POLITECNICO DI MILANO

Computer Science and Engineering

Design Document

CodeKataBattle

Software Engineering 2 Project

Academic year 2023 - 2024

3 January 2024

Version 1.0

Authors:
Massara Gianluca
Nguyen Ba Chiara Thien Thao
Nicotri Flavia

Professor:
Matteo Giovanni Rossi

Contents

Chapter 1

Introduction

1.1 Purpose

In the last years, more and more students are becoming interested in programming and many educators have realized the need of new innovative methods to improve coding skills.
CodeKataBattle aims to assist students in enhancing their programming skills by challenging them with creative tasks in a competitive and stimulating environment.
The following document wants to describe the system focusing on the design choices, describing at a high level the technologies and components used to implement the platform, paying attention also to the interactive behavior of system users.

1.2 Scope

CodeKataBattle is a web application that allows students to compete in tournaments created by educators. During a tournament, educators can create battles and students can join them. During a battle, students can create a team and complete the project with their code, which will be evaluated by the system and optionally by the educators. Battle scores contribute to both battle and tournament ranks, and at the end of a tournament students can also collect badges based on their performance.
For a more detailed description of the features that the system offers to end users, please refer to the RASD.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Definitions	Meaning
Code Kata	A code kata battle is a programming exercise in a programming language of choice. The 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.
Tournament	A tournament is a set of battles. It is created by an educator and it is composed by a set of rules and possible badges.
Battle	A battle is a competition between teams. It is created by an educator and it is composed by a set of rules.
Team	A team is a group of students that compete in a battle.
Badge	A badge is a reward that a student can obtain by participating in a tournament.
Application User Interface	The application user interface defines the operations that can be invoked on the component which offers it.
Demilitarized zone	A demilitarized zone is a middle ground between an organization's trusted internal network and an untrusted, external one.

1.3.2 Acronyms

Acronyms	Meaning
CKB	CodeKataBattle
UI	User Interface
UX	User Experience
RASD	Requirements Analysis and Specification Document
DD	Design Document
S2B	System To Be

1.3.3 Abbreviations

Abbreviations	Meaning
WP	World Phenomena
SP	Shared Phenomena
G	Goal
R	Requirement
UC	Use Case
e.g.	exempli gratia
i.e.	id est
DB	Database
API	Application Programming Interface
DMZ	Demilitarized Zone

1.4 Reference Documents

- Course slides on WeBeep.
- RASD and DD assignment document.
- CodeKataBattle Requirements Analysis and Specification Document.

1.5 Document Structure

The document is structured as follows:

- **Introduction**: The first chapter includes the introduction in which the purpose of the document is explained, then, a brief recall of the concepts introduced in the RASD is given. Finally, important information for the readers is given, i.e. definitions, acronyms, synonyms and the set of reference documents.
- **Architectural Design**: This chapter includes the description of the system architecture, the design choices and the interaction between the components. It includes a component view, a deployment view and a runtime view.
- **User Interface Design**: This chapter includes the description of the user interface design, with mockups and UX diagrams.
- **Requirements Traceability**: This chapter includes the mapping between the requirements defined in the RASD and the design elements specified in the DD.
- **Implementation, Integration and Test Plan**: This chapter includes the description of the implementation, integration and test plan to which developers have to stick in order to produce the correct system in a correct way.
- **Effort Spent**: This chapter includes the description of the effort spent by each member of the group.
- **References**: This chapter includes the list of the documents used to write this document.

Chapter 2

Architectural Design

This section aims to present and analyze the architecture of the S2B in a top-down manner. We discuss about the architectural design choices and the reasons behind them.

2.1 Overview: High-level components and their interactions

The figure shown below represents a high-level description of the components which make up the system.

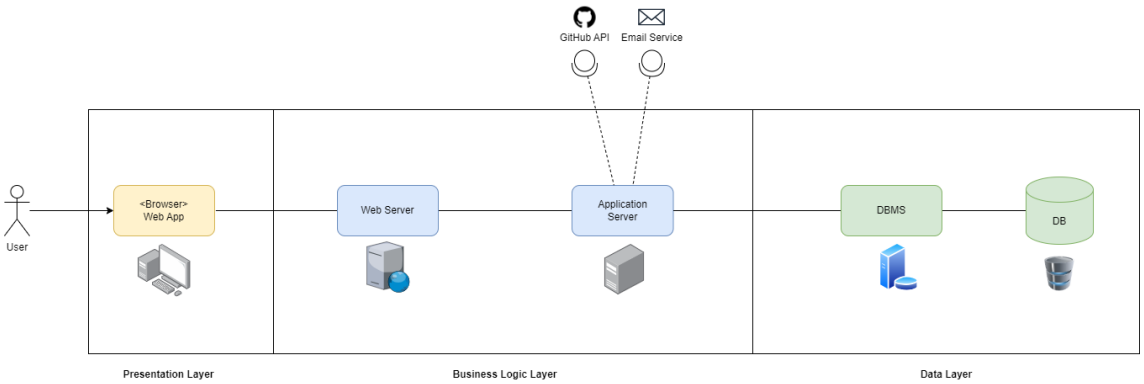

Figure 2.1: Overview CKB architecture

A web interface will be used to access the platform. The overall architecture of the system is based on a three-tier architecture, with the application servers interacting with a database management system and using APIs to retrieve and store data.
The three logical layers correspond to three different physical layers and each layer can communicate only with the adjacent ones. The interactions between clients and server are stateless according to the REST architectural style.
The web server is responsible for the communication with the clients and for the management of the requests.
The application server holds the business logic of the application and it can communicate with the database server to retrieve and store data, also with the GitHub platform and the Email Service.

2.2 Component view

2.2.1 High level view

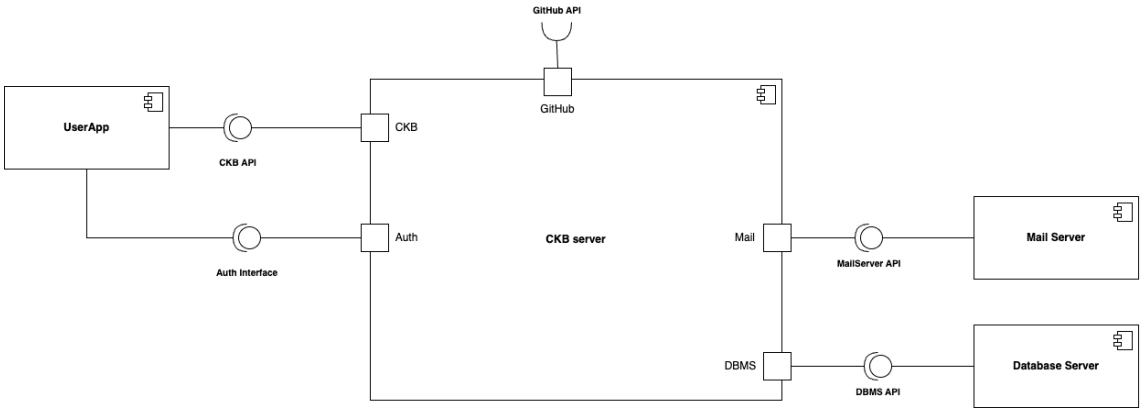

Figure 2.2: High level component view

The figure above shows the system components and interfaces at a high level.

- **CKBServer:** it represents the core of the CKB system and it contains all the business logic.
- **UserApp:** it represents the web application used by the users to access the CKB platform. Users register and log into the system through the **AuthInterface** and access the functionalities offered by the system through the **CKB API**.
- **DatabaseServer:** it represents the DBMS used to store the data of the system. The system can access the data through the **DBMS API**.
- **MailServer:** it represents the mail server used to send notifications to the users. The system can access the mail server through the **MailServer API**.

2.2.2 CKB Server detailed view

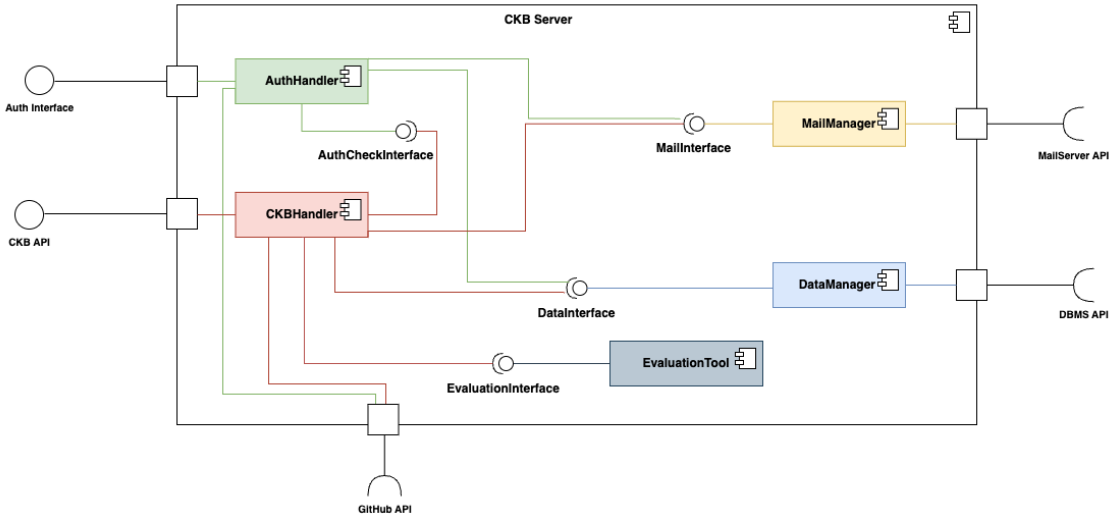

Figure 2.3: CKB server component view

The figure above shows the internal components of the CKB server.

- **AuthHandler:** it handles the login and registration of the users. It offers the **AuthCheckInterface** to allow other components to check if a user is authorized. It communicates with the **MailInterface** to manage confirmation emails and with the **DataInterface** to check the credentials correctness.
- **CKBHandler:** it handles all the functionalities offered by the system. It communicates with the **DataInterface** to retrieve and store data, with the **AuthCheckInterface** to check if a user is authorized to perform a certain operation, with the **MailInterface** to send notifications to the users, and with the **EvaluationInterface** to perform the evaluation of the students' projects. It also communicates with the **GitHubInterface** to retrieve data (nickname and pushed code) from the GitHub platform.
- **MailManager:** it handles the access to the mail server, it provides the **MailInterface** that allows components to send emails.
- **DataManager:** it handles the access to the persistent data saved on the DB, almost every component communicates with it through the **DataInterface**.
- **EvaluationTool:** it handles the evaluation of the students' code based on the battle rules. It retrieves this information from the **CKBHandler**. The specific analysis tools are chosen during the implementation phase based on the programming languages that the system will support (e.g., Pylint for Python, CLang for C/C++, ...).

2.2.3 Auth component view

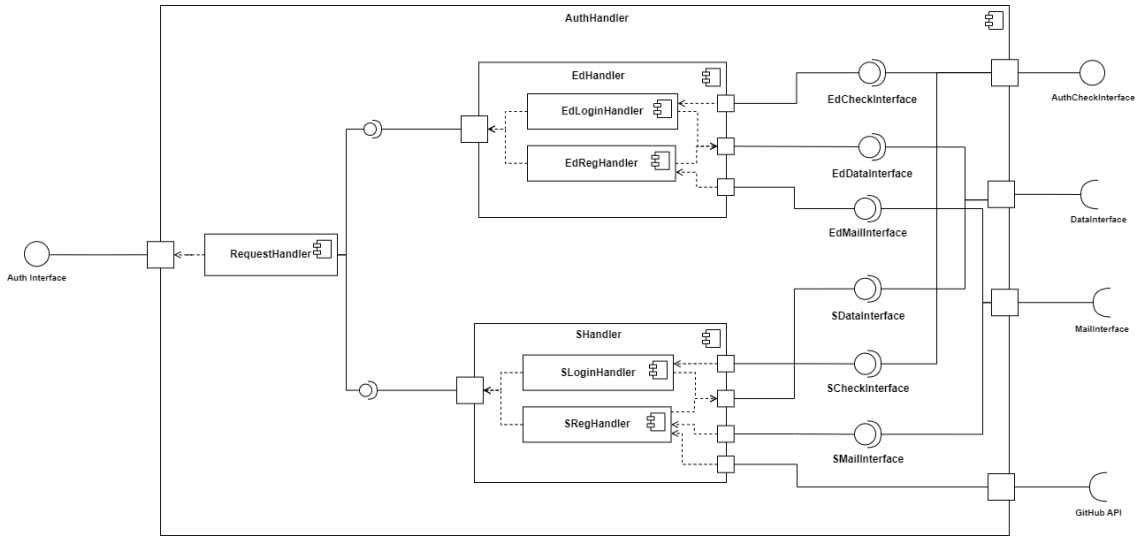

Figure 2.4: Auth component view

The figure above shows the internal components of the AuthHandler.

- **RequestHandler:** it handles the requests acting as a router dispatching requests to the right handler.
- **EdHandler:** it is composed of the **EdLoginHandler** and the **EdRegHandler**. The former handles the login of educators, the latter handles their registration.
- **SHandler:** it is composed of the **SLoginHandler** and the **SRegHandler**. The former handles the login of students, the latter handles their registration.

LoginHandlers communicate with the **AuthCheckInterface** to check the authorization of a user.
RegistrationHandlers communicate with the **MailInterface** to send confirmation emails and with the **DataInterface** to store the data of the new user.
SRegistrationHandler also communicates with **GitHubAPI** to link the CKB account with the GitHub one.

