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

CKB - CodeKataBattle

DD
Design Document

January 7, 2024

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

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

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

1.2 Scope

The CodeKataBattle system is designed with the primary objective of providing a comprehensive platform for students to hone their programming skills through the collaborative resolution of coding exercises. The system follows the principles of *codekata*, emphasizing the iterative practice of problem-solving for effective learning.
In addition to catering to students, the system accommodates a second user category: educators. Educators are empowered to oversee various aspects of the tournaments they create, wielding a range of functionalities. This includes the creation of battles, the assignment of problems to students, and the evaluation of the projects submitted by participants.
Each battle within the system is defined by specific deadlines, imposing a structured timeline on student endeavors. Participants are required to diligently work on their projects, ensuring compliance with the technical specifications set for their respective workspaces. Furthermore, the platform extends its utility by offering features such as the visualization of tournament and battle ranks, as well as the opportunity for students to earn badges upon completion of a tournament.
For a more detailed exploration of these features, refer to the Requirements Analysis and Specification Document (RASD).

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

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

1.3.2 Acronyms

- **CKB**: CodeKataBattle
- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **UML**: Unified Modelling Language
- **UI**: User Interface item **REST**: Representational state transfer
- **DBMS** - database management system
- **API** - Application Programming Interface
- **ER** - entity-relationship
- **HTTP** - hypertext transfer protocol
- **SPA** - Single Page Application
- **JSON** - JavaScript Object Notation
- **XML** - eXtensible Markup Language

1.3.3 Abbreviations

- [Rn] - the n-th functional requirement
- [Fn] - the n-th feature
- S2B - software to be

1.4 Revision History

- Version 1.0 (07/01/2024)

1.5 Reference Documents

This document is based on:

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

1.6 Document Structure

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

- **Introduction**: this chapter explains what are the purpose and the scope of the project, adding some technical information such as definitions and acronyms of terms used in the document and references.
- **Architectural Design**: it introduces the main architectural choices made for the system. This section includes an overview of the components and interfaces to communicate, a description of the infrastructure and also diagrams to represent both static and runtime views.
- **User Interface Design**: this chapter contains the mock-ups both for educator and student interfaces, including a brief description of the main functionalities provided by the interface.
- **Requirement Traceability**: this section shows the links between requirements (previously defined in RASD) and components described in the previous chapters.
- **Implementation, Integration and Test Plan**: this chapter describes the correct order to deploy the system, starting from the subsystems and components implementation, then integrating and testing.
- **Effort spent**: this section shows the time spent (in hours) by each member to work on this document.
- **References**: it contains the references to any documents and to the Software used in this document.

2 Architectural Design

The purpose of this section is to present and analyze the architecture of the software to be system in a top-down manner. We first introduced the overall architecture and then provided a diagram of the system's components, focusing on the tournaments and battles sub-components. Next, we used an ER diagram to describe the system's logical data and presented the system's deployment view, including the layers and tiers involved. We also used sequence diagrams to depict important runtime views and class diagrams to analyze the component interfaces. Finally, we discussed the architectural design choices and the reasons behind them.

2.1 Overview

The figure shown below represents a high-level description of the components which make up the System. It is a distributed system with a 4-tier architecture: presentation, web server, application server and database. In this document, the presentation layer and the Client will be referred to as the Frontend, while the Application Layer and the Data Layer will be referred to as the Backend.

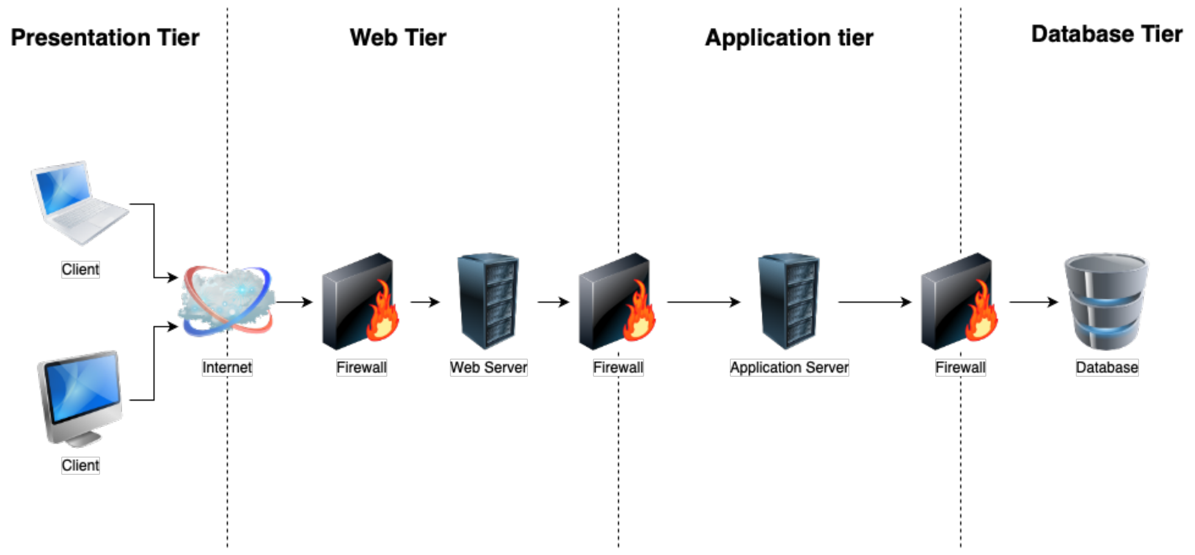


Figure 1: High level system architecture

The presentation tier is the client side, where users interact with the system through a web browser. A single page application will be developed both for educators and students to use the system. The reason behind this choice is the ease of interaction without the need for frequent page reloading, providing a better user experience.
The web server is responsible for communication between the Frontend and the Backend. It handles HTTP requests, routing them to the appropriate components. It supports secure communication with clients through HTTPS, and load balancing ensures optimal performance during periods of high traffic.
The application tier is the logic core of our system. It processes data received from the presentation tier, performs business logic operations, and communicates with the database. It tier ensures data integrity, security, and authentication, with role-based access controls. It interfaces with external services or APIs as needed.
Finally, the data tier is responsible for storing, reading and updating all information necessary for the system. We use a relational database that offers high scalability potential

for structured data. It interfaces with the application tier, granting data access and manipulation while respecting security protocols.
The client web-server and web-server application-server communication use HTTP, while app-server DBMS communication relies on APIs. The app-servers are designed to be stateless according to REST standards. The system also includes firewalls to enhance security.

2.2 Component view

In this section we offer a more detailed view of the S2B. We will focus on the components and their interactions, along with the interfaces. To meet performance and availability criteria, most of said components will be duplicated or replicated.

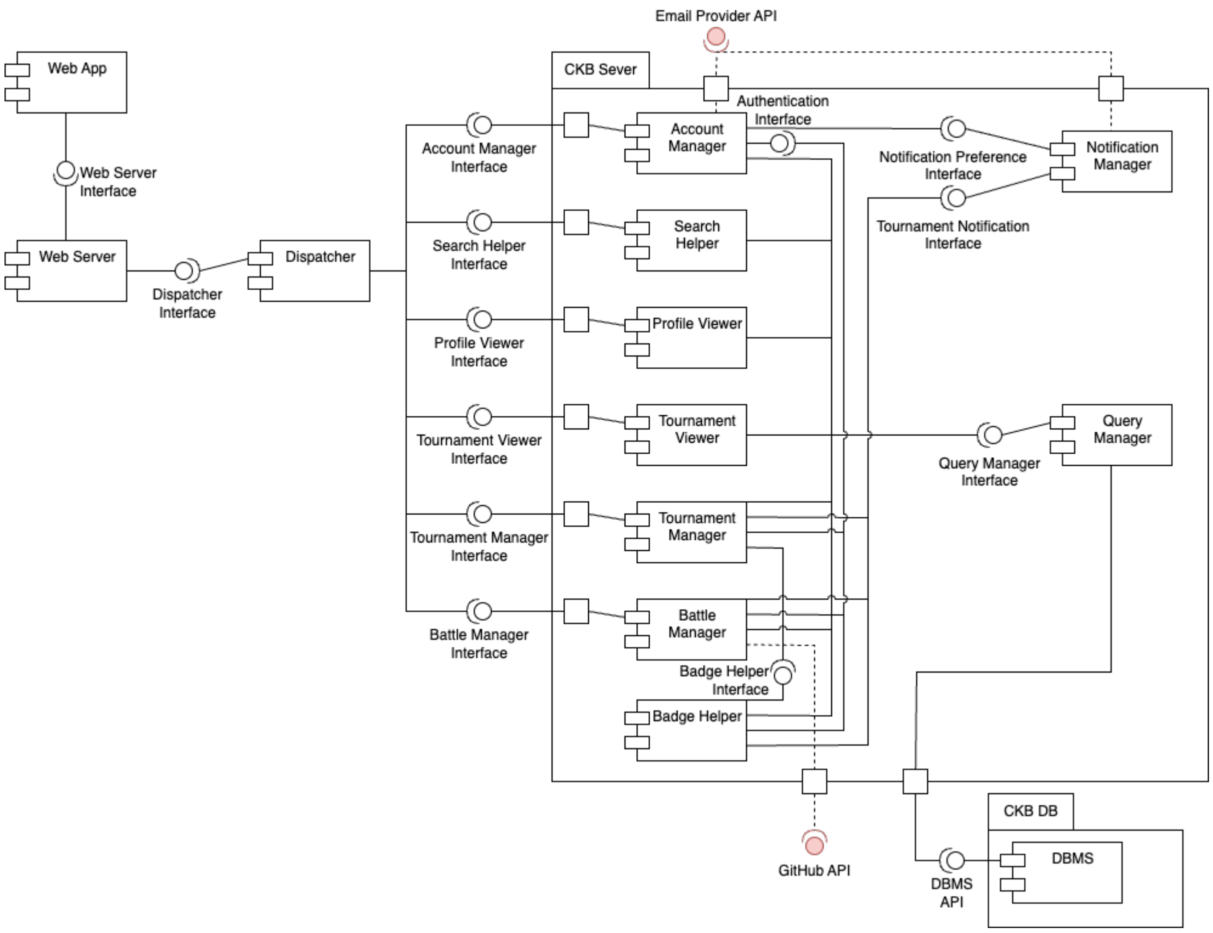


Figure 2: Component Diagram of the CKB System

- **Web server**
  The web server is designed to handle HTTP requests from the client, redirecting them to the appropriate components. It uses client-side rendering. As the user interacts with the page, further requests may be automatically made and the page will be updated dynamically without requiring a full page reload.

- **CKB server**
  The CKB server is responsible for the business logic and provide the full functionality to users. Since most of the functionalities offered are shared between educators and students, there will not be a different server for each user category. It is further composed of different components.

  – **CKB's Account Manager**
    This component handles all the account operations related to the users and offers an interface to authenticate the requests. It offers functionality to create new account, logging in, setting preferences and verify the authentication of the user at any time. To create a new account, It interacts with the external email provider API to make the user receive a code to verify the identity.

  – **Search helper**
    This component enables the search functionality, letting users and non-users search for a specific user or a specific tournament. It also allows filtering based on date posted, number of awards, ongoing/finished, number of participants and much more (tournaments).

  – **Profile viewer**
    This component allows viewing profiles of users subscribed to CKB, both educators and students. Anyone can see profiles, even unsubscribed users. Within the profile page, there may be tournaments the user took part in (as a student or as an educator) and clicking them will redirect that tournament page.

  – **Tournament viewer**
    This component allows viewing tournaments, along with the battle pages in the scope of It. Anyone can see tournaments, even unsubscribed users. You can find all the information related to that tournament, as well as the battles, including teams, final rank for each battle and much more.

  – **Tournament manager**
    This component allows managing and creating tournaments for educators and subscribing to them for students. The creator can grant other educators permission to create battles. For ongoing tournaments, allowed educators can create new battle clicking the right button. Such functionality is provided through the next component.

  – **Battle manager**
    This component allows managing and creating battles for educators and joining them for students. It also manages the team functionality, allowing students to team up for each battle as well as the scores for each team (including manual

evaluation)

- **Badge helper**
  This component enables the gamification aspects of CKB. It allows educators to create badges and define new rules as well as new variables associated with them. Badges are then linked to the profile of users achieving them, making them visible to anyone.

- **CKB Notification Service**
  This component enables the CKB notifications. Users will receive notifications about important events, like new tournaments created, tournament to which the user is subscribed closes and more.

  External APIs

- **GitHub API**
  This API enables the GitHub integration: new repositories automatically created (e.g. when a new battle begins) and sources automatically downloaded (e.g. when a user pushes new code to the main branch of his repository)

- **Email provider API** This service is used for authentication during registration.

- **Query Manager**
  This component is responsible for communicating with the Database Management System. It follows the adapter pattern, allowing easier communication between other components and the DBMS.

2.3 Deployment view

This chapter describes the deployment of the system.

