POLITECNICO DI MILANO

Computer Science and Engineering

Design Document

CodeKataBattle

Software Engineering 2 Project

Academic year 2023 - 2024

7 December 2023

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 realised 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 choiches, descripting at a high level the technologies and components used to implement the platform, paying attention also to the interactive behaviour 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
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
w.r.t.	with reference to
e.g.	exempli gratia
i.e.	id est
etc.	etcetera
DB	Database
API	Application Programming Interface
DMZ	Demilitarized Zone

1.4 Revision history

1.5 Reference Documents

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

1.6 Document Structure

The document is structured as follows:

- **Introduction**: The first chapter includes the introduction in which is explained the purpose of the document, then, s 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.

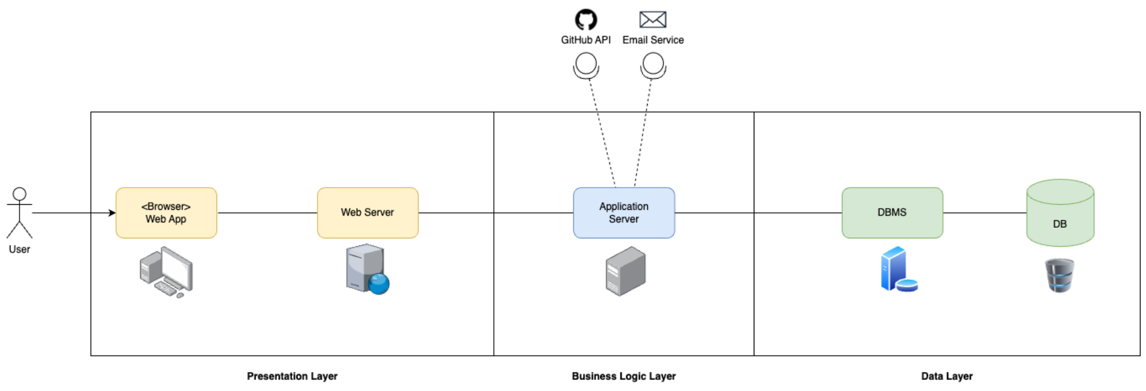

Figure 2.1: Overview CKB architecture

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 corrspond 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

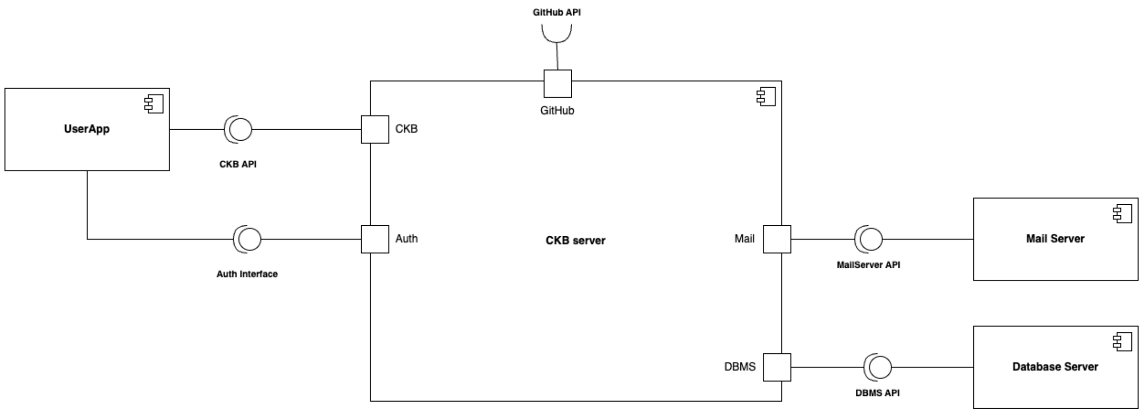

Figure 2.2: High level component view

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

- **CKB Server:** 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 **Auth interface** and access the functionalities offered by the system through the **CKB API**.
- **Database Server:** it represents the DBMS used to store the data of the system. The system can access the data through the **DBMS API**.
- **Mail Server:** 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

