POLITECNICO DI MILANO

SOFTWARE ENGINEERING II

CODE KATA BATTLE PLATFORM

Design Document

Version 1.2

Simona Cai 10752734
Yang Hao Mao 10705881
Jiaxiang Yi 10765316

Contents

1. Introduction

1.1 Purpose

The primary objective of this document is to facilitate the development team in bringing the envisioned system to fruition. It furnishes a comprehensive overview of the chosen system architecture, delving into an intricate breakdown of individual components. Furthermore, it elucidates the intricate interactions among these components, providing clarity on the dynamics within the system.

Moreover, the Design Document (DD) serves to articulate detailed plans for implementation, integration, and testing. These plans are meticulously crafted, taking into consideration the priority of tasks, the required effort for successful execution, and the potential impact of each component on stakeholders. In essence, this document acts as a guiding framework, aligning development efforts with strategic priorities and ensuring a coherent and efficient realization of the envisioned system.

1.2 Scope

CodeKataBattle (abbreviated CKB) is a comprehensive software platform designed to facilitate the improvement of students' software development skills through engaging coding challenges. The primary goal is to provide a seamless and effective environment for students to enhance their coding proficiency without unnecessary complexities.

CodeKataBattle (CKB) - Advancing Coding Skills for All
The CKB platform enables students to participate in code kata battles, where they collaboratively engage in programming exercises to hone their coding skills. The key objectives include:

1. **Skill Enhancement:**
   - Empower students to improve their software development skills through hands-on coding challenges.
   - Encourage a collaborative and competitive environment that fosters continuous learning.
2. **Educator-Facilitated Challenges:**
   - Allow educators to create and manage code kata battles, setting challenges and evaluating student performances.
   - Provide a platform for educators to monitor and assess the progress of their students.
3. **Test-Driven Development:**
   - Promote a test-first approach, requiring teams to develop solutions that pass predefined test cases within a specified timeframe.
4. **Tournament Structure:**
   - Organize code kata battles within the context of tournaments, creating a structured and competitive framework.

- Allow educators to define tournament-specific rules, badges, and scoring criteria.

5. **Notification and Subscription:**
   - Notify students about upcoming tournaments and battles to encourage participation.
   - Allow students to subscribe to specific tournaments and receive timely notifications.
6. **Badge System:**
   - Implement a badge system to recognize and showcase students' achievements based on their participation and performance in tournaments.
   - Allow educators to define custom badges, rules, and variables associated with the badge assignment.
7. **Dynamic Badge Visualization:**
   - Enable both students and educators to visualize collected badges on individual profiles, showcasing accomplishments.
8. **Educator Collaboration:**
   - Facilitate collaboration among educators by allowing them to create and manage tournaments within the platform.
   - Grant permissions for colleagues to create battles within the context of specific tournaments.

The outlined scope emphasizes creating a user-friendly and collaborative platform for students and educators, fostering a culture of continuous improvement in software development skills. Further details and specific functionalities can be explored in the detailed Requirements Analysis and Specification Document (RASD).

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Student**: An individual engaged in learning or academic activities, typically enrolled in an educational institution or a course. In a model, a Student might have attributes like name, email, titles, and associated possible battles records.

- **Educator**: A person who provides education or instruction, often a teacher, instructor, or professor. In a system, an Educator might have attributes like name, email, tournament admin, battles creator.

- **Battle**: In an educational or competitive context, a Battle could refer to a contest or a challenge where participants compete, often in teams of one or more Student. Attributes might include battle name, tournament reference, participating teams, start and end times, and status.

- **Tournament**: A series of competitions or battles organized around a particular activity or subject. In a model, a Tournament might have attributes like tournament name, list of battles or contests, registration details, badges, and overall status.

- **Code Kata**: A term often used in programming and software development, referring to a small and focused coding exercise to practice programming skills. In a system, a Code Kata might have attributes like problem statement, language, expected outcomes,

and associated test case resources.

- **Badge**: Typically a symbol or indicator of achievement, skill, or qualification. In educational or gamified systems, badges are often awarded to participants for completing tasks, reaching titles, or achieving certain standards. Attributes might include badge name, criteria for earning, and the badge's visual representation.
- **Practice Mode**: This could refer to a non-competitive or learning-oriented mode in a system where participants can practice skills, solve problems, or engage in activities without the pressure of competition or assessment. Attributes might include available exercises, and resources.

1.3.2 Acronyms

- **[CKB]:** Code Kata Battle Platform
- **[UI]:** User Interface
- **[UML]:** Unified Modelling Language
- **[RASD]:** Requirement Analysis and Specification Document
- **[API]:** Application Programming Interface
- **[DD]:** Design Document
- **[DB]:** Database
- **[Rn]:** the n-th functional requirement

1.4 Revision History

- Version1.0 (January 05th 2024 : first version);

1.5 Reference Documents

This document is strictly 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 codewar: https://www.codewars.com/
- DD Sample from A.Y. 2021-2022

1.6 Document Structure

CodeKataBattle (CKB) Document Structure:

1. **Introduction:**
   - Provides an overview of the document's purpose and significance for the CodeKataBattle platform.

- Briefly recaps key concepts from the Requirements Analysis and Specification Document (RASD).
- Offers definitions, acronyms, synonyms, and references to relevant documents for reader clarity.

2. **Architectural Design:**
   - Offers a detailed description of the system's architecture, including high-level views of elements and software components within CKB.
   - Utilizes runtime diagrams to illustrate various functionalities of the system.
   - Provides an in-depth explanation of the architectural pattern employed in CKB.
3. **User Interface Design:**
   - Presents mockups of the user interfaces within the CKB application.
   - Illustrates the flow between different interfaces with links to aid in understanding the user experience.
4. **Requirements Traceability:**
   - Describes the connections between requirements outlined in the RASD and the components detailed in the architectural design.
   - Serves as evidence that design decisions align with the specified requirements, ensuring the system meets its goals.
5. **Implementation, Integration, and Test Plan:**
   - Details the step-by-step process of implementation, integration, and testing that developers should follow to ensure the correct and robust development of the CodeKataBattle platform.
6. **Effort Spent:**
   - Summarizes the efforts invested in the development of the CodeKataBattle platform, providing insights into the project's timeline and resource allocation.
7. **References:**
   - Lists all references to documents and software tools used in the creation of this document and the development of the CodeKataBattle platform.

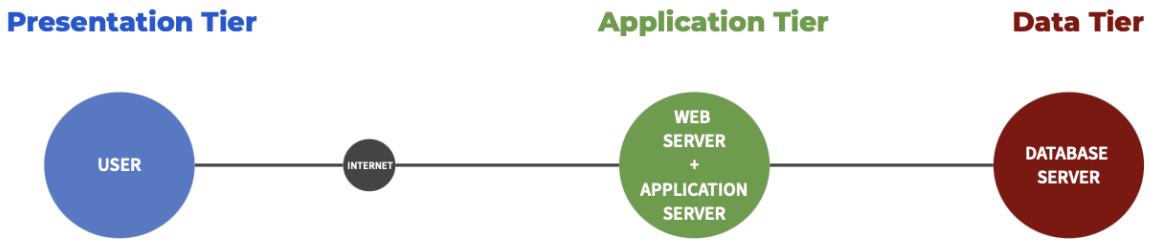
2. Architectural Design

2.1 Overview

In this section the architecture will be described in an easy way, justifying all the choice for adopted patterns.

2.1.1 High-level components and tier interaction

The CKB is distributed as a system with a tree-tier architecture:
The **Presentation Tier**: also known as the user interface layer, its primary responsibility is to interact with users and present information in a human readable format. This tier handles user input and displays the output in a way that users can understand, includes components that users interact with directly. The main goal of this tier is to provide a user-friendly and responsive interface for interacting with the application.
The **Application Tier**: contains the application logic of the system. It's responsible for processing user request and coordinating the application's functionality. The application tier is where the core processing and decision making occur. It acts as a bridge between the presentation tier and the data tier, ensuring that data manipulation is carried out effectively.
The **Data Tier**: is responsible for managing and storing data. Its primary function is to handle data storage, retrieval, and manipulation. This tier is involving database systems that store and organize data efficiently. The data tier ensures data integrity, security, and availability. It also responds to requests from the application tier to retrieve or update information.
This separation of concerns enhances modularity, maintainability, and scalability in development, each tier can communicate only with adjacent one, so to communicate between the presentation tier and the data tier is necessary through the application tier.



The system is a distributed application which follows the common known client-server paradigm.

In particularly, the CKB platform will be accessed from a **web browser** by a registered **User** (Student or Educator), which can send an https request to a **Web Server**, which will return a valid web page via the application server. The **Application Server** will retrieve the desired data from a **Shared database** (e.g. User Profile, score, rank, badges, battle, tournament, etc.) or a **Private Database** (e.g. User account passwords, analyzers, team information, etc.). In addition, Application Server has other important roles, including but not limited to:

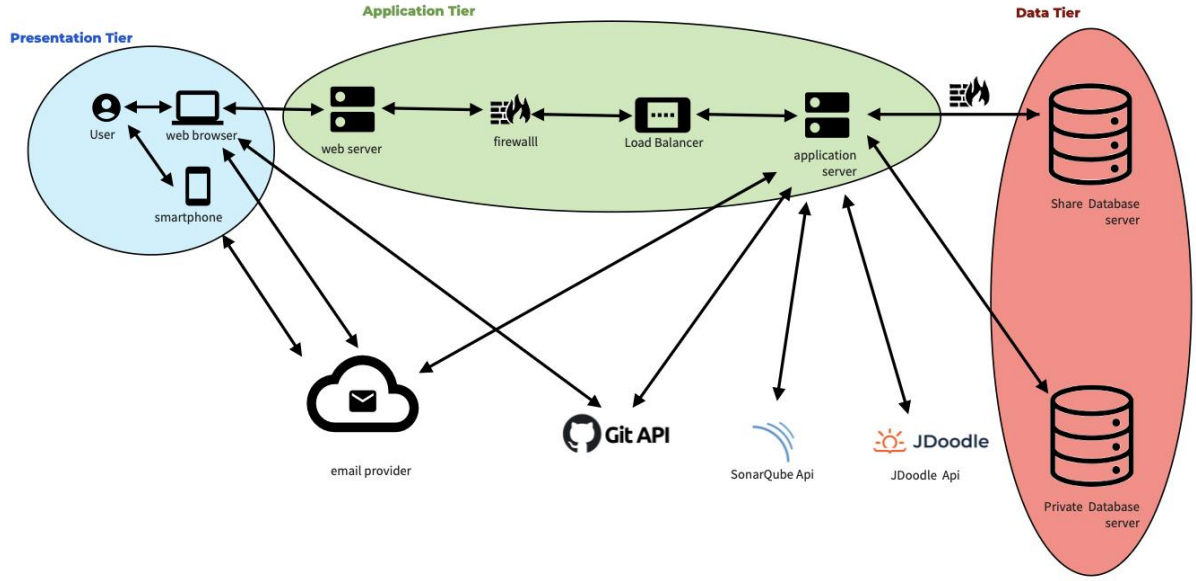
Email provider: students will receive information from tournament start, end, battle start, end, ranking, reward information through their computer or cell phone, Educators can receive other Educators' battle invitations and Students' private information, as well as the platform to send tournament or battle end information.

GitHub Api: After students create a successful team, the platform will send the information to Students through the Application Server and Github binding information to join the better participation in the battle.

SonarQube API: Educators will use more advanced analytics tools to parse the code submitted by Students to verify their reliability, and Application Server will score the code based on this tool.

JDoodle API: The platform will also use the compiler tool provided by JDoodle to run the code, together with the analysis tool to make the code analysis more efficient.

To ensure an adequate level of security, a front-end **firewall** has been installed on the router connecting the internal CKB network and the internet. Additionally, a back-end firewall can be installed between the presentation layer and application layer and the application layer and the data layer to provide an even higher level of security.



Furthermore, the system employs **Load Balancing** to distribute incoming requests evenly across multiple servers, enhancing performance and preventing overload on a single server.

Especially, to dispatch the requests to Web Server and Application Server, all the request coming from web browsers will receive from load balancer which redirects the request to the Web Server to handle it. Then the web server manipulates the request and forwards it to the Application Server.

Additionally, **Clustered Database Management Systems** (DBMS) are implemented to improve scalability, fault tolerance, and data availability by distributing and replicating data across multiple database nodes within the cluster.

2.2 Component View

The diagram below represents the components of the CKB Platform. The WebApp, shown in purple, belongs to the Presentation tier. It includes elements related to the user interface and provides users with a seamless and interactive web experience. The Web Server is coloured in blue, and it represents the element not belonging to the application server, yet still being part of the application layer. All the components illustrated in yellow represent the core application layer of the CKB platform, which includes major business logic and management functionalities. The green components related to the Database Management System (DBMS) belong to the data tier, which manages shared and private data, providing a structured storage solution for various platform functionalities. The rest of the red components consists of external services integrated into the platform provided by third parties, such as SonarQube for code analysis, JDoodle for online code compilation, and GitHub for repository management.
These colour-coded groups help categorise and visually organise the components based on their functionalities and roles within the CKB platform.