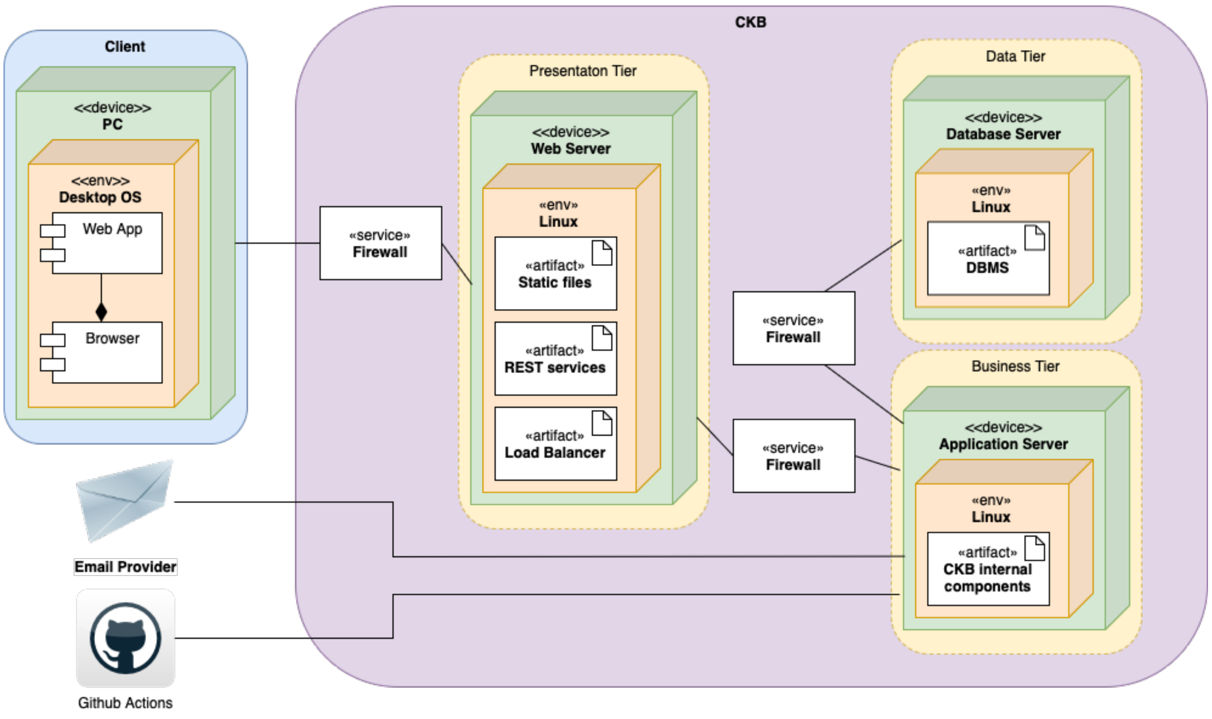


Figure 3: Deployment View of the system

- **Web-server**
  The web-server will be the entry point for the SPA. It forwards the requests to the application-server. A modern device with access to a web browser is required to interface with the web-server.
- **Load Balancer**
  This device is responsible for load balancing, which means, distributing incoming network traffic and requests across multiple servers. Its primary purpose is to optimise resource utilization, enhance reliability while ensuring high availability.
- **Firewall**
  Firewalls are used to filter connections to the application and data layers of a system. They are located between the internet and the system intranet. They offer additional security blocking or allowing traffic based on predetermined rules.

2.4 Runtime View

This section illustrates the interactions between actors, subsystems and interfaces of the system showing the specific method called.

2.4.1 Runtime View Preconditions

- **Web Server views**
  The calls to the web server for retrieve the views of each page are implied and not explicitly shown in the runtime views below. Notice that the Web server will act as a proxy when it has already the requested page.
- **Token Validation**
  Users can only search and view details of tournaments and profiles without first being authenticated and authorized. All other actions that a user (both students and educators) can perform are dependent on the validation of a token. The token is passed in the API request and the process of verifying it is shown in the sequence diagram below.

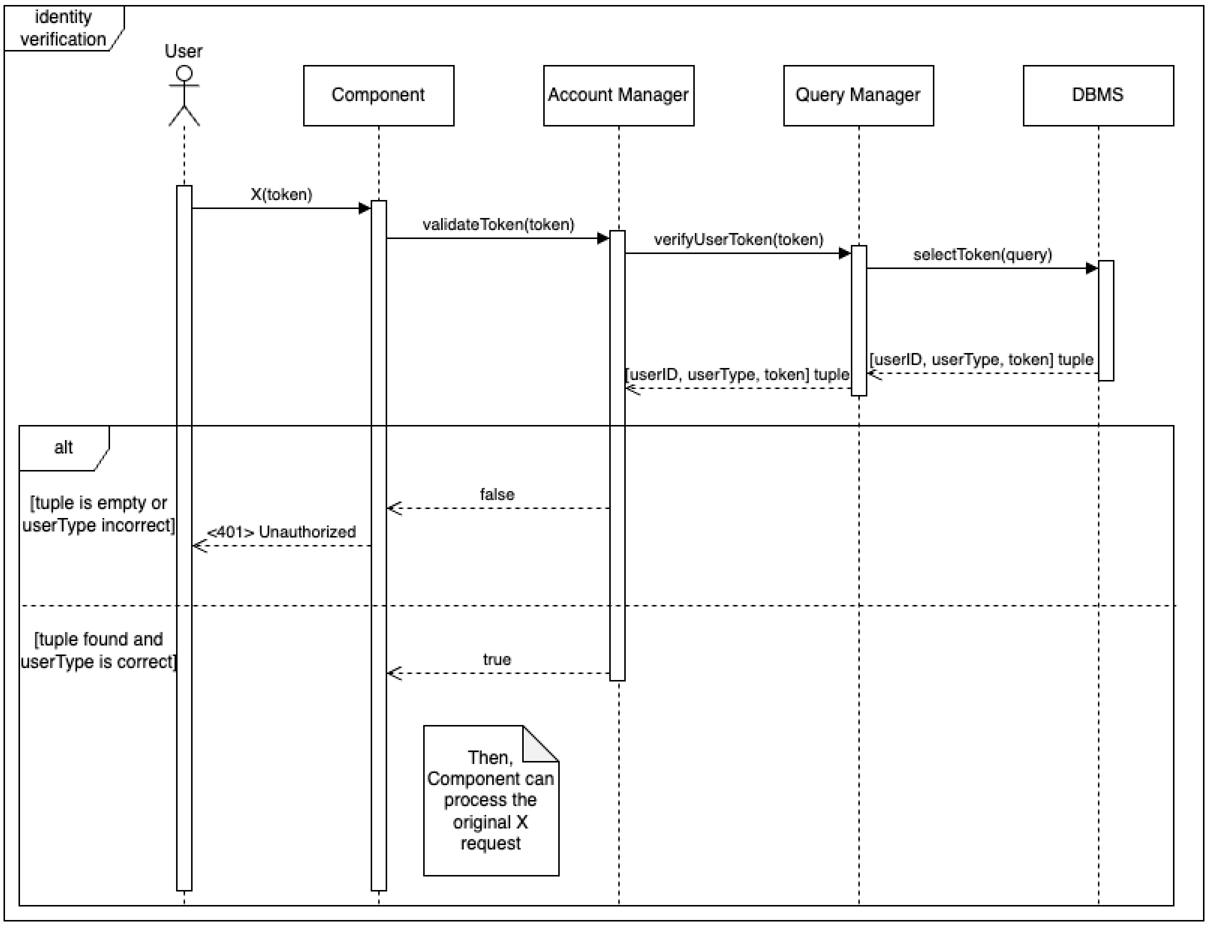

identity verification
User
Component
Account Manager
Query Manager
DBMS
X(token)
validateToken(token)
verifyUserToken(token)
selectToken(query)
[userID, userType, token] tuple
[userID, userType, token] tuple
alt
[tuple is empty or userType incorrect]
false
<401> Unauthorized
[tuple found and userType is correct]
true
Then, Component can process the original X request

Figure 4: Token validation

- **Log in**
  The user must login in order to join or create tournaments and battles. The user enters the credentials (email and password) and, if they are valid, he will be authenticated, otherwise, an error will be displayed.

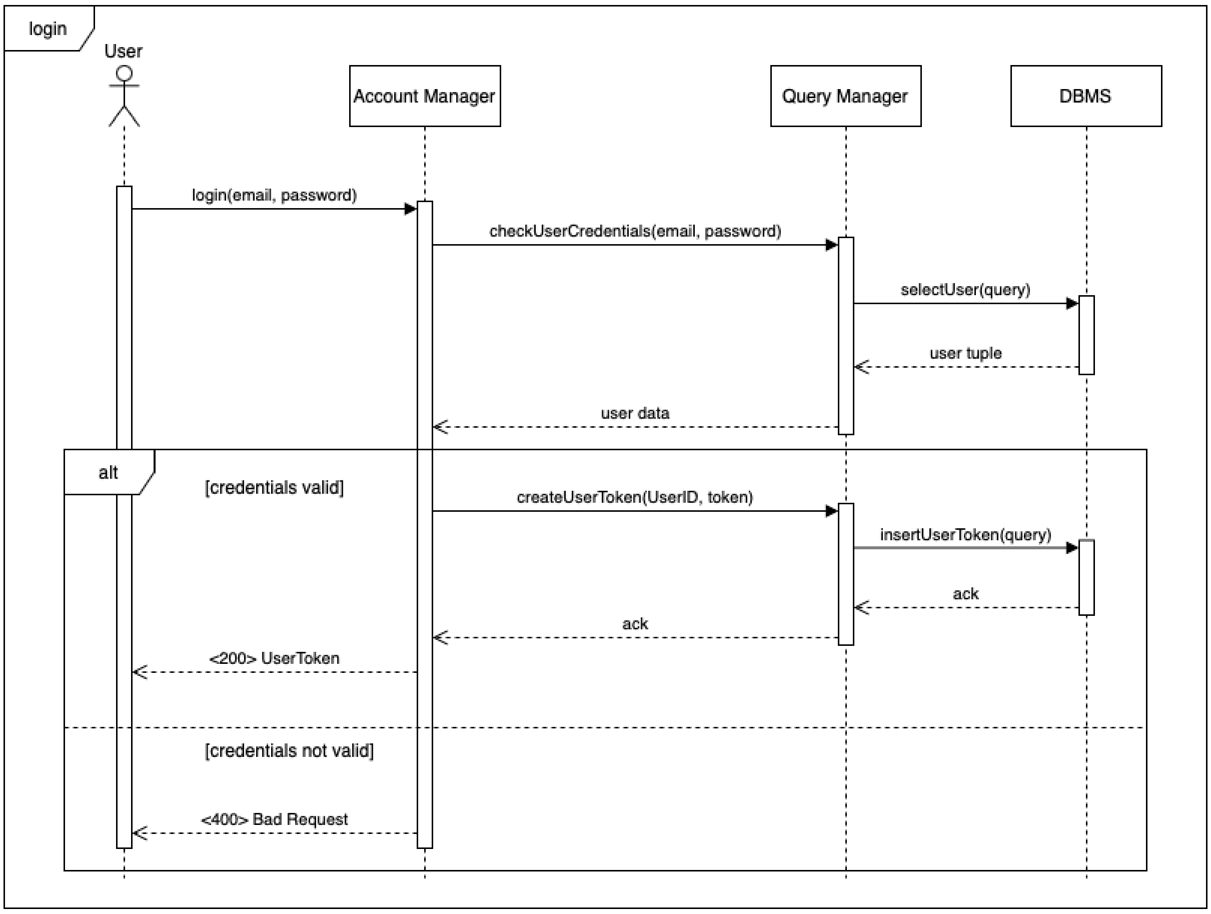


Figure 5: Login

- **Registration**
  This is the process a new user must follow to create a new account. First, they enter their desired credentials. Then, the system checks if the email provided already belongs to another user. If It does not, a new account is created and a confirmation email is sent. Otherwise, an error will be displayed.

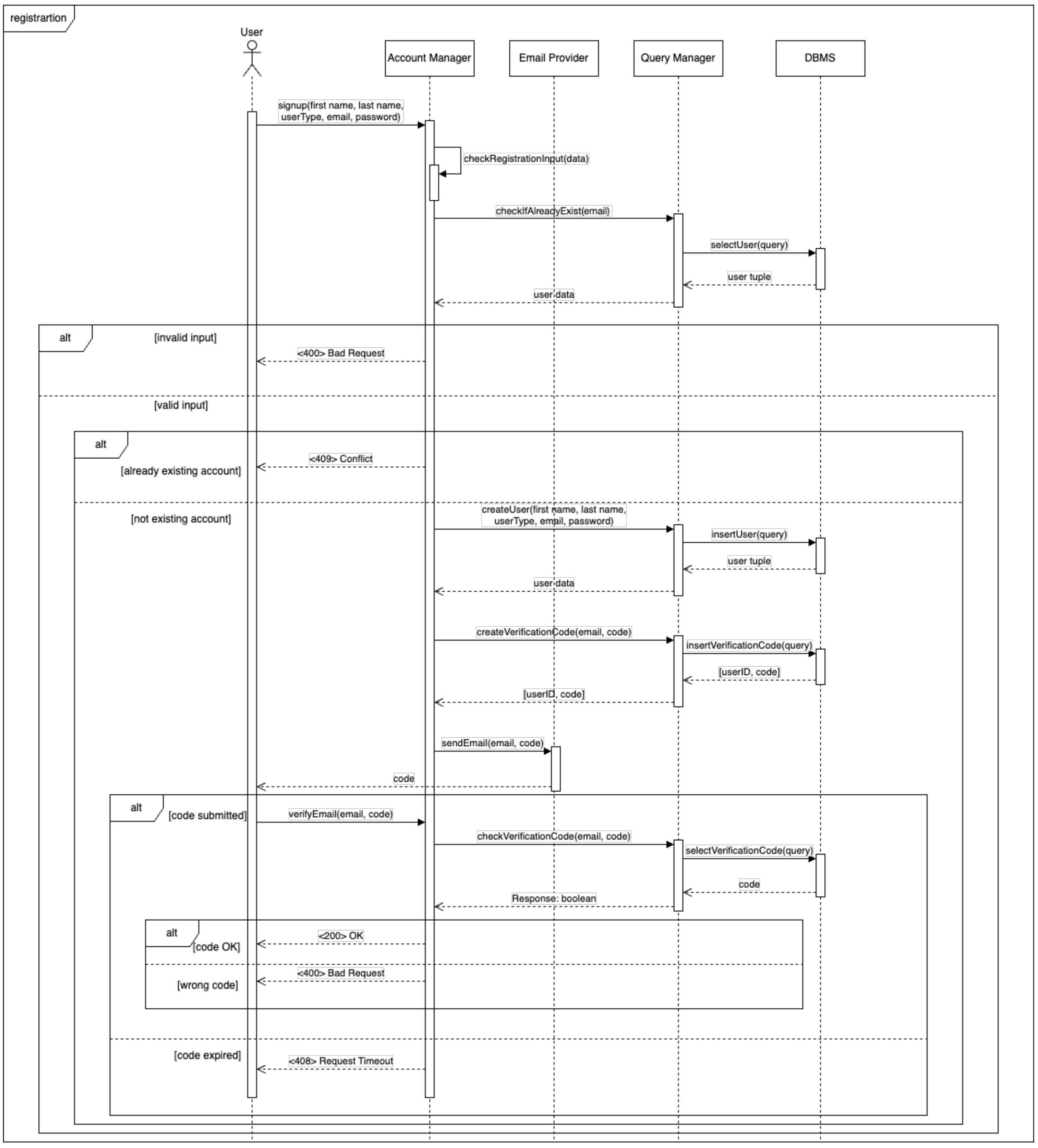


Figure 6: Registration

- **See tournament**
  Users can search for tournaments by name and applying filters. Requests will be handled by the search helper. After a user clicks on their desired tournament, the right page will be shown thanks to the tournament viewer.

