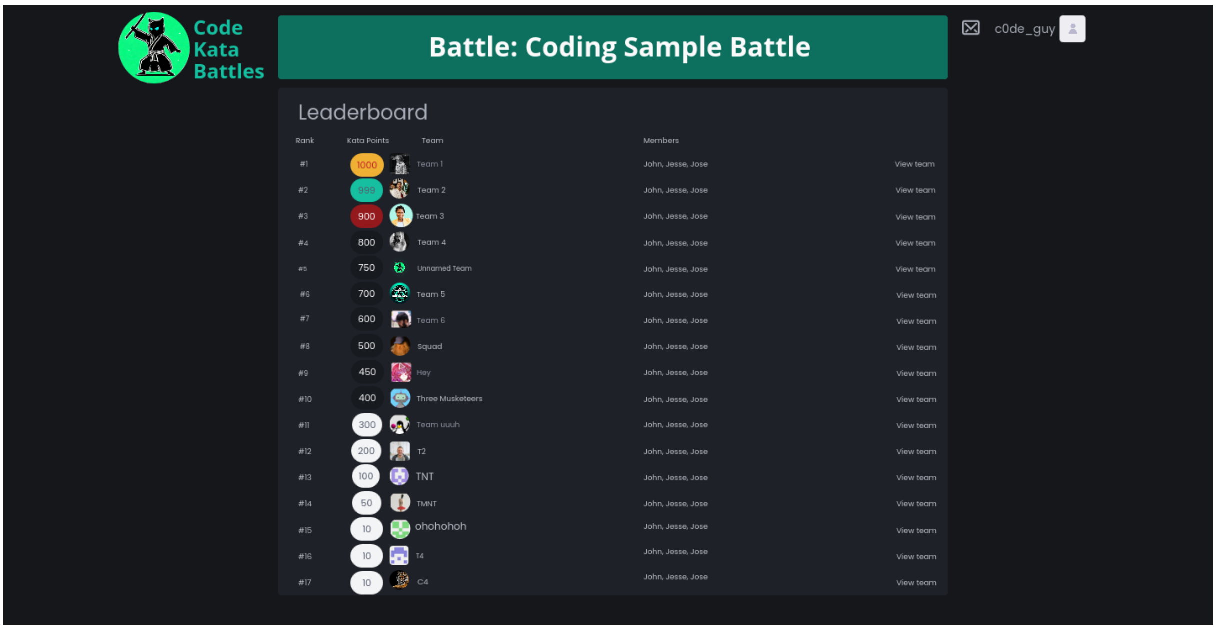
Tournament: Coding Sample Tournament

Leaderboard

Rank	Kata Points	Student	Earned badges	
#1	1000	g964	Top committer, CodeProdigy	View profile
#2	999	myjinxin2015		View profile
#3	900	jhoffner	Most lines of code	View profile
#4	800	PG1		View profile
#5	750	Unnamed		View profile
#6	700	wichu		View profile
#7	600	xDranik		View profile
#8	500	Bind4Basics		View profile
#9	450	Voie		View profile
#10	400	canleay tto		View profile
#11	300	donaldsebleung		View profile
#12	200	BattleRatte		View profile
#13	100	bkaes		View profile
#14	50	dfhwze		View profile
#15	10	eheheheh		View profile
#16	10	joh_pot		View profile
#17	10	KenKamau	Worst player	View profile

Figure 3.8: Tournament leaderboard page

The Tournament Leaderboard page (Figure 3.8) will show the ranking of all the students enrolled in the tournament, in descending order. The Students who earned more points appear upper in the list. The points are calculated as the sum of the points earned in each battle of the tournament. Each row of the leaderboard will show a Student's ranking (position), their score, their username, the badges they earned in the tournament (if any) and a button to view their profile. Clicking on 'View Profile' will open the profile page of the selected user.