2.2.4 CKB component view

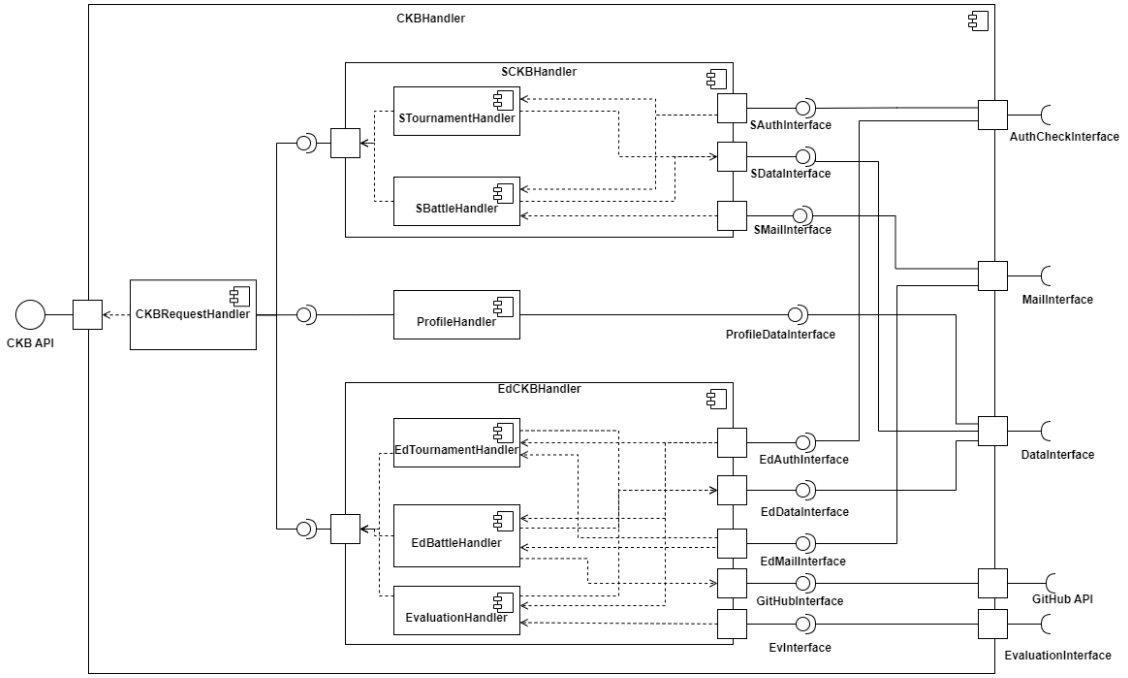

Figure 2.5: CKB component view

The figure above shows the internal components of the CKBHandler.

- **CKBRequestHandler:** it handles the requests acting as a router dispatching requests to the right handler.
- **EdCKBHandler:** it is composed of the **EdTournamentHandler**, the **EdBattleHandler** and the **EvaluationHandler**. EdTournamentHandler is responsible for the creation and the conclusion of tournaments, EdBattleHandler is responsible for the creation of battles, and EvaluationHandler is responsible for the evaluation of the students' code.
- **SCKBHandler:** it is composed of the **STournamentHandler** and the **SBattleHandler**. STournamentHandler is responsible for the visualization and the subscription to tournaments, SBattleHandler is responsible for the subscription to battles.
- **ProfileHandler:** it is responsible for the visualization of the profile of a student.

All components communicate with the **AuthCheckInterface** to check the authorization of a user to perform the requested operation.
TournamentHandlers and BattleHandlers communicate with the **DataInterface** to retrieve and store data, and with the **MailInterface** to send notifications to the users.
EdBattleHandler also communicates with the **GitHubAPI** to retrieve the students' pushed code.
EvaluationHandler communicates with the **EvaluationInterface** to evaluate the students' code and with the **DataInterface** to store the results of the evaluation.
ProfileHandler communicates with the **DataInterface** to retrieve the data of the searched student.

2.3 Deployment view

The figure below shows the architecture of the system. All the users access to the WebApp through the browser, which communicates with the Web Server. Both Web Server and Application Server are hosted on a Cloud Provider. This choice offers many advantages, such as:

- **Scalability and flexibility:** the ability of adding and removing resources efficiently through the use of load balancing services which allow the application server to manage traffic and workload.
- **Security:** the ability to protect the application server using firewall and DMZ, against cyber attacks and possible threats.
- **Cost efficiency:** the ability to pay only for the resources used which can help to lower the overall cost.

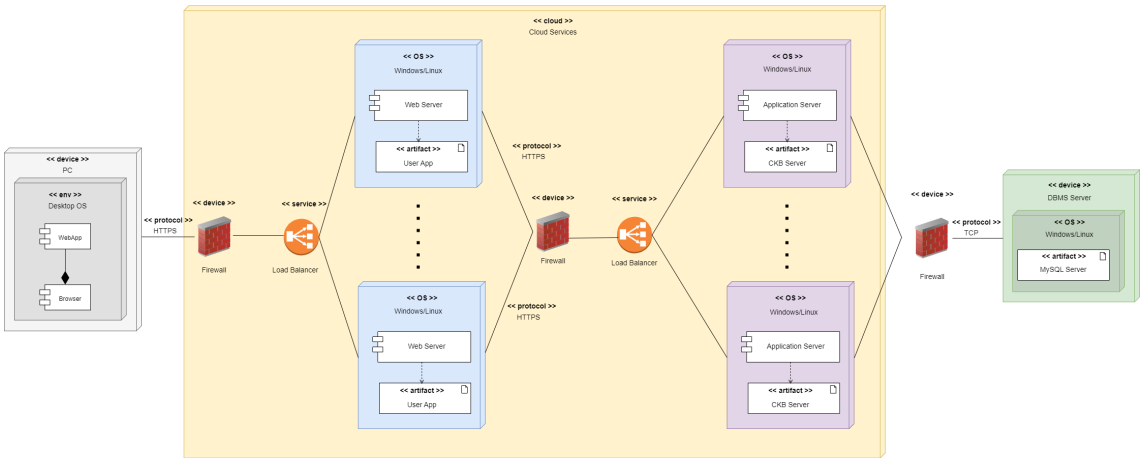

Figure 2.6: Deployment diagram

The deployment diagram offers a more detailed view over the hardware and software components of the system.

- **PC:** any device having a browser capable of running the JavaScript code.
- the Cloud Services will host all the business and data logic for the system. It is characterized by
  - **Firewall:** device used to protect and filter incoming connections to the logic and data layers of the system. It protects the system from unauthorized access and malicious attacks.
  - **Load Balancer:** service used to distribute the workload across multiple servers. It helps to improve the performance and reliability of the system. It also helps to ensure that application can handle a large volume of requests, without any downtime.

  - **Multiple copies of Web Server and Application Server:** they are used to ensure that the system is always available. The different instances can be created and destroyed dynamically, based on the workload. It also helps to achieve fault tolerance by allowing traffic to be redirect to a different instance, if one instance becomes unavailable.

- **Database:** used to store all the data of the system. It uses MySQL as DBMS to retrieve and store data.

2.4 Runtime view

2.4.1 Registration Student

The figure below shows the sequence diagram of the registration of a student. The student fills the form his their email and password, and his personal information (name, surname, date of birth) and also his GitHub nickname. The database checks if the email is already used, if not the system sends a confirmation email to the student to complete the registration. Also, the system checks if the GitHub nickname of the student is valid, if so the system saves the student's credentials and the registration is completed.

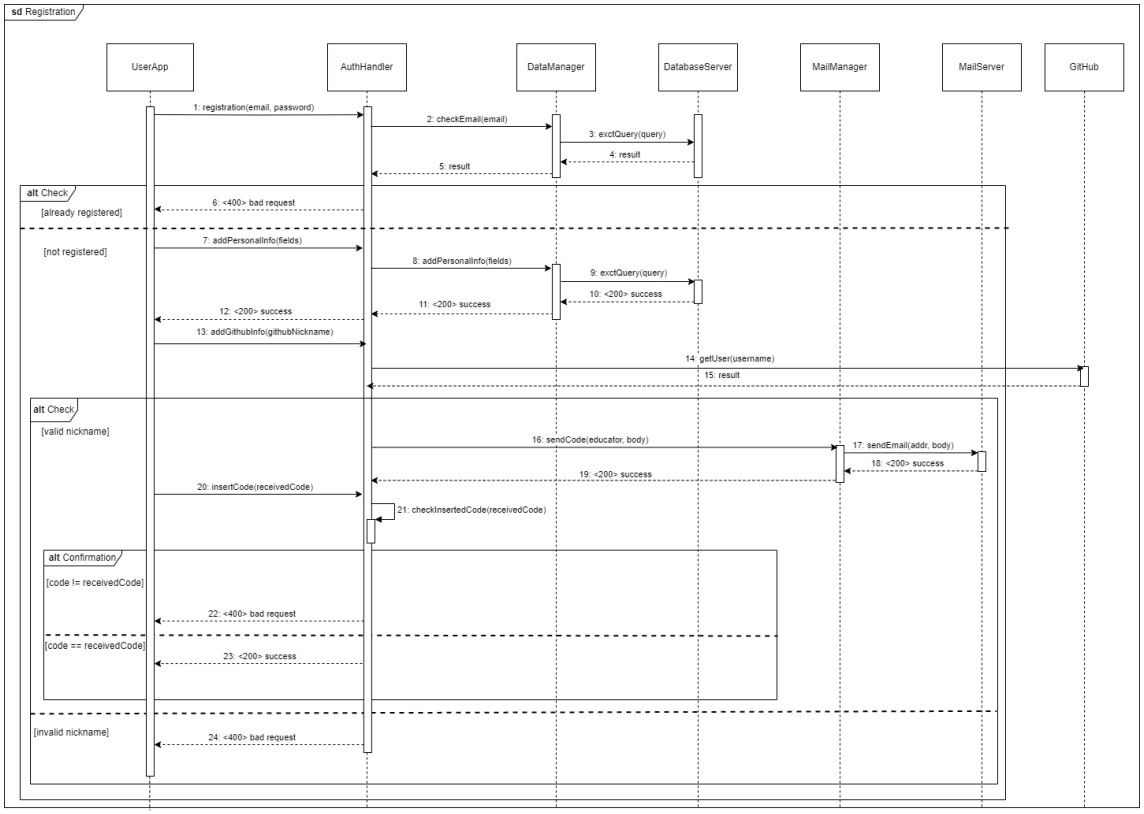

Figure 2.7: Registration Student

2.4.2 Registration Educator

The figure below shows the sequence diagram of the registration of an educator. The educator fills the form with his email and password, and his personal information (name, surname, date of birth) and submits it. The database checks if the email is already used, if not the system sends a confirmation email to the educator to complete the registration. The system saves the educator's credentials and the registration is completed.

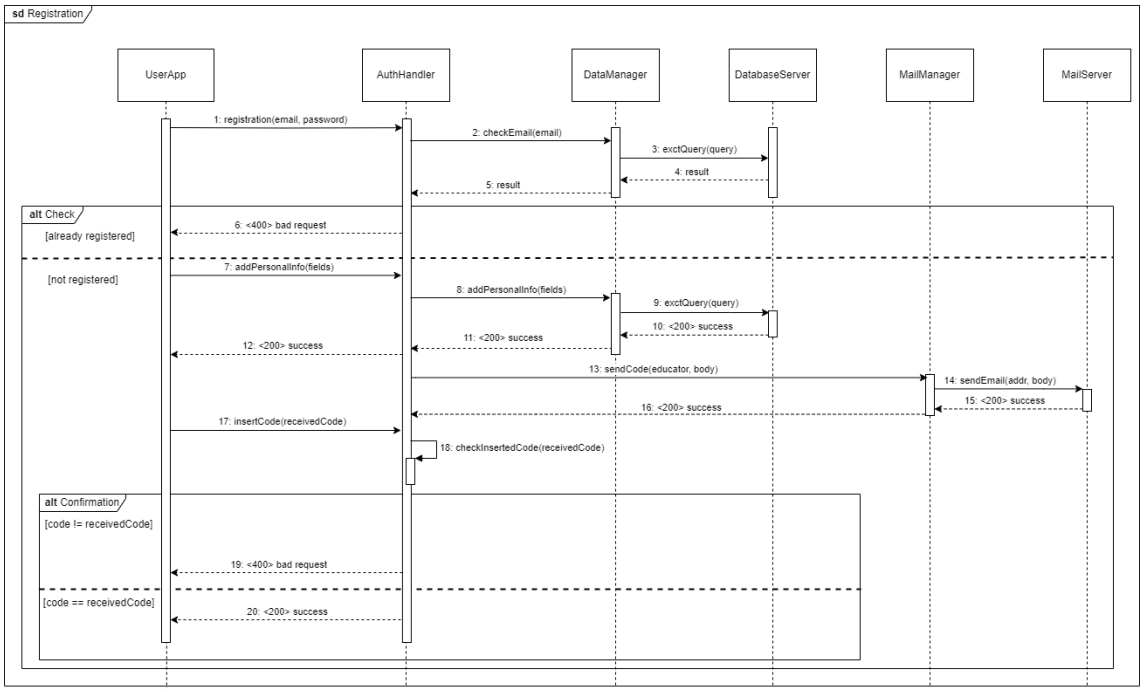

Figure 2.8: Registration Educator

2.4.3 Login

The figure below shows the sequence diagram of the user login. The user fills the form with his email and password and submits it. The system checks if the credentials are stored and valid, if so the user is logged in, otherwise the system shows an error message.