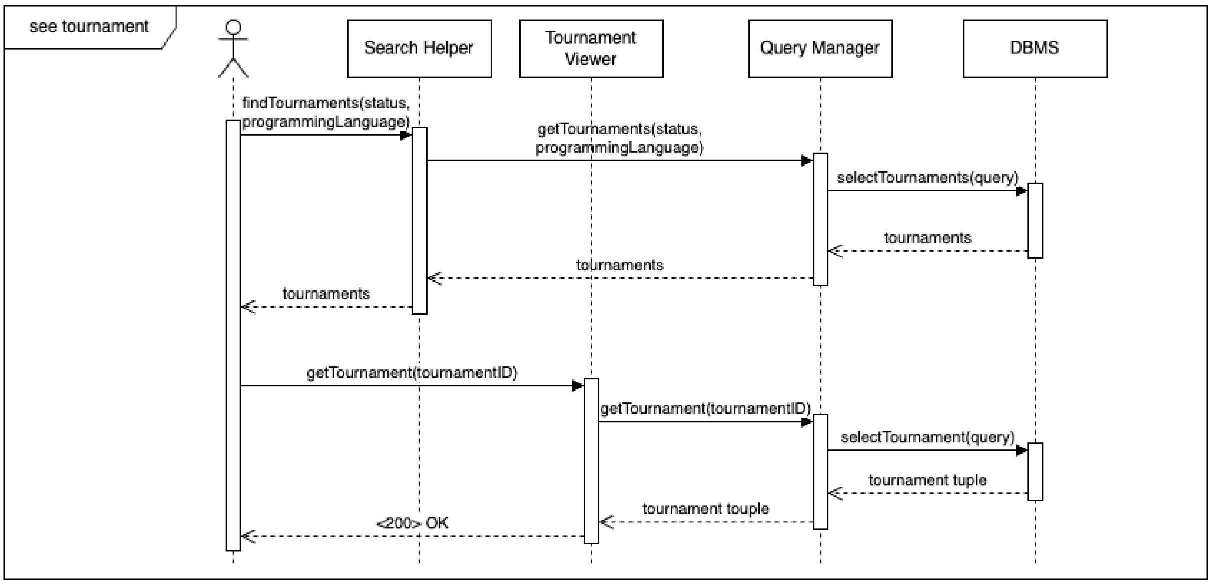


Figure 7: See tournament

- **See profile**
  Users can search for profiles by name and applying filters. Requests will be handled by the search helper. After a user clicks on their desired profile, the right page will be shown thanks to the profile viewer.

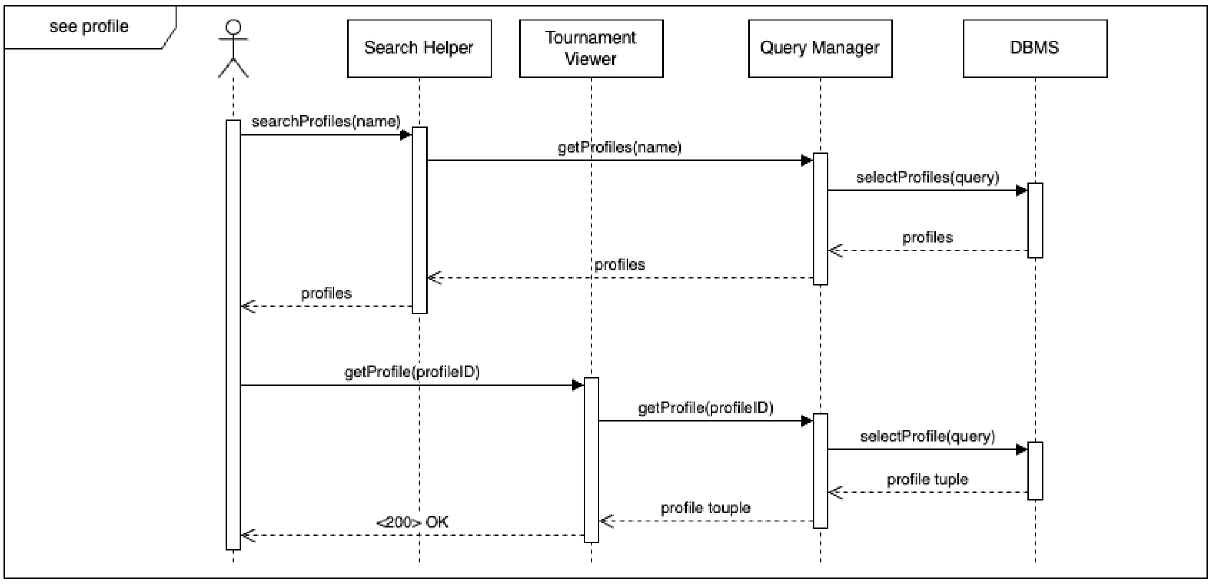


Figure 8: See profile

- **Manage account**
  Requests are handled by CKB's account manager. Users can update their desired preferences.

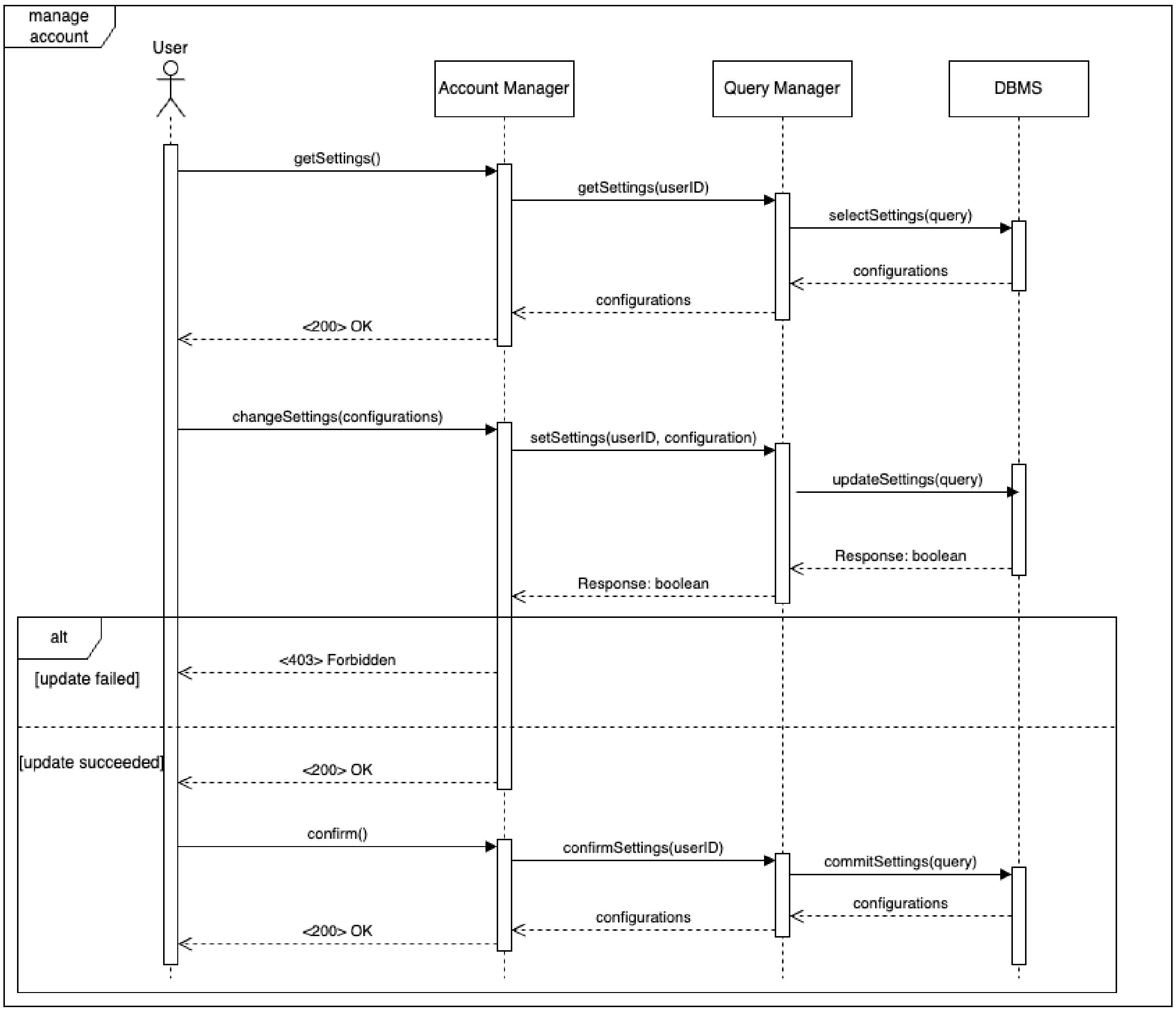


Figure 9: Manage account

- **Join tournament**
  The user navigates to the tournament they are interested in thanks to the tournament viewer as shown before. If the user is logged in as a student, a 'join tournament' button is displayed that the user can click. The request is handled by the tournament manager. If the request is valid, the user successfully joins the tournament. Internal databases are updated accordingly.

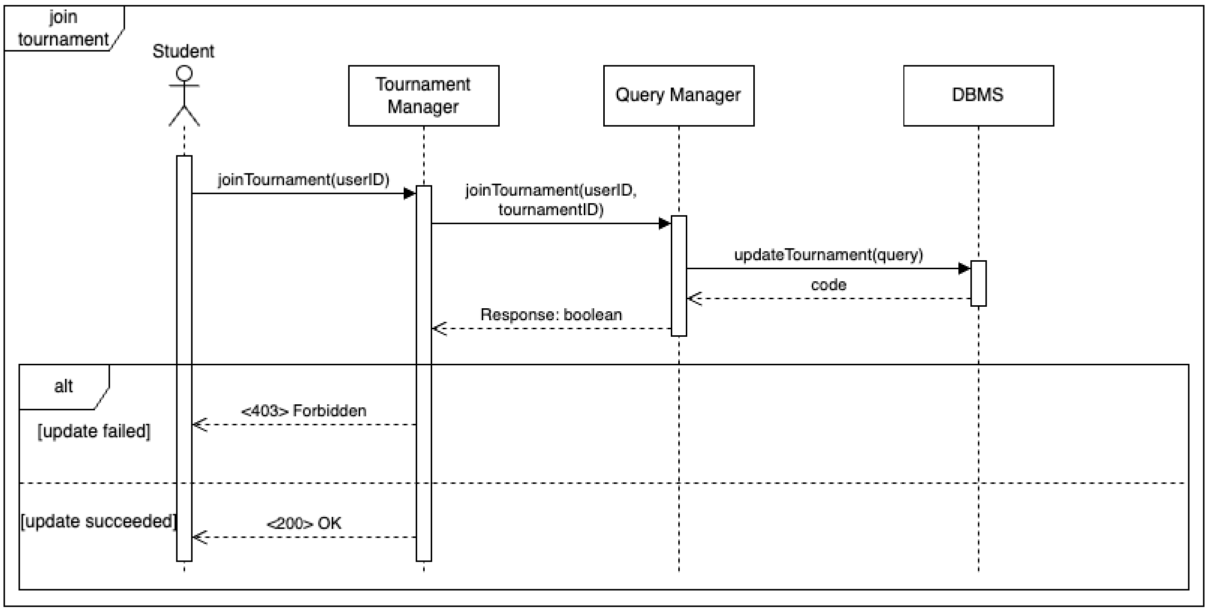


Figure 10: Join tournament

- **Create team**
  The user navigates to their desired battle thanks to the tournament viewer. If the user is logged in as a student and is not part of any team for that battle, a 'create team' button is displayed that the user can click. This request is handled by the battle manager. If the request is valid, a new team is created for the user. Internal databases are updated accordingly.