Battle: Coding Sample Battle

Leaderboard

Rank	Kata Points	Team	Members	
#1	1000	Team 1	John, Jesse, Jose	View team
#2	999	Team 2	John, Jesse, Jose	View team
#3	900	Team 3	John, Jesse, Jose	View team
#4	800	Team 4	John, Jesse, Jose	View team
#5	750	Unnamed Team	John, Jesse, Jose	View team
#6	700	Team 5	John, Jesse, Jose	View team
#7	600	Team 6	John, Jesse, Jose	View team
#8	500	Squad	John, Jesse, Jose	View team
#9	450	Hey	John, Jesse, Jose	View team
#10	400	Three Musketeers	John, Jesse, Jose	View team
#11	300	Team uuuh	John, Jesse, Jose	View team
#12	200	T2	John, Jesse, Jose	View team
#13	100	TNT	John, Jesse, Jose	View team
#14	50	TMNT	John, Jesse, Jose	View team
#15	10	ohohohoh	John, Jesse, Jose	View team
#16	10	T4	John, Jesse, Jose	View team
#17	10	C4	John, Jesse, Jose	View team

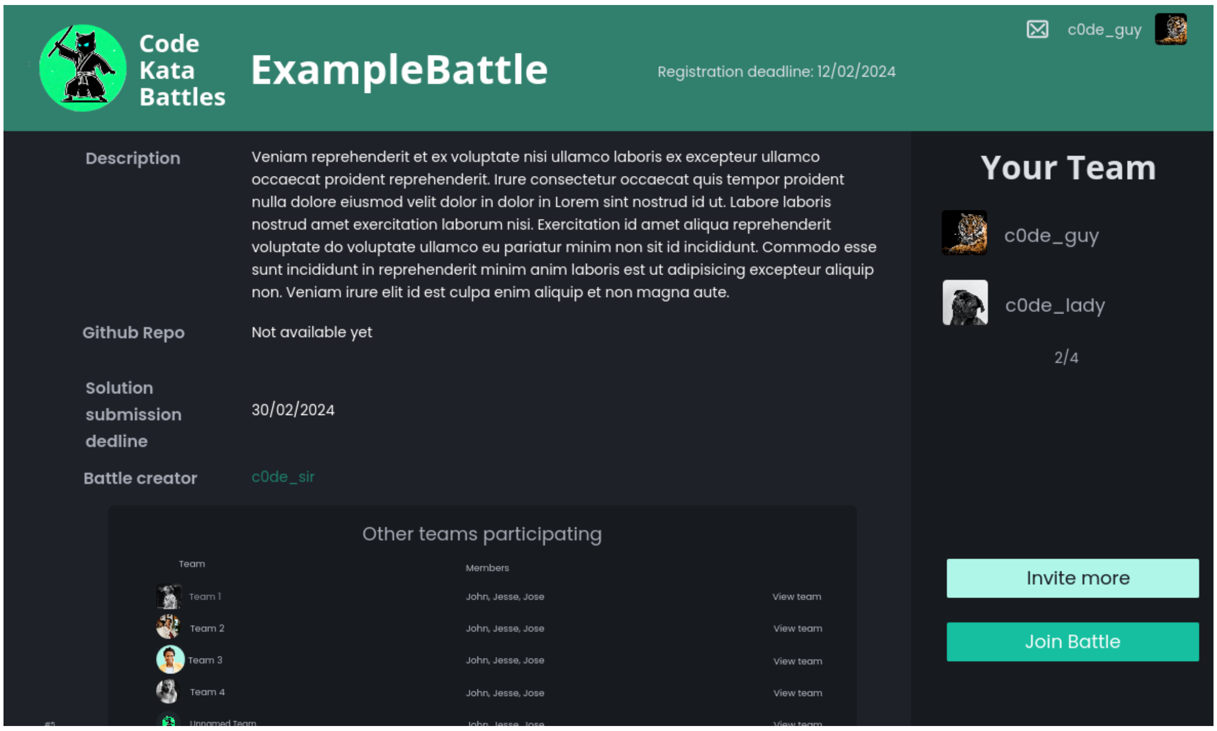
Figure 3.9: Battle leaderboard page

The Battle Leaderboard page (Figure 3.9) will show the ranking of all the Teams enrolled in the battle, in descending order. The Teams who earned more points appear upper in the list. Each row of the leaderboard will show a Team's ranking (position), their score, their name, the usernames of the members of the team and a button to view the Team Page. Clicking on 'View Team' will open the Team page of the selected team. No mockups are provided for the Team page as it is very simple: a list of the members of the team along with the public info of the members.

Battle Page from both Educators and Students' perspective

The `Battle Page` will be different depending on the role of the user.