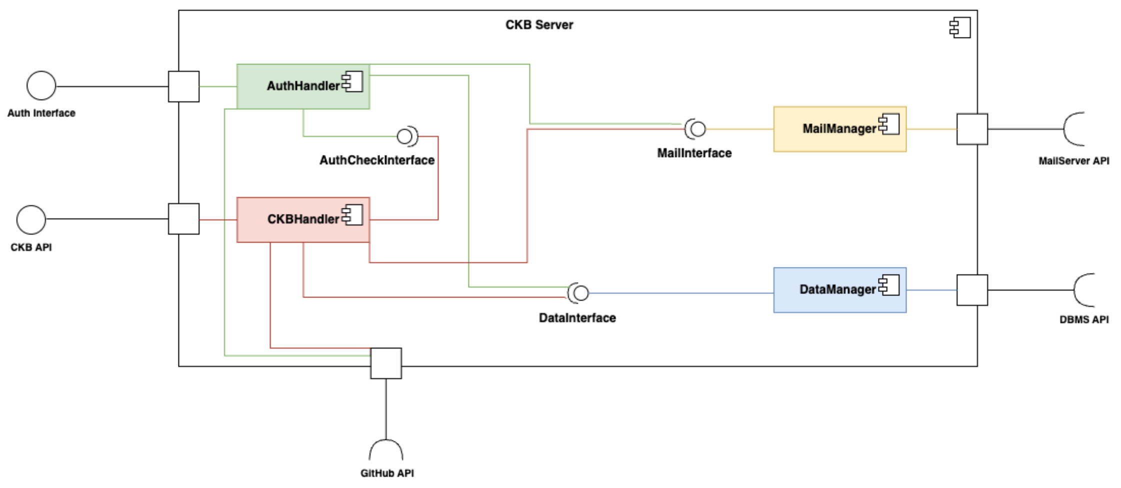

Figure 2.3: CKB server component view

The figure above shows the internal componenents 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 and with the **MailInterface** to send notifications to the users. 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**.

2.2.3 Auth component view

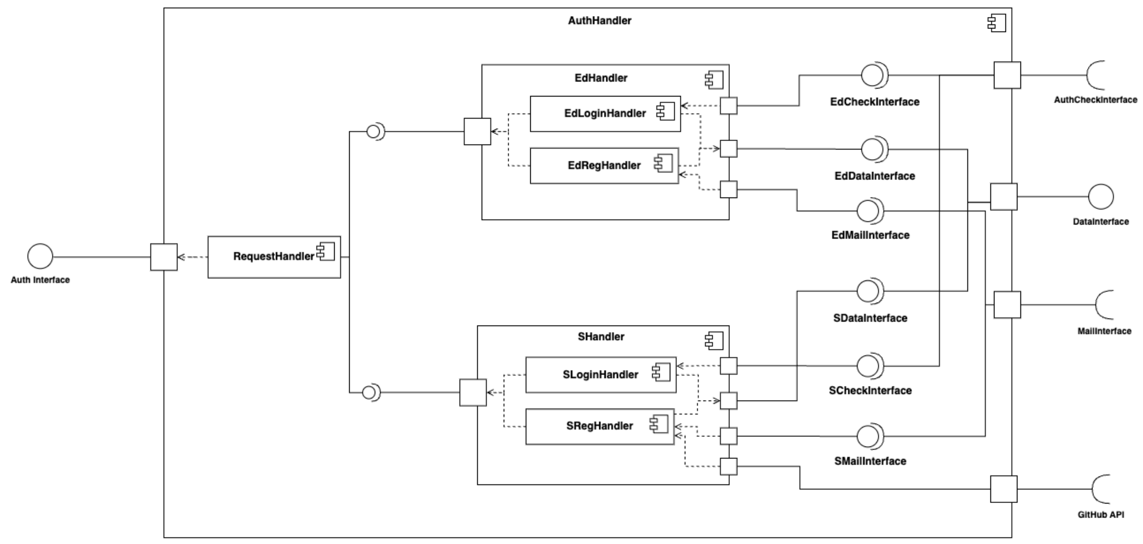

Figure 2.4: Auth component view

The figure above shows the internal componenents of the AuthHandler.

- **RequestHandler:** it handles the requests acting as a router dispatching requests to the right handler.
- **EdHandler:** it is composed by the **EdLoginHandler** and the **EdRegHandler**. The former handles the login of educators, the latter handles their registration.
- **SHandler:** it is composed by 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 communicate with **GitHubAPI** to link the CKB account with the GitHub one.

2.3 Deployment view

The figure below shows the architecture of the system. All the users access to the WebApp through the browser, which communicate 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 throw the use of load balancing services which allows the application server to manage traffic and workload.

- **Security:** the ability to protect the application server using firewall and DMZ, againist cyberattacks and possible threats.
- **Cost efficiency:** the ability to pay only for the resources used which can help to lower the overall cost.

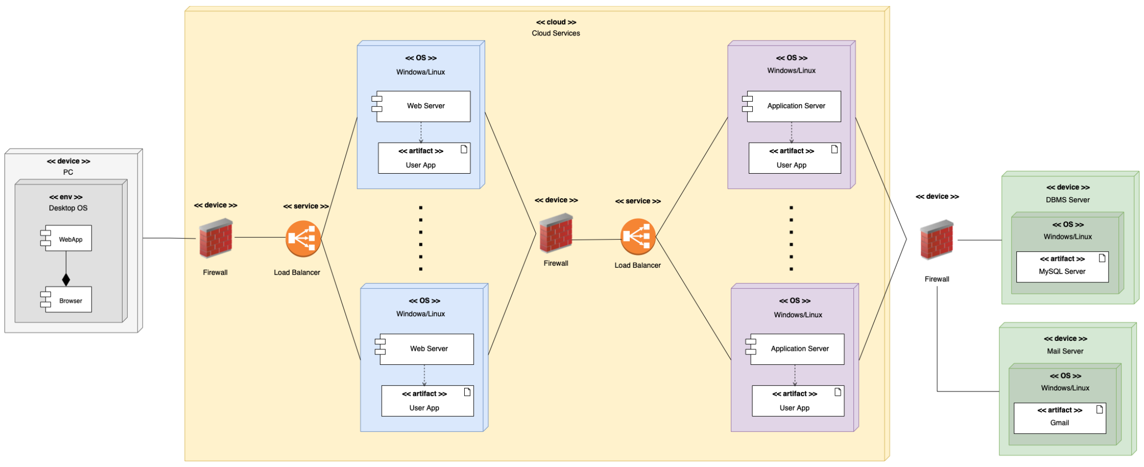

Figure 2.5: Deployment diagram

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

- **PC:** is 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:** devices are 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:** services are 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:** 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:** is used to store all the data of the system. It uses MySQL as DBMS to retrieve and store data.
- **Mail Provider:** is used to send notifications to the users. It uses Gmail as mail provider.