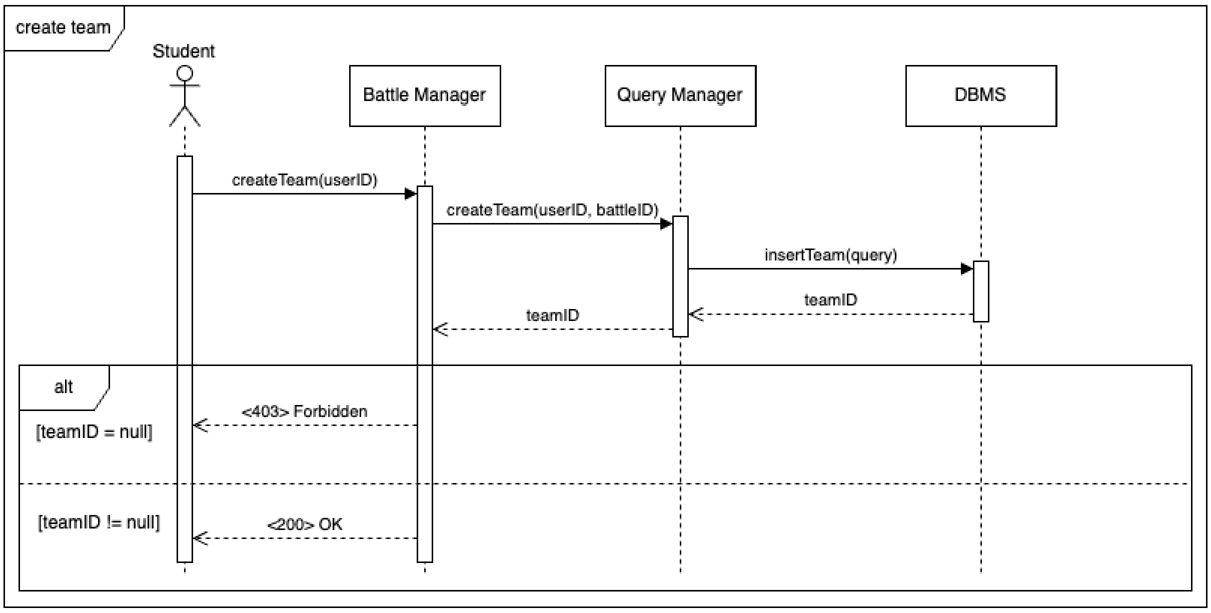


Figure 11: Create team

- **Join team**
  The user navigates to their desired battle thanks to the tournament viewer. If the user is logged in as a student and is not part of any team for that battle, a 'join team by id' button is displayed that the user can click. This request is handled by the battle manager. First the system checks if the id provided is valid for that tournament. If It is not, an error is displayed, otherwise, the user joins that team. Internal databases are updated accordingly.

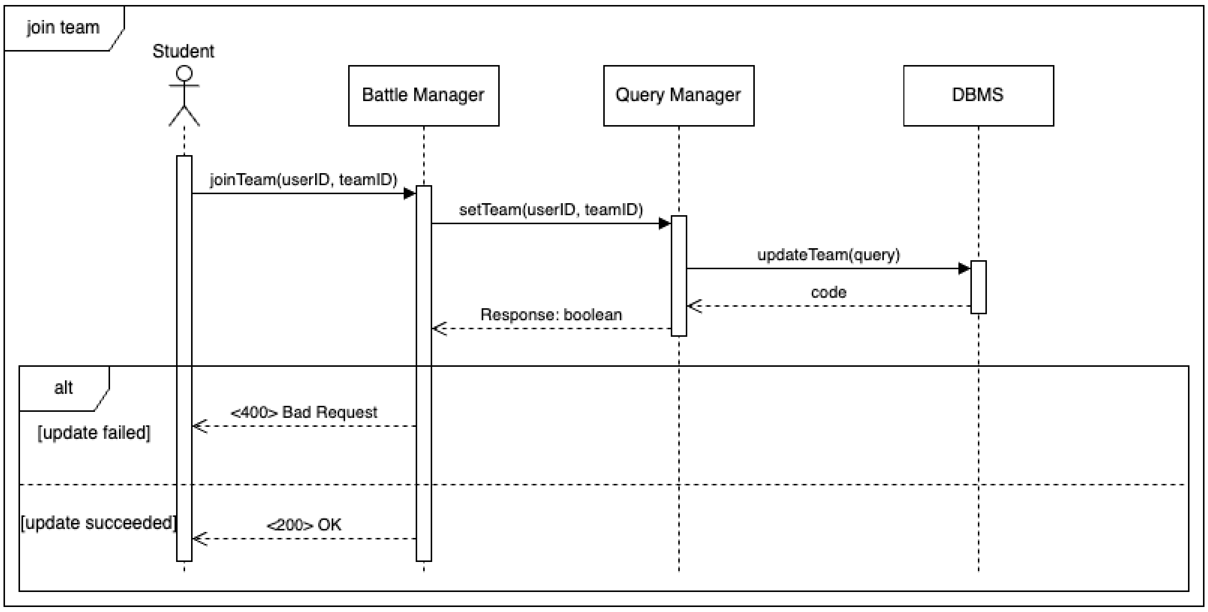


Figure 12: Join team

- **Join battle**
  The user navigates to their desired battle thanks to the tournament viewer. If the user is logged in as a student and is the creator of a team for that battle, a 'join battle' button is displayed that the user can click. This request is handled by the battle manager. First the system checks if the team respects the criteria for that battle. If It does not, an error is displayed, otherwise, the entire team joins that battle. Internal databases are updated accordingly.

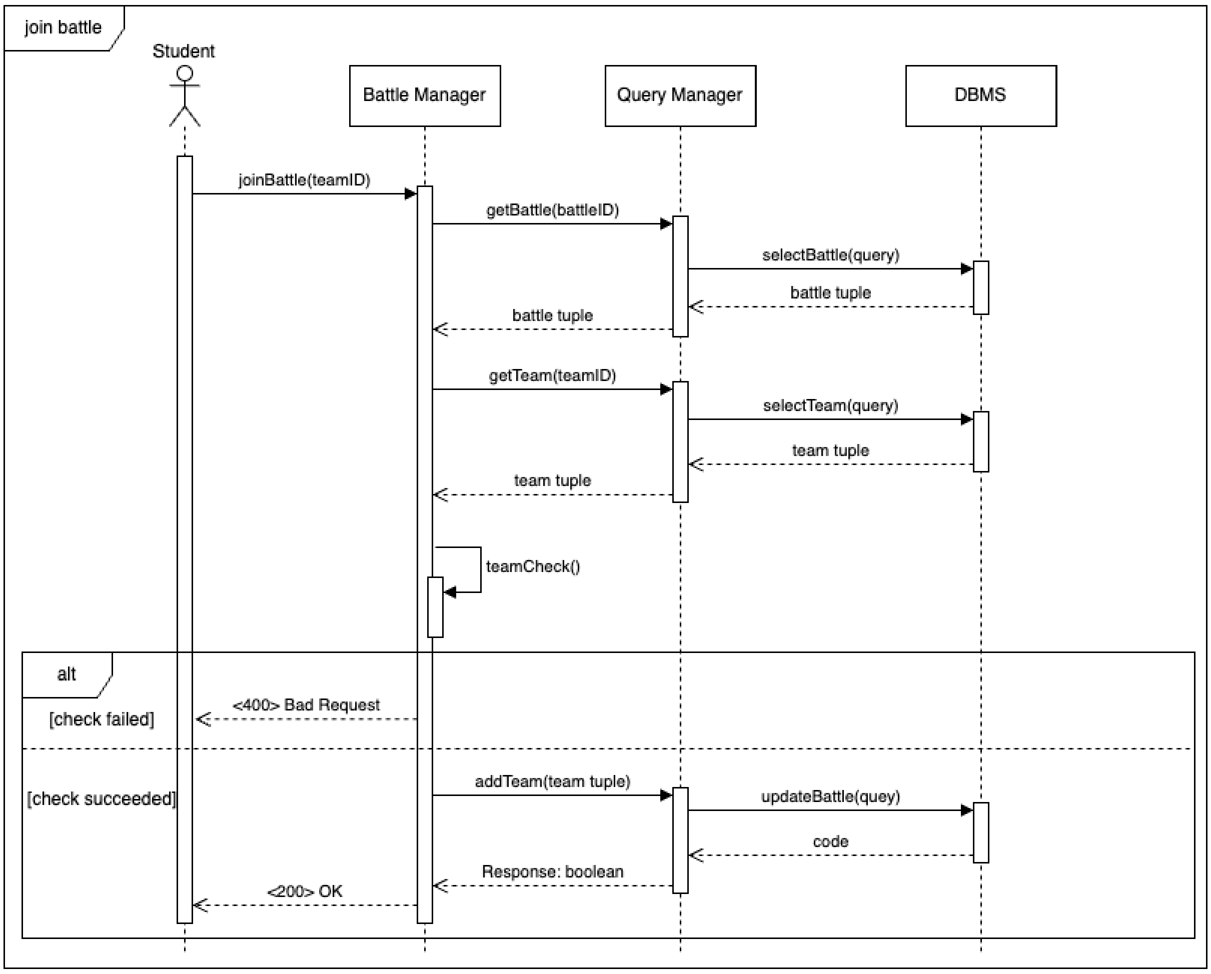


Figure 13: Join battle

- **Submit code**
  When a user commits to the main branch of a GitHub repository, GitHub updates its sources. If the user set up a proper automated workflow, the system will be notified by GitHub through the dedicated GitHub API. The system responds updating Its sources, using GitHub API again. After that, the sources are evaluated and the scores are updated. Such update and evaluation is handled by the battle manager.

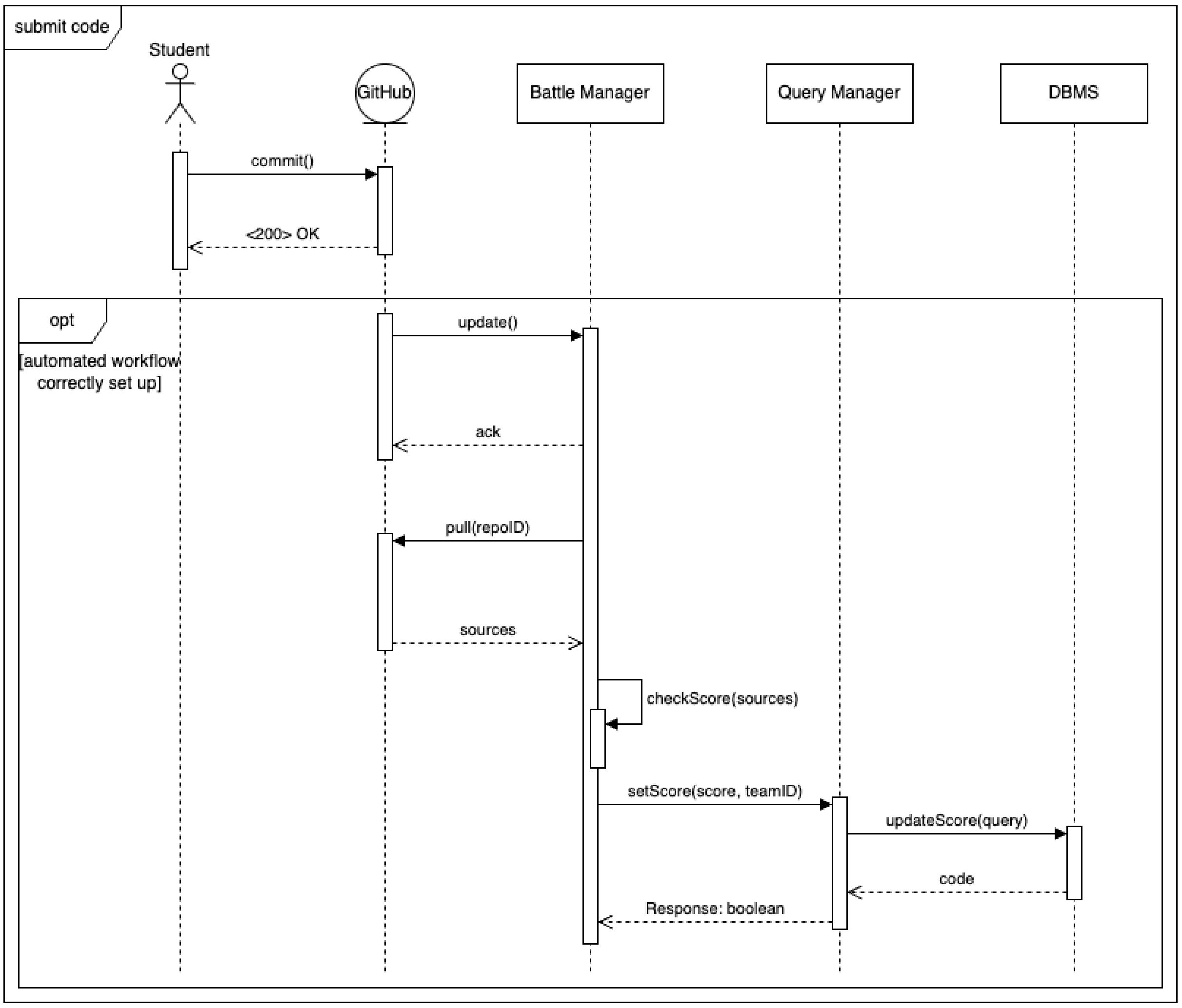


Figure 14: Submit code

- **Create tournament**
  If a user is logged in as an educator, a 'create new tournament' button will show up in the main page that the user can click. The whole creation process is handled by the tournament manager. The user chooses their desired settings and submits. The system checks if such settings are applicable (for instance, if the tournament name is not already taken). If they are, the tournament is created, otherwise, an error message will be displayed. Internal databases are updated accordingly.