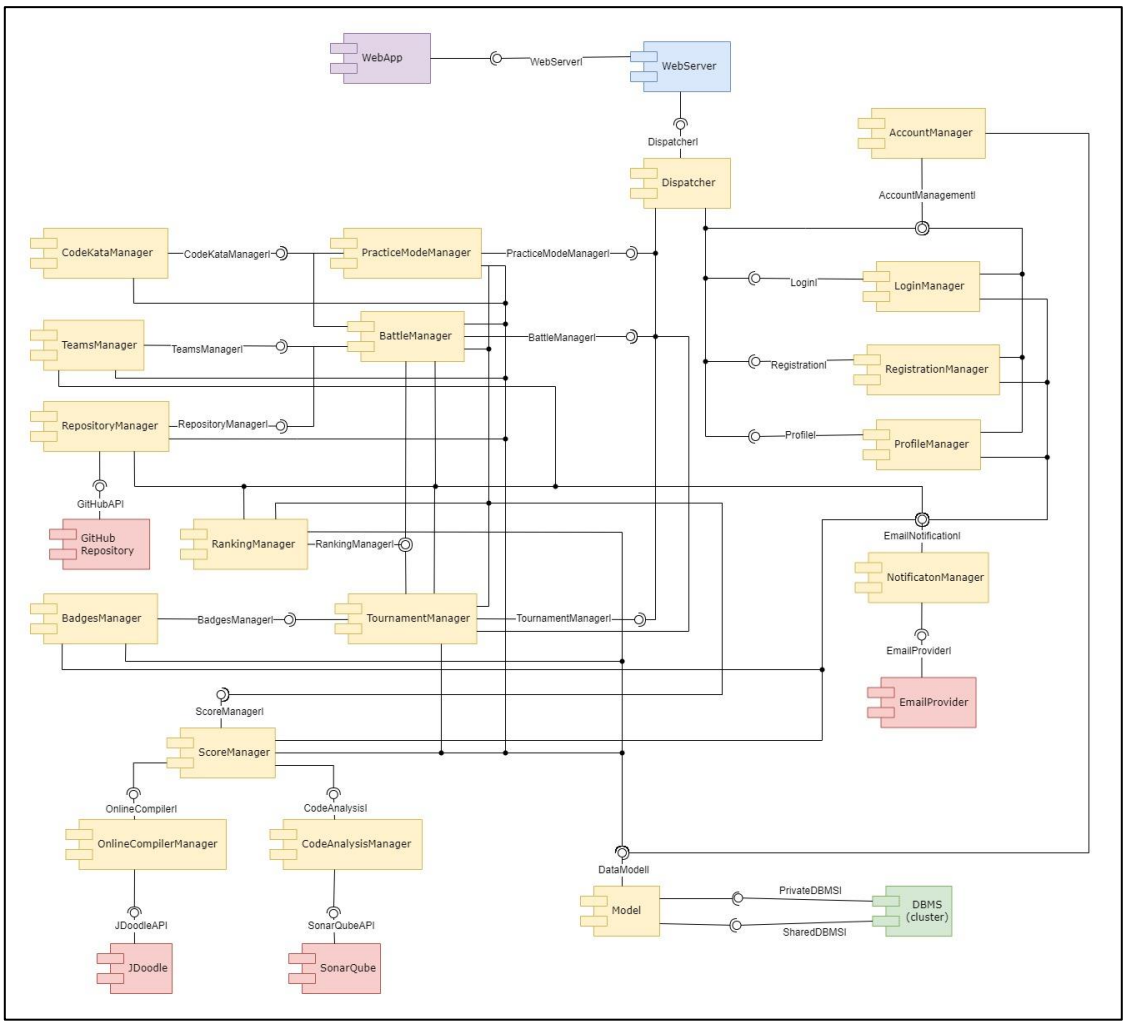


Figure 2.2 - CKB component diagram

- The **WebServer** serves as the interface for users (students and educators) to interact with the various managers and functions of the CKB platform. The system processes user requests, presents relevant information, and ensures a user-friendly interface.
- The **WebApp** serves as the interface for students and educators to interact with the CKB platform. The system communicates through HTTP requests to the WebServer, allowing it to be easily executed on a web browser.
- The **Dispatcher** coordinates communication between different managers to ensure a seamless workflow and data exchange.
- The **NotificationManager** is responsible for handling notifications related to various events, such as tournament creation, battle updates, score changes, and badge achievements. It uses email providers to send notifications to students and educators based on their preferences.
- The **TournamentManager** is responsible for creating, modifying, and closing tournaments. It allows educators to set up code kata battles, define deadlines, upload challenges, and configure scoring parameters.
- The **BadgesManager** is responsible for creating and assigning gamification badges. Educators can define rules and variables for badge achievements.
- The **BattleManager** is responsible for managing code kata battles in tournaments. This includes creating, tracking progress, and closing individual code kata battles, as well as managing information related to battles such as deadlines, team compositions, and GitHub integration.
- The **RepositoryManager** coordinates with the Battle Manager to create repositories for each battle and handles the generation and distribution of GitHub repository links to students upon battle registration. Additionally, it monitors GitHub repositories for new commits and triggers the CKB platform for automated scoring. This tool enables the integration of student code repositories with the CKB platform using GitHub Actions, ensuring secure and efficient communication with the GitHub platform.
- The **TeamsManager** is responsible for forming and managing teams for each code kata battle. They enforce team size limits and facilitate collaboration among students.
- The **CodeKataManager** handles the upload, storage, and retrieval of code kata challenges. Educators can upload descriptions and software projects, while students can access them to participate in battles.

- The **ScoreManager** is responsible for managing the scoring system for code kata battles. It handles both automated scoring, which includes functional aspects, timeliness, and quality level of sources, as well as manual scoring by educators. The Score Manager utilises information from code analysis, test cases, and possibly educator evaluations.
- The **OnlineCompilerManager** handles integration with JDoodle, an online compiler, for compiling and running code submitted by students.
- The **CodeAnalysisManager** integrates with the SonarQube API to perform automated code analysis. It evaluates code quality based on factors such as security, reliability, and maintainability.
- The **RankingManager** maintains rankings for individual battles and overall tournaments, providing a visual representation of a student's performance in comparison to others.
- The **PracticeModeManage**r improves the CKB platform by providing a dedicated area for students to enhance their coding skills outside of competitive code kata battles. It integrates with the Code Kata Manager to access and retrieve practice code katas. This feature allows students to select specific code katas for practice and provides a simulated environment similar to code kata battles for individual learning. Additionally, this feature allows students to receive automated feedback on their code, similar to the feedback provided during code kata battles. It is a valuable resource for continuous learning and self-improvement.
- **DBMS** (Database Management System):
    - The **SharedDBMS** manages shared data, such as profiles, scores, ranks, badges, battles, and tournaments. It utilises a clustered database system for scalability and reliability.
    - The **PrivateDBMS** stores sensitive information such as user accounts, code analysis results, and team data.
- The **AccountManager** is responsible for managing user accounts. This includes creating and updating personal information, managing subscription status, and handling authentication-related tasks. It interacts with the private account database.
- The **RegistrationManager** is responsible for managing the student registration process on the CKB platform. Depending on the client type, a specific form will be provided that must be completed with the appropriate information. For example, an educator will need to submit a teaching certificate.
- The **LoginManager** is responsible for managing user authentication and authorisation. It handles user login and registration, ensuring secure access to the platform.

- The **ProfileManager** handles user profiles, displaying personal details, ongoing tournaments, and earned badges. It uses shared profile data.
- The **Model** component determines how information is stored and processed within the CKB platform by defining its data structures and business logic. It is the only component that interacts with the DBMS service.

2.3 Deployment View

This deployment diagram illustrates the basic structure of the CodeKataBattle platform, illustrating the physical deployment of software components within the system, showing how software artefacts are distributed across hardware nodes.

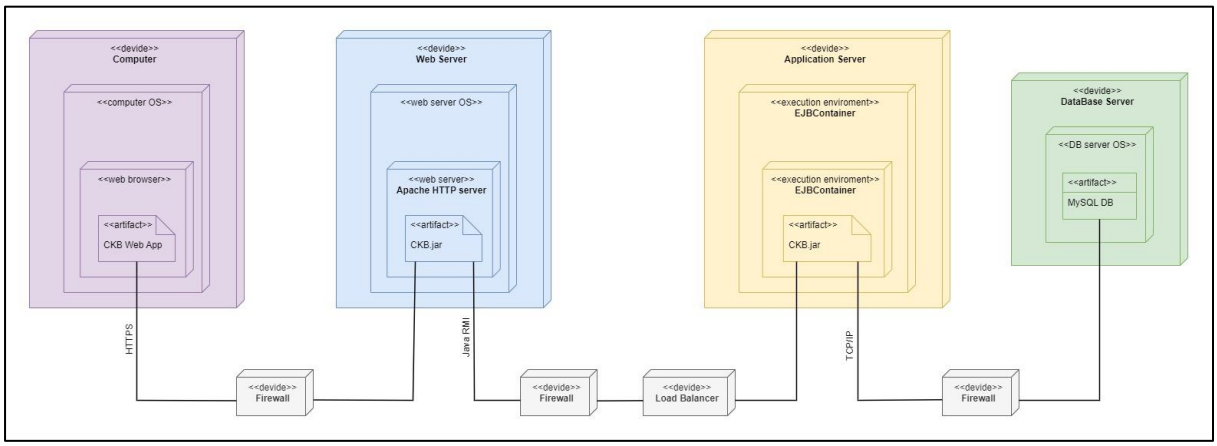


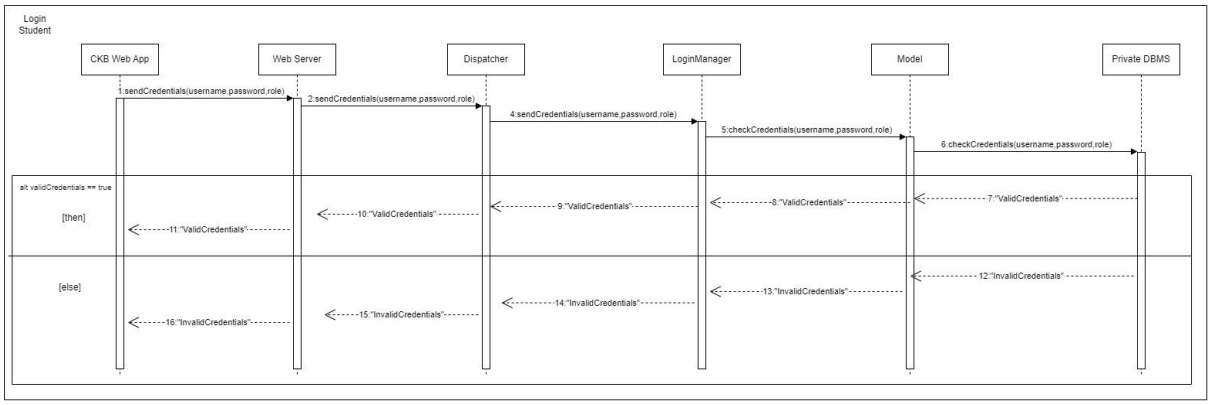
Figure 2.3 - CKB deployment diagram

- The **Computer** represents the user's device, through which they access the CodeKataBattle (CKB) platform. Users access the CKB platform through web browsers on their devices, using the graphical user interface provided by the Web Server. It acts as the client, sending requests to the Web Server for various actions, such as creating tournaments, joining battles, or checking tournament rankings. The Computer serves as the interface for both educators and students to navigate the platform, contributing to a seamless and user-friendly experience.
- The **Firewall** is a security component in the CodeKataBattle (CKB) platform that protects against unauthorized access and cyber threats. It enforces rules and policies to control network traffic, safeguarding the platform from web vulnerabilities like Cross-Site Scripting and SQL injection. The Firewall collaborates with the Web Server to secure data transmission between users' devices (Computer) and the CKB platform, implementing protocols HTTPS for encrypted communication. By defining and enforcing access rules, the Firewall prevents unauthorised users from compromising the platform's security. It allows legitimate requests while blocking or filtering malicious traffic that could pose a threat to the integrity of the CKB platform.

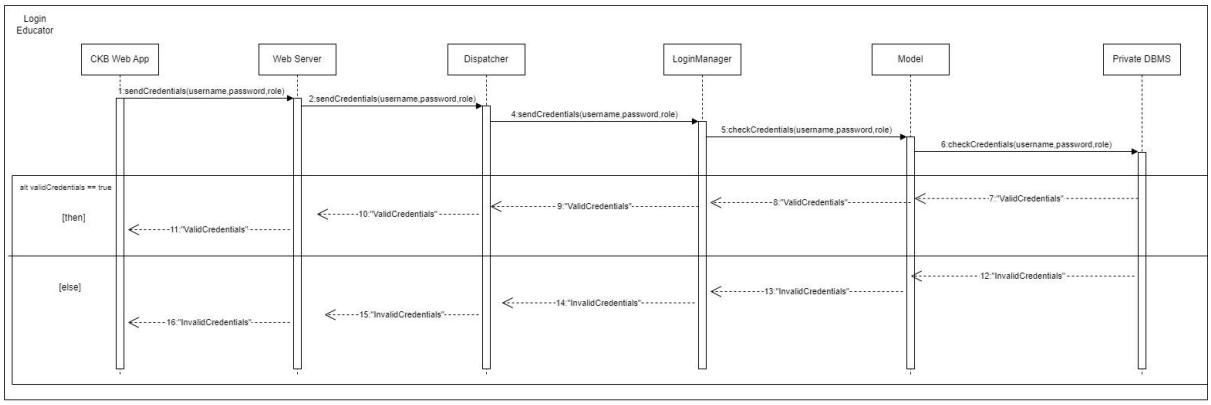
- The **Web Server** functions as the entry point for users accessing the CodeKataBattle (CKB) platform. It processes requests, validates user authentication, and dynamically manages interactions. For dynamic requests, such as creating tournaments or submitting code, the Web Server collaborates with the Application Server via established communication channels. The Application Server handles complex operations like GitHub integration and code evaluations. After completion, the Application Server returns results to the Web Server, which generates dynamic web pages containing relevant data (e.g., updated scores) and delivers them to users. For static content requests, such as tournament descriptions, the Web Server efficiently handles these locally, ensuring a responsive user experience without involving the Application Server.
- The **Load Balancer** serves as a critical component in the CodeKataBattle (CKB) platform's infrastructure, playing a pivotal role in optimising performance, ensuring fault tolerance, and efficiently managing incoming web traffic. Its primary function is to evenly distribute user requests across multiple instances of the Web Server. By monitoring the health of individual Web Server instances, the Load Balancer intelligently directs traffic only to healthy servers, preventing service disruptions and enhancing overall system reliability. In cases of increased demand or potential server failures, the Load Balancer dynamically adjusts the distribution of incoming requests, optimising resource utilisation and maintaining an efficient user experience. This ensures that no single server becomes overwhelmed, contributing to improved scalability and responsiveness. By distributing the load among multiple servers, it supports high availability and scalability, enabling the CodeKataBattle platform to handle many concurrent users without performance degradation.
- The **Application Server** is the processing backbone of the CKB platform, managing business logic, executing critical computations, and integrating with external services for efficient code kata evaluation and scoring. When the Web Server forwards requests, the Application Server takes charge of processing dynamic operations, such as GitHub integration, automated code evaluations, and updates to battle scores. It acts as the engine that executes the critical functionalities of the platform. Specifically, during code kata battles, the Application Server communicates with the GitHub repository and triggers automated workflows through GitHub Actions. It pulls the latest code, conducts static analysis using various tools to assess code quality, and executes the required tests to calculate and update the battle score for each team. The Application Server plays a key role in maintaining data integrity by interacting with the Database Server via TCP/IP, storing and retrieving essential information such as user profiles, tournament details, and battle scores. Its ability to dynamically handle complex computations ensures a smooth and efficient user experience. Additionally, the Application Server is designed to update battle scores in real-time as users submit new code, allowing participants and educators to track the evolving competition during a battle.