2.4 Runtime view

2.4.1 Creation of the tournament

The figure below shows the sequence diagram of the creation of a tournament. The educator compiles the form with the tournament details and submits it. The system checks if the data are valid, and if the badge option is set to true, the system creates the badge. After that the system send an email to all the granted colleagues. The tournament is created and the educator is redirected to the tournament page.

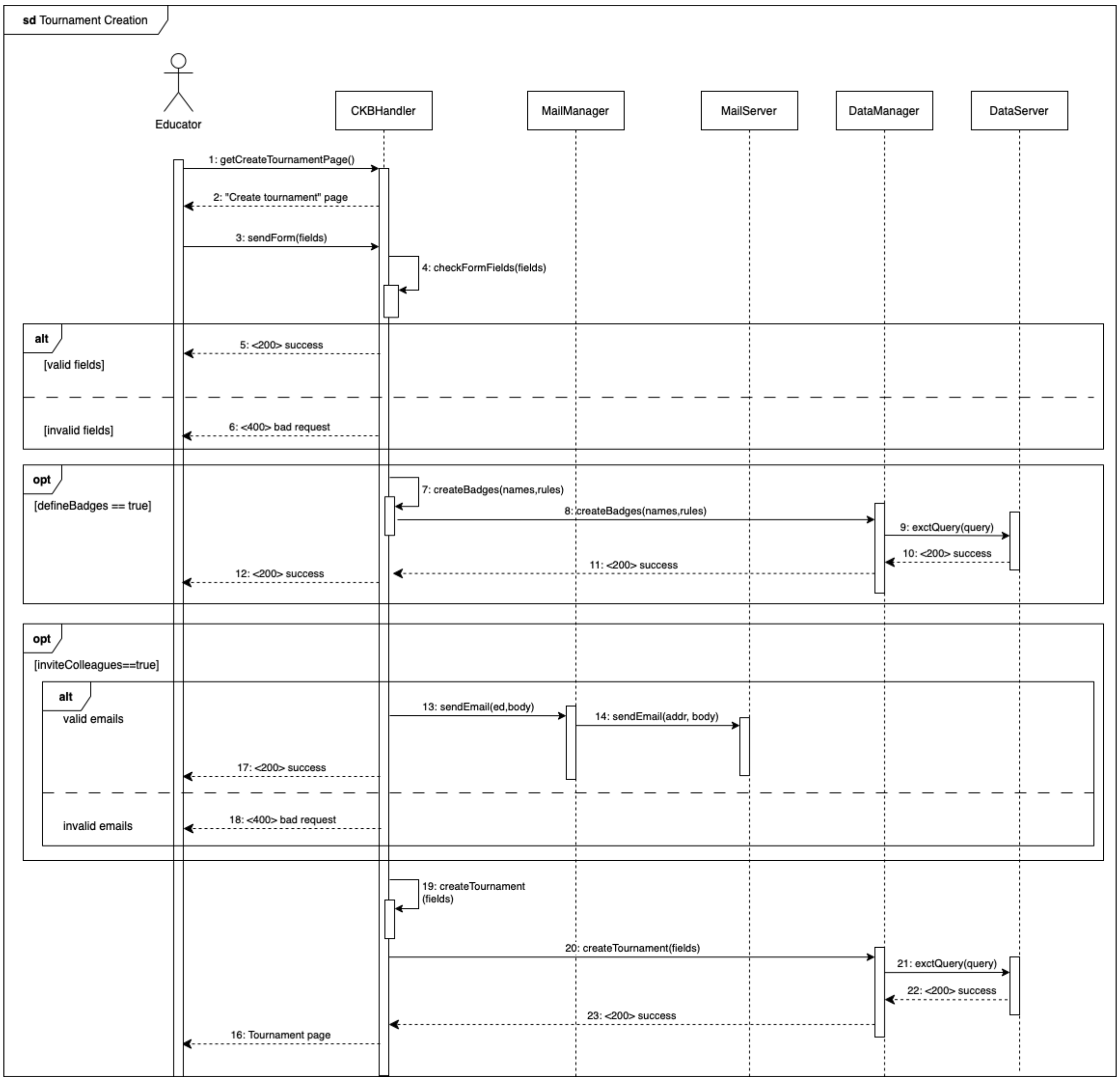

Figure 2.6: Creation of the tournament

2.4.2 Creation of the battle

The figure below shows the sequence diagram of the creation of a battle. After the educator clicked ok the "Create Battle" button, he compiles the form with the battle details and submits it. The system checks if the data are valid. The battle is created, so the system send an email to all the submitted students.