A Student will see the description of the battle, the link of the GitHub repository they are supposed to work on, the list of the other Teams enrolled in the battle, the username of the Educator who created the battle, the submission deadline for solutions and the registration deadline. In case the Battle has already started, the list of Teams participating will be replaced by the leaderboard of the Battle. The GitHub link will be visible only after the Battle starts. A sidebar on the right shows the Students belonging to the Team of the logged in user. In case the user and their team are not enrolled in the Battle, the sidebar will show a button to enroll in the Battle and a button to invite other Students to join the Team. Minimum and maximum Team capacity are shown in the sidebar. If the team is composed of less than the minimum number of students, the `Join Battle` button will be disabled. If the team is composed of the maximum number of students, the `Invite more` button will be disabled.



ExampleBattle

Registration deadline: 12/02/2024

Description	Veniam reprehenderit et ex voluptate nisi ullamco laboris ex excepteur ullamco occaecat proident reprehenderit. Irure consectetur occaecat quis tempor proident nulla dolore eiusmod velit dolor in dolor in Lorem sint nostrud id ut. Labore laboris nostrud amet exercitation laborum nisi. Exercitation id amet aliqua reprehenderit voluptate do voluptate ullamco eu pariatur minim non sit id incididunt. Commodo esse sunt incididunt in reprehenderit minim anim laboris est ut adipisicing excepteur aliquip non. Veniam irure elit id est culpa enim aliquip et non magna aute.
Github Repo	Not available yet
Solution submission dedline	30/02/2024
Battle creator	c0de_sir

Your Team

c0de_guy

c0de_lady

2/4

Other teams participating

Team	Members	
Team 1	John, Jesse, Jose	View team
Team 2	John, Jesse, Jose	View team
Team 3	John, Jesse, Jose	View team
Team 4	John, Jesse, Jose	View team
Unnamed Team	John, Jesse, Jose	View team

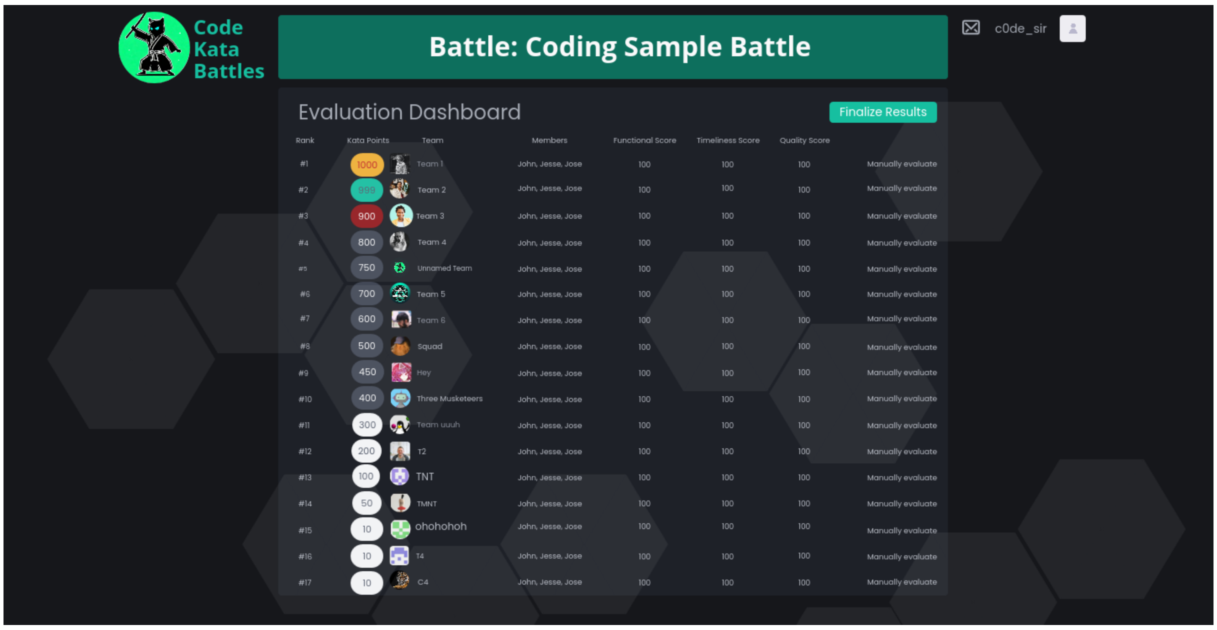
Figure 3.10: Battle page for student

An Educator will see the all the info shown to the Students except for the sidebar. The Educator, after the end of the Battle, will be able to evaluate the solutions submitted by the Teams. The evaluation page is shown in Figure 3.11. An initial score is assigned to each Team (calculated by the automated evaluation system), which can be modified by the Educator. The Leaderboard shows for each row:

- the Team's ranking (position)
- the Team's score
- the Team's name
- the usernames of the members of the team
- the score automatically assigned by the system regarding the Functional Aspects of the solution
- the score automatically assigned by the system regarding the Timeliness (submission time) of the solution

- the score automatically assigned by the system regarding the Quality of Code aspects of the solution
- a button to manually modify the Team's score examining their solution

A button `Finalize Results` allows the Educator to finalize the evaluation of the Battle and make the results available to everyone.



Battle: Coding Sample Battle

Evaluation Dashboard

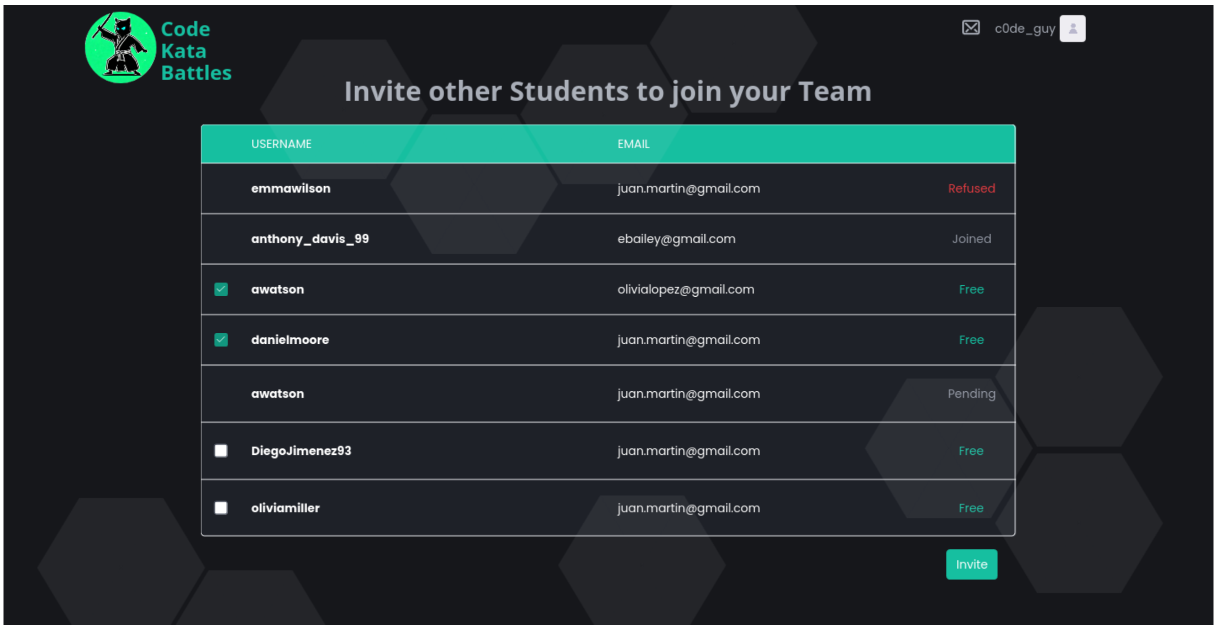
Finalize Results

Rank	Kata Points	Team	Members	Functional Score	Timeliness Score	Quality Score	
#1	1000	Team 1	John, Jesse, Jose	100	100	100	Manually evaluate
#2	999	Team 2	John, Jesse, Jose	100	100	100	Manually evaluate
#3	900	Team 3	John, Jesse, Jose	100	100	100	Manually evaluate
#4	800	Team 4	John, Jesse, Jose	100	100	100	Manually evaluate
#5	750	Unnamed Team	John, Jesse, Jose	100	100	100	Manually evaluate
#6	700	Team 5	John, Jesse, Jose	100	100	100	Manually evaluate
#7	600	Team 6	John, Jesse, Jose	100	100	100	Manually evaluate
#8	500	Squad	John, Jesse, Jose	100	100	100	Manually evaluate
#9	450	Hey	John, Jesse, Jose	100	100	100	Manually evaluate
#10	400	Three Musketeers	John, Jesse, Jose	100	100	100	Manually evaluate
#11	300	Team uuuh	John, Jesse, Jose	100	100	100	Manually evaluate
#12	200	T2	John, Jesse, Jose	100	100	100	Manually evaluate
#13	100	TNT	John, Jesse, Jose	100	100	100	Manually evaluate
#14	50	TMNT	John, Jesse, Jose	100	100	100	Manually evaluate
#15	10	ohohohoh	John, Jesse, Jose	100	100	100	Manually evaluate
#16	10	T4	John, Jesse, Jose	100	100	100	Manually evaluate
#17	10	C4	John, Jesse, Jose	100	100	100	Manually evaluate

Figure 3.11: CKB Battle Evaluation Dashboard

3.6. Teams and Invitations

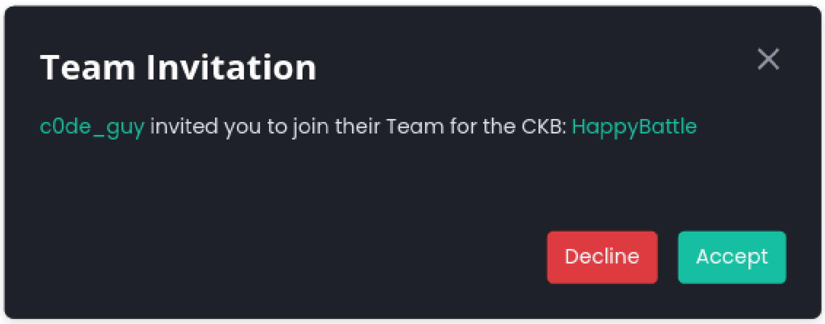
Clicking on the `Invite more` button in the Battle page (Figure 3.10) will open the `Invite other Students` page (Figure 3.12). The page will show a list of all the Students enrolled in the Tournament the Battle belongs to who are not already enrolled in the Battle with a Team. The user can select one or more Students from the list and click on the `Invite` button to send them an invitation to join the Team. The list of Students show their username, their email and the state of the invitation (Refused/Free/Joined/Pending). Clicking on the `Invite` button will send invitations to the selected Students.