- The **Database Server** is a central repository within the CodeKataBattle (CKB) platform, responsible for storing and managing persistent data critical to the platform's functionality. This server stores and manages critical persistent data for the platform, such as user profiles, tournament details, code kata descriptions, battle scores, and other vital information. It employs cluster of databases managed via the MariaDB Galera Cluster. The Database Server is actively accessed by the Application Server, which communicates with it to store and retrieve data during various operations. For instance, when educators create new tournaments or battles, the details are stored in the database. Likewise, user profiles, including tournament scores and earned badges, are stored and updated in this central data repository. By maintaining a structured and organized database, the platform ensures the persistence and reliability of information. The Database Server contributes to the overall robustness of the CKB platform by providing a secure and efficient storage solution for the diverse data generated and managed within the system.

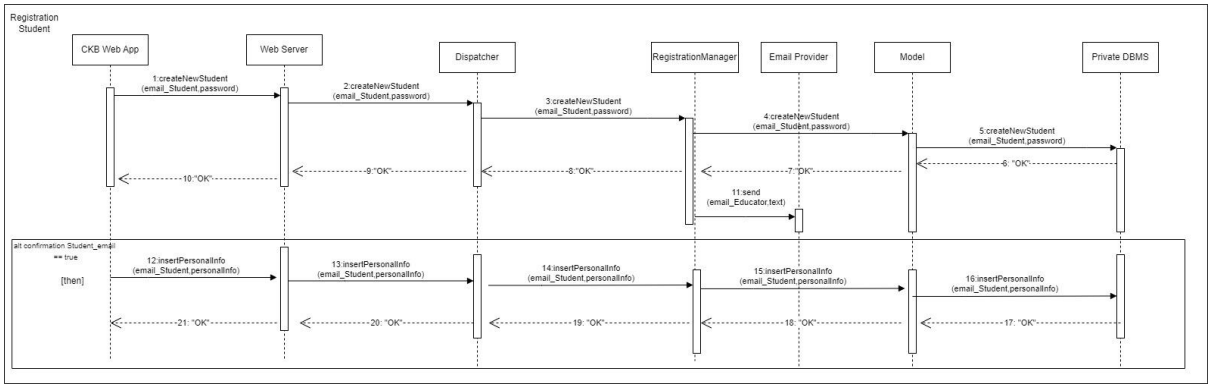
2.4 Runtime View



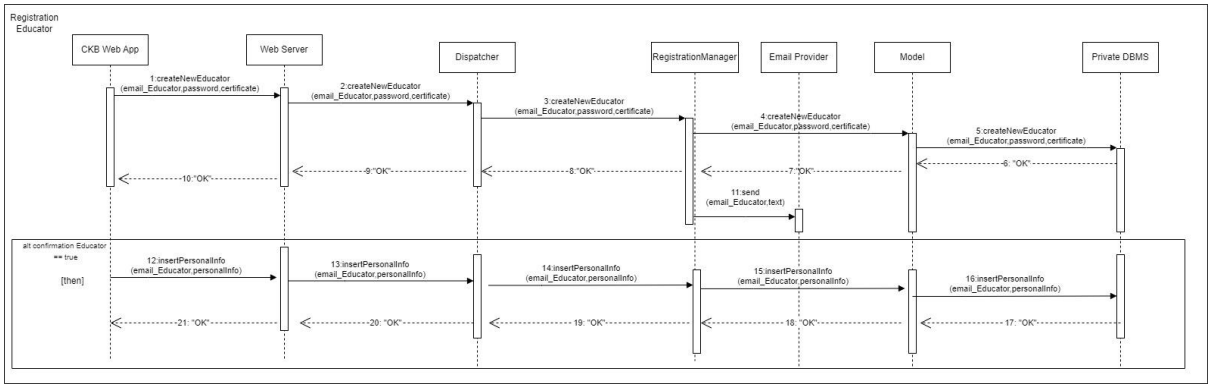
Student Login



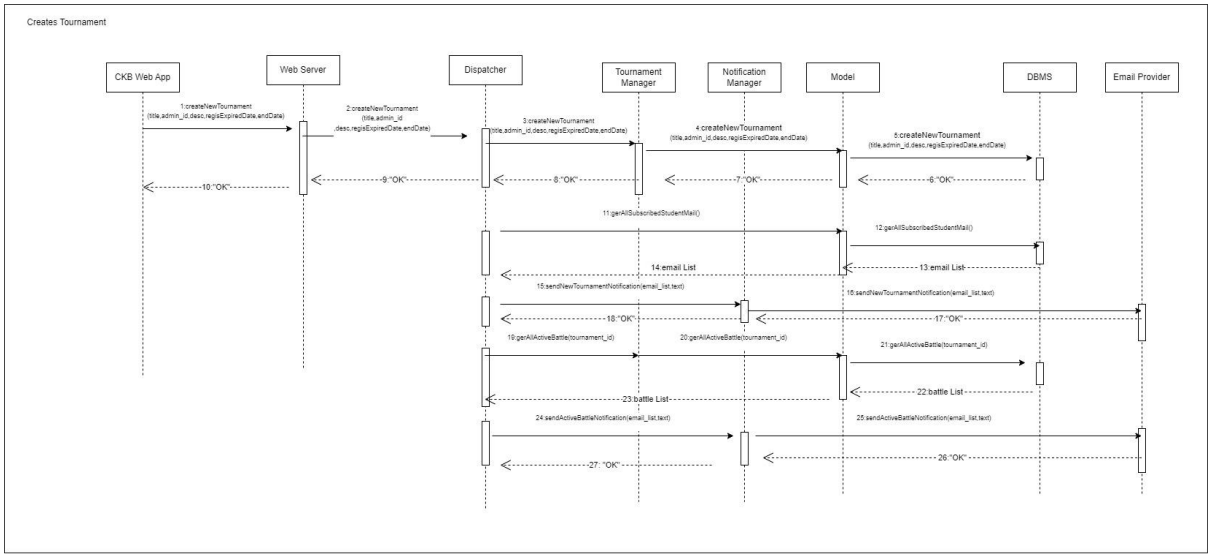
Educator Login



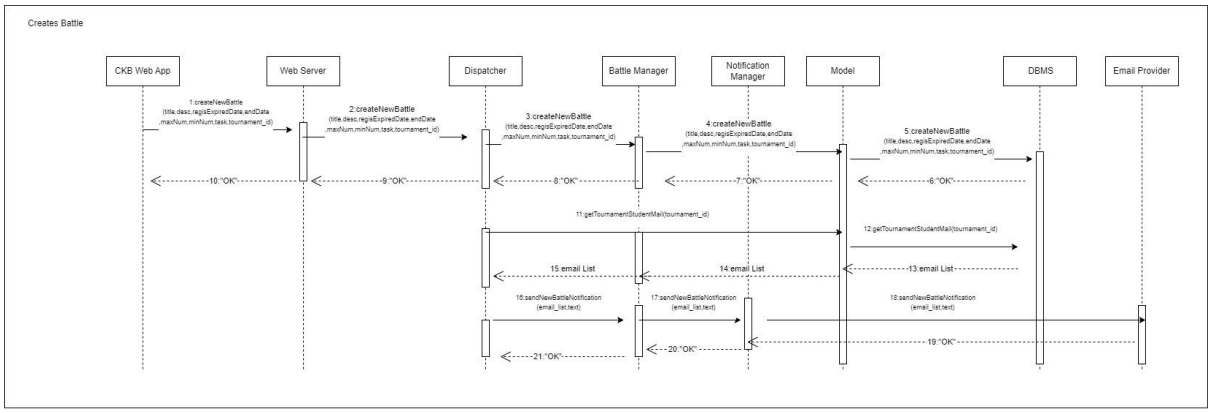
Student Registration



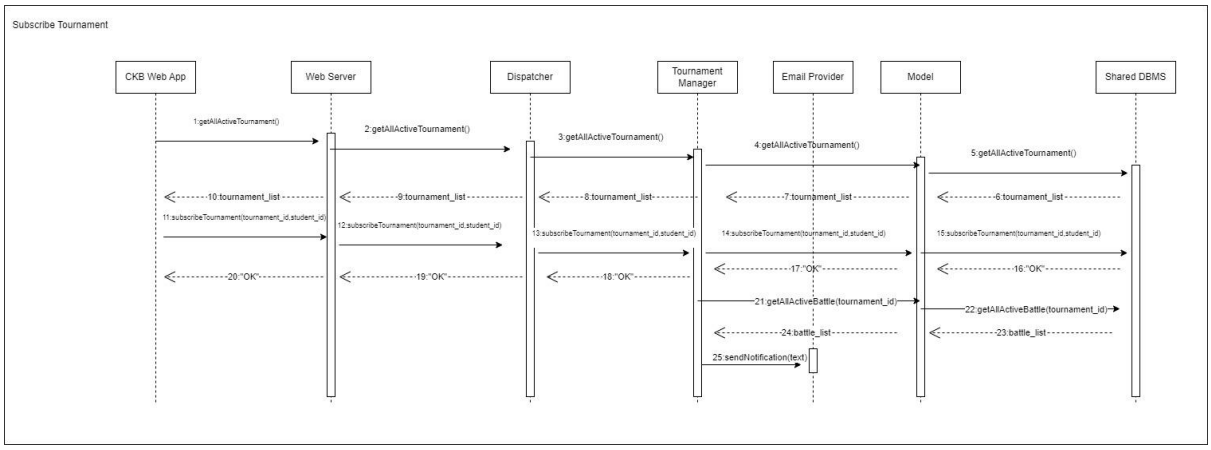
Educator Registration



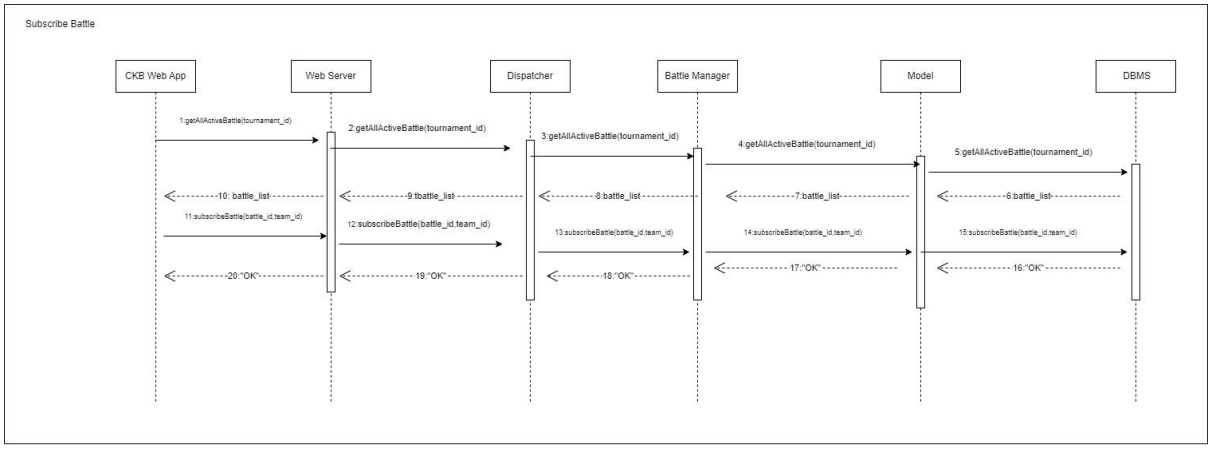
Create Tournament



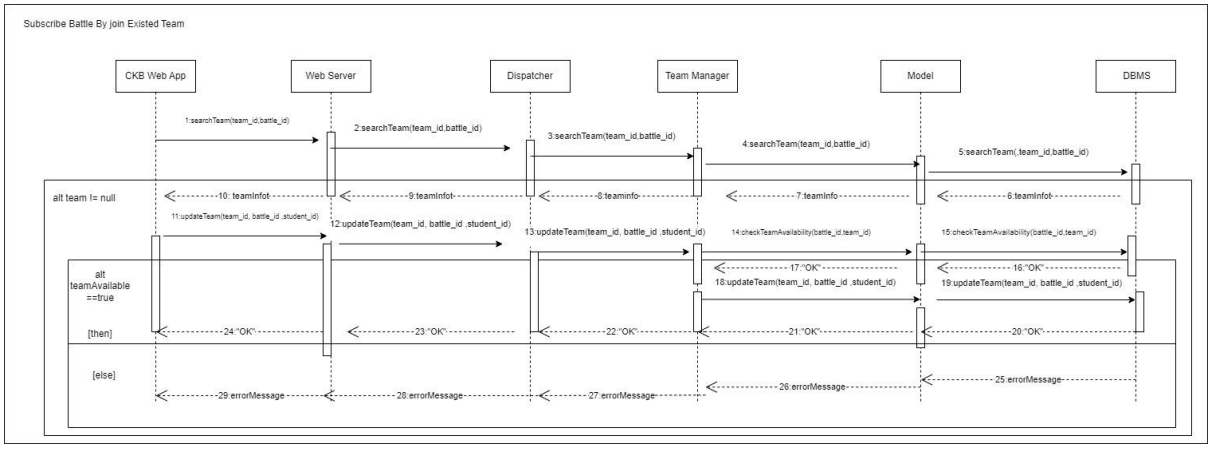
Create Battle



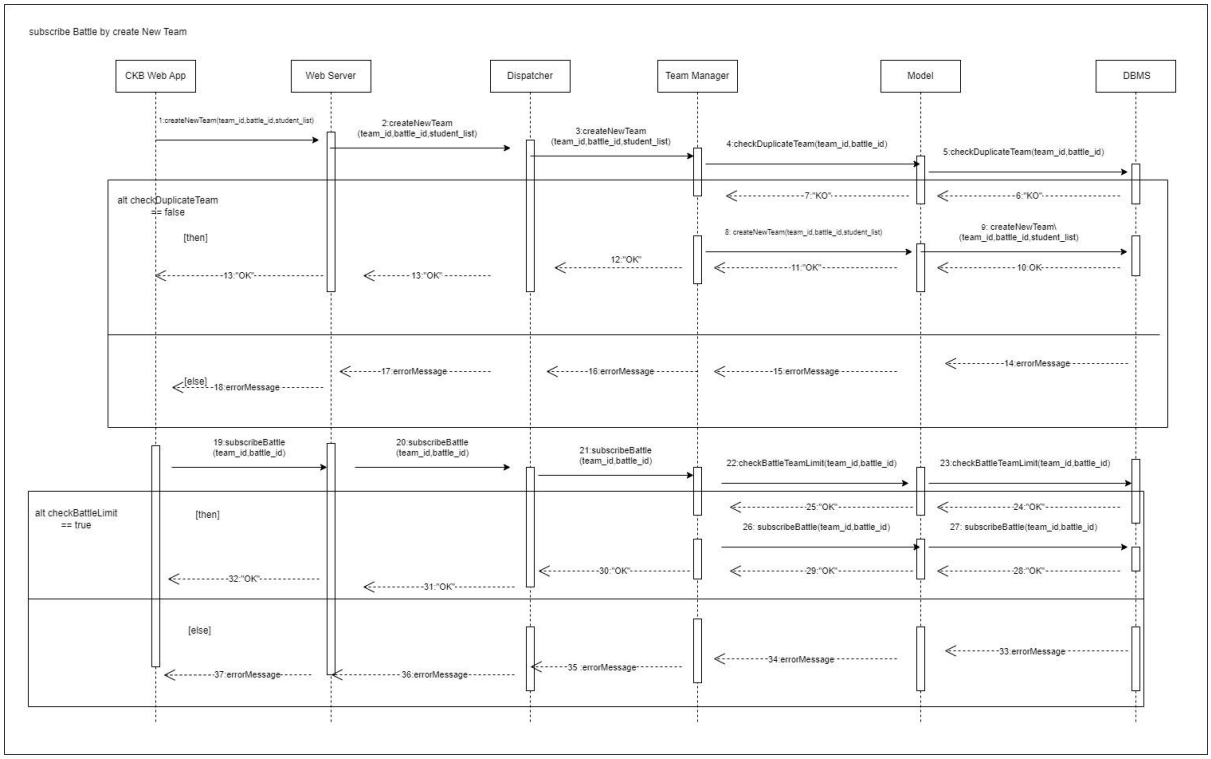
Subscribe Tournament



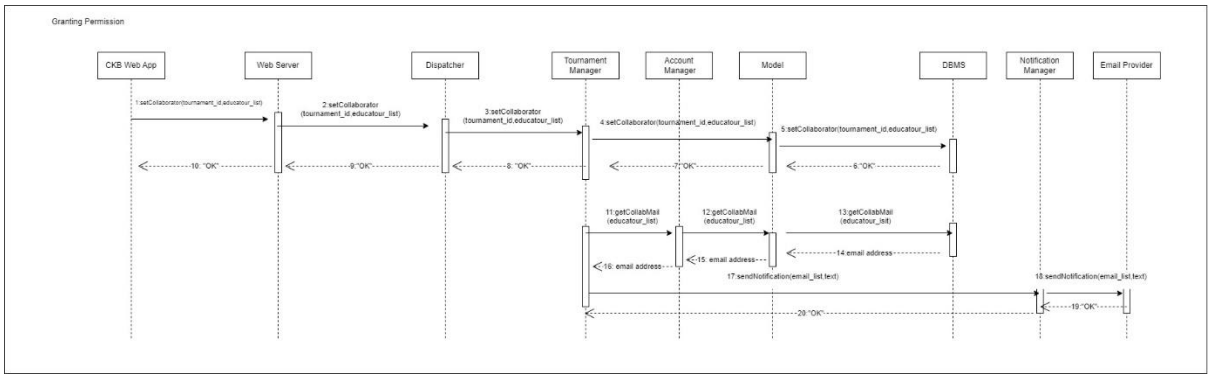
Subscribe Battle



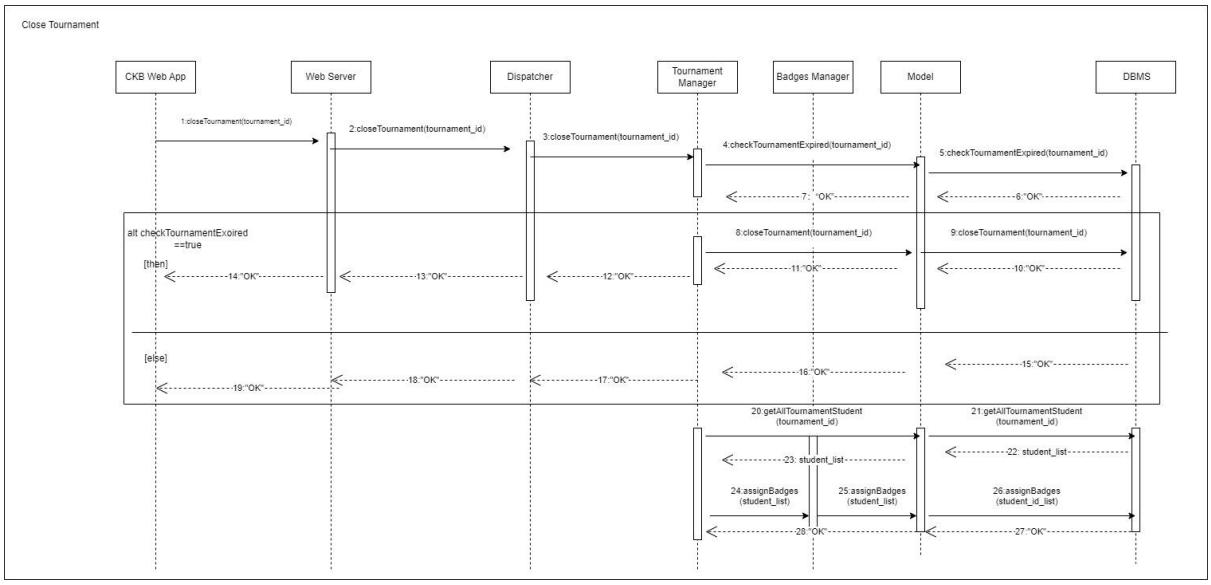
Subscribe Battle By Join Existing Team



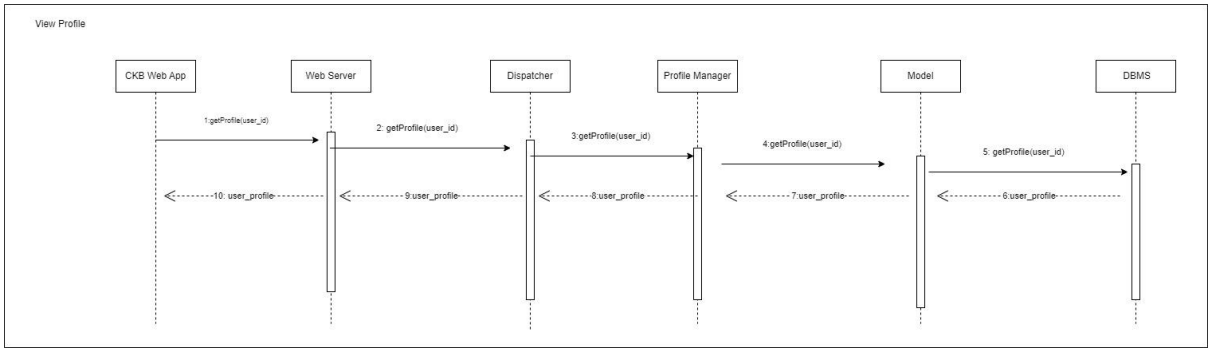
Subscribe Battle By Creating New Team



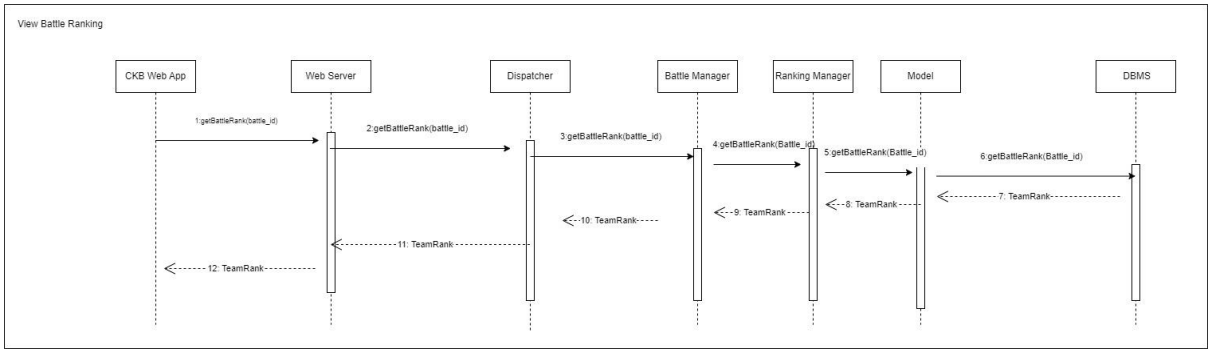
Granting Permission



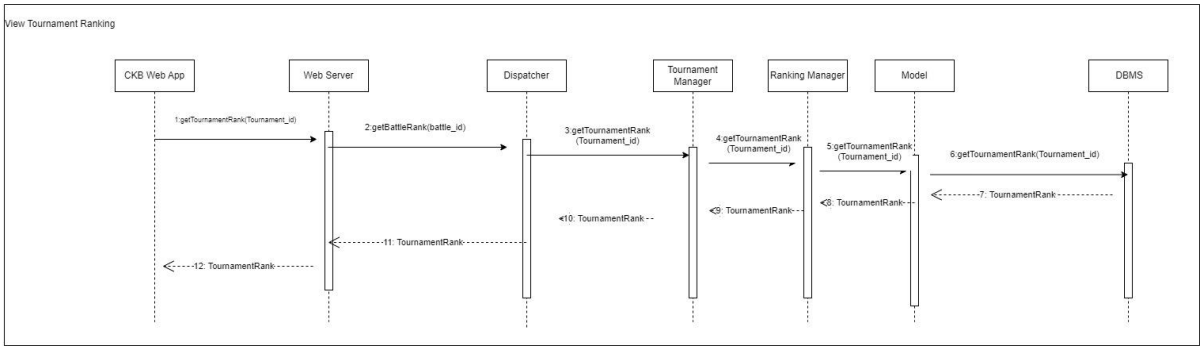
Closing Tournament



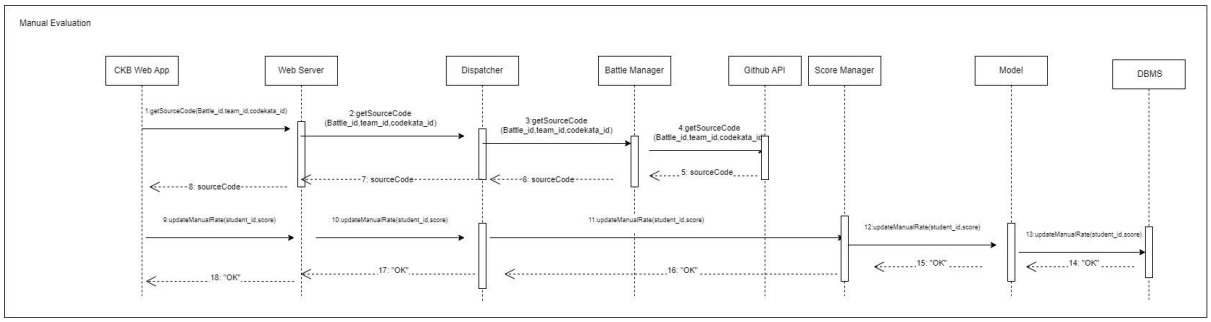
View Profile



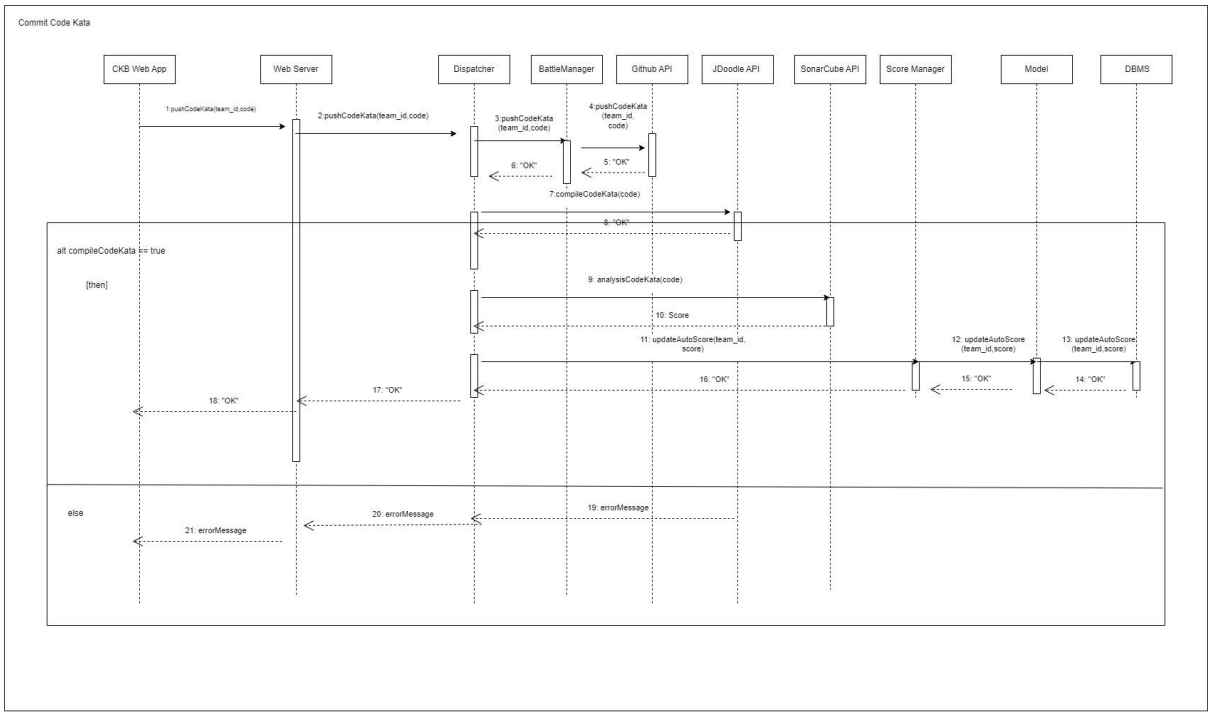
View Battle Ranking



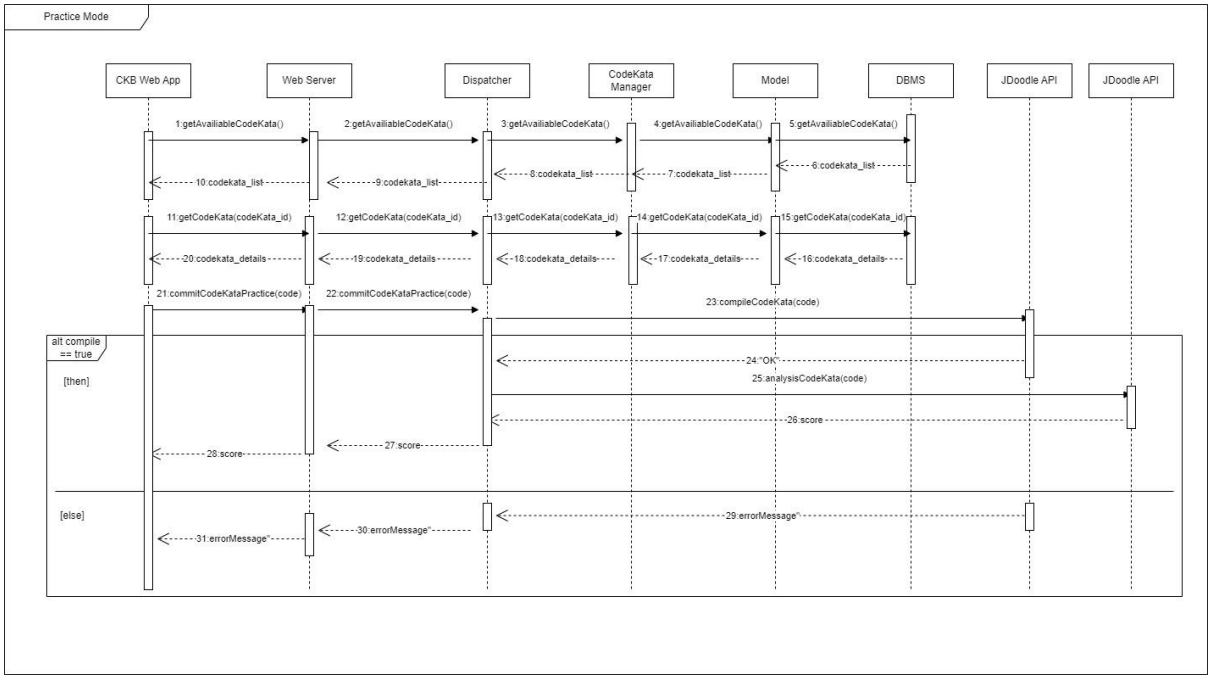
View Tournament Ranking



Manual Evaluation By Educator



Upload Code Kata Solution



Practice Mode

2.5 Component Interfaces

DispatcherI

-sendCredentials(username,password,role)
-createNewEducator(email_Educator,password,certificate)
-createNewStudent(email_Student,password)
-createNewTournament(title, admin_id,desc,regisExpiredDate,endDate)
-getAllSubscribedStudentMail()
-createNewBattle(title,desc,regisExpiredDate,endDate,maxNum,minNum,task,tournament_id)
-getTournamentStudentMail(tournament_id)
-getAllActiveBattle(tournament_id)
-subscribeBattle(battle_id,team_id)
-searchTeam(team_id,battle_id)
-updateTeam(team_id,battle_id,student_id)
-checkTeamAvailability(battle_id,team_id)
-createNewTeam(team_id,battle_id,student_list)
-setCollaborator(tournament_id,educatour_list)
-closeTournament(tournament_id)
-getProfile(user_id)
-getBattleRanking(battle_id)
-getTournamentRanking(Tournament_id)
-getSourceCode(Battle_id,team_id,codekata_id)
-updateManualRate(student_id,score)
-pushCodeKata(team_id,code)

-compileCodeKata(code)
-analysisCodeKata(code)
-updateAutoScore(team_id,score)
-getAvailiableCodeKata()

WebServerI

-sendCredentials(uername,password,role)
-createNewEducator(email_Educator,password,certificate)
-createNewStudent(email_Student,password)
-createNewTournament(title, admin_id,desc,regisExpiredDate,endDate)
-getAllSubscribedStudentMail()
-createNewBattle(title,desc,regisExpiredDate,endDate,maxNum,minNum,task,tournament_id)
-getTournamentStudentMail(tournament_id)
-getAllActiveBattle(tournament_id)
-subscribeBattle(battle_id,team_id)
-searchTeam(team_id,battle_id)
-updateTeam(team_id,battle_id,student_id)
-createNewTeam(team_id,battle_id,student_list)
-setCollaborator(tournament_id,educatour_list)
-closeTournament(tournament_id)