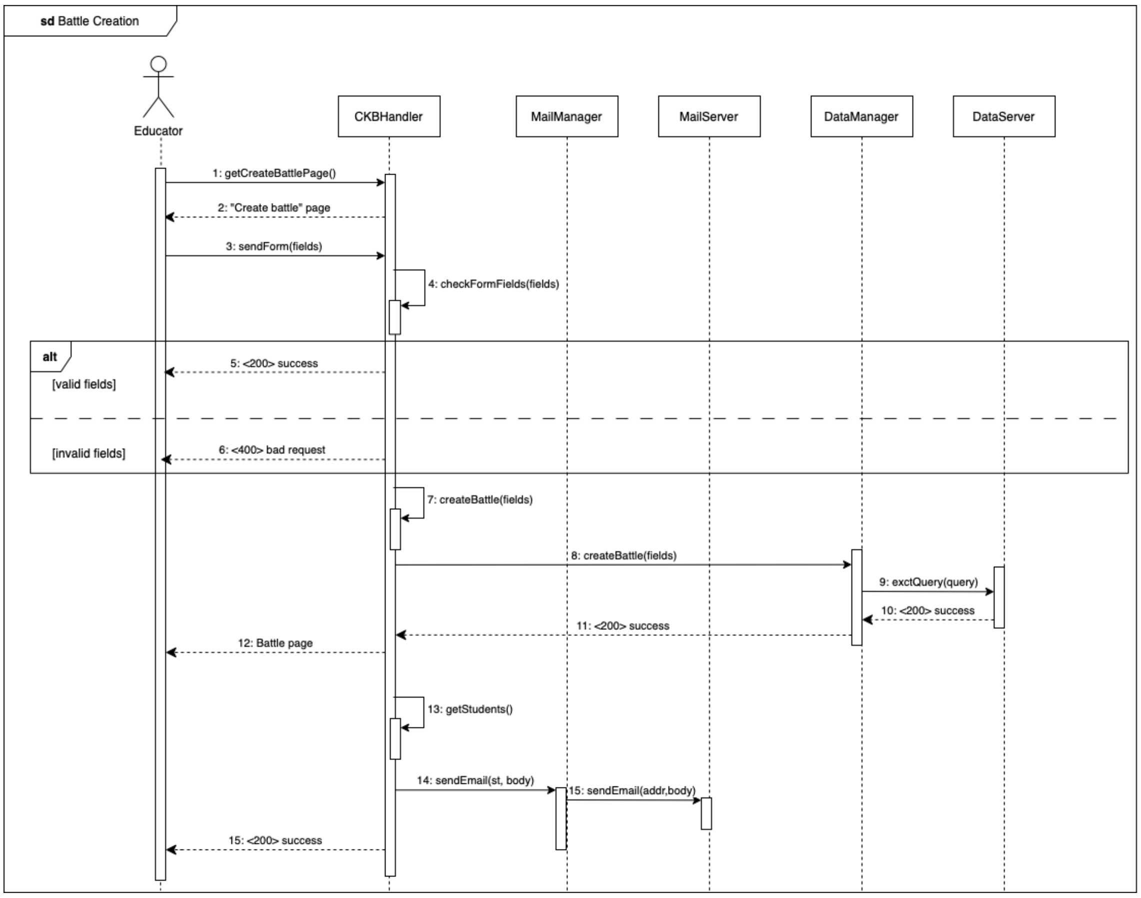

Figure 2.7: Creation of the battle

2.4.3 Execution of the battle

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

The figure below shows how a student can sign up for a tournament and receive an email notification about it.

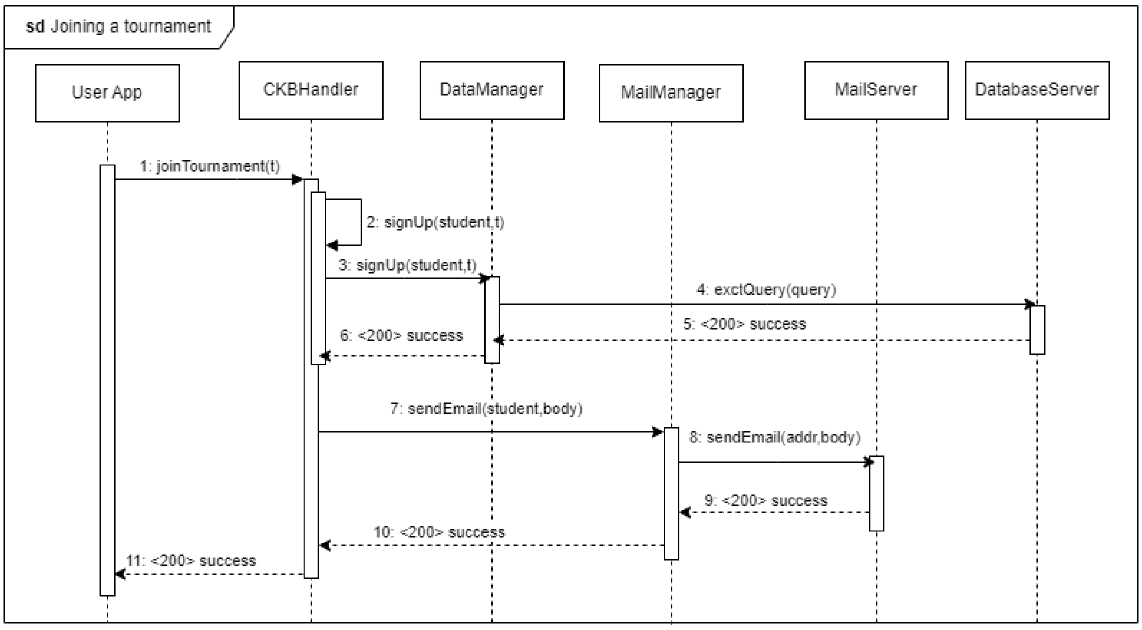

Figure 2.8: Execution of the battle

The figure below shows a student joining a battle as a singleton and receiving an email notification about it.