	USERNAME	EMAIL	
	emmawilson	juan.martin@gmail.com	Refused
	anthony_davis_99	ebailey@gmail.com	Joined
✓	awatson	olivialopez@gmail.com	Free
✓	danielmoore	juan.martin@gmail.com	Free
	awatson	juan.martin@gmail.com	Pending
☐	DiegoJimenez93	juan.martin@gmail.com	Free
☐	oliviamiller	juan.martin@gmail.com	Free

Figure 3.12: Invite other students page

A Student who receives an invitation will see a notification in the `Messages` page. Opening the notification will show the invitation popup (Figure 3.13). The popup will show the name of the Student that sent the invitation, the name of the Battle. Clicking on the name of the Battle will open the Battle page, allowing the user to develop an informed decision about accepting or refusing the invitation. Clicking on the `Accept` button will enroll the user in the Battle with the Team that sent the invitation. Clicking on the `Decline` button will refuse the invitation.



c0de_guy invited you to join their Team for the CKB: HappyBattle

Figure 3.13: Invitation popup

4 | Requirements Traceability

This chapter shows how the functional and non-functional requirements of the CKB platform described in the RASD are met. Firstly, we describe the mapping between the requirements listed in the 3.2.1 *Requirements* section in the RASD and the components identified in section 2.2 of this document. Then, we comment on how *Performance Requirements* and *Software System Attributes* described in the 3.3 and 3.5 sections of the RASD are met thanks to the design adopted for the CKB platform.

4.1. Functional Requirements Traceability

The following table lists all the requirements that are satisfied by the components. Texts of the requirements are inserted in the table to facilitate comprehension.

UserManager	R1. The CKB platform shall allow educators to create an account. R2. The CKB platform shall allow educators to log in.
Educator Dashboard Manager	R3. The CKB platform shall allow educators to create a new tournament. R4. The CKB platform shall allow educators to set the minimum and maximum number of students per group for a tournament. R5. The CKB platform shall allow educators to upload a code kata battle. R6. The CKB platform shall allow educators to grant permissions to other educators to create battles within a specific tournament. R7. The CKB platform shall enable educators to include a battle to a specific tournament. R8. The CKB platform shall allow educators to include a description for a battle.