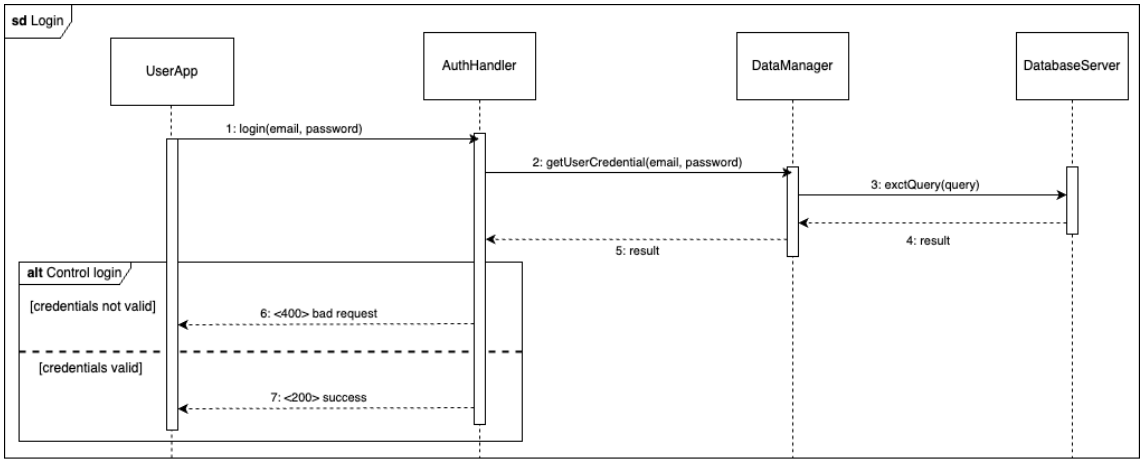

Figure 2.9: Login

2.4.4 Creation of the tournament

The figure below shows the sequence diagram of the creation of a tournament. The educator fills the form with the tournament details and submits it. The system checks if the data is valid, and if the badge option is set to true, the system creates the badge. After that the system sends an email to all the invited colleagues and to all the students subscribed to the CKB platform. The tournament is created and the educator is redirected to the tournament page.

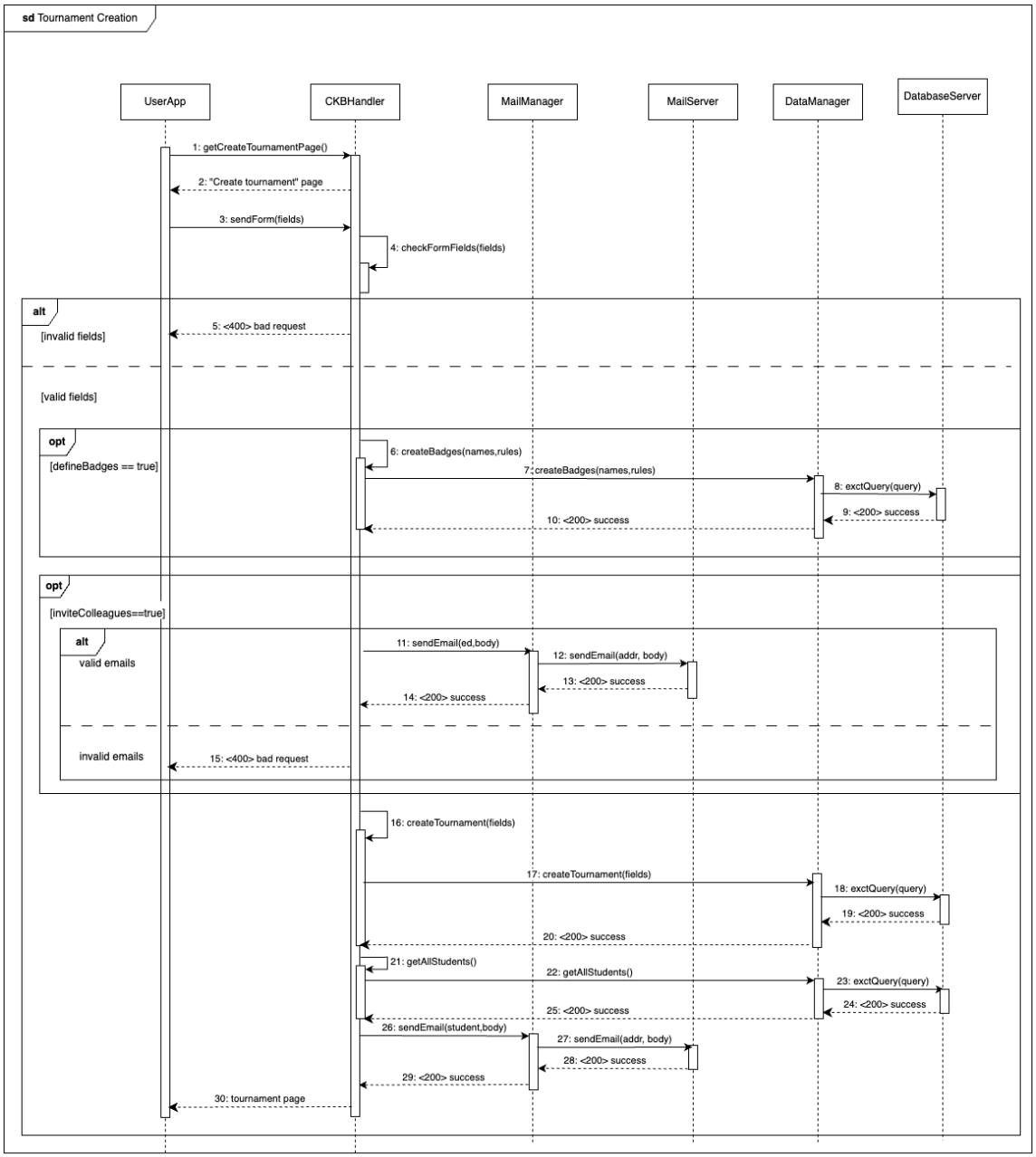

Figure 2.10: Creation of the tournament

2.4.5 Creation of the battle

The figure below shows the sequence diagram of the creation of a battle. After the educator clicked on the "Create Battle" button, he fills the form with the battle details and submits it. The system checks if the data is valid. When the battle is created, the system sends an email to all the students subscribed to the correspond tournament.

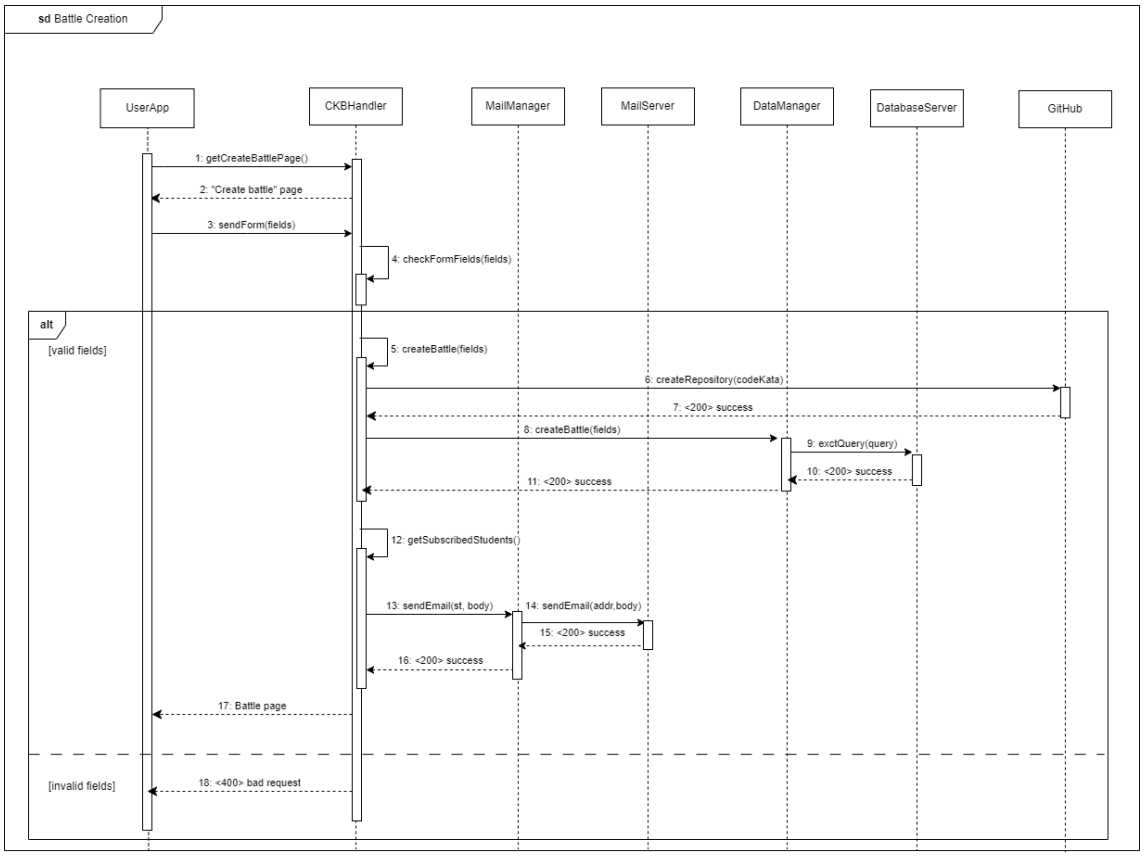

Figure 2.11: Creation of the battle

2.4.6 Tournament visualization

The figure below shows the sequence diagram of the visualization of a tournament. First the user requests the list of ongoing tournaments, then the user selects the tournament he wants to visualize the information.

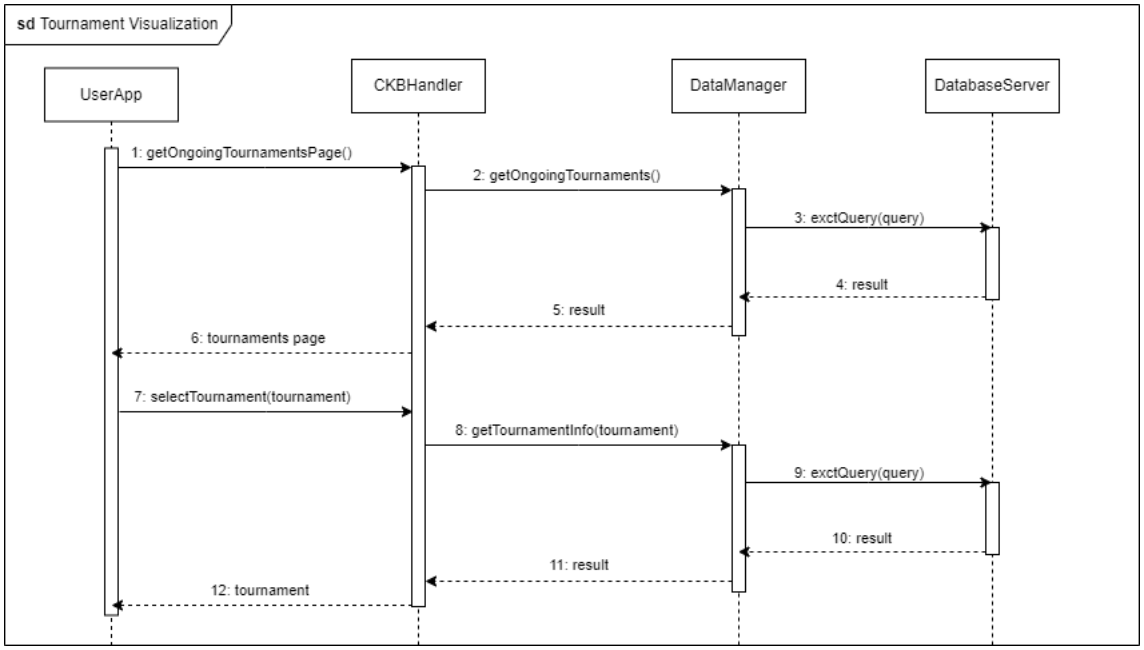

Figure 2.12: Tournament visualization

2.4.7 Joining a tournament

The figure below shows a student signing up for a tournament and receiving an email notification about it.

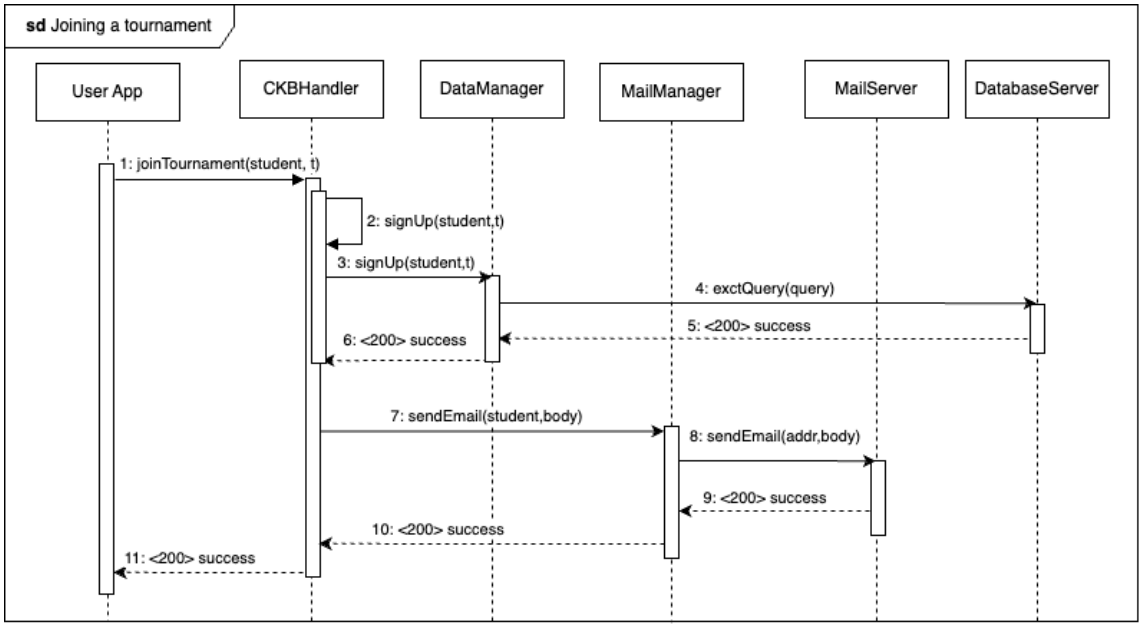

Figure 2.13: Joining a tournament

2.4.8 Joining a battle as a singleton

The figure below shows a student joining a battle as a singleton and receiving an email notification about it. We assume that the battle configuration allows students to participate on their own.