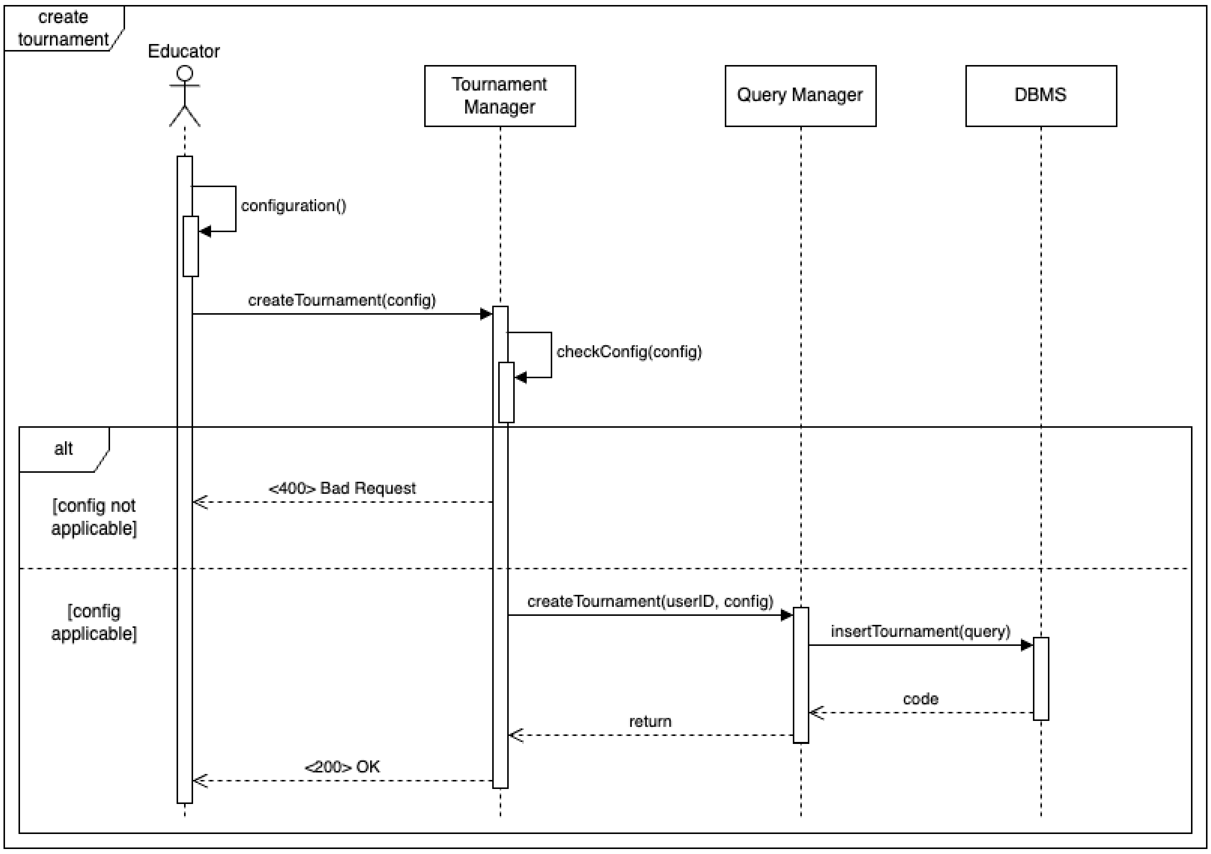


Figure 15: Create tournament

- **Edit tournament settings**
  The user navigates to their desired tournament thanks to the tournament viewer. If the user is logged in as an educator and is the creator of that tournament, a 'manage tournament' button is displayed that the user can click. After that, the process is similar to the previous one.

- **Create battle**
  The user navigates to their desired battle thanks to the tournament viewer. If the user is logged in as an educator and has been granted permission to create battles or is the tournament creator themselves, a 'create new battle' button is displayed that the user can click. The user chooses their desired settings and submits. The system checks if such settings are applicable (for instance, if the battle name for that tournament is not already taken). If they are, the battle is created, otherwise, an error message will be displayed. Internal databases are updated accordingly.

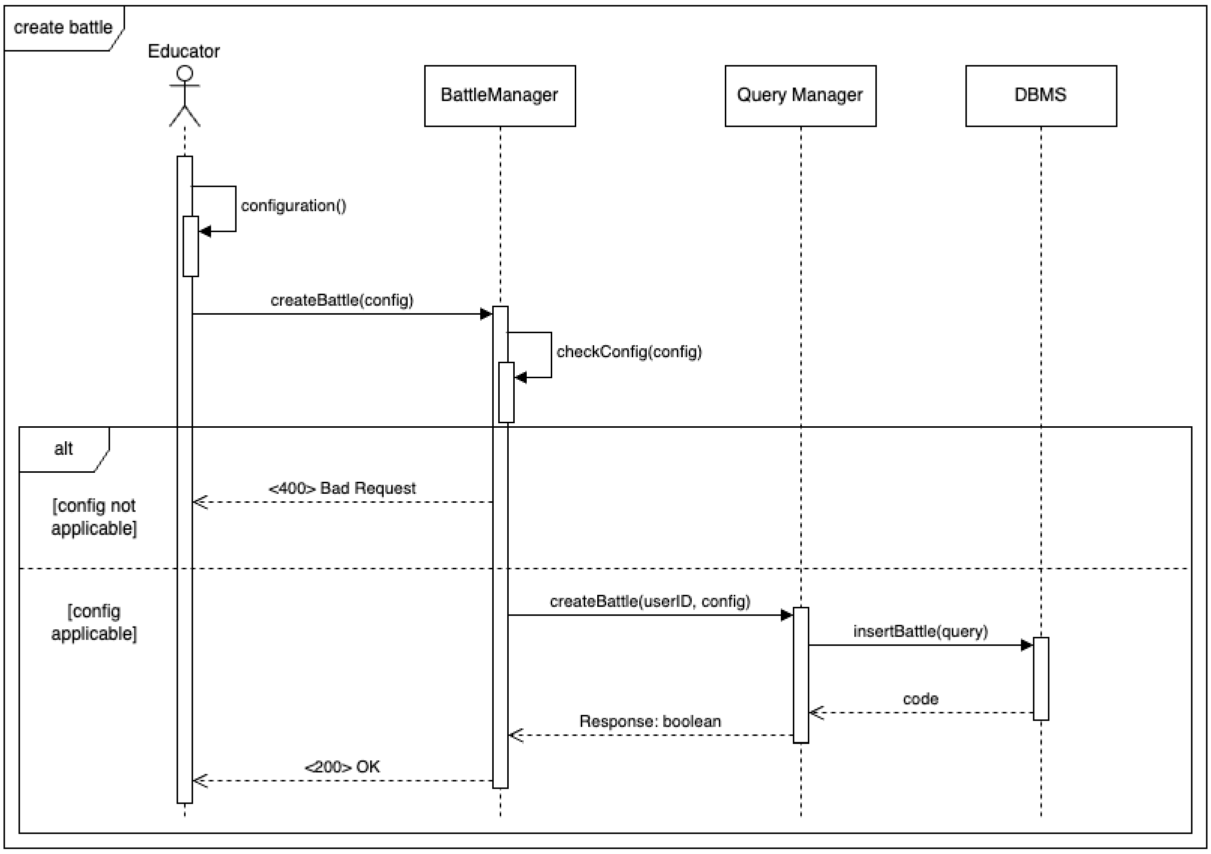


Figure 16: Create battle

- **Edit battle settings**
  The user navigates to their desired battle thanks to the tournament viewer. If the user is logged in as an educator and has been granted permission to create battles or is the tournament creator themselves, a 'manage battle' button is displayed that the user can click. After that, the process is similar to the previous one.

- **Review scores**
  This process can happen only when a battle is in consolidation phase. The user navigates to their desired battle thanks to the tournament viewer. If the user is logged in as an educator and has been granted permission to create battles or is the tournament creator themselves, a 'review scores' button is displayed that the user can click. The process is handled by the battle manager. Upon submitting, internal databases will be updated accordingly.

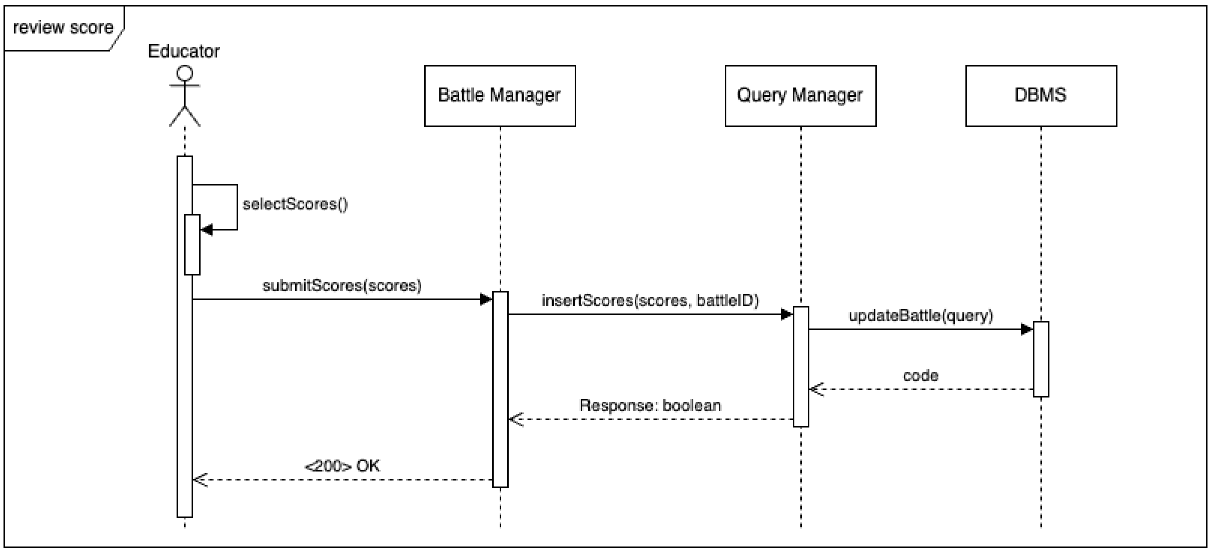


Figure 17: Review scores

- **Close tournament**
  This process can only happen when a tournament is ongoing. The user navigates to their desired tournament thanks to the tournament viewer. If the user is logged in as an educator and is the tournament creator, a 'close tournament' button is displayed that the user can click. The process is handled by the tournament manager. Internal databases are updated accordingly.

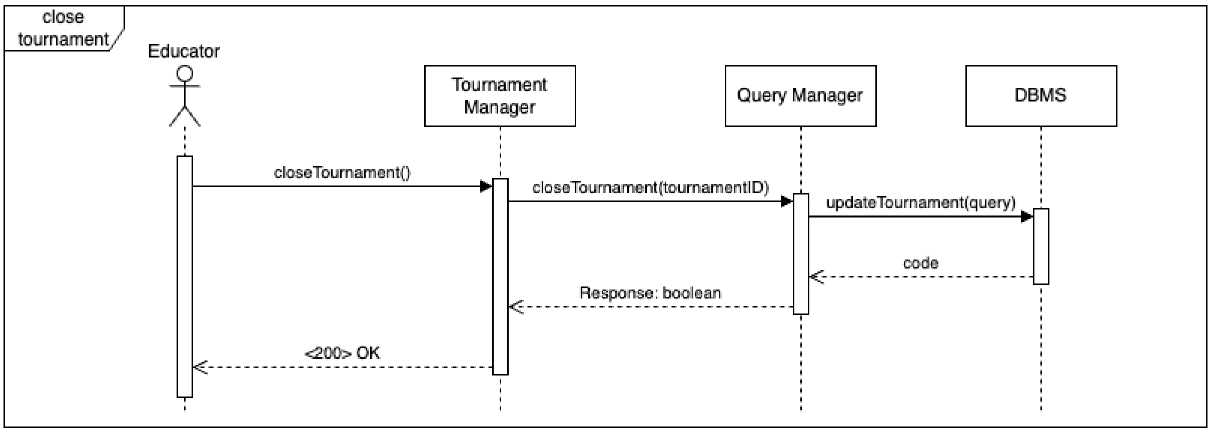


Figure 18: Close tournament

- **New tournament notification**
  When a new tournament is created in CKB, the tournament manager communicates with the CKB notification service in order to send a notification to all users.

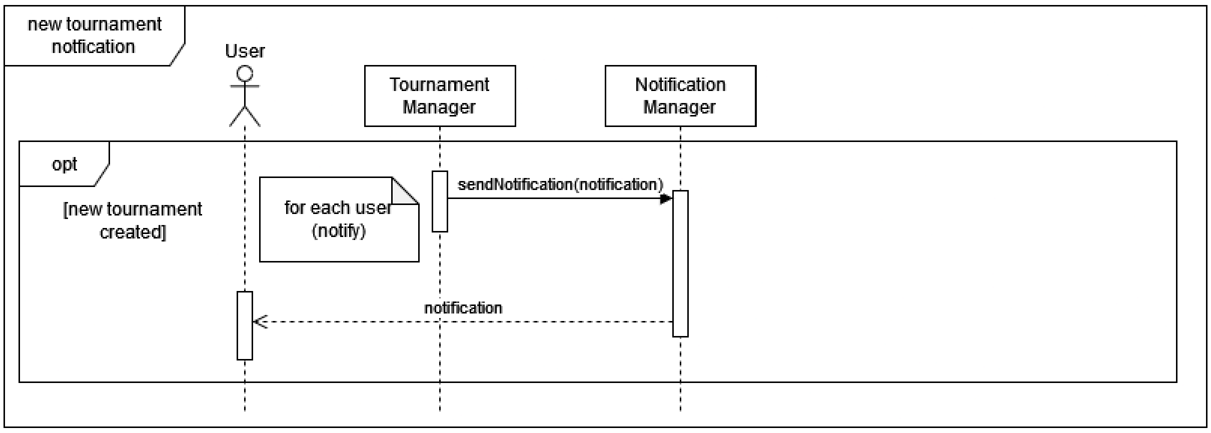


Figure 19: New tournament notification

- **New battle notification**
  When a new battle is created in the scope of a tournament, the battle manager communicates with the CKB notification service in order to send a notification to all users subscribed to that tournament.
- **Battle begins notification**
  When a battle begins, the battle manager communicates with GitHub through the dedicated GitHub API in order to create a repository. After that, It communicates with the CKB notification service, which notifies user subscribed to that battle.