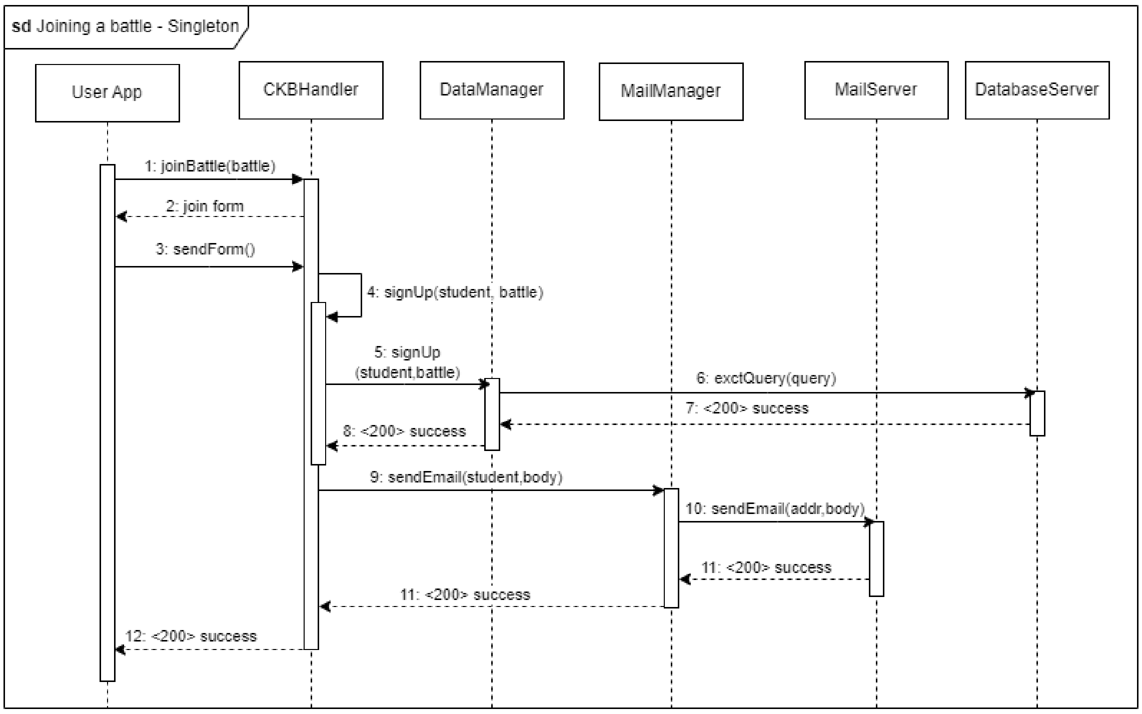

Figure 2.9: Execution of the battle

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

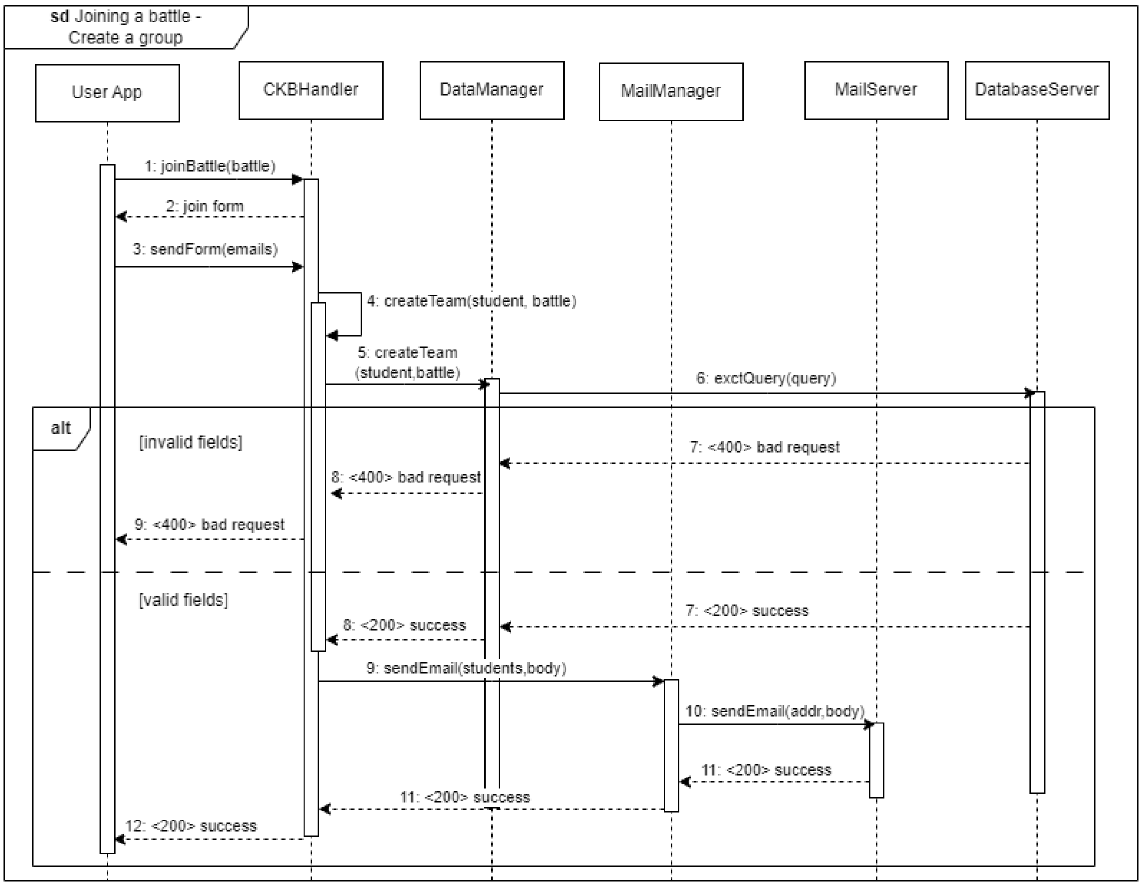

Figure 2.10: Execution of the battle

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.