	R9. The CKB platform shall allow educators to include a software project with test cases. R10. The CKB platform shall allow educators to set a registration deadline for a battle within a tournament. R11. The CKB platform shall allow educators to set a final submission deadline for a battle within a tournament. R12. The CKB platform shall allow educators to set additional configurations for scoring, including functional aspects and quality level criteria. R13. The CKB platform shall allow educators to close a tournament. R14. The CKB platform shall allow an educator to define optional manual evaluation criteria for score assignment in battles. R15. The CKB platform shall allow educators to manually evaluate and assign scores to teams. R17. The CKB platform shall allow educators to define new badges for gamification. R18. The CKB platform shall allow educators to define new rules associated with the badges. R19. The CKB platform shall allow educators to define new variables associated with the badges.
Badge Manager	R17. The CKB platform shall allow educators to define new badges for gamification. R18. The CKB platform shall allow educators to define new rules associated with the badges. R19. The CKB platform shall allow educators to define new variables associated with the badges.
Gamification Rules Manager	R18. The CKB platform shall allow educators to define new rules associated with the badges. R19. The CKB platform shall allow educators to define new variables associated with the badges.
Student Dashboard Manager	R21. The CKB platform shall not allow students to participate in a tournament after the registration deadline. R22. The CKB platform shall not allow students to participate in a battle after the registration deadline.

	R23. The CKB platform shall allow students to subscribe to a tournament within a specified deadline. R24. The CKB platform shall allow students to create teams for a specific battle within a tournament. R25. The CKB platform shall allow students to join teams for a specific battle within a tournament. R26. The CKB platform shall allow students to invite other participants to the same group. R27. The CKB platform shall allow students to accept an invitation. R28. The CKB platform shall allow students to reject an invitation. R29. The CKB platform shall allow students to join a battle without a team.
Battle Manager	R3. The CKB platform shall allow educators to create a new tournament. R5. The CKB platform shall allow educators to upload a code kata battle. R7. The CKB platform shall enable educators to include a battle to a specific tournament. R8. The CKB platform shall allow educators to include a description for a battle. R9. The CKB platform shall allow educators to include a software project with test cases.
Tournament Manager	R3. The CKB platform shall allow educators to create a new tournament. R4. The CKB platform shall allow educators to set the minimum and maximum number of students per group for a tournament. R10. The CKB platform shall allow educators to set a registration deadline for a battle within a tournament. R11. The CKB platform shall allow educators to set a final submission deadline for a battle within a tournament. R12. The CKB platform shall allow educators to set additional configurations for scoring, including functional aspects and quality level criteria.

	R13. The CKB platform shall allow educators to close a tournament. R20. The CKB platform shall allow all students and educators to see the ranking of each ongoing tournament with the score of each student subscribed.
Automated Evaluation Manager	R36. The CKB platform shall be able to run the test cases on the code uploaded by students and determine if the code is a valid solution for the exercise. R37. The CKB platform shall inform students of the mandatory automated evaluation criteria, including functional aspects, timeliness, and source code quality. R38. The CKB platform shall automatically update the battle score of a team based on GitHub commits and test results. R39. The CKB platform shall automatically close a finished battle. R40. The CKB platform shall assign or update battle scores to each team of the battle. R41. The CKB platform shall calculate and update the personal tournament score of each student based on their performance in battles. R42. The CKB platform shall be able to create a ranking of teams for every tournament. R44. The CKB platform shall be able to use static analysis tools to evaluate the quality of the code submitted by teams in terms of security, reliability, maintainability and other aspects defined by the educator who created the battle.
Team Manager	R24. The CKB platform shall allow students to create teams for a specific battle within a tournament. R25. The CKB platform shall allow students to join teams for a specific battle within a tournament. R26. The CKB platform shall allow students to invite other participants to the same group.
Invitation Manager	R26. The CKB platform shall allow students to invite other participants to the same group. R27. The CKB platform shall allow students to accept an invitation.