-getProfile(user_id)
-getBattleRanking(battle_id)
-getTournamentRanking(Tournament_id)
-getSourceCode(Battle_id,team_id,codekata_id)
-updateManualRate(student_id,score)
-pushCodeKata(code)
-getAvailiableCodeKata()

PrivateDBMSI

-checkCredentials(username,password,role)
-createNewEducator(email_Educator,password,certificate)
-createNewStudent(email_Student,password)
-getAllSubscribedStudentMail()
-getTournamentStudentMail(tournament_id)
-getCollabMail(educator_list)
-getAllTournamentStudent(tournament_id)
-updateManualRate(student_id,score)

SharedDBMSI

-createNewTournament(title, admin_id,desc,regisExpiredDate,endDate)

-createNewBattle(title,desc,regisExpiredDate,endDate,maxNum,minNum,task,tournament_id)
-getAllActiveBattle(tournament_id)
-checkDuplicateTeam(team_id,battle_id)
-subscribeBattle(battle_id,team_id)
-searchTeam(team_id,battle_id)
-checkTeamAvailability(battle_id,team_id)
-updateTeam(team_id,battle_id,student_id)
-setCollaborator(tournament_id,educatour_list)
-getProfile(user_id)
-getBattleRanking(battle_id)
-getAvailiableCodeKata()

TournamentManagerI

-createNewTournament(title,admin_id,desc,regisExpiredDate,endDate)
-sendNotification(email_list,text)
- getCollabMail(educatour_list)
-setCollaborator(tournament_id,educatour_list)
-closeTournament(tournament_id)
-getAllTournamentStudent(tournament_id)
-assignBadges(student_list)
-getTournamentRanking(Tournament_id)

BattleManagerI

-createNewBattle(title,desc,regisExpiredDate,endDate,maxNum,minNum,task,tournament_id)
-getAllActiveBattle(tournament_id)
-subscribeBattle(battle_id,team_id)
-getBattleRanking(battle_id)
-getSourceCode(Battle_id,team_id,codekata_id)
-pushCodeKata(code)

EmailNotificationManagerI

-sendNewTournamentNotification(email_list,text)
-sendNewBattleNotification(email_list,text)
-sendActiveBattleNotification(email_list,text)

TeamManagerI

-searchTeam(team_id,battle_id)
-updateTeam(team_id,student_id)
-checkTeamAvailability(battle_id,team_id)
-createNewTeam(team_id,battle_id,student_list)

-checkDuplicateTeam(team_id,battle_id)
-checkBattleTeamLimit(team_id,battle_id)

LoginI
-checkCredentials(uername,password,role)

RegistrationI
-createNewEducator(email_Educator,password,certificate)
-createNewStudent(email_Student,password)

AccountManagerI
-getCollabMail(educatour_list)

BadgesManagerI
-assignBadges(student_list)

ProfileI
-getProfile(user_id)

RankingManagerI
-getBattleRanking(battle_id)
-getTournamentRanking(Tournament_id)

Github API
-getSourceCode(Battle_id,team_id,codekata_id)

Score ManagerI
-updateManualRate(student_id,score)
-updateAutoScore(team_id,score)

CodeKata ManagerI
-getAvailiableCodeKata()
-getCodeKata(codekata_id)

OnlineCompilerMnagerI
-compileCodeKata(code)
-compileCodeKataPractice(code)

CodeAnalysisManagerI
-analysisCodeKata(code)

2.6 Selected Architectural Style and Patterns

2.6.1 Three-tiered architecture

We chose this architecture for many reasons:

- **Modularity and Maintainability**: this architecture divides the application into independent modules, handling user interface and data management separately. This modular design makes the system easier to understand, develop, and maintain.
- **Scalability**: by dividing the system into three layers, it becomes easier to scale and expand different functionalities. For instance, additional application servers can be added to handle more database servers can be added to manage a larger volume of data.
- **Reusability**: this architecture encourages code and component reuse. The functionalities can be shared across multiple user interfaces, and the data tier can be shared across multiple applications, enhancing code and functionality reuse.
- **Security**: placing the data tier in an internal network, accessed through the application tier, enhances access control and security for the database. Additionally using firewalls between the presentation tier and application tier, as well as between the application tier and data tier, contributes to overall system security.