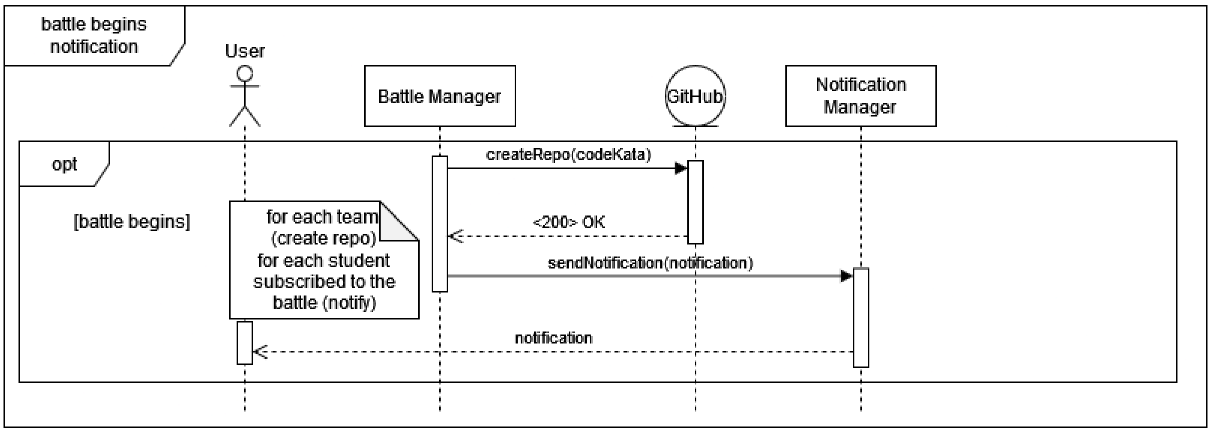

battle begins notification
User
Battle Manager
GitHub
Notification Manager
opt
[battle begins]
for each team (create repo)
for each student subscribed to the battle (notify)
createRepo(codeKata)
<200> OK
sendNotification(notification)
notification

Figure 20: Battle begins notification

- **Badge assignment**
  When a tournament closes, the tournament manager communicates with the badge helper, which checks for badges associated with that tournament. If there is any, It will check the requirements for each student. If a user is assigned a badge, the CKB notification service will be called to notify them.

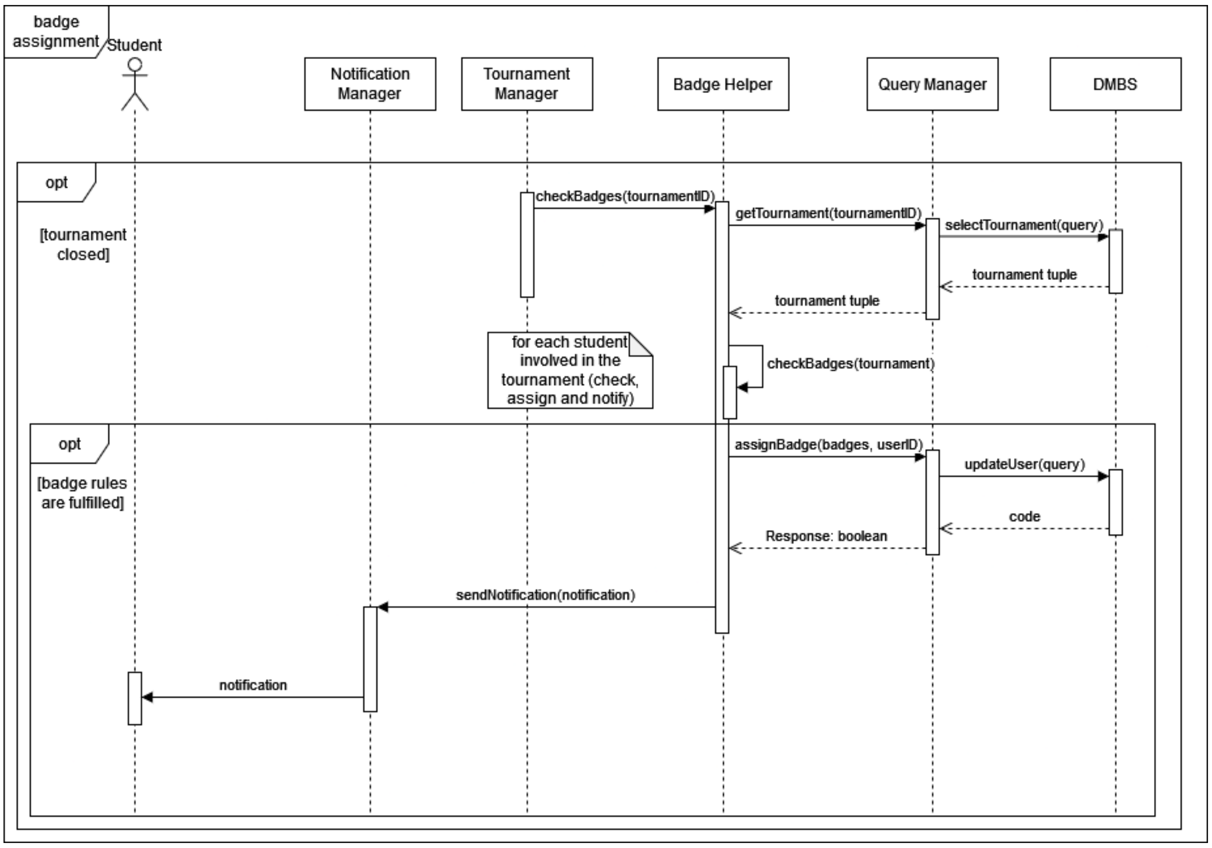


Figure 21: Badge assignment

2.5 Component Interfaces

This section lists all the methods offered by each component interface to the other components.

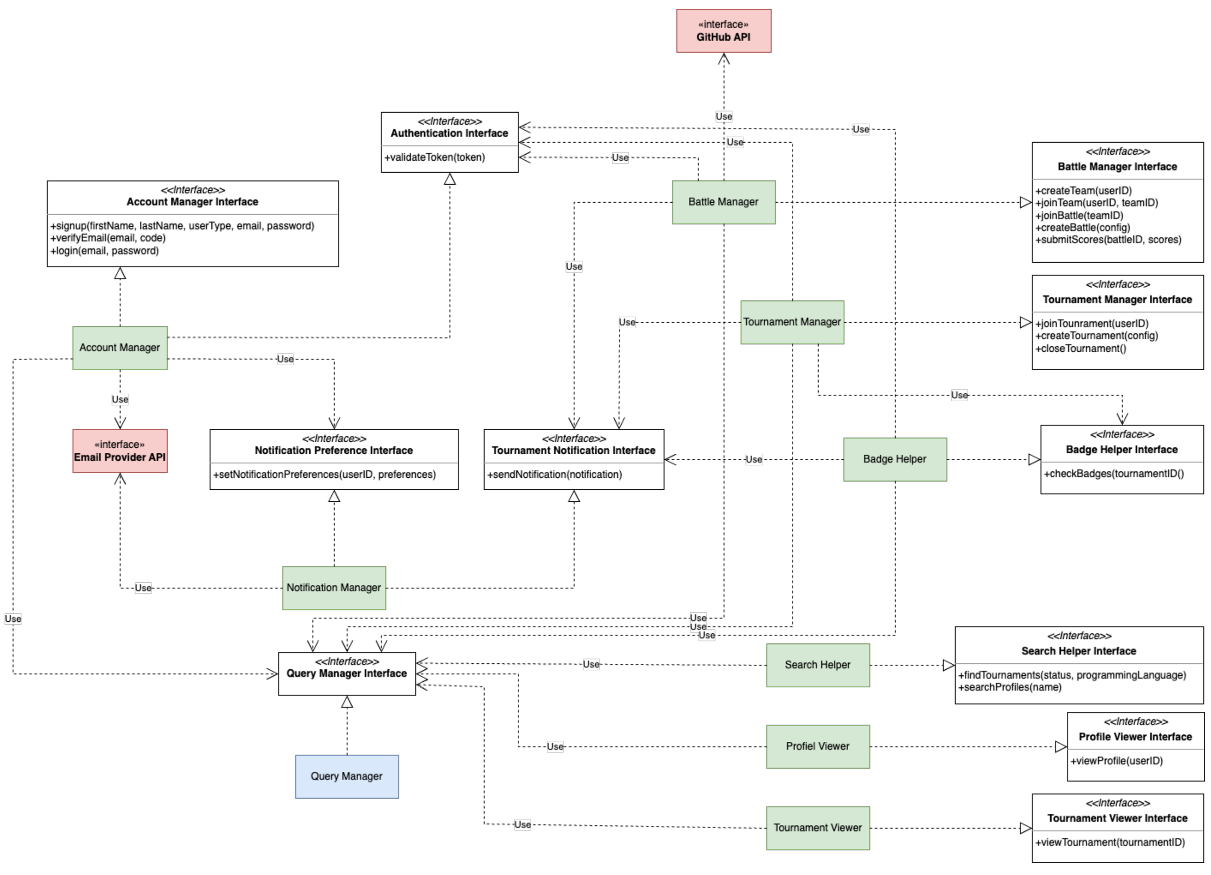


Figure 22: Class Diagram with interfaces of the CKB System

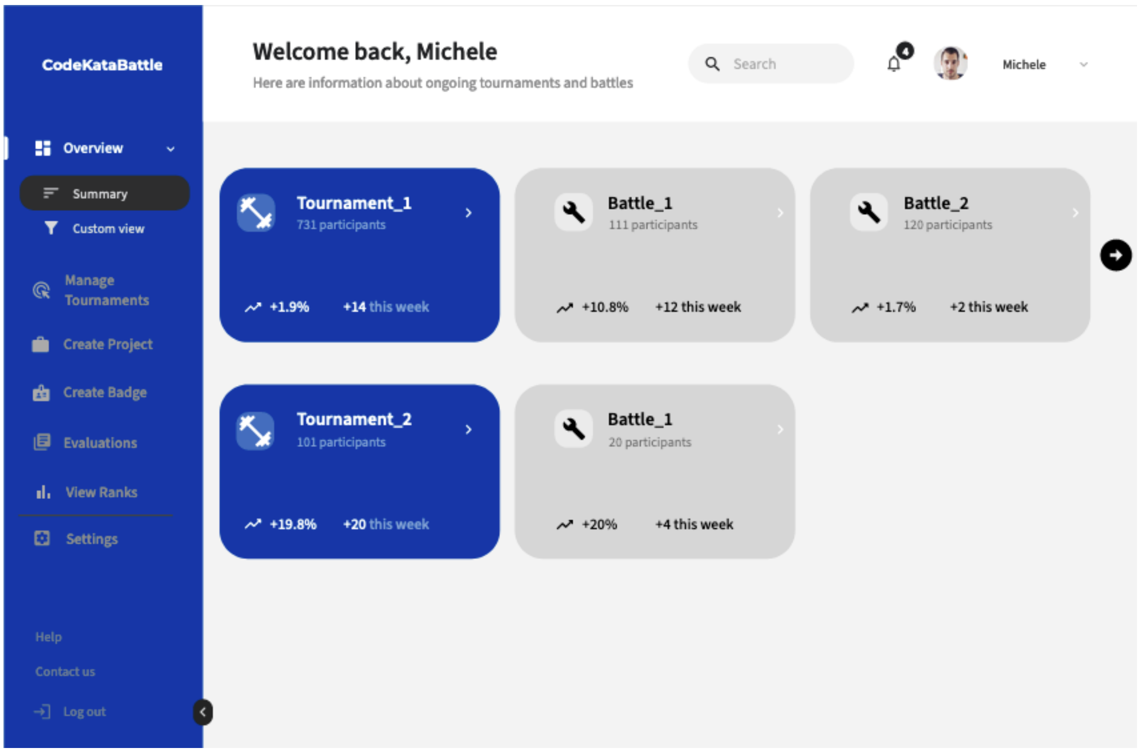
2.6 Selected architectural styles and patterns

- **Three layer four tier**
  This architecture offers several benefits. It allows the distinction of layers (presentation, logic, data) and tiers (client, web server, application server, DBMS), which makes the system more modular and easier to maintain. Such separation also allows better scalability and better workload distribution across multiple servers. Finally, load balancing and caching techniques are available to further increase performance and availability.
- **RESTful APIs**
  The API is based on standard web technologies, like JSON or XML, allowing better integration with other systems. It is stateless, which makes the development easier. It also promotes simplicity and scalability.
- **Adapter pattern**
  The Adapter Pattern seamlessly integrates different components, standardizing interfaces and ensuring flexibility. It hides the underlying complexity and provides a set of high-level functions.

2.7 Other design decisions

- SQL database SQL databases offer high scalability and decent performance for structured data. It follows ACID principles and ensures data integrity. Overall, It is a good compromise for performance, consistency, reliability and ease of use.