	R28. The CKB platform shall allow students to reject an invitation.
CKB Notification System	R30. The CKB platform shall notify all subscribed students of a new battle and its details within a specific tournament. R31. The CKB platform shall notify all subscribed students of a new tournament and its details. R45. The CKB platform shall notify all students involved in a tournament when it is closed and the final ranking is available.

Table 4.1: Mapping between components and requirements.

4.2. Non Functional Requirements Traceability

4.2.1. Performance Requirements

Performance Requirement	**Architectural Component**
Response Time (2s avg)	Presentation Tier, Logic Tier, WebSocket with Pub/Sub Pattern
Concurrent Users (1000 users)	Logic Tier, Data Tier, Worker Pooling, Reverse Proxy
GitHub Integration (5 mins)	Logic Tier, Microservices
Automated Evaluation (1 min)	Automated Evaluation Manager, Microservices
Data Retrieval Time (3s)	Data Tier, Database (NoSQL with SQL)
Consolidation Stage (3 days)	Logic Tier, Educator Dashboard Manager
Badge Assignment (1 day)	Badge Manager, Gamification Rules Manager, Factory Pattern
Platform Uptime (99.9%)	All Tiers, Microservices Architecture
Notification Latency (1 min)	CKB Notification System, WebSocket with Pub/Sub Pattern
Gamification Features (3s)	Gamification Rules Manager, Decorator Pattern

Table 4.2: Traceability Matrix for Performance Requirements

4.2.2. Software System Attributes

System Attribute	**Architectural Component/Design Choice**
Reliability	Automated Evaluation Manager, Microservices Architecture, Worker Pooling
Availability	All Tiers (Presentation, Logic, Data), Reverse Proxy, Cloud Hosting/Infrastructure
Security	HTTPS Implementation, Encryption of Stored Data, Firewall and Reverse Proxy
Maintainability	Modular Architecture, Clear Documentation, MVC Pattern
Portability	Cross-Platform Web and Mobile Accessibility, Separate Implementations for Android and iOS

Table 4.3: Traceability Matrix for Software System Attributes

5 | Implementation, Integration and Test Plan

5.1. Implementation

The implementation process begins with setting up the databases. MongoDB, a NoSQL database, is chosen for its flexibility and scalability, making it ideal for storing diverse data types such as users, educators, tournament results, badges, notifications, and more. This database will be designed with a schema-less structure, allowing for easy modifications as the data requirements evolve over time.
Concurrently, MySQL, a SQL database, is used for its strength in handling structured data and complex queries. It will store data related to tournaments, teams, and battles. Given the nature of these data types, they are likely to involve numerous queries, especially during an ongoing tournament. The structured nature of SQL databases allows for efficient querying and retrieval of data, ensuring smooth operation during high-load scenarios.
Once the databases are set up, the backend development begins. This involves creating core functionalities such as user manager, battle manager, tournament manager, badge manager, and the notification system. Each of these components is crucial for the functioning of CKB and will involve interactions with the databases. For instance, user manager will handle user registration, login, and profile management, interacting primarily with the MongoDB database. Battle manager, on the other hand, will involve interaction with the MySQL database. The order in which each components needs to be developed is not a strict one. Here, we present a suggestion on the order.
We first suggest creating the user manager, since this will allows us to immediately test subscription and login, as well as interaction with MongoDB. Then, it would be auspicious to develop the Tournament manager. This entity relies on the existence of an Educator to create and manage the Tournament. Therefore, this is purely a logical step in the development of the back end. Once the Tournament manager is properly functioning, we can focus on the production of the Battle manager and Team manager, two entities

strictly related to the Tournament Manager. Then, all the other aspects can be creating without a specific order, since the core of the logic of the backend is already up.
Last steps for the backend development will be the integration of the CKB platform with all the external tools required to achieve the desired implementation. This includes integrating the notification system with the mail system, the communication with GitHub, and the implementation with Google Spreadsheets and Universities Authentication.
Following the completion of the backend, the focus shifts to frontend development. This includes designing and implementing the user interface for various operations such as viewing and joining battles, viewing scores, user registration, login, and profile management. Additionally, real-time updates using WebSocket will be set up for live scoring and notifications of new battles or tournaments. This feature is crucial for enhancing the user experience by providing immediate feedback and updates.
To ensure the security of the application, a firewall is set up to protect the servers from potential cyber attacks and unauthorized access. The firewall is configured to allow WebSocket traffic. In addition, a reverse proxy is set up for load balancing, SSL termination, and simplified network routing. These measures enhance the security and performance of the application.
Throughout the implementation process, careful management is required to handle data synchronization and maintain data integrity across both databases. A reliable mechanism is implemented to keep the databases updated and consistent with each other. This comprehensive implementation plan provides a robust and adaptable data management solution for CKB, ensuring its efficiency and reliability.