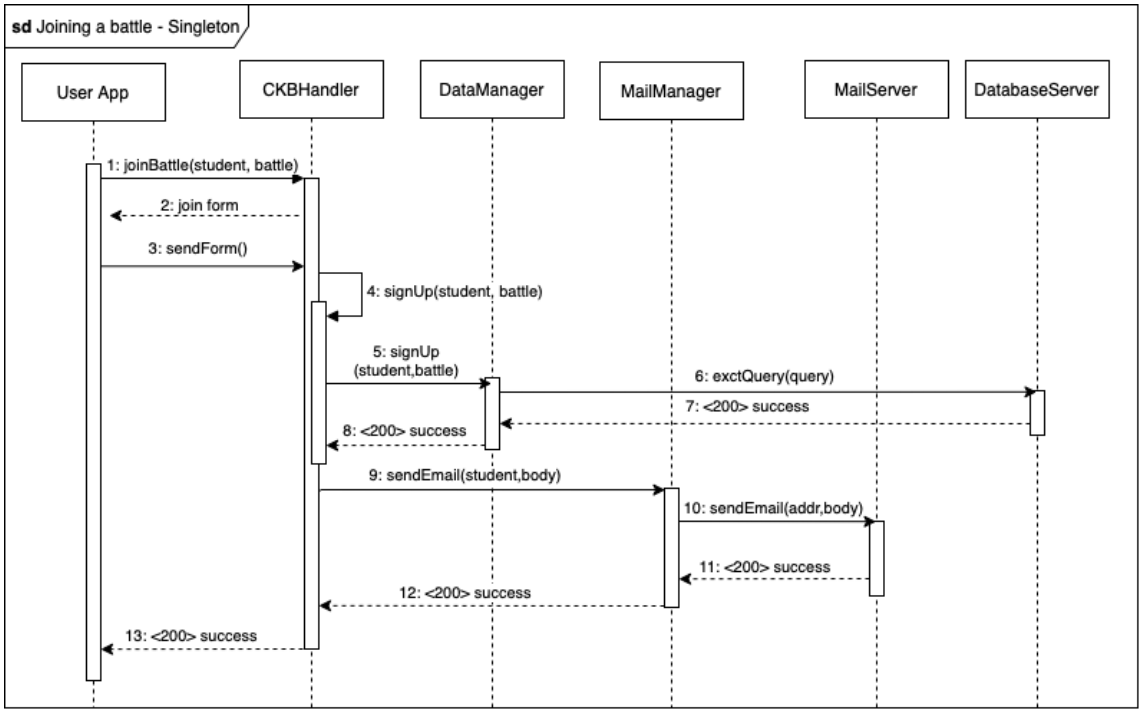

Figure 2.14: Joining a battle as a singleton

2.4.9 Joining a battle by creating a group

The figure below shows a student creating a group for a battle and receiving an email notification about it.

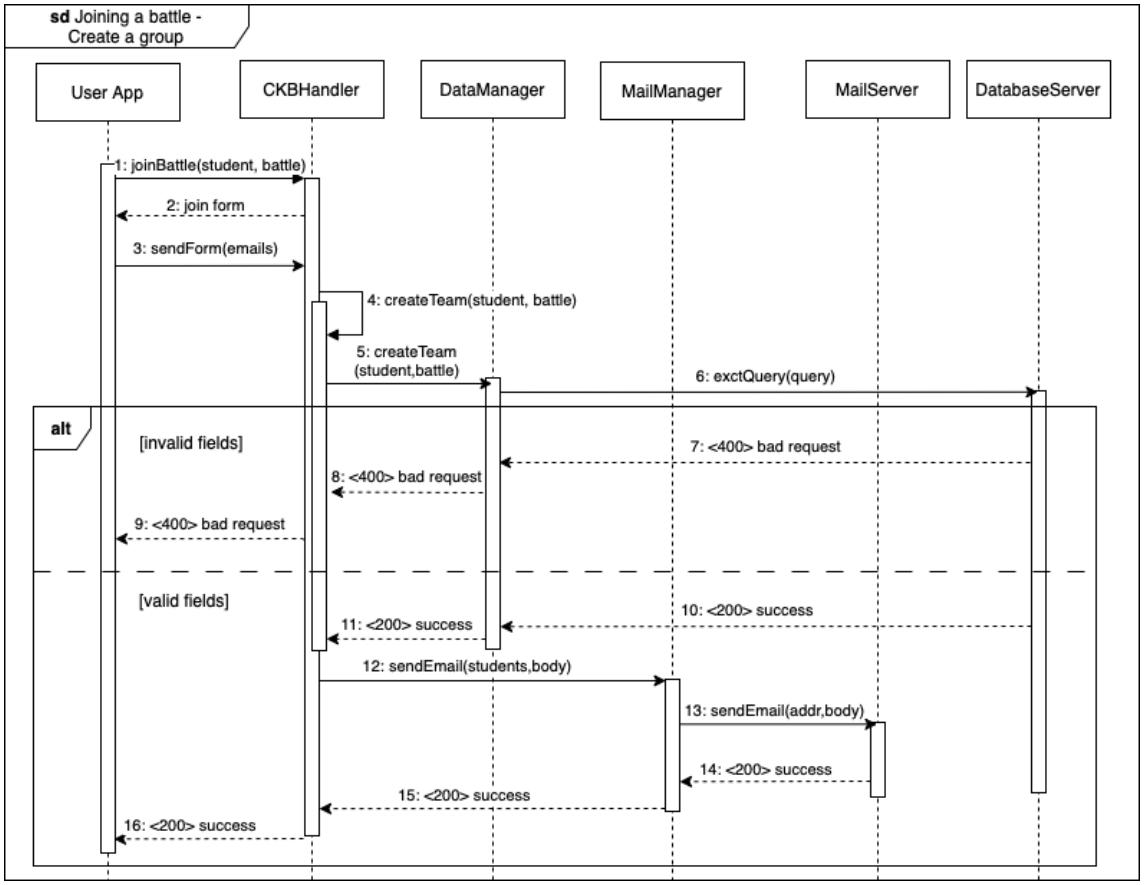

Figure 2.15: Joining a battle by creating a group

2.4.10 Joining a battle by joining a group

The figure below shows a student joining a group for a battle and receiving an email notification about it. If the minimum number of participants is reached, the team is signed up for the battle.

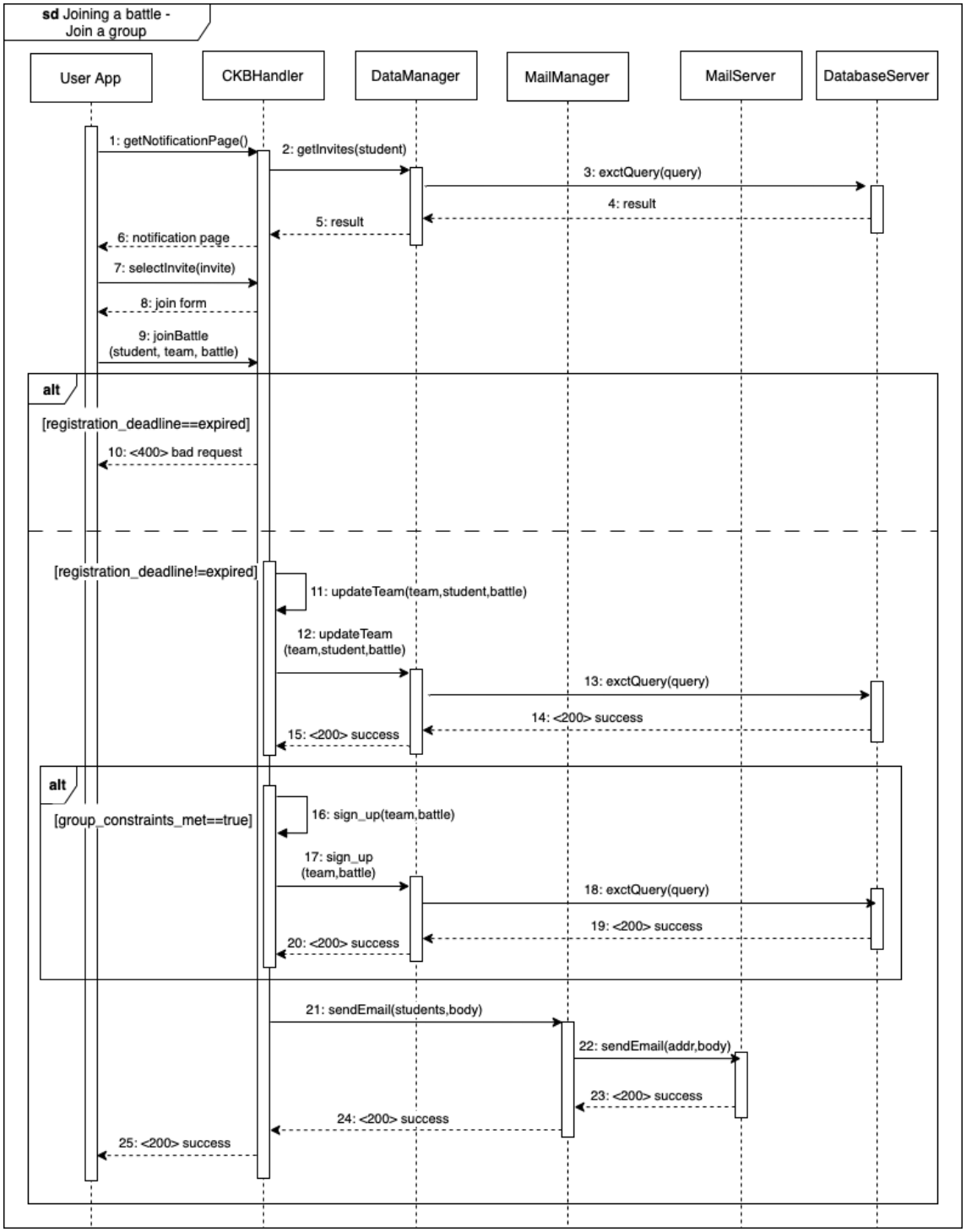

Figure 2.16: Joining a battle by joining a group

2.4.11 Execution of the battle

The figure below shows the sequence diagram of the execution of a battle. After GitHub notifies the system that a student pushed code, the system evaluates automatically the code based on the requirements chosen by the educator at the creation of the battle and updates the battle score of the student.

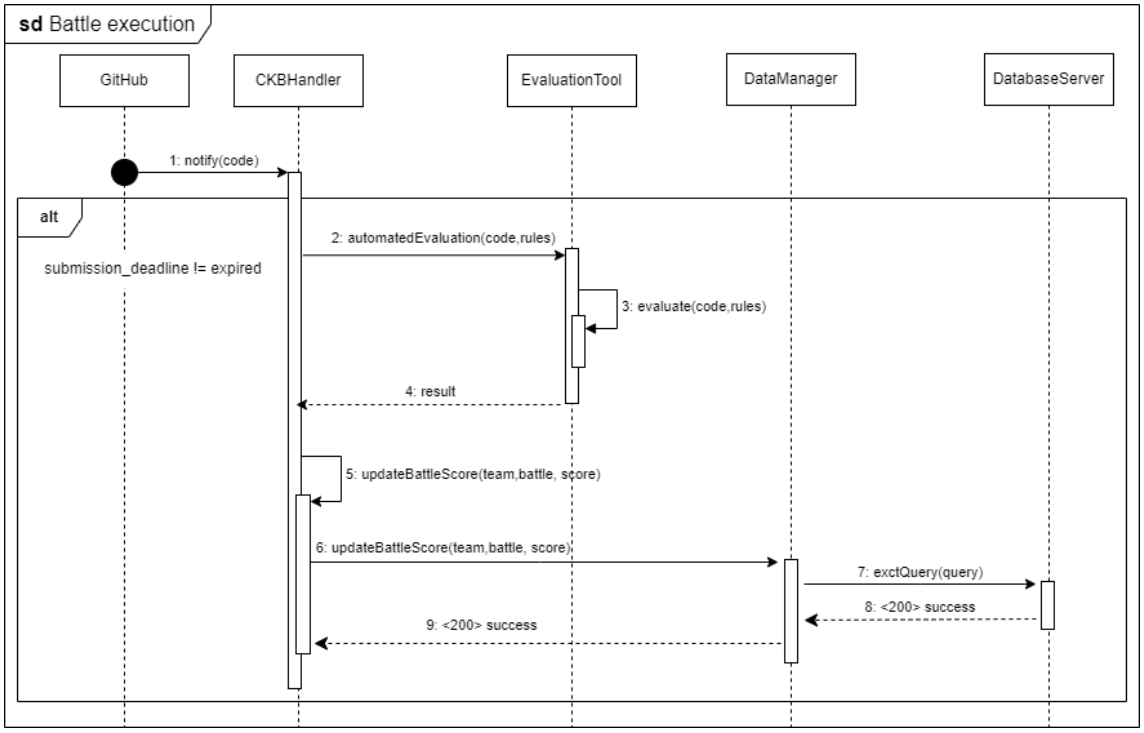

Figure 2.17: Execution of the battle

2.4.12 Conclusion of the battle

The figure below shows the sequence diagram of the conclusion of a battle. If required, the educator performs the manual evaluation and the system updates the battle scores. Then, the system updates the tournament score of the students who participated in the battle and the tournament rank. The system sends an email to all the students subscribed to the tournament to notify them that the updated tournament rank is available.