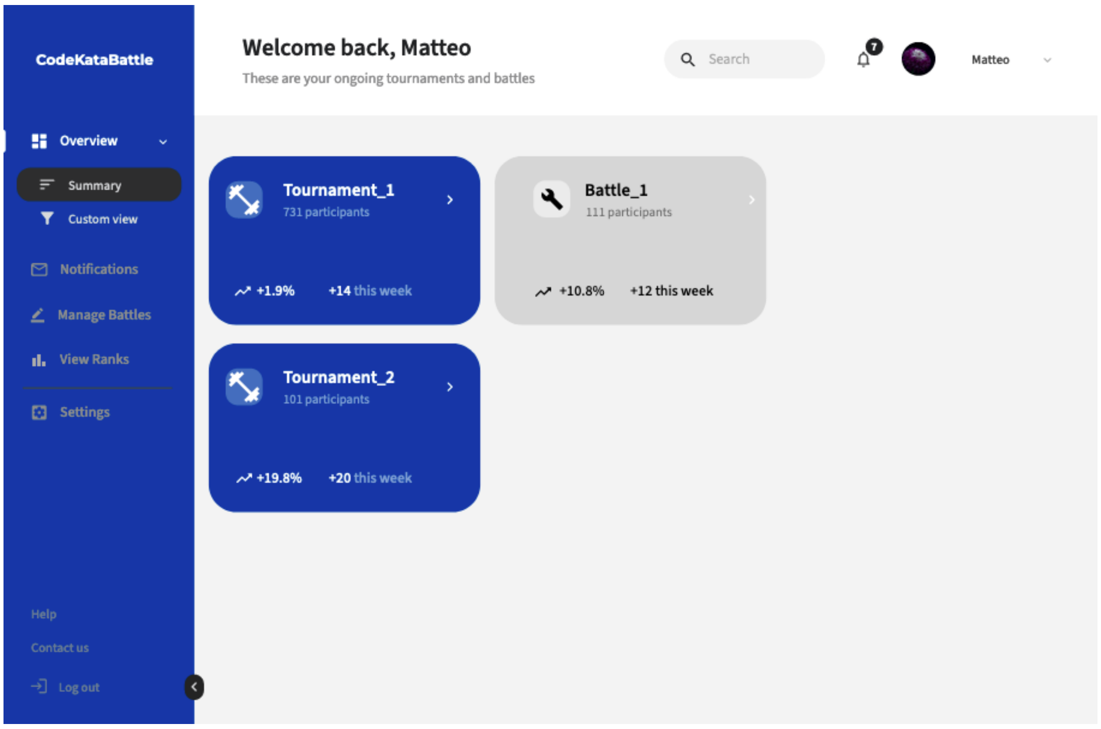
3 User Interface Design



Educator Interface

In the Educator Interface the main page shows the ongoing tournaments, followed by every ongoing battle for that tournament.
The search bar allows educator to search for a nickname and find users associated to it, so that it's possible to visualize their profile.
The main functions an educator can perform are shown on the left column:

- **Manage Tournaments**: allows an educator to create and close a tournament, create (with all the possible settings) and close battles and eventually grant the permission to create battles to another educator.
- **Create Project**: allows an educator to create a new project to assign to students.
- **Create Badge**: allows an educator to create a new badge that could be achieved by the students in a tournament.
- **Evaluations**: allows an educator to manually evaluate students' projects.
- **View Ranks**: view all ongoing tournaments and battles rank.



Student Interface

In the Student Interface the main page shows the ongoing tournaments, followed by the battles to which the student is subscribed.
The search bar allows student to search for a nickname and find users associated to it, so that it's possible to visualize their profile.
The main functions a student can perform are shown on the left column:

- **Notifications**: allows a student to view the notifications the platform sends when a new tournament or a new battle within a tournament the student is subscribed to is created.
- **Manage Battles**: this section allows a student to create teams and contains the link to the GitHub repository for each battles he/she is subscribed to.
- **View Ranks**: view all ongoing tournaments and battles rank.

4 Requirement Traceability

This section aims to map each requirement previously defined in the RASD document to the design elements that compose the CKB system and that satisfy it.

Requirements:	[$R1$]: The system allows educators to sign up [$R2$]: The system allows students to sign up
Components:	• Web App • Web Server • Dispatcher • CKB Server – CKB's Account Manager • Query Manager • DBMS • Email Provider API

Requirements:	[$R3$]: The system allows registered educators to log in [$R4$]: The system allows registered students to log in
Components:	• Web App • Web Server • Dispatcher • CKB Server – CKB's Account Manager • Query Manager • DBMS

Requirements:	[$R5$]: The system allows educators to create a new tournament [$R6$]: The system allows an educator to grant permission to create battles within the context of a specific tournament to another educator [$R24$] The system allows educators who created a tournament to close it
Components:	• Web App • Web Server • Dispatcher • CKB Server – CKB's Account Manager – Tournament Manager • Query Manager • DBMS

Requirements:	[*R*7]: The system allows educators to create a new battle, if they have the permission to do it
Components:	• Web App • Web Server • Dispatcher • CKB Server – CKB's Account Manager – Tournament Manager – Battle Manager • Query Manager • DBMS

Requirements:	[*R*8]: The system notifies all the students subscribed to the platform when a new tournament is created
Components:	• Web App • Web Server • Dispatcher • CKB Server – Tournament Manager – CKB Notification Service • Query Manager • DBMS

Requirements:	[*R*9]: The system allows students to subscribe to a tournament
Components:	• Web App • Web Server • Dispatcher • CKB Server – CKB's Account Manager – Search Helper – Tournament Manager • Query Manager • DBMS

Requirements:	[R10]: The system notifies all the students subscribed to a specific tournament when a new battle in the context of that tournament is created
Components:	• Web App • Web Server • Dispatcher • CKB Server – Battle Manager – CKB Notification Service • Query Manager • DBMS

Requirements:	[R11]: The system allows educators that created a battle to upload the code kata [R12] The system allows educators that created a battle to set minimum and maximum number of students per group [R13] The system allows educators that created a battle to set a registration deadline [R14] The system allows educators that created a battle to set a final submission deadline [R15] The system allows educators that created a battle to set the possibility to manually evaluate projects during the consolidation stage [R16] The system allows students to create teams for ongoing battles inviting other students [R17] The system allows students to join a battle [R20] The system allows educators to manually evaluate projects during the consolidation stage in the context of a specific battle
Components:	• Web App • Web Server • Dispatcher • CKB Server – CKB's Account Manager – Battle Manager • Query Manager • DBMS

Requirements:	[R18]: The system sends to each student that belongs to a team subscribed to a battle the link of the GitHub repository assigned to that team
Components:	• Web App • Web Server • Dispatcher • CKB Server – Battle Manager – CKB Notification Service • GitHub API • Query Manager • DBMS

Requirements:	[R19]: The system allows both students and educators to see the current rank evolving during a battle [R22] The system allows both educators and students to see the list of ongoing tournaments [R23] The system allows both educators and students to see the rank of each ongoing tournament
Components:	• Web App • Web Server • Dispatcher • CKB Server – Tournament Viewer • Query Manager • DBMS

Requirements:	[R21] The system notifies all students participating in the battle when the final battle rank becomes available [R25] The system notifies all students involved in a tournament when the final rank of that tournament becomes available
Components:	• Web App • Web Server • Dispatcher • CKB Server – Tournament Viewer – CKB Notification Service • Query Manager • DBMS

Requirements:	[R26] The system allows educators that have created a tournament to create badges
Components:	• Web App • Web Server • Dispatcher • CKB Server – CKB's Account Manager – Tournament Manager – Badge Helper • Query Manager • DBMS

Requirements:	[R27] The system allows both educator and students to visualize the profile and badges of a student
Components:	• Web App • Web Server • Dispatcher • CKB Server – Profile viewer • Query Manager • DBMS

5 Implementation, Integration and Test Plan

5.1 Overview

The aim of this chapter is to show how to implement the system, then how to integrate the various components with each other and finally how to plan the test phases, in order to offer simplifications during the development of the whole system.
First, there is a description about how to implement, integrate and test single components. After that, the second step consists in defining the methods to follow to test the whole system and last there are some concerns about additional information regarding the testing process.

5.2 Implementation, Component Integration and Testing Plan

Here is a summary regarding the components that together constitute the CKB system:

- **Web Server**
- **CKB Server**
  - **CKB's Account Manager**
  - **Search Helper**
  - **Profile Viewer**
  - **Tournament Viewer**
  - **Tournament Manager**
  - **Battle Manager**
  - **Badge Helper**
  - **CKB Notification Service**
- External components
  - **Github API**
  - **Email Provider API**
- **Query Manager**

To implement the system, we decided to opt for a bottom-up approach. By focusing on individual components and building gradually, it yields a more modular and scalable design. This facilitates specific modifications without impacting the entire system.
Moreover, it promotes adaptability and flexibility, enabling optimization of distinct parts without compromising the overall technical project.
To follow the bottom-up approach, we highlighted the difficulty level to implement each component and we also provided a development order based on the dependencies between each component with the others, leading to a decision about the sequence of steps of implementation, component integration and testing.
In the following steps we assume that DBMS component is already implemented and available (it is not represented for simplicity), to focus on the main components and sub-components of CKB system.

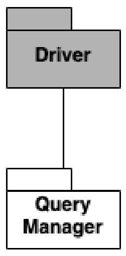
Order	Stage
0	[$F0$] DB Interaction
1	[$F1$] User Notifications
2	[$F2$] Create Account and Log in
3	[$F3$] View Profiles
4	[$F4$] Search Features
5	[$F5$] Managing Battles
6	[$F6$] Managing Tournaments
7	[$F7$] Tournaments and Battles Features

Features Identification - Server Side

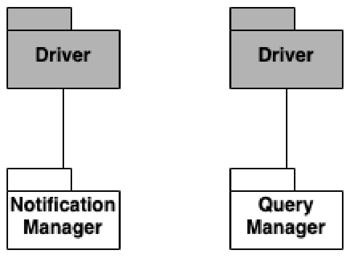
5.3 Development Stages Definition

Here is a description about the steps to follow to implement, integrate and test the components of the system following the features enumeration described above.
Each component uses a driver to simulate other components not already implemented.

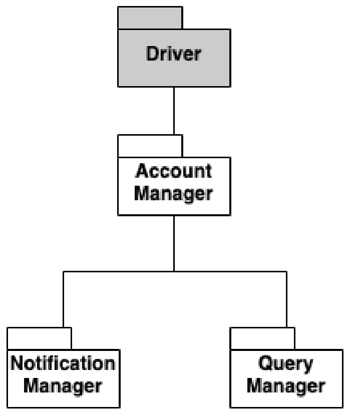
1. **DB Interaction**: the first component that must be implemented in the system is Query Manager, that offers to almost all other components in CKB server an interface to communicate indirectly with the DBMS.



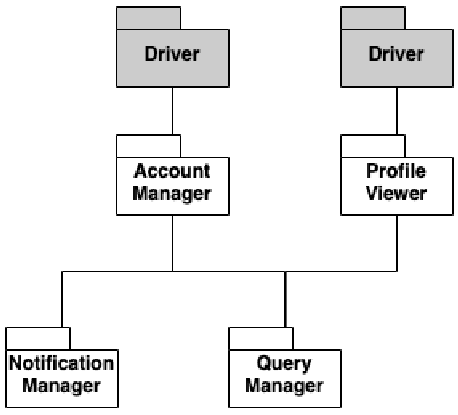
2. **User Notifications**: this is the second step of the implementation and integration of components that satisfy the requirements for the CKB system. In this phase it's introduced and tested the Notification Manager to handle the notifications that the system sends to users. It offers interfaces to other components that will later be developed.



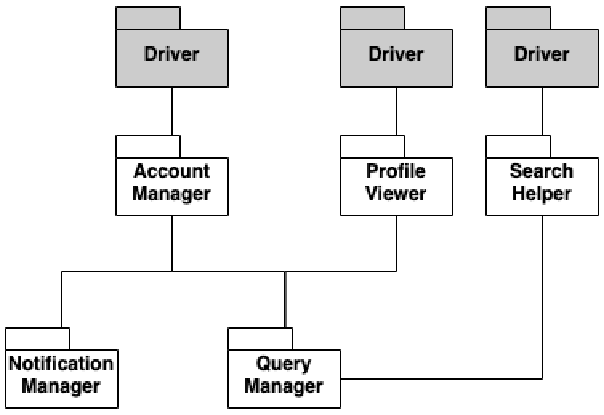
3. **Create Account and Log in**: than component that have to be implemented is the one who manage the basic features to access the systems, that is creating a new account or logging in, with all the features that result from it, such as setting preferences or verifying an account whenever needed. The component that manages these features is CKB's Account Manager, the third one to be implemented and for which the unit tests have to be provided. Account Manager is integrated with Notification Manager as it leverages functionalities exposed by that component and with Query Manager to interact with DBMS.



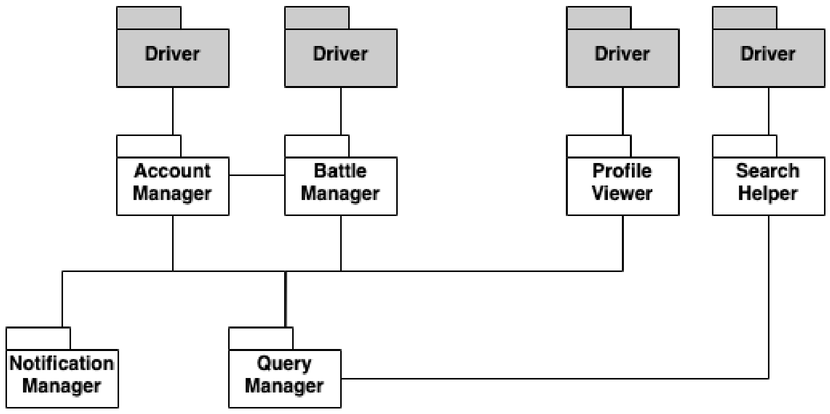
4. **View Profiles**: the next phase consists in implementing and test Profile Viewer component, the one that manages user's profiles. It is integrated with Query Manager as it interacts with the DBMS to accomplish its operations.



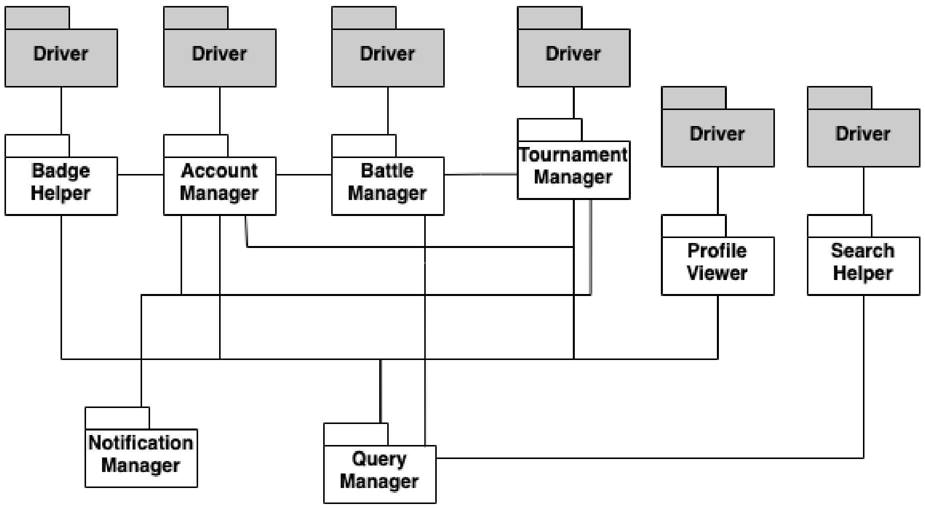
5. **Search Features**: the next component that has to be implemented and tested is Search Helper. It offers the possibility to search for user profiles and to search for a tournament with filtering options. It is integrated with Query Manager as it interacts with the DBMS to accomplish its operations.