5.2. Integration

The integration phase is a critical step in the development process of CodeKataBattle (CKB), where the individual components of the system, developed during the implementation phase, are combined and tested as a whole.
The integration process begins with the combination of the frontend and backend components of CKB. This involves connecting the backend with the user interface elements in the frontend. The WebSocket connection between the client and the server, set up during the implementation phase, is also integrated into the system to enable real-time updates.
Next, the databases are integrated with the backend. This involves setting up the database connections and ensuring that the backend can correctly query the databases and handle the returned data. The MongoDB database, which stores non-relational data such as users, educators, tournament results, badges, notifications, and more, is integrated with the user management, badge management, and notification system components of the

backend. The MySQL database, which stores relational data related to tournaments, teams, and battles, is integrated with the battle management and tournament management components of the backend.
The firewall and reverse proxy, set up during the implementation phase for security and performance enhancement, are also integrated into the system. The firewall is configured to allow WebSocket traffic, which is essential for the real-time updates in CKB. The reverse proxy is set up for load balancing, SSL termination, and simplified network routing.
The final step in the integration process is the integration of the various components of the backend. This includes integrating the user management, battle management, tournament management, badge management, and notification system components with each other. This ensures that these components can work together seamlessly to provide the core functionalities of CKB.

5.3. Testing

5.3.1. Unit Testing

Unit testing is a critical process in software development, where individual components of a program are tested in isolation to ensure they function correctly. This method focuses on the smallest units of code, like functions or methods, to identify issues early, enhancing overall software quality. This type of test will be performed in the early stage of the development. In particular, during the implementation of each manager, unit testing will be required in order to ensure that each component as a standalone works properly.

5.3.2. Integration Testing

Integration testing is a level of software testing where individual units are combined and tested as a group. The goal is to identify any problems or bugs that arise when different components are combined and interact with each other.
The integration testing can follow the order of the implementation of the back end logic, and be conducted in parallel with it (this means implementation, integration and testing steps for the back end are done simultaneously). This will ensure that everything works properly and as intended. If an error arises at this point of the testing, it will be easier to spot and correct it, instead of waiting for testing after the end of the implementation and integration.

5.3.3. System Testing

System testing validates the complete and fully integrated software product. The purpose is to evaluate the end-to-end system specifications. System testing will require to verify every input in the application to check for desired outputs, users' experience and integration as a whole including external services

5.3.4. Performance Testing

Performance testing ensures software applications perform properly under their expected workload. It focuses on evaluating the performance and scalability of a system. The goal is to identify bottlenecks, measure system performance under various loads and conditions, and ensure the system can handle the expected number of users or transactions. In particular, we want to make sure in this phase that even under stress, they system can still provide real time updates to the users, and that no data is lost.

6 | Effort Spent

Member of group	Effort spent	
	Introduction	$4h$
	Architectural Design	$10h$
	User Interface Design	$17h$
Federico Zanca	Requirements Traceability	$1h$
	Implementation, integration and test plan	$0h$
	Introduction	$3h$
	Architectural Design	$12h$
	User Interface Design	$2h$
Federico Costantini	Requirements Traceability	$2h$
	Implementation, integration and test plan	$10h$
	Introduction	$1h$
	Architectural Design	$15h$
	User Interface Design	$2h$
Michele Zhenghao Zhuge	Requirements Traceability	$10h$
	Implementation, integration and test plan	$2h$

Table 6.1: Effort spent by each member of the group.

7 | References

7.1. Paper References

- The specification document Assignment RDD AY 2023-2024.pdf
- RASD document

7.2. Used Tools

- GitHub for project versioning
- StarUML for UML diagrams
- DrawIo for UML diagrams
- Notion for notes and project organization between team members
- VSCode as LaTeX editor
- Visily for UI/UX mockups
- Hotpot for CKB Logo in UI/UX mockups

List of Figures

List of Tables