2.6.2 Model View Controller (MVC)

The Model-View-Controller architectural pattern is a widely adopted design paradigm in software development that aims to provide a structured organized approach to building applications. MVC divides an application into three interconnected components, each with a specific responsibility:

- **Model:** represents the application's data. It is responsible for managing and processing data, as well as enforcing the business rules of the application. The model component is independent of the user interface and communicates any changes in data to the View.
- **View:** is responsible for presenting the application's data to the user. It receives inputs from the Model and renders it in a format suitable for display. The View is passive and does not handle user input or manipulate date directly. Instead, it listens for updates from the Model and adjusts the user interface accordingly.
- **Controller**: acts as an intermediary between the Model and the View. It is responsible for handling user input, processing it, and updating the Model accordingly. The Controller receives input from the user through the View, interprets it, and invokes the necessary actions in the Model. Any changes in the Model are then communicated back to the View for display.

The distribution of tasks between these three elements translates into an increased decoupling of the components which leads to a series of benefits such as reusability, simplicity of implementation, etc.

2.6.3 Observer Pattern

The choice of the Observer Pattern is justified in the context of the CKB for several reasons:
Real-time Notification and Updates: this pattern enables real-time communication between different components of the system. It allows the CKB platform to act as an. observer, promptly receiving notifications and updates when Students submit new code to GitHub. This is crucial for ensuring timely automated scoring and update during battle.
Decoupling of Components: this pattern helps decouple the sender (Students submitting code) from receiver (CKB platform). This decoupling promotes flexibility and maintainability, as changes one component to do not directly affect the others.
Support for Automated Evaluation: for automated evaluation of submitted code, this pattern facilitates the seamless integration of external tools (e.g., static analysis tools) by notifying the CKB platform of new code submissions. This supports the automated analysis of code quality, security and reliability, contributing to the overall team score calculation.
Real time Score Updates: this pattern ensures that the CKB platform receives immediate updates whenever Students push new commits to the main branch of their GitHub repositories. This real time feedback allows both Students and Educators to monitor the evolving battle score throughout the battle duration.
Scalability and Extensibility: by using this pattern, the system can easily scale to handle an increasing number of battles and participants. It provides a flexible and extensible architecture that accommodates future enhancements without requiring significant modifications to the existing codebase.

2.7 Other Design Decisions

In this section are presented some of the design decisions made for the system to make it work as expected.

Availability

Load balancing and replication are concepts that have been introduced in the system to increase the availability and reliability of the system. In this way the system would be as fault-tolerant as possible regarding data management and service availability.

3. User Interface Design

The majority of the key interfaces have already been introduced in the Requirements Analysis and Specification Document (RASD). In this Detailed Design (DD) document, in addition to the crucial interfaces, some additional ones have been included.