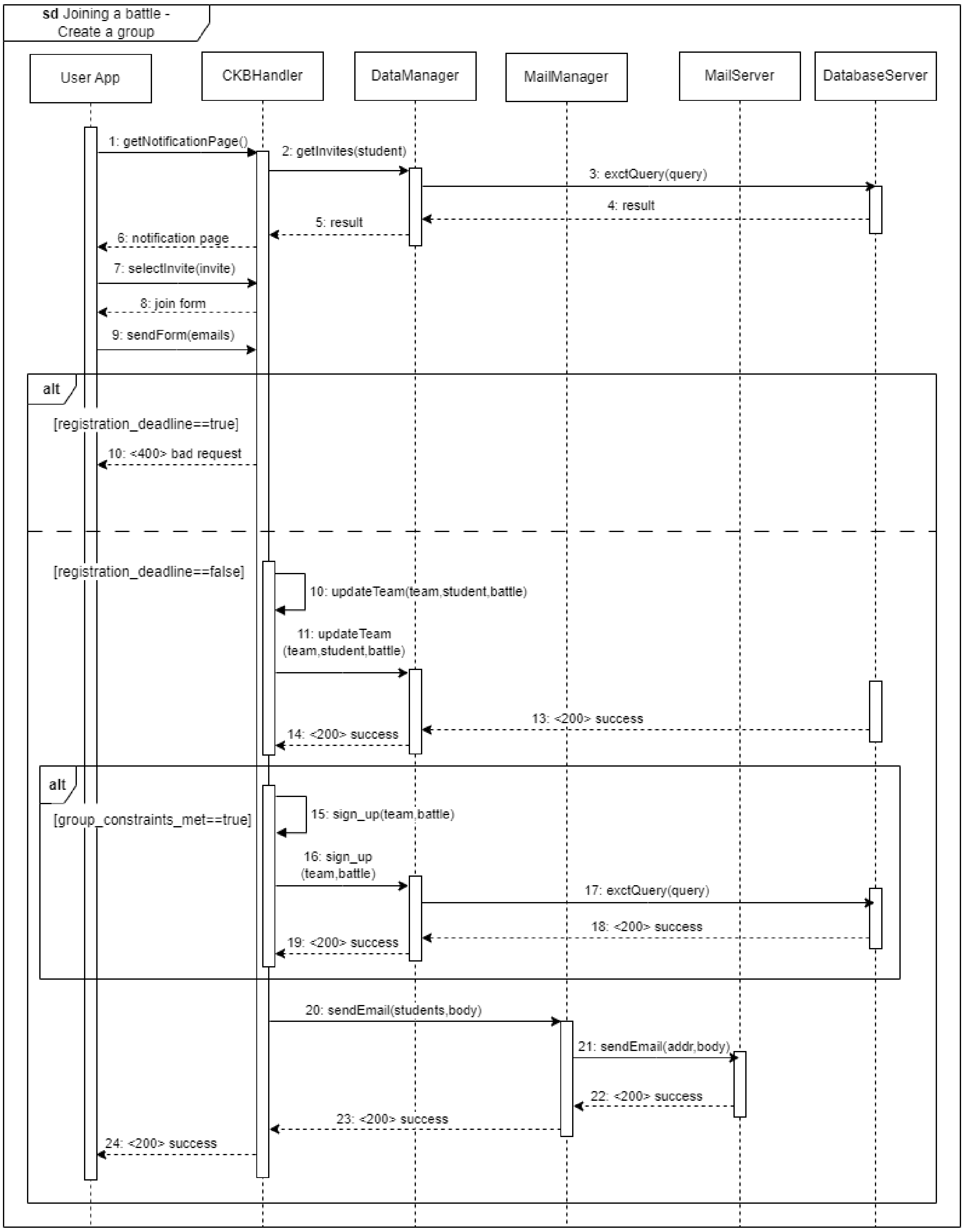

Figure 2.11: Execution of the battle

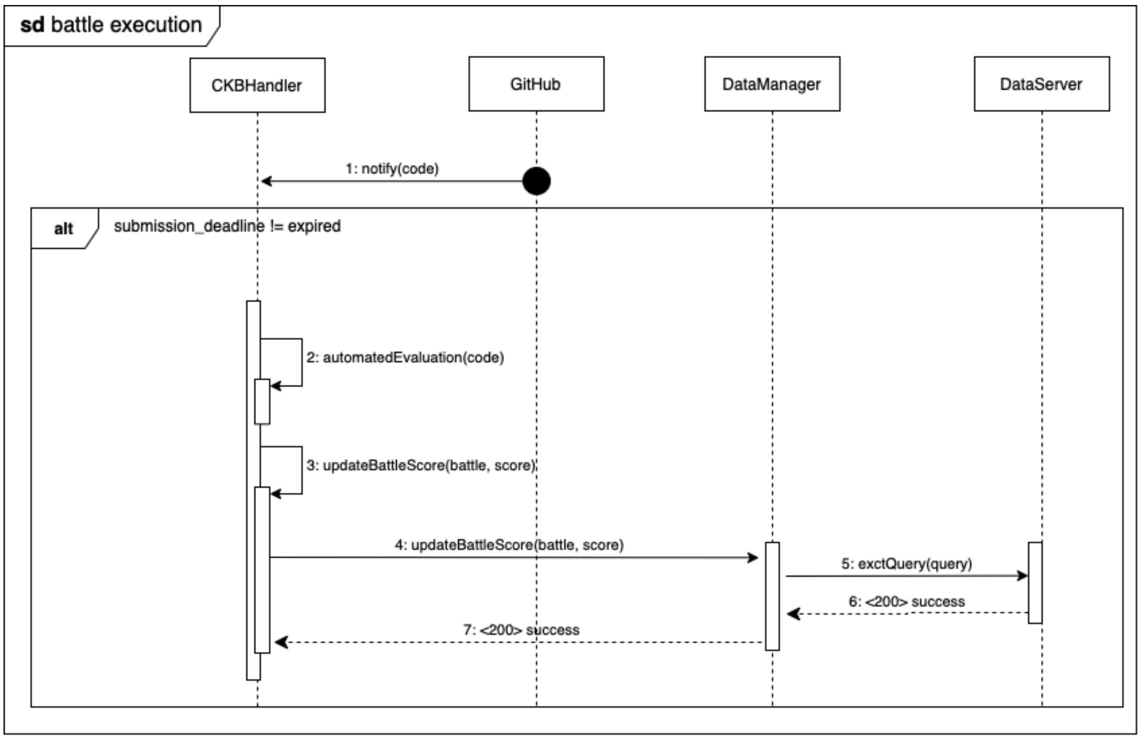
sd battle execution
CKBHandler
GitHub
DataManager
DataServer
1: notify(code)
alt
submission_deadline != expired
2: automatedEvaluation(code)
3: updateBattleScore(battle, score)
4: updateBattleScore(battle, score)
5: exctQuery(query)
6: <200> success
7: <200> success

Figure 2.12: Execution of the battle

The figure below shows evaluation process of a students' solution: first the CKBHandler asks the EvaluationTool to evaluate the code according to the rules for automated evaluation, then, after the battle submission deadline expires and if required, the educator can manually evaluate the code.