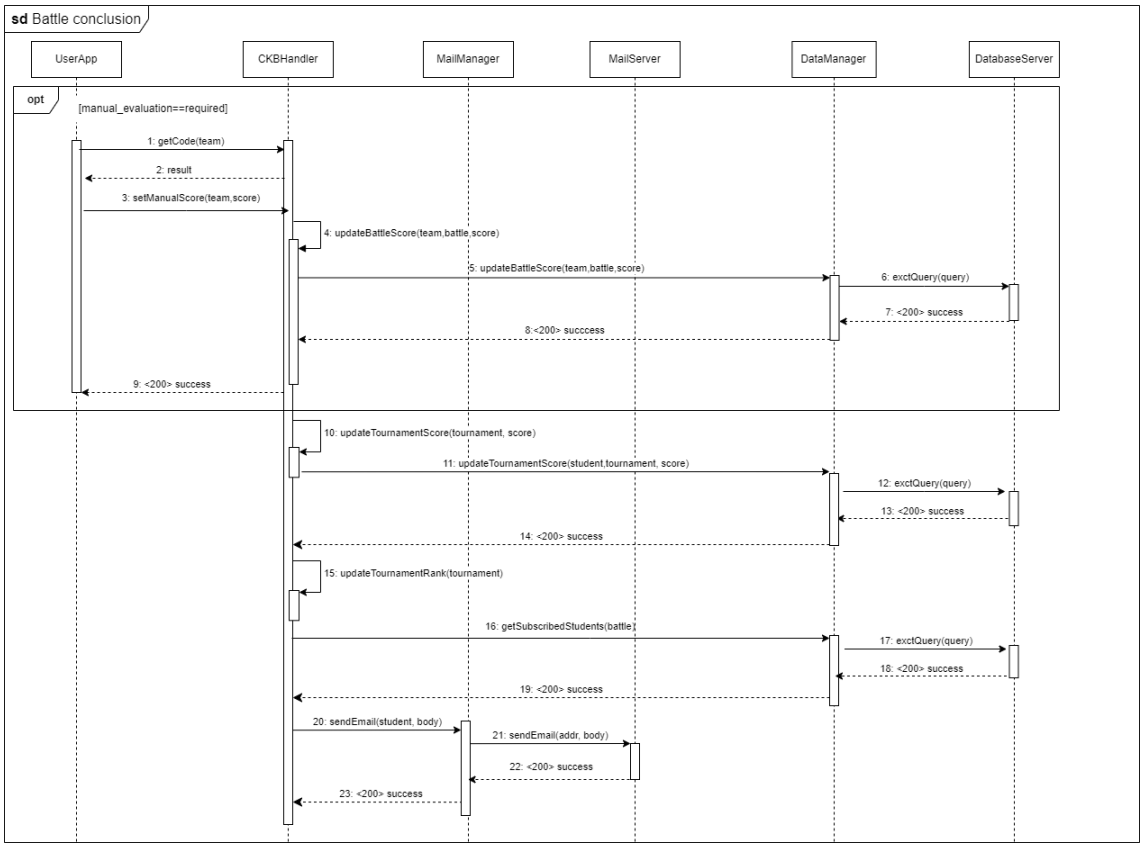

Figure 2.18: Conclusion of the battle

2.4.13 Conclusion of a tournament

The figure below shows the sequence diagram of the conclusion of a tournament. The educator chooses to conclude the tournament, so the system awards badges to eligible students and sends an email notification to all subscribed students.

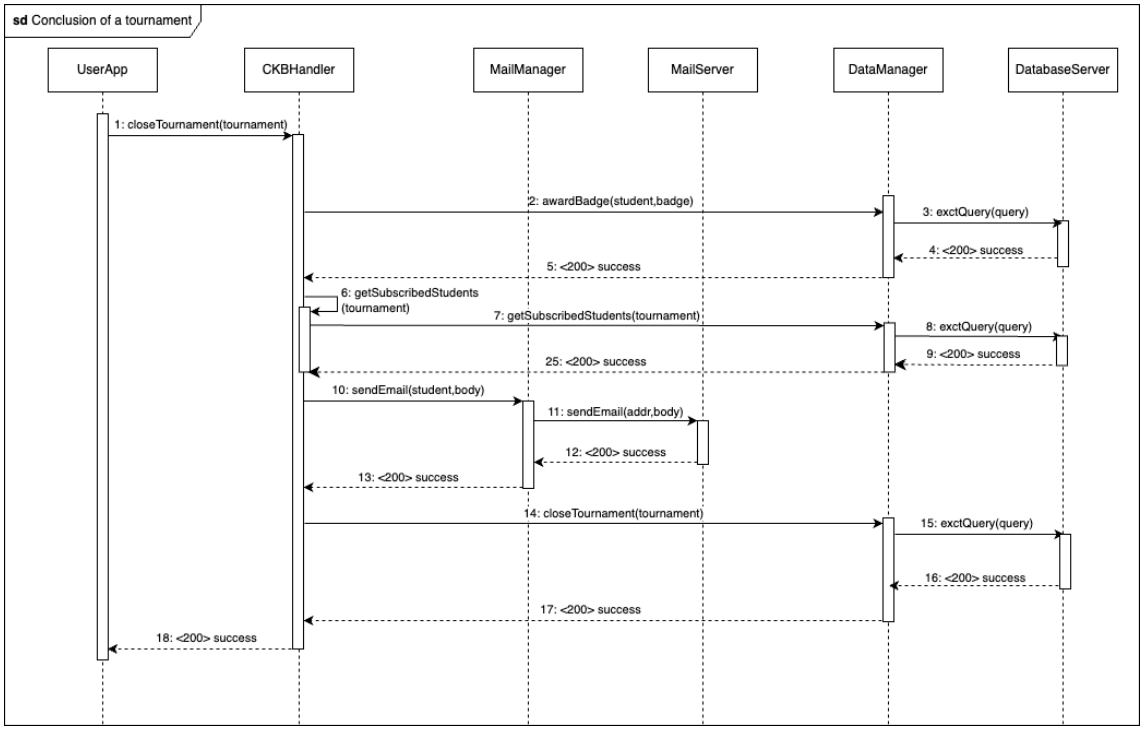

Figure 2.19: Conclusion of a tournament

2.4.14 Profile visualization

The figure below shows the sequence diagram of the visualization of the profile of a student. The system retrieves the data of the searched student from the database if the student is registered to the system, otherwise it sends an error message.

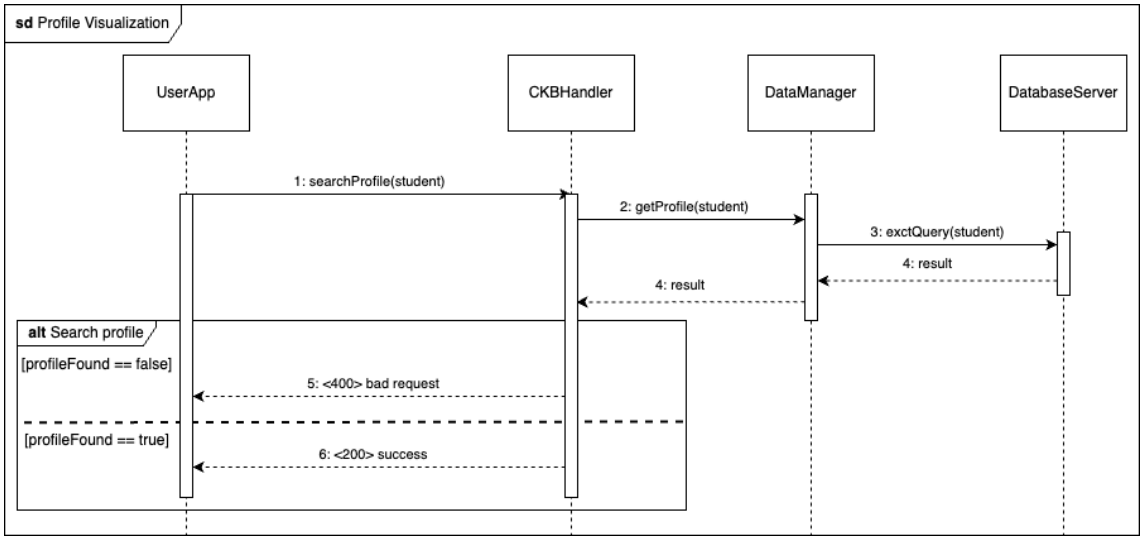

Figure 2.20: Profile visualization

2.5 Component interfaces

In this section, the figure below depicts the interface diagram with the component interfaces and their specific methods and connections.

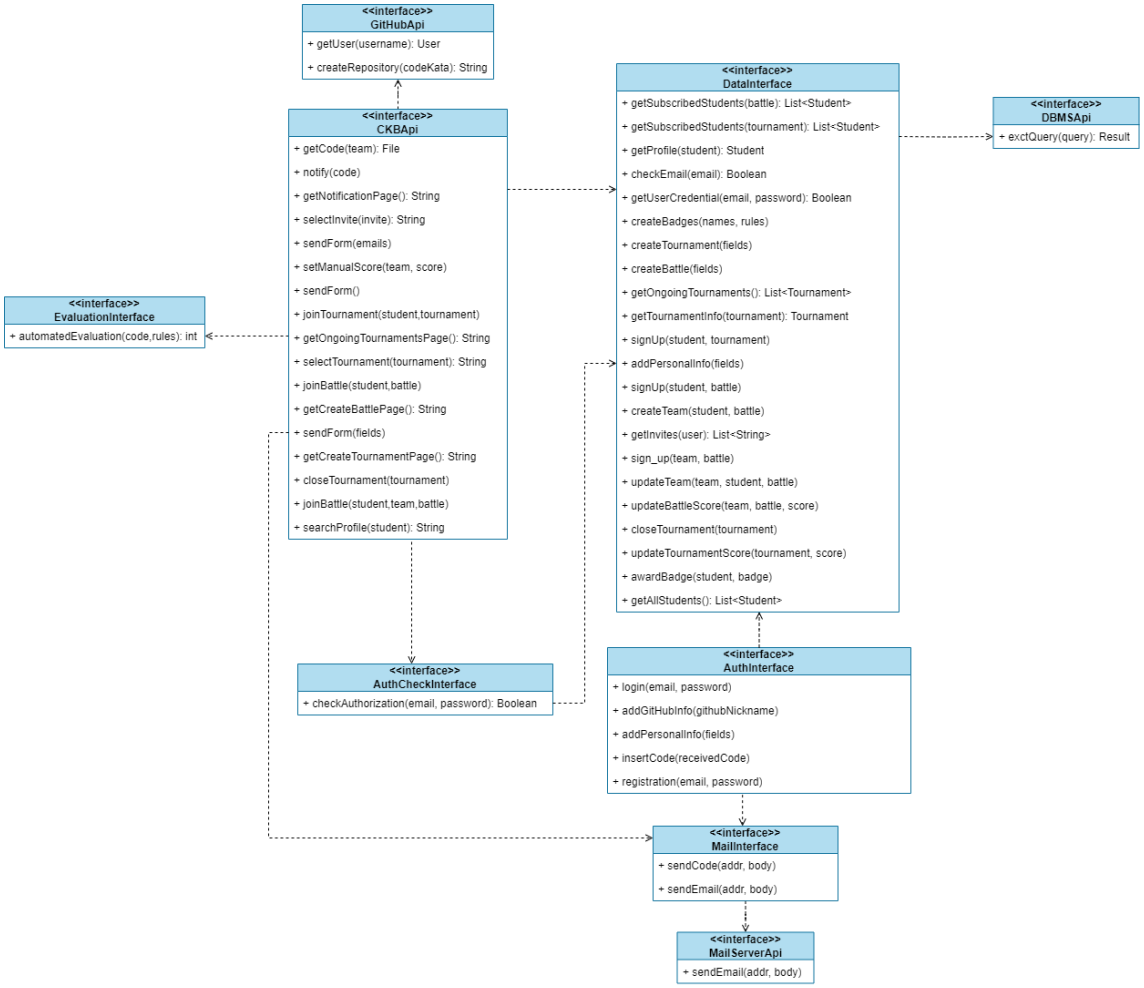

Figure 2.21: Interface diagram

2.6 Selected architectural styles and patterns

- **Three-tier architecture:** the system is based on a three-tier architecture, with every tier (presentation, application and data) having a corresponding layer. The separation of the layers allows to scale the application efficiently by distributing the workload across multiple servers. It also allows to improve the security of the system by protecting the data layer. Finally, it improves the maintainability of the system by allowing to modify a single layer without affecting the other ones.
- **REST architectural style:** the system is based on the REST architectural style. REST is a software architectural style that defines a standard language for Web services. It is stateless and it makes easy to integrate with other systems thanks to the uniform interface. It also allows for a clear separation between the client and the server. Clients are not concerned with the internal state of the server, promoting loose coupling and enhancing system maintainability.

2.7 Other design decisions

2.7.1 ER diagram

The figure below shows the Entity-Relationship diagram of the database used by the system.