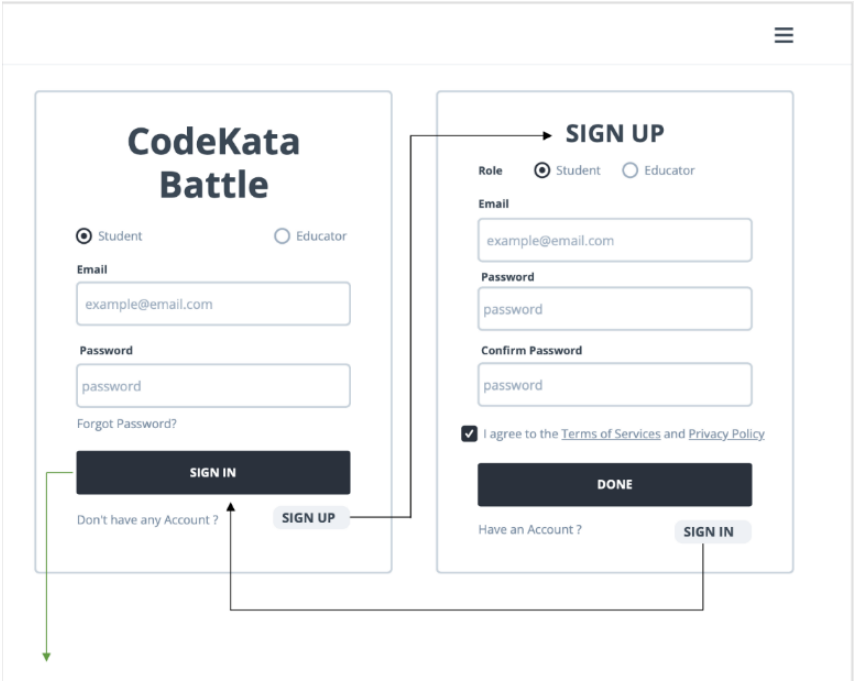


Figure 3.1 - CKB Login and Sign up Interface

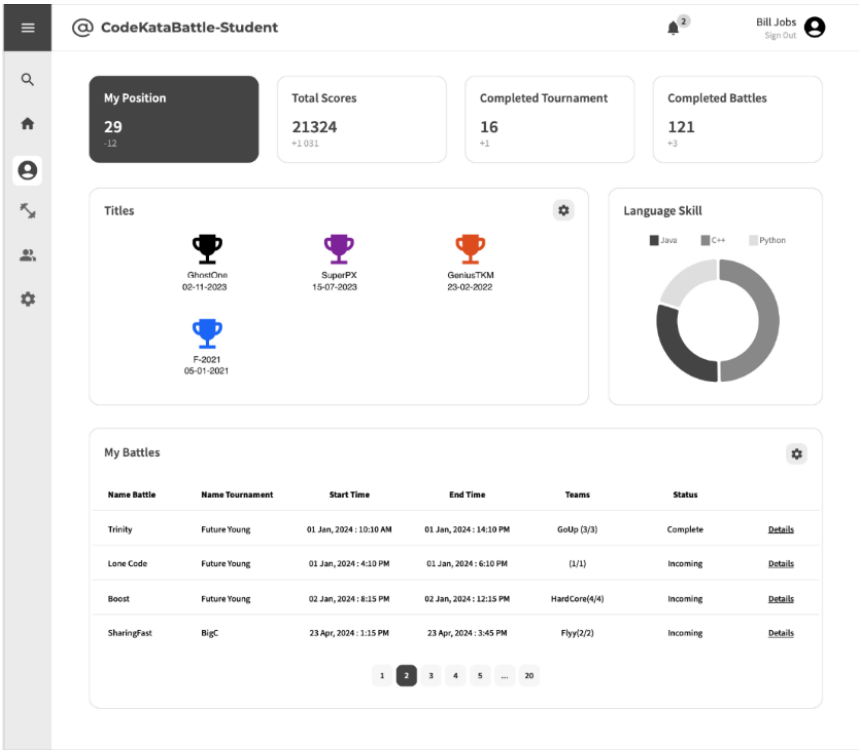


Figure 3.2 - CKB Student Home Page

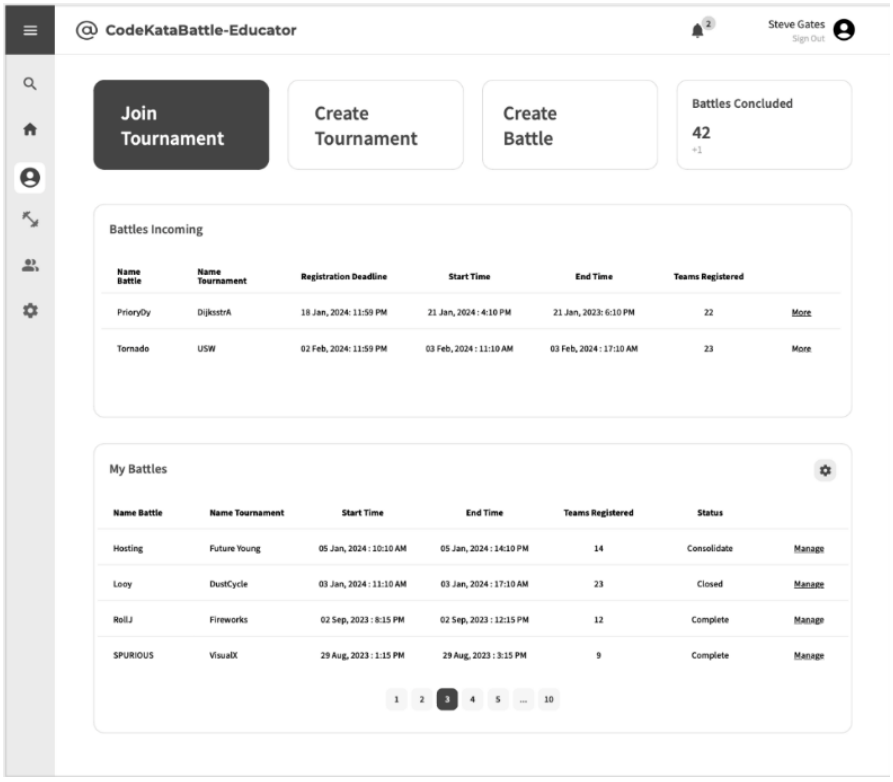


Figure 3.3 - CKB Educator Home Page

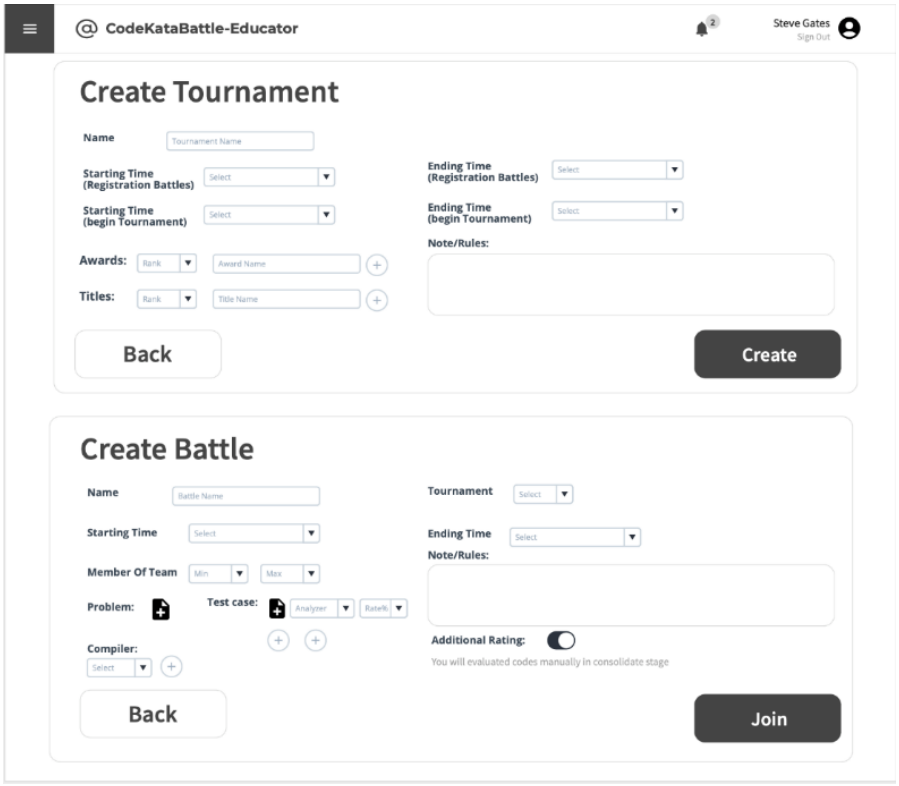


Figure 3.4 - CKB Educator Create Tournament Page

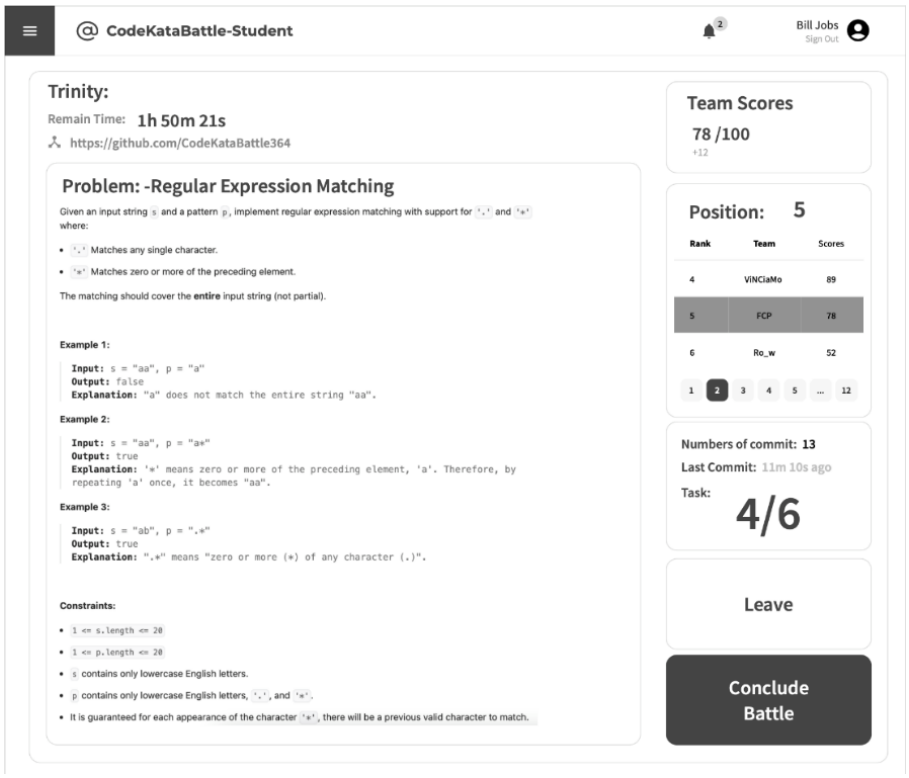


Figure 3.5 - CKB Code Kata Page

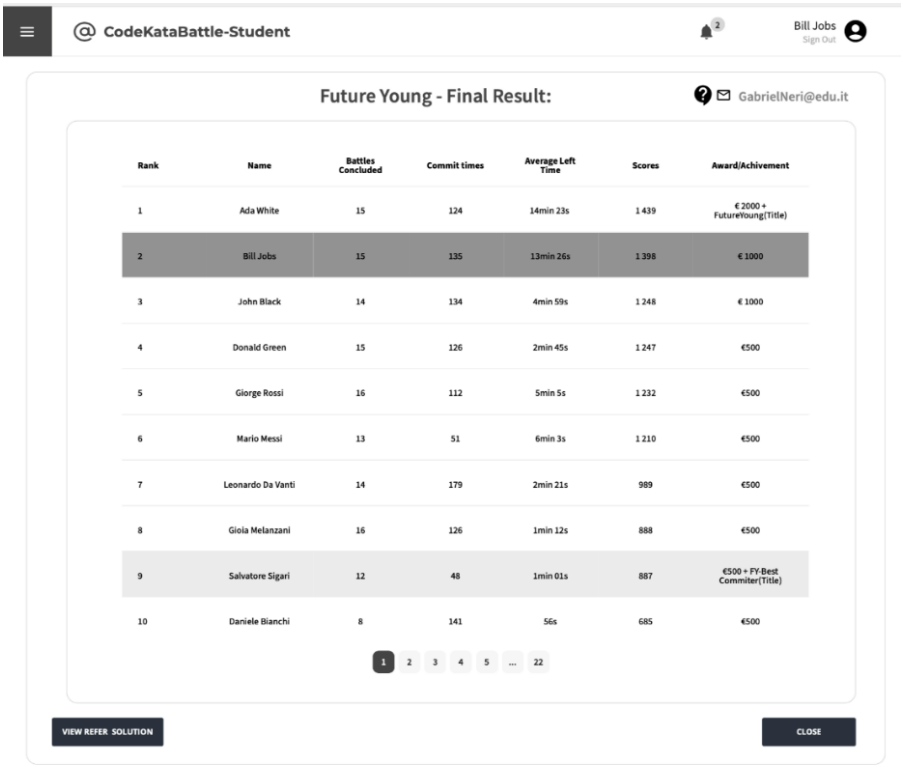


Figure 3.6 - CKB Result of a Tournament

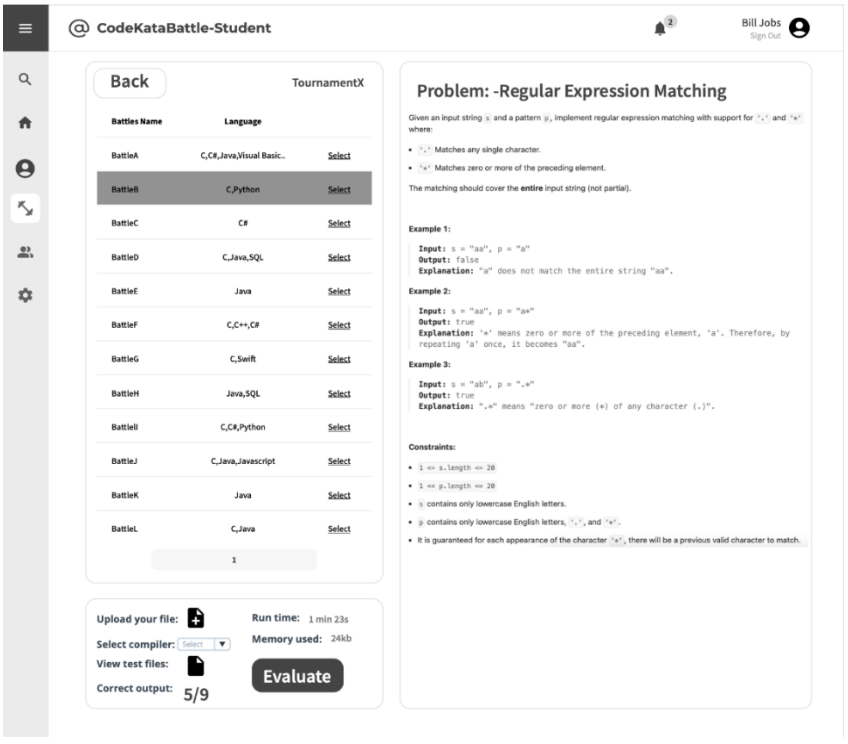


Figure 3.7 - CKB Battle Page

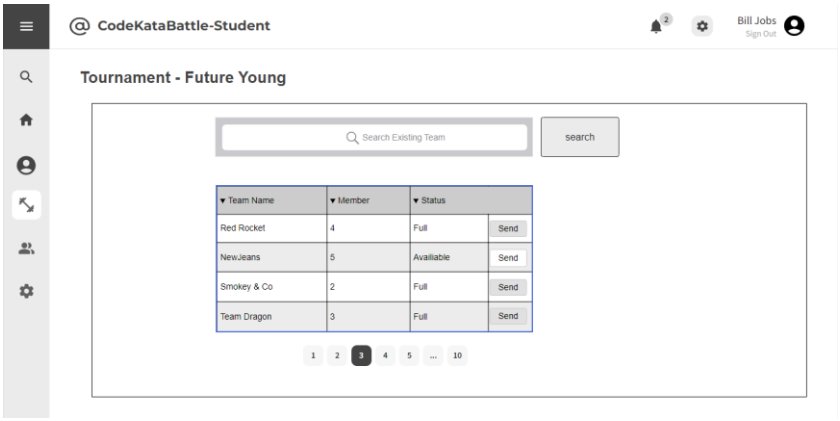


Figure 3.9 - CKB Search Existing Team

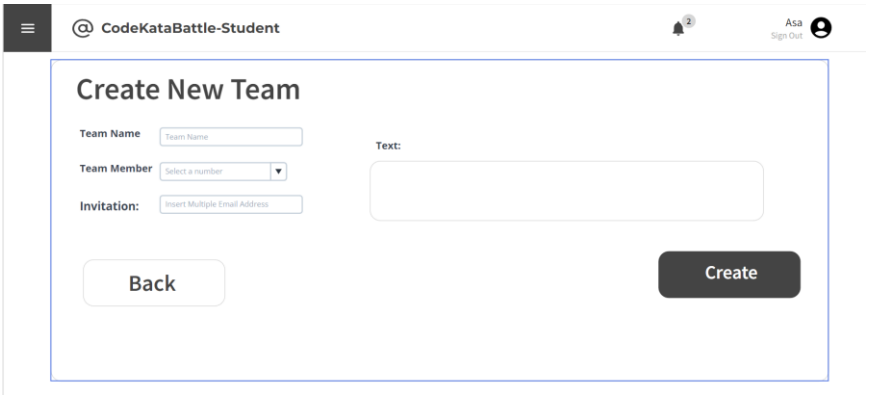


Figure 3.10 - CKB Create New Team

4. Requirements Traceability

This mapping demonstrates how each requirement is fulfilled by particular design components within the CodeKataBattle (CKB) system. It establishes a clear connection between the functional requirements and the system's architecture.

Requirements:	**[R1]** The system allows Students to sign up. **[R3]** The system allows Educators to sign up.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ RegistrationManager
	  ○ AccountManager
	  ○ Model
	• DBMS:
	  ○ PrivateDBMS

Requirements:	**[R2]** The system allows Students to sign in. **[R4]** The system allows Educators to sign in.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ LoginManager
	  ○ AccountManager
	  ○ Model
	• DBMS:
	  ○ PrivateDBMS

Requirements:	**[R5]** The system allows Students to access the profile section.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ ProfileManager
	  ○ AccountManager
	  ○ Model
	• DBMS:

	○ SharedDBMS

Requirements:	**[R7]** The system allows Students to choose their preferred programming language. **[R22]** The system allows Educators to upload code kata with referenced answer. **[R23]** The system allows Students to access referenced answer.
Components:	• WebApp • WebServer • Application Server: ○ Dispatcher ○ BattleManager / PracticeModeManager ○ CodeKataManager ○ Model • DBMS: ○ SharedDBMS

Requirements:	**[R8]** The system allows Students to explore and view details of individual tournaments. **[R9]** The system allows Students to subscribe to tournaments. **[R16]** The system allows Students to access a comprehensive list of battles in which they have previously participated. **[R18]** The system allows Educators to create new tournaments. **[R20]** The system allows Educators to grant specific individuals the authority to create battles within a tournament.
Components:	• WebApp • WebServer • Application Server: ○ Dispatcher ○ TournamentManager ○ Model • DBMS: ○ SharedDBMS

Requirements:	**[R10]** The system allows Students to explore and select specific battles with tournaments. **[R11]** The system allows Students to subscribe to battles. **[R19]** The system allows Educators to create new battles.

	[R24] The system allows Educators to set participant limits, registration deadlines and submission deadlines for each battle. **[R25]** The system allows Educators to configure grading parameters (e.g. security percentages...).

Components:	• WebApp
	• WebServer
	• Application Server:
	  ◦ Dispatcher
	  ◦ BattleManager
	  ◦ Model
	• DBMS:
	  ◦ SharedDBMS

Requirements:	**[R12]** The system notifies Students of a GitHub repository link associated with the selected battle. **[R32]** The system allows Educators to access to the code submitted by Students through GitHub commits.

Components:	• Application Server:
	  ◦ Dispatcher
	  ◦ BattleManager
	  ◦ RepositoryManager
	  ◦ NotificationManager
	• DBMS:
	  ◦ PrivateDBMS

Requirements:	**[R13]** The system allows Students to view their scores obtained in battles. **[R33]** The system allows Educators to send manual scores to the platform.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ◦ Dispatcher
	  ◦ BattleManager
	  ◦ ScoreManager
	  ◦ OnlineCompilerManager
	  ◦ CodeAnalysisManager
	  ◦ Model
	• DBMS:
	  ◦ SharedDBMS

Requirements:	**[R14]** The system allows Students to access battle raking.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ BattleManager
	  ○ RankingManager
	  ○ Model
	• DBMS:
	  ○ SharedDBMS

Requirements:	**[R15]** The system allows Students to access tournament ranking.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ TournamentManager
	  ○ RankingManager
	  ○ Model
	• DBMS:
	  ○ SharedDBMS

Requirements:	**[R17]** The system allows Students to choose the practice mode for ongoing learning and skill enhancement.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ PracticeModeManager
	  ○ OnlineCompilerManager
	  ○ CodeAnalysisManager
	  ○ Model
	• DBMS:
	  ○ SharedDBMS

Requirements:	**[R21]** The system allows Educators to setting titles and rules for tournaments, referred to as “Tournament Badges”.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ TournamentManager
	  ○ BadgesManager
	  ○ Model
	• DBMS:
	  ○ SharedDBMS

Requirements:	**[R26]** The system notifies Students upon the creation of a new tournaments. **[R29]** The system notifies Educators and Students once the tournament has concluded.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ TournamentManager
	  ○ NotificationManager
	• DBMS:
	  ○ SharedDBMS

Requirements:	**[R27]** The system notifies Students upon the updates of a new battle. **[R28]** The system notifies Students in advance of upcoming battles. **[R30]** The system notifies Educator and Students once the battle has concluded.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ BattleManager
	  ○ NotificationManager
	• DBMS:
	  ○ SharedDBMS

Requirements:	**[R31]** The system notifies Students of the battle results and the overall tournament results, including scores and rankings.
Components:	• WebApp

	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ RankingManager
	  ○ ScoreManager
	  ○ TournamentManager
	  ○ BattleManager
	  ○ NotificationManager