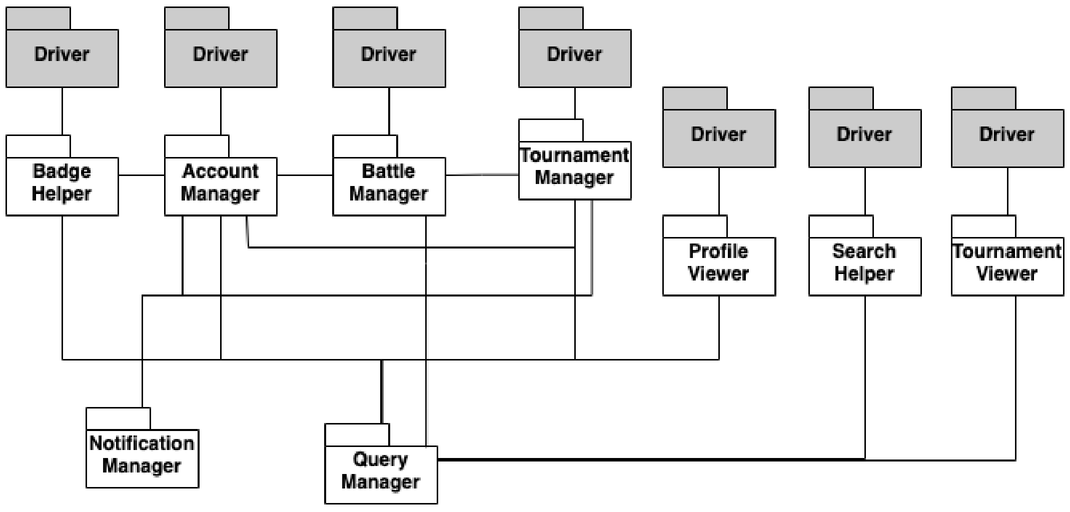
6. **Managing Battles**: in this step must be implemented one of the core components for the system, Battle Manager. It offers an interface for the following component and it interacts with Account Manager so these two components have to be integrated. It is integrated also with Query Manager as it interacts with the DBMS to accomplish its operations.



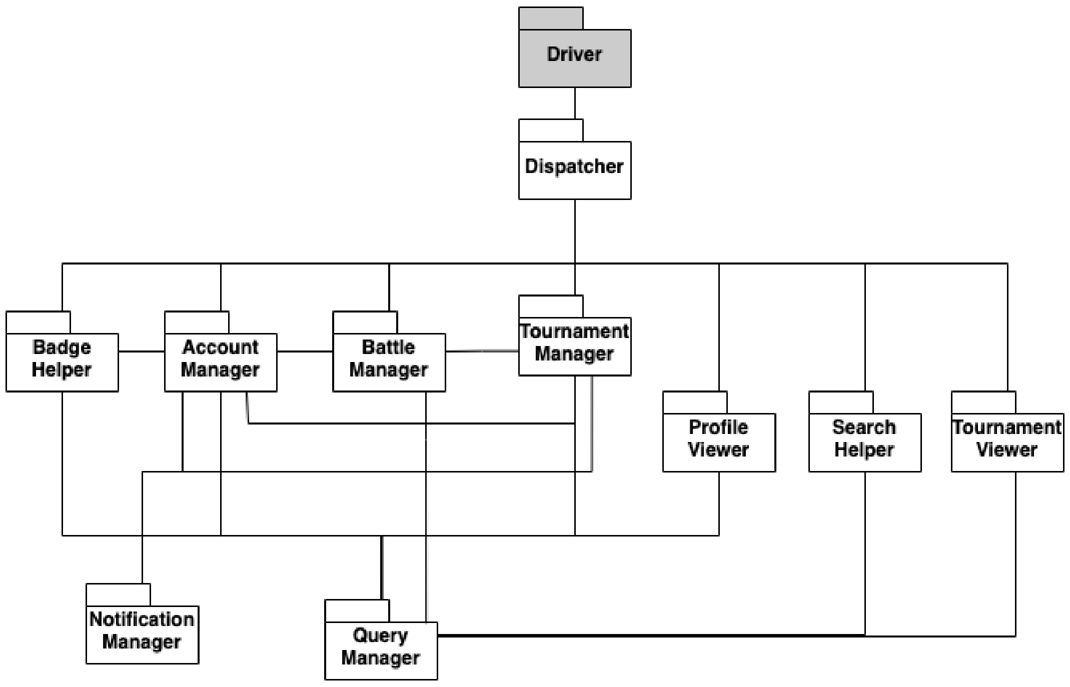
7. **Managing Tournaments**: this is one of the most important feature of the system and the component that fulfill it is Tournament Manager, that offers all the features described previously in this document. Moreover, due to the possibility for educators to create badges before a tournament, in this phase is implemented also the Badge Helper component. Both components are integrated with Account Manager because many operations require the profile authentication to be executed. For both of these components must be provided the appropriate unit tests. The Tournament Manager component is also integrated with Battle Manager and Notification Manager to fulfill the main operations regarding the life cycle of a tournament. Finally, the component is integrated with Query Manager to interact with DBMS.



8. **Tournaments and Battles Features**: this step implements and integrates in the system the Tournament Viewer component. It is integrated with Query Manager as it interacts with the DBMS to accomplish its operations.

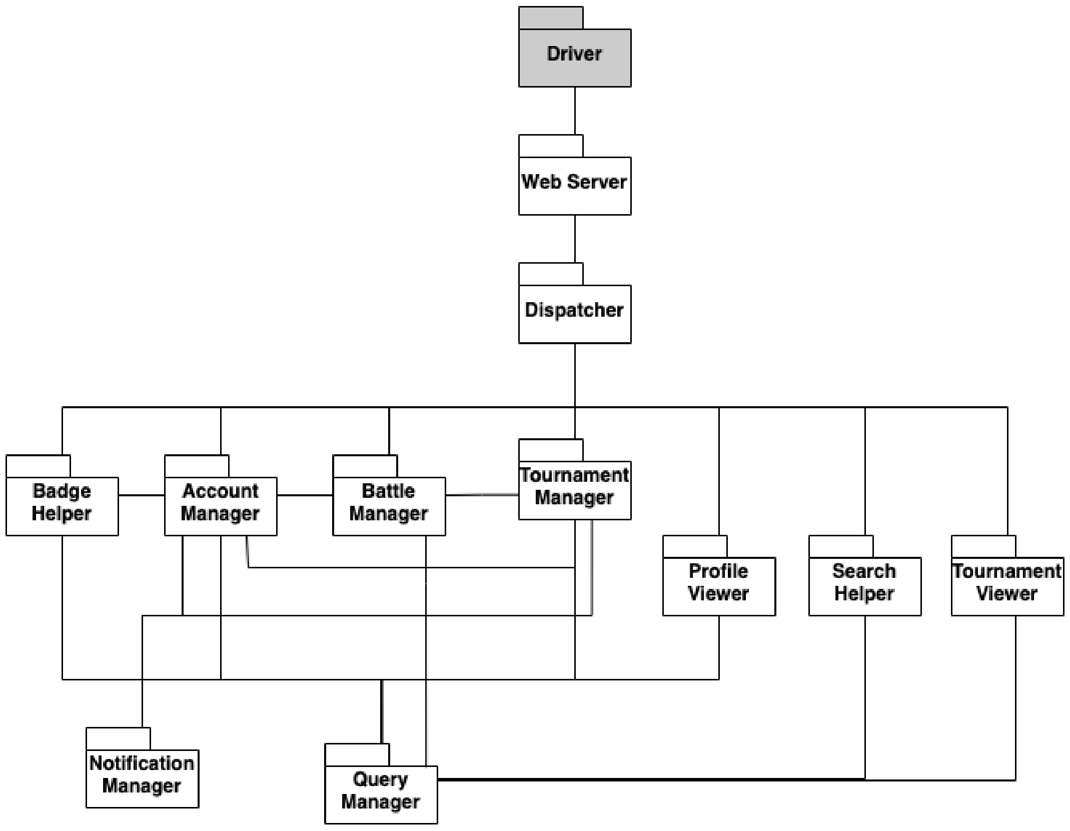


9. **Dispatcher**: this component has to be implemented and tested to allow the correct interaction with different components. It's integrated with almost all the components previously described, so this phase is crucial to build and develop a working system.

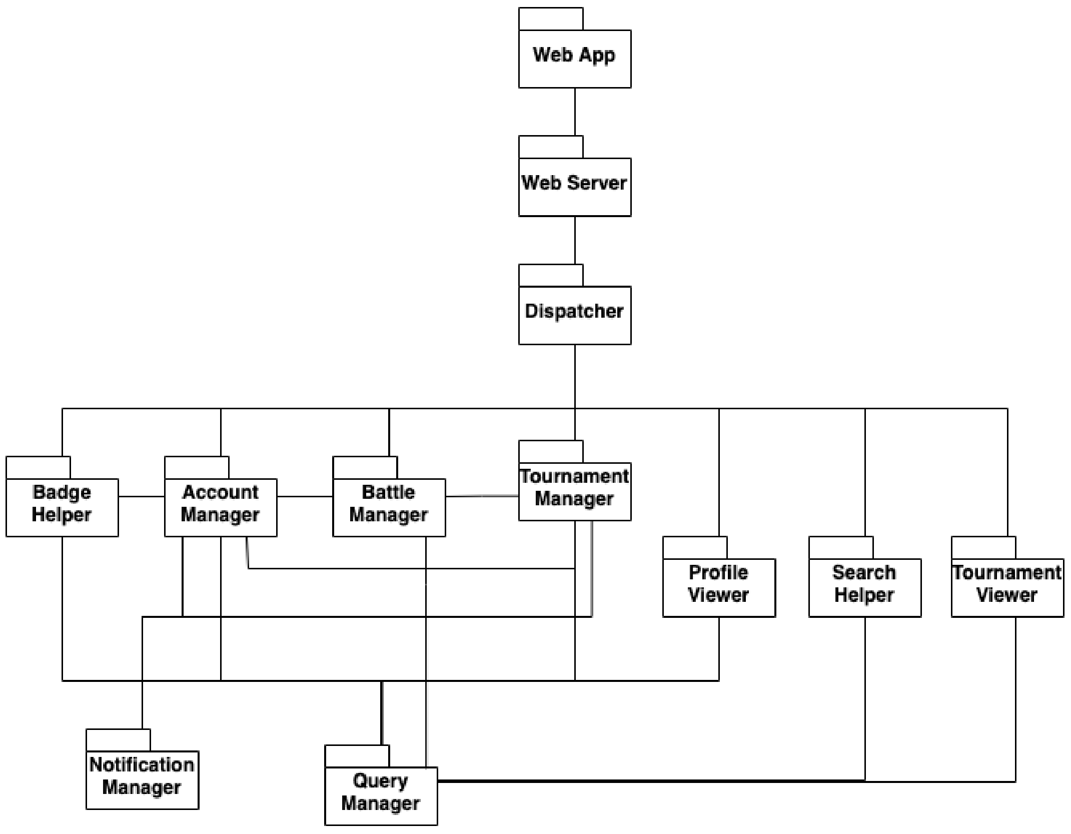


From now on are implemented components to allow the communication between the client-side and the server-side.

10. **Web Server**: this component interacts from the server-side with the Dispatcher, so when the Web Server is implemented and tested, it has to be integrated with it.



11. **Web App**: this component represents the system on the client-side. It offers an interface to communicate with the server-side (it has to be integrated with Web Server). This interaction must be carefully curated down to the smallest details as it represents the main interaction between users and system.



5.4 System testing

This phase consider the system as a whole and concerns the testing of all the functionalities that the system has to fulfill, analyzing the requirements defined in the RASD.
System testing is a more complicated step with respect to unit testing, and has to be performed by developers but with also an interaction with stakeholders, to make sure that the system is as close as possible to the desired final product.

Different system testing techniques can be adopted and executed:

- **Functional testing**: to check whether to software meets the requirements.
- **Performance Testing**: to highlight bottlenecks and measure response time and throughput and also hardware and network issues. To do so the testers have to load the system with expected workload and compare performances.
- **Load Testing**: to highlight bugs concerning the memory usage. To do so the testers have to load the system with increasing workload and for a long period.

- **Stress Testing**: to make sure that the system recovers gracefully after failure. The testers have to try to break the system under test by overwhelming its resources or by reducing resources.

5.5 Additional specifications on testing

A key point that must be underlined is that this is only a first version of the CKB system, so whenever new functionalities or changes in the requirements are made, the developers must check as first thing the functionalities added, and then also executing integration tests and system tests that can find new bugs before releasing a new version of the system. Once more, during the whole process, is important having an interactions with the stakeholders to take right decisions to create the right product.

6 Effort Spent

Emanuele Pocelli

Chapter	**Effort (in hours)**
1	1
2	25
3	2
4	2
5	2

Fabrizio Sordetti

Chapter	**Effort (in hours)**
1	1
2	25
3	1
4	3
5	3

Andrea Varesi

Chapter	**Effort (in hours)**
1	3
2	3
3	5
4	6
5	15

7 References

- Diagrams made with: draw.io
- Mockups made with: moqups.com
- Testing models and component architecture made with: draw.io
- High level architectures are made with: draw.io