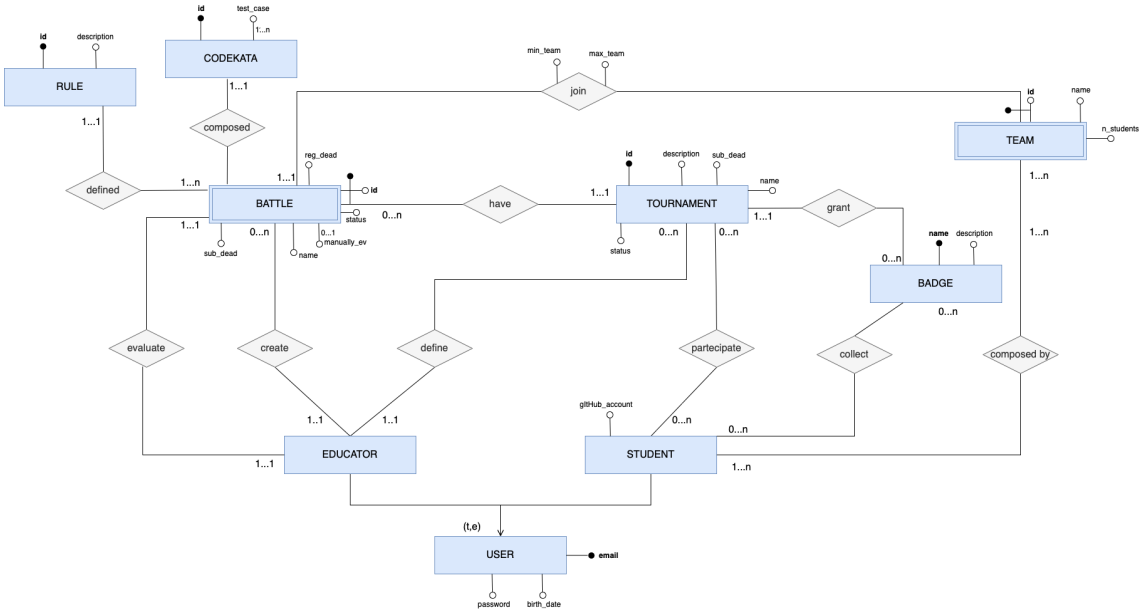

Figure 2.22: ER diagram

Chapter 3

User Interface Design

3.1 Mockups

The user can access or register to the application using a Web Interface. The Mockups of the interface are shown in the following figures.