	• DBMS:
	  ○ SharedDBMS

Requirements:	**[R6]** The system allows Students to send invitations for specific battles. **[R34]** The system allows Students to invite other students to join the team for the battle.

Components:	• WebApp
	• WebServer
	• Application Server:
	  ○ Dispatcher
	  ○ BattleManager
	  ○ TeamsManager
	  ○ NotificationManager
	  ○ Model
	• DBMS:
	  ○ SharedDBMS

5. Implementation, Integration and Test Plan

In this section the plans to follow for the implementation, integration and testing of the system are described.

5.1 Implementation plan

The implementation of this system would follow a planned struct, followed a bottom-up logic, to avoid stub structures that would be more difficult to implement and test.
In particularly divided as follow:

- Internal Database: implemented to store essential data such as user information, tournaments, battles, scores, badges etc.;
  - Choose a database system, like MySQL,
  - Design and implemented the database schema based on model requests,
- External services: integrate external services such as GitHub API, JDoodle API, SonarQube API, email provider;
  - Set up API keys and credentials for external services
  - Develop modules to interact with GitHub API for repository management and commit tracking
  - Integrate JDoodle API for code execution during automated testing
  - Implement communication with SonarQube API for static code analysis
  - Set up email integration for notifications
- Application Server: implement the core logic and functionality of the application server which acts as the logic layer;
  - Choose a web framework
  - Implement user authentication and authorization
  - Develop APIs for tournament and battle creation, team formation, user registration etc.
  - Integrate with Internal Database for data retrieval and storage
  - Set up real time communication with the web server, like for scoring updates
- Web Server: created to handle user requests. Serve static assets and manage communication with application server;
  - Choose a web server platform
  - Configure the server to handle incoming HTTP requests and route the to the application server
  - Implement static file serving for fronted assets
  - Integrate with the application server for real time updates

- Client: develop the fronted client accessible through web browsers;
  - Choose a frontend framework
  - Design and implement user interfaces for Educators and Students
  - Implement user authentication on the client side

This structured approach allows for a systematic implementation, ensuring each component is functional before moving to the next layer.

5.1.1 Features identification

The features to implement in the system are extracted directly from the system requirements. It is important to notice that some of them require the implementation of entire components, while others require the implementation of more or less small portions of them. In the following there is a brief recap of the features of the system.

[F1] Sign-in and Sign-up
The two functionalities are strictly related, since the Sign-in result is somehow affected by the Sign-up function. Both parts (Student and Educator) are sharing the same interface. To implemented, we must develop a unified authentication system and ensure proper validation and security measures.

[F2] Code Kata Battle creation
This is one of the features used by Educator parts, it can be divided into two sub features: Tournament creation and Battles creation, the last one is requires uploading the code kata materials, like, description, problem, setting team size, registration and submission deadlines, and scoring configurations; it's also given in presence of the former one, Tournament creation.

[F3] Team formation and Battle participation
These features are used by Student parts, involves joining battles individually or inviting others, respecting group size limits set by Educators.

[F4] Scoring and ranking
The scoring features is divided by: automation evaluation and manual evaluation; the first one, automated evaluated of functional aspects, timeliness and source code quality, this is real time scoring updates during the code kata battle based on GitHub commits; the second one, manually evaluated by Educator if required at consolidation stage after the submission deadline, executing by tools available includes code compiler and code analyzer.
The ranking features calculate personal tournament scores for each student based on battle scores, updated after each battle conclusion, it generates ranking to measure student

performance in the context of a tournament.

[F5] Practice mode management
This is a feature used to all user registered on the platform, which can practice the code kata problem available stored in share databases.

[F6] User notification
This is the feature that manages the email notifications on user's PCs and/or smartphones.

5.2 Component Integration and Testing

In this section there is a description about how the components are integrated and communicate, and how the components are tested.

First, in figure 5.2.1 it clear that the first component to implemented is Model, and tested using a driver for the components that are still under implementation. These components are responsible for all the interactions with the database and are needed by the majority of all the other components.

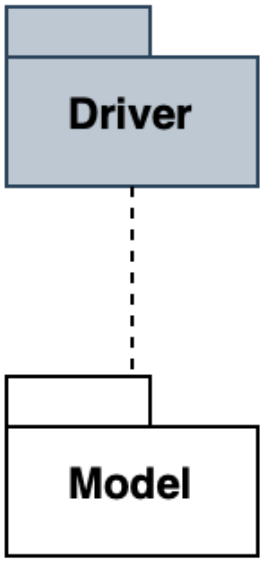


Figure 5.2.1

[F1] Sign-in and Sign-up implementation

In figure 5.2.2, it would be a good way to implement the sign-up method first and then the sign in one and then execute a unit and integration test with correspondence profileManager.

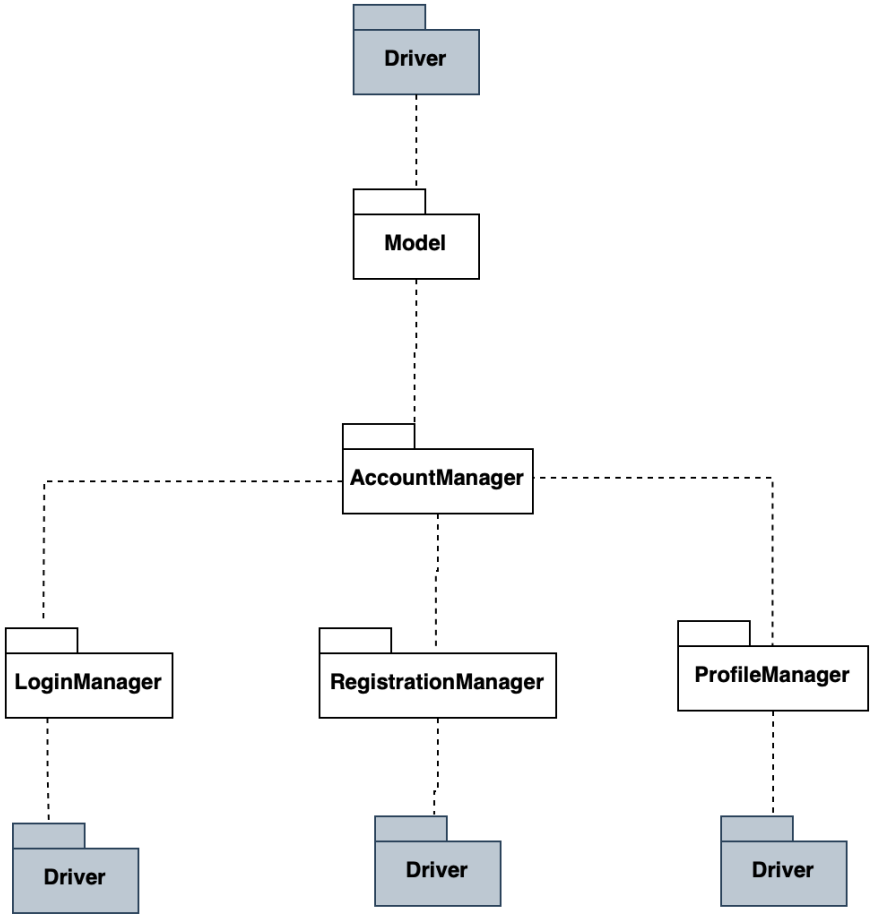


Figure 5.2.2

[F2] Code Kata Battle (Educator)

After F1, we can immediately implement features about Educator side, so in figure 5.2.3, we have Tournament and Battle manager for the core of the CKB platform.