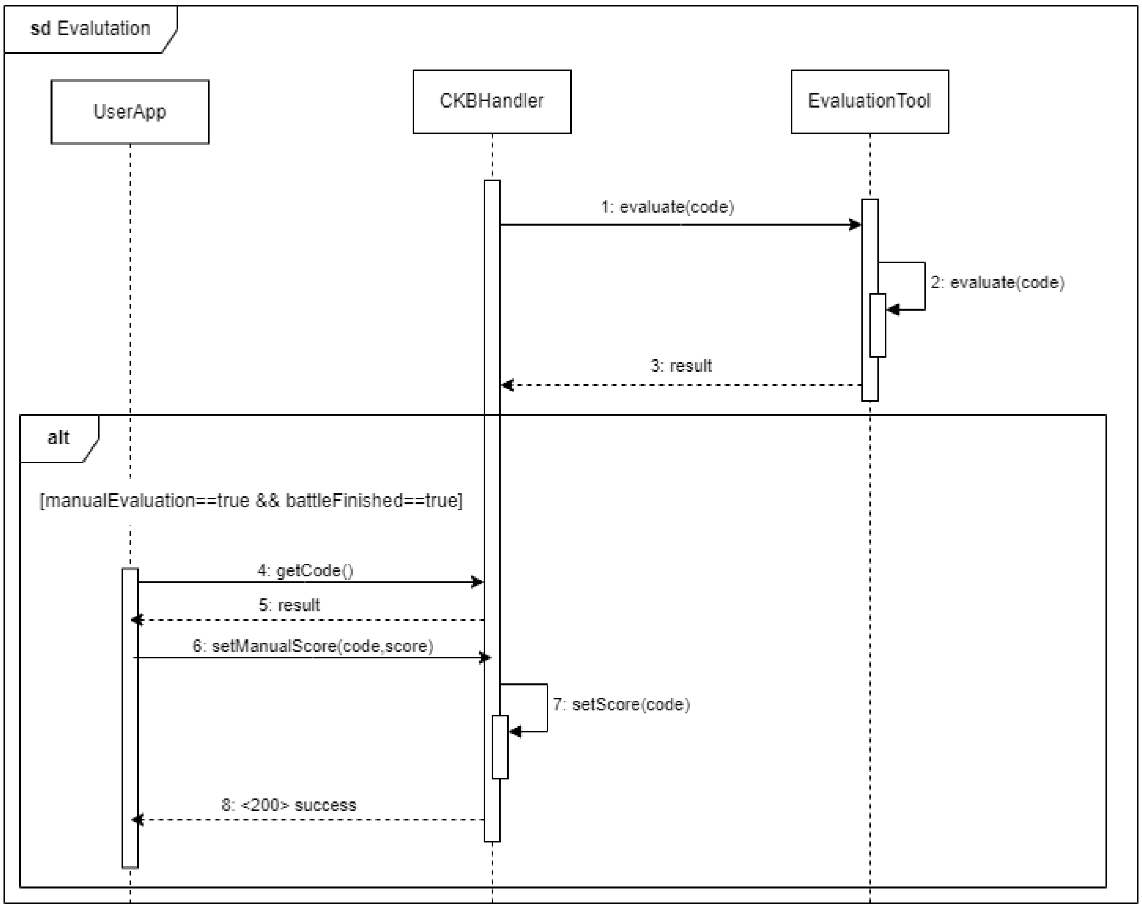

Figure 2.13: Execution of the battle

2.4.4 Conclusion of the battle

The figure below shows the sequence diagram of the conclusion of a battle. The system updates the tournament score of the students who participated to the battle and the tournament rank. The system send an email to all the students submitted to the tournament to notify them that the updated tournament rank is available.

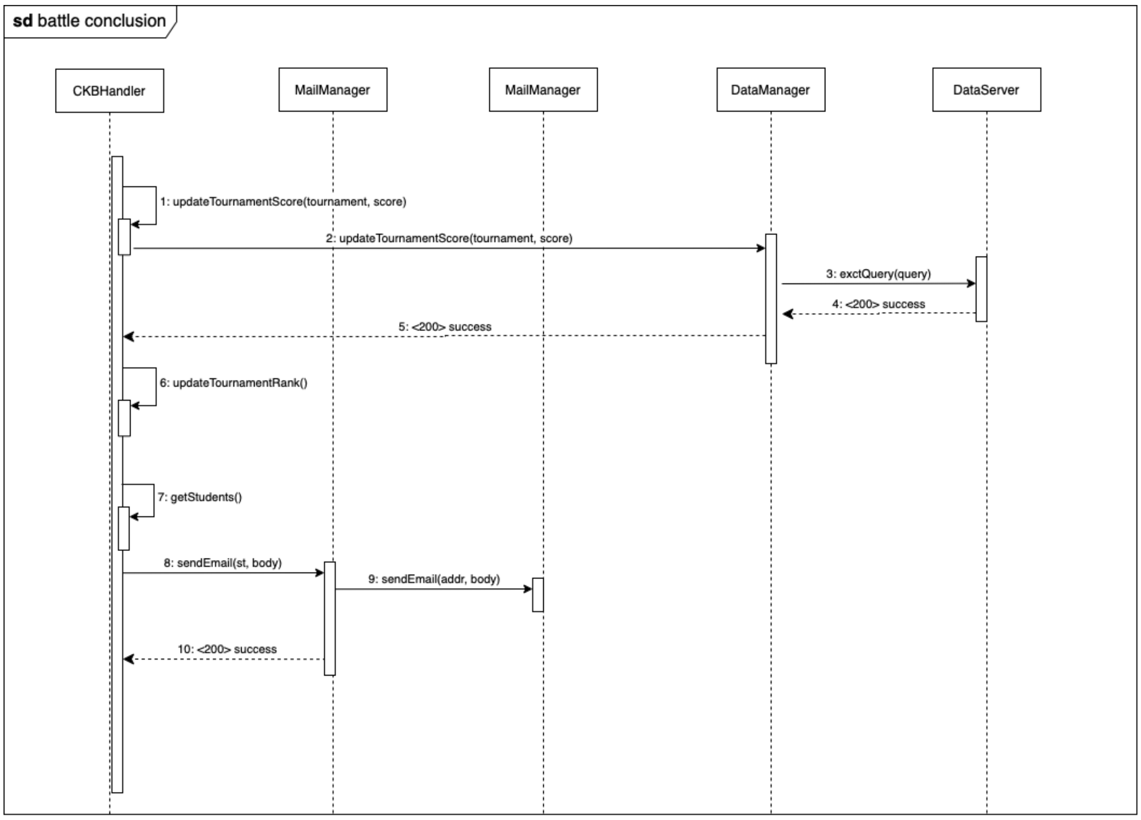

Figure 2.14: Conclusion of the battle

2.5 Component interfaces

2.6 Selected archititetural styles and patterns

2.7 Other design decisions

Chapter 3

User Interface Design

Chapter 4

Effort spent

Chapter 5

References