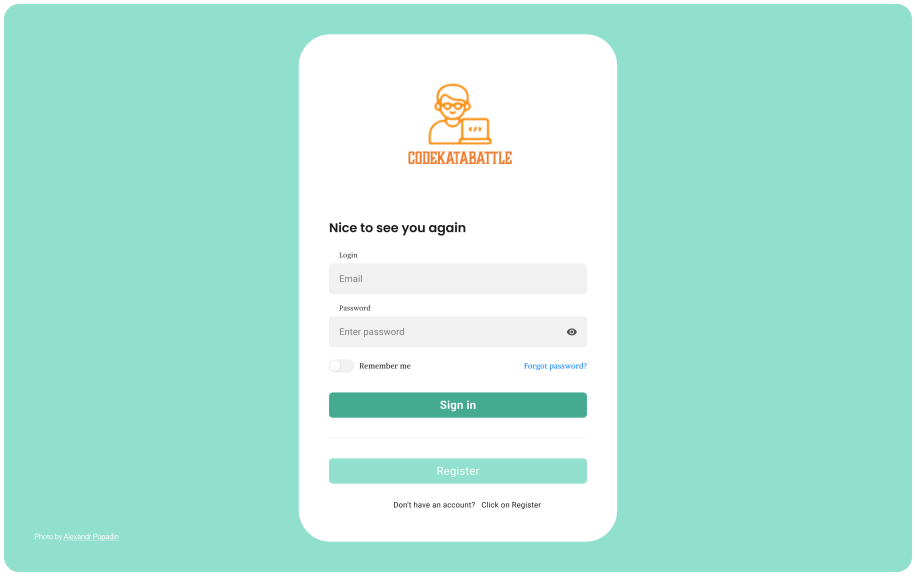

Figure 3.1: Login page - same for all types of users

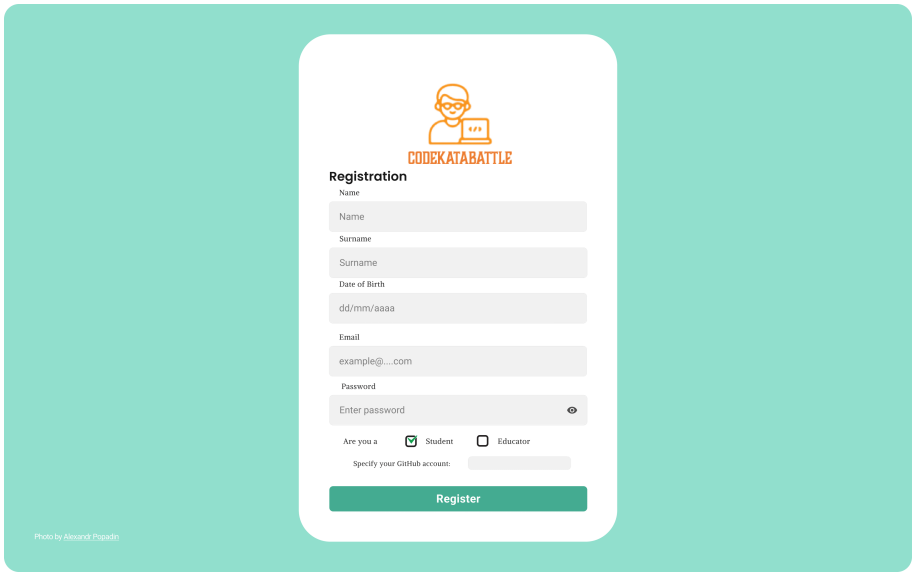

Figure 3.2: Registration page - same for all types of users

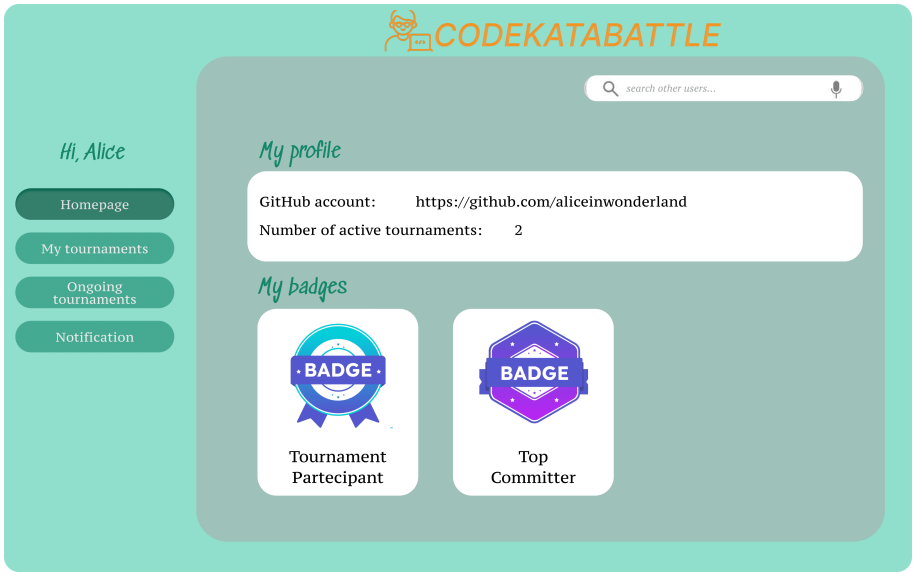

Figure 3.3: Student's homepage and profile

Figure 3.4: My tournaments - student's page

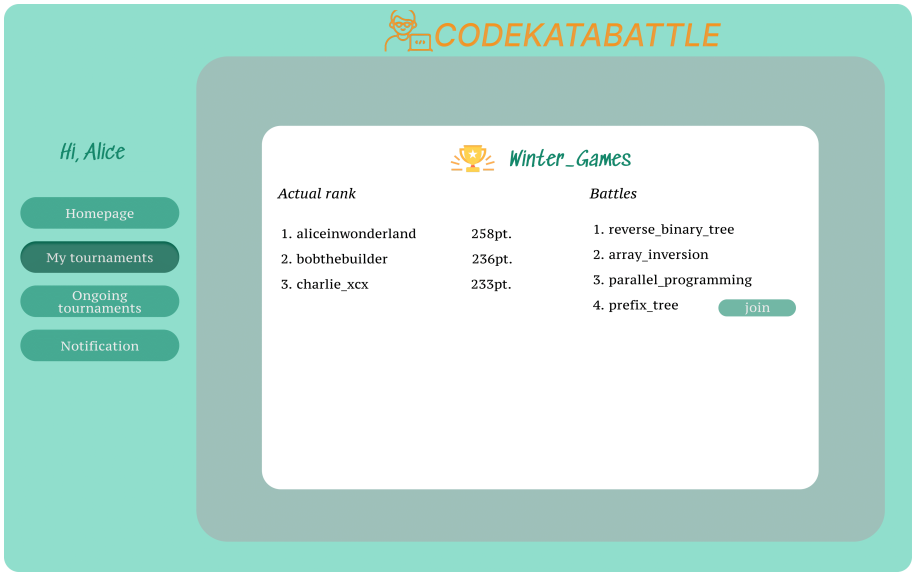

Figure 3.5: Tournament's details - student's page

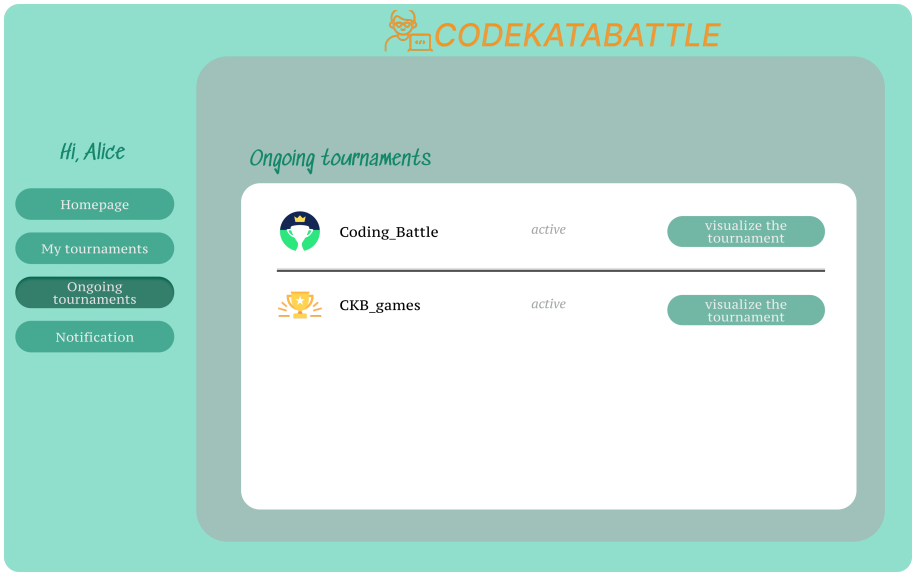

Figure 3.6: Ongoing tournaments - student's page

Figure 3.7: Join a new tournament - student's page

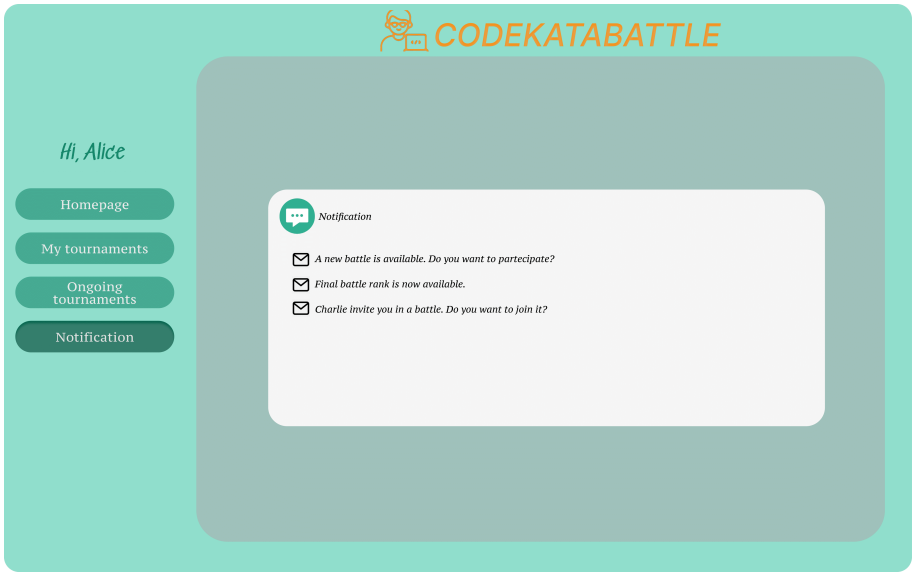

Figure 3.8: Notification page - student's page

Figure 3.9: Join a new battle - student's page

Figure 3.10: Educator's homepage

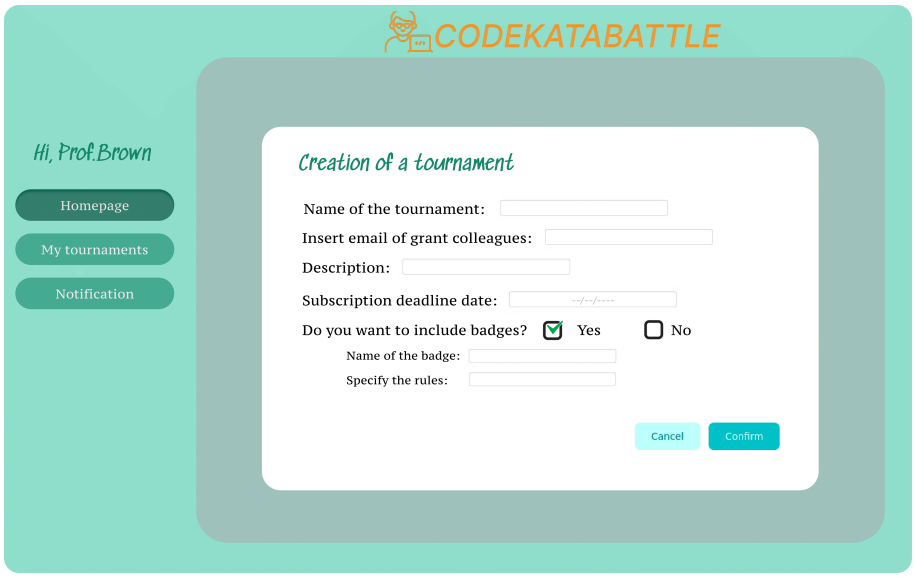

Figure 3.11: Creation of a tournament page - educator's page

Figure 3.12: My tournaments page - educator's page

Figure 3.13: Tournament's details - educator's page

Figure 3.14: Creation of a battle page - educator's page

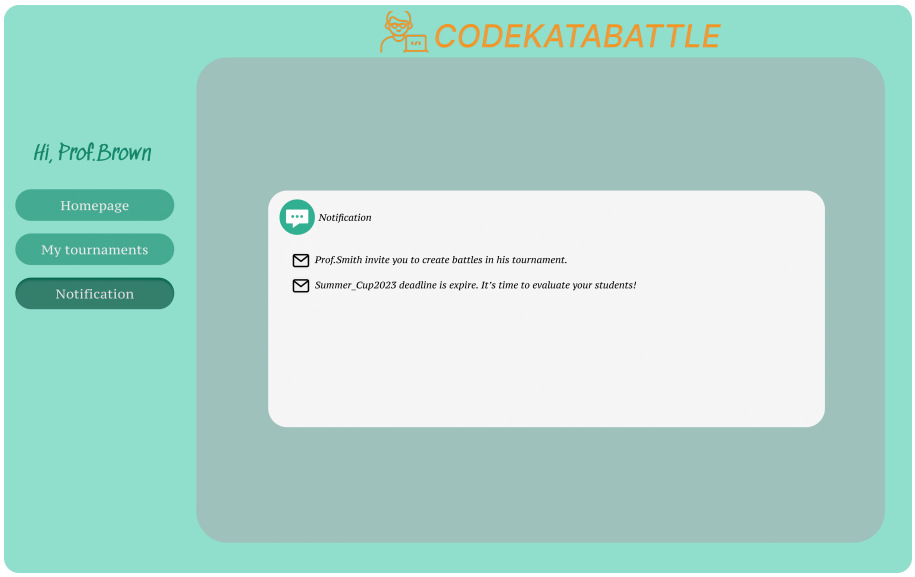

Figure 3.15: Notification page - educator's page

- **Figure 3.1:** the login page allows registered users (both educators and students) to access the application.
- **Figure 3.2:** the registration page allows new users (both educators and users) to register to the application. They have to specify if they are educators or students.
- **Figure 3.3:** the student's homepage shows the profile of the student and the list of the collected badges.
- **Figure 3.4:** "my tournaments" page shows the list of the tournaments in which the student is enrolled.

- **Figure 3.5:** "tournament's details" page shows the details of the selected tournament. The student can see the actual rank and the completed battles. They can also join a battle, if available.
- **Figure 3.6:** "ongoing tournaments" page shows the list of the tournaments in the platform which deadline isn't already expired. The student can view the details of the tournament.
- **Figure 3.7:** "join a new tournament" page allows the student to join a new tournament. They can visualize the actual rank and the completed battles of that tournament.
- **Figure 3.8:** "notification" page shows the list of the notifications received by the student.
- **Figure 3.9:** "join a new battle" page allows the student to join a new battle available in the selected tournament. They can read all the details and rules of the battle and, if they want to join, they have to choose the team.
- **Figure 3.10:** the educator's homepage shows the educator's main page.
- **Figure 3.11:** "creation of a tournament" page allows the educator to create a new tournament. They have to specify the name, the description, the deadline, the rules and the granted colleagues. They can also choose to include badges.
- **Figure 3.12:** "my tournaments" page shows the list of the tournaments created by the educator.
- **Figure 3.13:** "tournament's details" page shows the details of the selected tournament to the educator. The educator can see the actual rank and the completed battles. They can also create a new battle, if authorized.
- **Figure 3.14:** "creation of a battle" page allows the educator to create a new battle. They have to specify the name, the code kata, the bound for each team, the final submission deadline. They can also choose to include a manual evaluation.
- **Figure 3.15:** "notification" page shows the list of the notifications received by the educator.

3.2 User Interface Flow Diagram

The following diagram shows the connection between the different pages of the application.

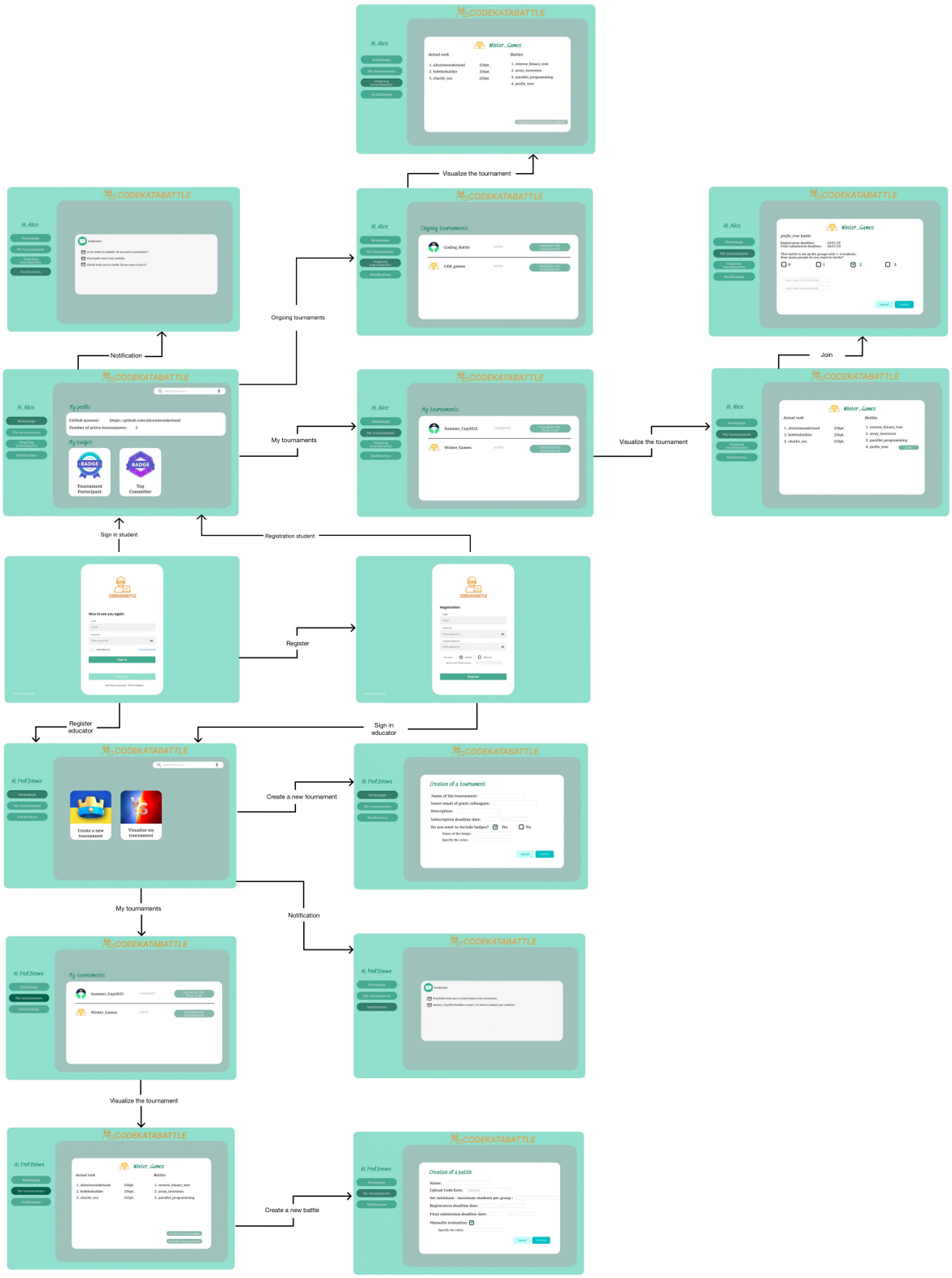

Figure 3.16: User Interface Flow Diagram

Chapter 4

Requirements Traceability

In this section, it is provided the mapping between the requirements defined in the RASD and the components defined in the DD.

R1 The system must allow the user to register himself to the system by providing all the mandatory information.
UserApp, **AuthHandler**, **RequestHandler**, **EdRegHandler** if the user is an educator, **SRegHandler** if the user is a student, **DataManager**, **DatabaseServer**

R2 The system must allow the students to link their GitHub account.
UserApp, **AuthHandler**, **RequestHandler**, **SRegHandler**, **DataManager**, **DatabaseServer**

R3 The system must verify the uniqueness of the email of the user to be registered.
AuthHandler, **RequestHandler**, **EdRegHandler** if the user is an educator,**SRegHandler** if the user is a student, **DataManager**, **DatabaseServer**

R4 The system must send a confirmation on the user's email after his registration.
AuthHandler, **RequestHandler**, **EdRegHandler** if the user is an educator, **SRegHandler** if the user is a student, **MailManager**, **MailServer**, **DataManager**, **DatabaseServer**

R5 The system must allow the user to log into their account by entering their email and password.
UserApp, **AuthHandler**, **RequestHandler**, **EdLoginHandler** if the user is an educator, **SLoginHandler** if the user is a student, **DataManager**, **DatabaseServer**

R6 The system must allow educators to create tournaments.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EdTournamentHandler**, **DataManager**, **DatabaseServer**

R7 The system must allow the tournament creator to invite other educators.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EdTournamentHandler**

R8 The systems must allow the tournament creator to end the tournament.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EdTournamentHandler**, **DataManager**, **DatabaseServer**

R9 The system must allow the tournament creator to set the tournament configuration.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EdTournamentHandler**, **DataManager**, **DatabaseServer**

R10 The system must allow the tournament creator to define badges.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EdTournamentHandler**, **DataManager**, **DatabaseServer**

R11 The system must allow students to collect badges.
CKBHandler, **DataManager**, **DatabaseServer**

R12 The system must assign badges to eligible students.
CKBHandler, **DataManager**, **DatabaseServer**

R13 The system must notify via email students when a tournament they subscribed to ends.
CKBHandler, **MailManager**, **MailServer**

R14 The system must notify via email students when they are awarded a badge.
CKBHandler, **MailManager**, **MailServer**

R15 The system must notify via email educators when it is time to evaluate students' project.
CKBHandler, **MailManager**, **MailServer**

R16 The system must notify via email educators when they are invited to join a tournament.
CKBHandler, **MailManager**, **MailServer**

R17 The system must allow authorized educators to create battles.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EdBattleHandler**, **DataManager**, **DatabaseServer**

R18 The system must allow the battle creator to upload the code kata.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EdBattleHandler**, **DataManager**, **DatabaseServer**

R19 The system must allow the battle creator to set the battle configuration.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EdBattleHandler**, **DataManager**, **DatabaseServer**

R20 The system must allow the educator to manually evaluate the students' project.
UserApp, **CKBHandler**, **CKBRequestHandler**, **EvaluationHandler**, **DataManager**, **DatabaseServer**

R21 The system must send the link to the repository to all students subscribed to the battle.
CKBHandler, **MailManager**, **MailServer**, **DataManager**, **DatabaseServer**

R22 The system must create the GitHub repository containing the code kata.
CKBHandler, **DataManager**, **DatabaseServer**

R23 The system must notify via email students when a new tournament is created.
CKBHandler, **MailManager**, **MailServer**

R24 The system must allow students to join a tournament.
UserApp, **CKBHandler**, **CKBRequestHandler**, **STournamentHandler**, **DataManager**, **DatabaseServer**

R25 The system must notify via email students when a new battle is created within the context of a tournament they signed-up to.
CKBHandler, **MailManager**, **MailServer**

R26 The system must allow students to subscribe to a battle within the context of a tournament they signed-up to.
UserApp, **CKBHandler**, **CKBRequestHandler**, **SBattleHandler**, **DataManager**, **DatabaseServer**

R27 The system must allow students to invite other students to a battle.
UserApp, **CKBHandler**, **CKBRequestHandler**, **SBattleHandler**

R28 The system must notify via email students when they are invited to join a battle by another student.
CKBHandler, **MailManager**, **MailServer**

R29 The system must notify via email students when the final battle rank becomes available.
CKBHandler, **MailManager**, **MailServer**

R30 The system must notify via email students when the final tournament rank becomes available.
CKBHandler, **MailManager**, **MailServer**

R31 The system must pull the latest submitted sources before the submission deadline expired.
CKBHandler, **DataManager**, **DatabaseServer**

R32 The system must analyze the students' projects.
CKBHandler, **EvaluationTool**, **DataManager**, **DatabaseServer**

R33 The system must run the students' projects.
CKBHandler, **EvaluationTool**, **DataManager**, **DatabaseServer**

R34 The system must evaluate the students' projects.
CKBHandler, **EvaluationTool**, **DataManager**, **DatabaseServer**

R35 The system must update the battle scores of the teams.
CKBHandler, **DataManager**, **DatabaseServer**

R36 The system must compute the battle rank.
CKBHandler, **DataManager**, **DatabaseServer**

R37 The system must update the students' personal tournament scores.
CKBHandler, **DataManager**, **DatabaseServer**

R38 The system must compute the tournament rank.
CKBHandler, **DataManager**, **DatabaseServer**

R39 The system must allow users to visualize ongoing tournaments.
UserApp, **CKBHandler**, **CKBRequestHandler**, **STournamentHandler** if the user is a student, **EdTournamentHandler** if the user is an educator, **DataManager**, **DatabaseServer**

R40 The system must allow users to visualize ongoing tournament ranks.
UserApp, **CKBHandler**, **CKBRequestHandler**, **STournamentHandler** if the user is a student, **EdTournamentHandler** if the user is an educator, **DataManager**, **DatabaseServer**

R41 The system must allow users to visualize students' profiles.
UserApp, **CKBHandler**, **CKBRequestHandler**, **ProfileHandler**, **DataManager**, **DatabaseServer**

R42 The system must notify via email students when they join a tournament.
CKBHandler, **MailManager**, **MailServer**

Chapter 5

Implementation, integration and testing plan

This chapter presents the order of implementation of subsystems and components, the integration strategy and the testing plan.
Subsystems are developed and integrated using a bottom-up approach, starting from the individual parts and then linking them together.