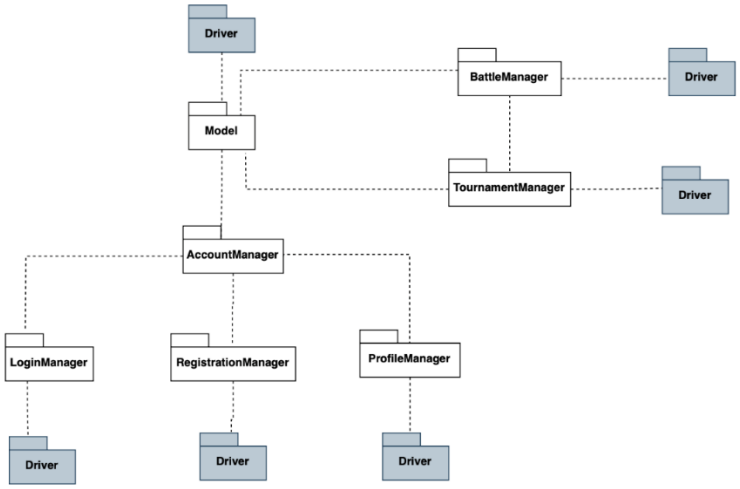


Figure 5.2.3

[F3] Code Kata Battle (Student)

In Sequence, the other side of Educator, Student side, we implemented TeamsManager component and the part of CodeKataManager for CKB battle's problems creation as follow figure 5.2.4

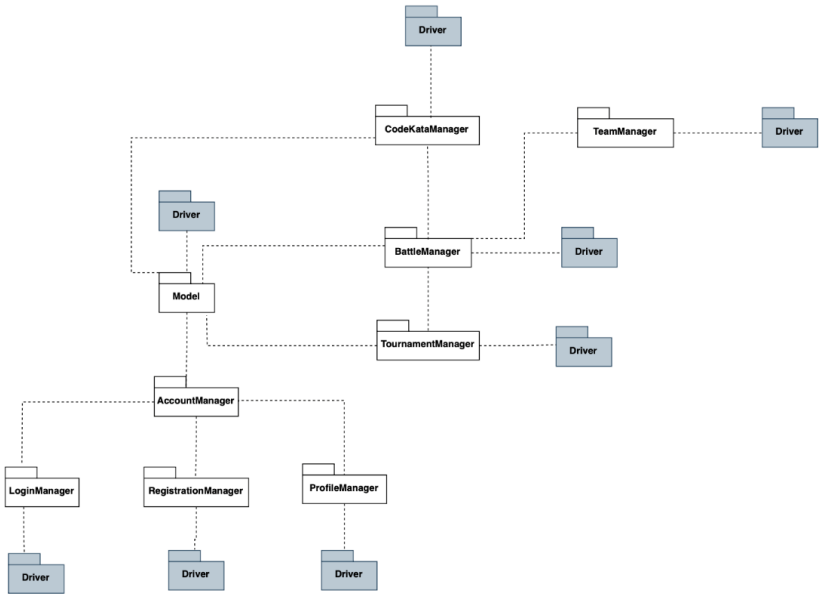


Figure 5.2.4

[F4] Scoring, ranking and badges

Now the scoring, in figure 5.2.5, ranking and badges Manager component is integrated in, noticed that badgesManager it does not need tested.

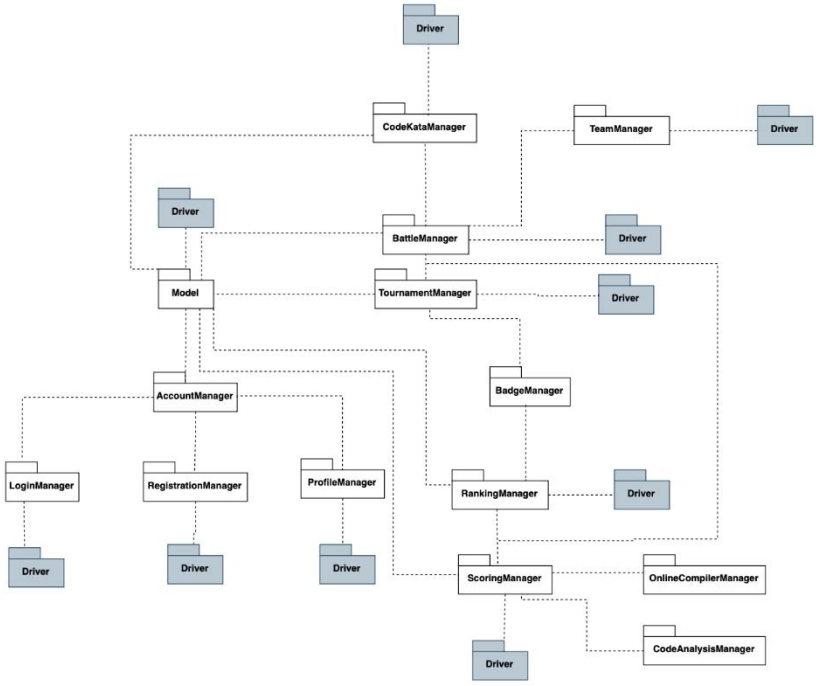


Figure 5.2.5

[F5] Practice Mode management

In additionally, we implemented, PracticeModeManager component for practice mode [figure 5.2.6]

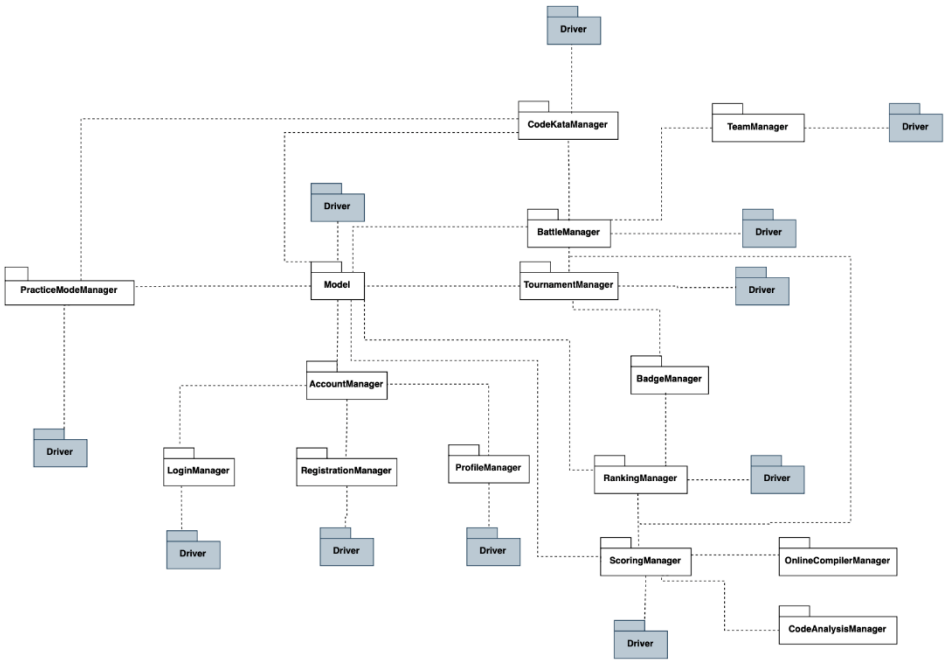


Figure 5.2.6

[F6] User notification

Finally, we are implemented the last component, NotificationManager, in figure 5.2.7

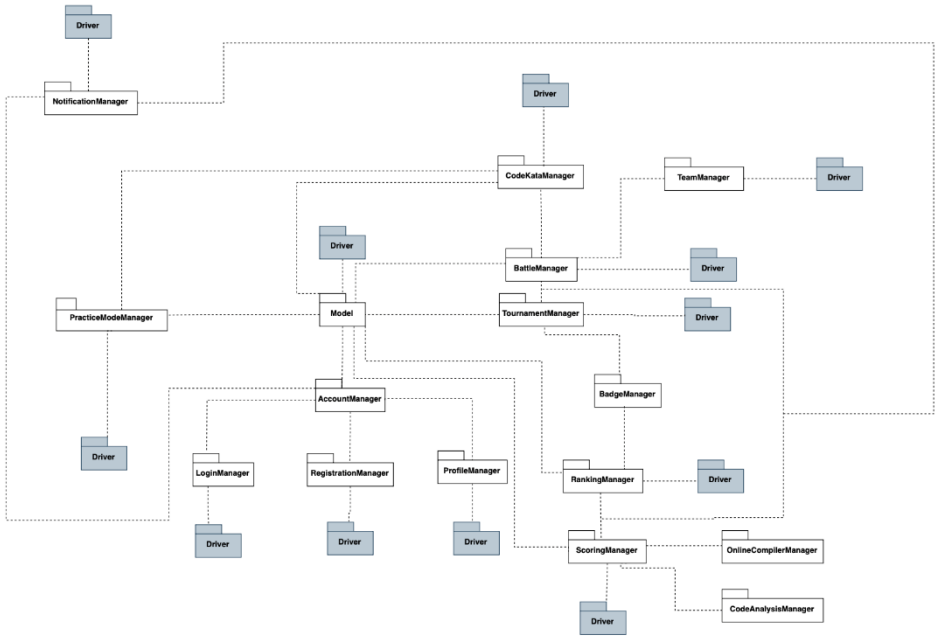


Figure 5.2.7

5.3 System testing

The CodeKataBattle (CKB) platform undergoes a comprehensive system testing process to ensure the reliability, performance, and usability of the system. The testing activities are conducted at different levels of granularity with varying scopes.

5.3.1 Component Testing

During the development phase, individual components or modules are tested in isolation to ensure they perform as expected. Some components may require the use of Driver and Stub components to simulate the behaviour of surrounding modules, including both expected and unexpected behaviours to assess the component's robustness.

5.3.2 Integration Testing

After individual component testing, the system undergoes integration testing. CKB employs a combination of the bottom-up and thread strategies for integration and testing, ensuring that components work seamlessly together.

5.3.3 System Testing

Once the system is implemented, unit tested, and integrated, it undergoes comprehensive system testing to verify that all features have been developed correctly and align with the functional and nonfunctional requirements outlined in the Requirements Analysis and Specification Document (RASD). This testing phase is not exclusive to developers and may involve stakeholders.

System Testing Steps:

1. **Functional Testing**
   - Verify that the system satisfies all the requirements specified in the RASD.
   - Explore opportunities for new features that could enhance the user experience.

2. **Performance Testing**
   - Identify bottlenecks and inefficient algorithms affecting response time, utilization, and throughput.
   - Uncover inefficiencies that may impact system performance.

- Conduct testing with an expected workload and establish acceptable performance targets.

3. **Usability Testing**
   - Evaluate how well users can utilize the CKB platform, including both the web application and smartphone application.
   - Prioritize ease of use, ensuring the system is user-friendly.

4. **Load Testing**
   - Expose bugs such as memory leaks, mismanagement of memory, or buffer overflow.
   - Identify upper limits of system components by increasing the load until reaching the threshold.

5. **Stress Testing**

   Ensure the system recovers gracefully after failures.

   Test the system's resilience by overwhelming its resources or taking resources away.

The system testing phase is crucial to ensure the overall quality and reliability of the CKB platform. It involves diverse testing methods to address functional, performance, usability, load, and stress aspects of the system.

5.4 Additional specifications on testing

In addition to the standard testing procedures mentioned earlier, the CodeKataBattle (CKB) platform incorporates specific testing specifications to ensure thorough validation and quality assurance.

5.4.1 Security Testing:

- **Authentication and Authorization**
  - Verify the effectiveness of user authentication and authorization mechanisms.
  - Test for vulnerabilities related to unauthorized access to sensitive information.

- **Data Encryption**
  - Ensure that data transmitted and stored within the CKB platform is adequately

encrypted.
    - Test for vulnerabilities related to data integrity and confidentiality.

- **Input Validation**
    - Validate user inputs to prevent common security threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

5.4.2 Compatibility Testing:

- **Cross-Browser Compatibility:**
    - Verify that the CKB platform functions correctly across multiple web browsers (e.g., Chrome, Firefox, Safari, Edge).
    - Identify and resolve any issues related to browser-specific functionalities.

5.4.3 Accessibility Testing:

- **Compliance with Accessibility Standards**
    - Confirm that the CKB platform adheres to accessibility standards (e.g., WCAG) to ensure inclusivity for users with disabilities.
    - Test features such as screen reader compatibility, keyboard navigation, and text alternatives.

5.4.4 Recovery and Redundancy Testing:

- **Data Recovery**
    - Verify the effectiveness of data recovery mechanisms in case of system failures or unexpected outages.
    - Test the restoration of data to ensure minimal data loss.

- **Redundancy Measures**
    - Validate the redundancy measures in place to maintain system availability.
    - Test failover mechanisms to ensure continuous operation during server or infrastructure failures.

5.4.5 Regression Testing:

- **Code Changes Impact:**
    - Perform regression testing after each code change or system update to ensure that existing functionalities remain unaffected.

- Automated testing tools may be employed to streamline regression testing processes.

5.4.6 User Acceptance Testing:

- **End-User Validation:**
  - Involve end-users and stakeholders in the testing process to validate the platform's usability and functionality.
  - Gather feedback to make iterative improvements based on user experience.

These additional testing specifications are integrated into the testing processes of CKB to address specific aspects such as security, compatibility, accessibility, recovery, and user acceptance. The aim is to ensure a robust, secure, and user-friendly platform that meets the diverse needs of its users.

6. Effort Spent

The time tables written below represent just an approximation of the time spent for the writing and discussions the team had for each specific chapter of this document. These times have not been measured while producing this document and are just based on the personal perception the team members have of the time spent.

Simona Cai

Chapter	Effort (in hours)
1	2
2	20
3	14
4	15
5	5

Yang Hao Mao

Chapter	Effort (in hours)
1	1
2	19
3	9
4	3
5	25

Jiaxiang Yi

Chapter	Effort (in hours)
1	3
2	17
3	15
4	9
5	3

7. References

- Mockups made with: moqups.com
- Testing models and component architecture made with: Draw.io
- Sequence diagram models made with: Draw.io
- High level architectures are made using www.draw.io
- Software Engineering 2 course material - Politecnico di Milano 2023-2024