5.1 Implementation Plan

The implementation of the system is realized following a bottom-up method, with a particular emphasis on the back-end development since users will interact with the system only through its APIs.
Particularly, the implementation order is the following:

1. **DBMS, DataManager:** this is the first part to be implemented since it is responsible for storing and accessing the data needed by the system.
2. **AuthManager:** this component is responsible for users authentication and authorization, which are key aspects of the system, and therefore is heavily linked to the DBMS and DataManager.
3. **MailManager, EvaluationTool:** these components can be implemented in parallel since they are independent from each other. In this phase we need to choose the programming languages that the system will support and consequently the static analysis tools that will be integrated into the system.
4. **CKBHandler:** this component is responsible for linking together the previous components, so it must be developed after them.
5. **UserApp:** this component is responsible for the interaction with the user, so it is to be developed after the back-end has been completed.

5.2 Integration Plan

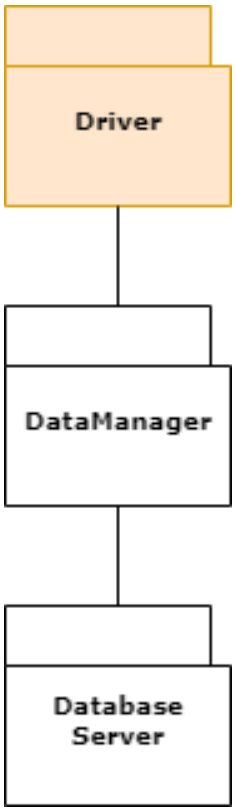

Figure 5.1: First level of integration

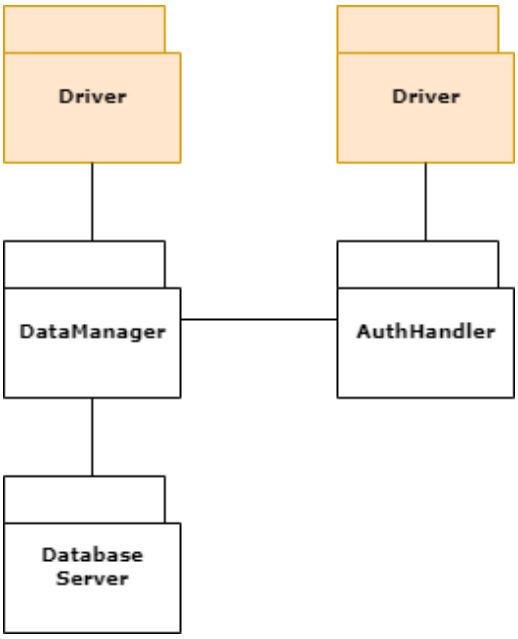

Figure 5.2: Second level of integration

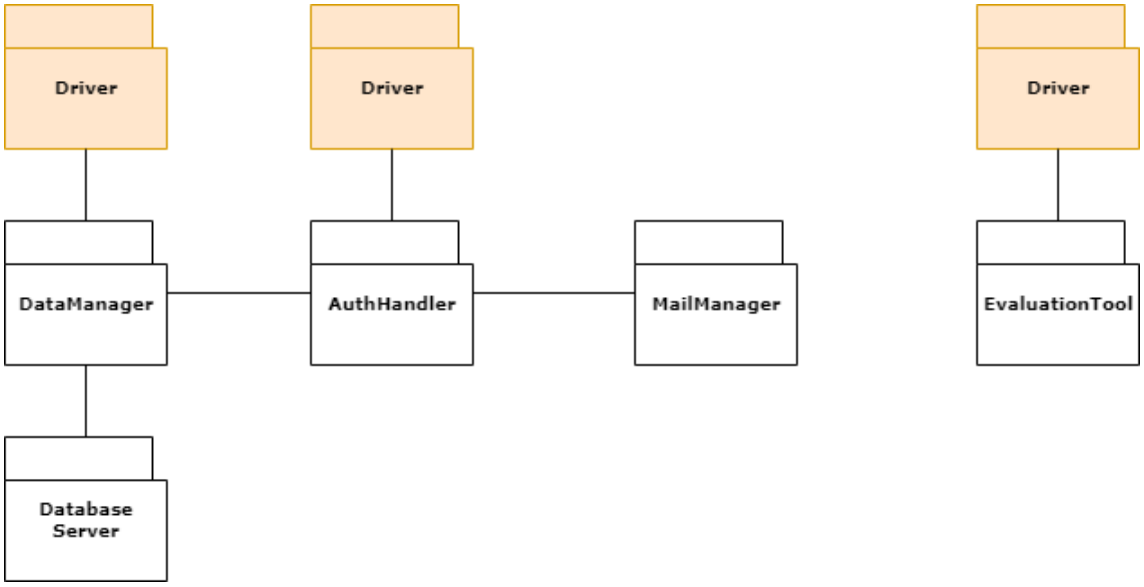

Figure 5.3: Third level of integration

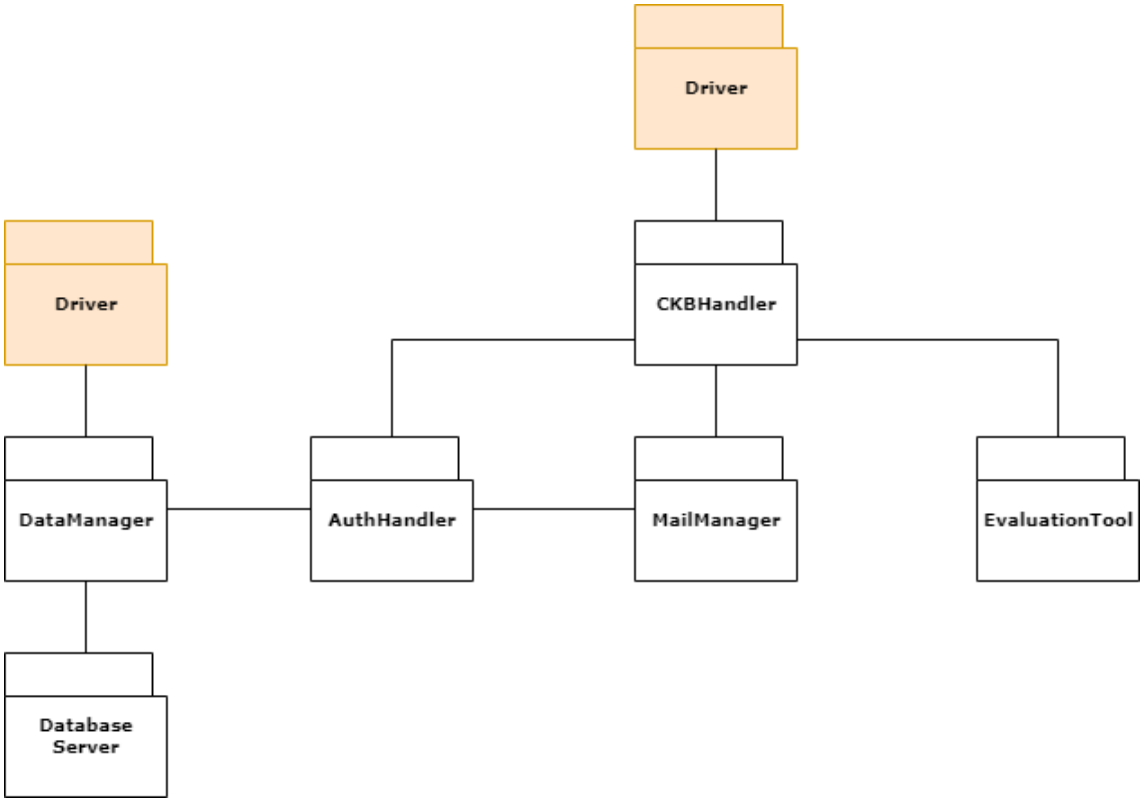

Figure 5.4: Fourth level of integration

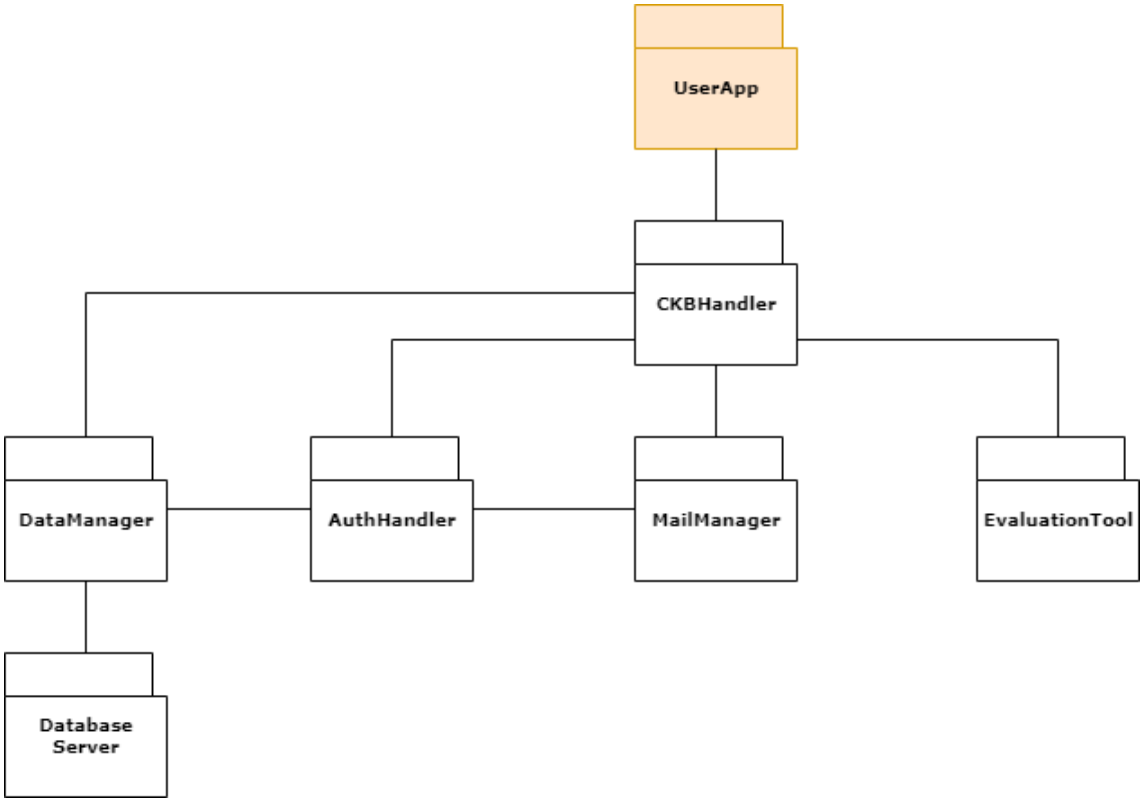

Figure 5.5: Fifth level of integration

5.3 System Testing

Testing should be performed both during and after the development of the system to ensure that all requirements are satisfied.
The testing plan is divided in two parts: unit testing and integration testing. Integration testing should also focus on the security aspects of the system to avoid any possible security breach.
Alpha testing should be performed whenever a new feature is implemented to receive feedback from the stakeholders regarding their level of satisfaction and eventual malfunctions.
A beta test phase could take place after the development of the system to ensure that the system is ready for a real-world usage and to collect information about performance and eventual scalability issues.

Chapter 6

Effort Spent

Group working

Introduction Section	**4h**
DD Structure analysis, overview architecture brainstorming	**3h**
Component View	**5h**
Deployment View	**2h**
Final Revision	**3h**

Chiara

Runtime View	**3h**
Component Interfaces	**2h**
Requirements traceability	**2.5h**

Flavia

Runtime View	**3h**
Design Choices description	**1.5h**
User Interface Design	**2.5h**

Gianluca

Runtime View	**4h**
Implementation Plan, integration plan and system testing	**3h**

Chapter 7

References

- Software Engineering II course slides
- RASD
- Use Case Diagrams and Sequence Diagrams were made using draw.io
  https://app.diagrams.net/
- UI Mockup
  https://www.figma.com/
- Three-tier architecture