POLITECNICO DI MILANO

SOFTWARE ENGINEERING II

CodeKataBattle

Design Document

Version 1.0

Daniel Maurio Ruiz Suarez
Sebastian Enrique Perea Lopez

January 07, 2024

Contents

1. INTRODUCTION

1.1. Purpose

The main objective of the design document (DD) is to serve as a crucial element for the design considerations, technical specifications and architectural decisions underlying the development of the CodekataBattle (CKB) project. It will serve as a guide for developers, providing insights of the sense and meaning behind the selected design, patterns and principles employed to build and develop the system.

In the context of CKB, an innovative platform for code-driven competitions, the DD outlines the intricacies of various components and their interactions. It delves into the user management system, tournament and battle management, team formation, automated evaluation processes, scoring mechanisms, and notification systems. By providing an in-depth understanding of these components, the DD facilitates the implementation of a robust and efficient system.

1.2. Scope

The Scope of this document searches to cover the entire of the design process of CKB, from high-level architecture to the detailed user interface elements. It sets the boundaries within which the design decisions have been made and clarifies the aspects of the system covered in this document.

It illustrates how the users, both students and educators, engage and interact with the platform, the students can form their teams, participate in code battles, and receive automated and manual evaluations. Additionally, the document covers aspects such as GitHub integration, static analysis, and real-time notifications, contributing to a comprehensive overview of the system's functionality.

1.3. Definitions, Acronyms, Abbreviations

A comprehensive list of definitions for terms, acronyms, and abbreviations used throughout the document to facilitate understanding.

1.3.1. Definitions

- **CodeKataBattle (CKB):** The collaborative platform designed to improve the students software development skills through code katas and team-based programming challenges.
- **Design Document (DD):** A comprehensive document that provides details about the architecture, components, and design of a software system.
- **User Interface (UI):** The visual elements and interactive components of a software application that users interact with.
- **Static Analysis:** The process of examining code without executing it, focusing on identifying potential issues and improving code quality.
- **Data Flow Diagram (DFD):** A graphical representation of the flow of data within a system, illustrating how information is processed and exchanged.
- **Database (DB):** A structured collection of data organized in a way that allows easy retrieval, management, and updating.
- **Application Programming Interface (API):** A set of rules that allows one software application to interact with another, facilitating the integration of different systems.

- **Graphical User Interface (GUI):** The visual interface that allows users to interact with electronic devices through graphical elements such as icons and buttons.

1.3.2. Acronyms

- **CKB:** CodeKataBattle
- **API:** Application Programming Interface
- **HTML:** Hypertext Markup Language

1.3.3. Abbreviations

- **[SQL]:** Structured Query Language
- **[URL]:** Uniform Resource Locator
- **[UMS]:** User Management System
- **[DBMS]:** Database Management System
- **[TBMS]:** Tournament and Battle Management System
- **[AES]:** Automated Evaluation System
- **[TGIS]:** Team Formation and GitHub Integration System
- **[SS]:** Scoring System
- **[NS]:** Notification System

1.4. Revision history

1.4.1. Version 1.0 (07/01/2024):

Initial release of the Design Document (DD) for CodeKataBattle.

1.5. Reference Documents

- 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.
- DD Sample from A.Y. 2022-2023.

1.6. Document Structure

The document is structured to provide a systematic exploration of the design elements, starting with an introduction to each major component and subsequently delving into detailed class diagrams, composite structure diagrams, and statecharts. This hierarchical structure aids developers in comprehending the system's architecture, behavior, and interactions.

1. **Introduction:** The introduction sets the context for the Design Document, offering a brief overview of its purpose, scope, and structure. It serves as a starting point for readers.
2. **Architectural Design:** This section explores the major components of the CKB system, presenting a high-level understanding of the architecture through class diagrams, composite structure diagrams, and statecharts.

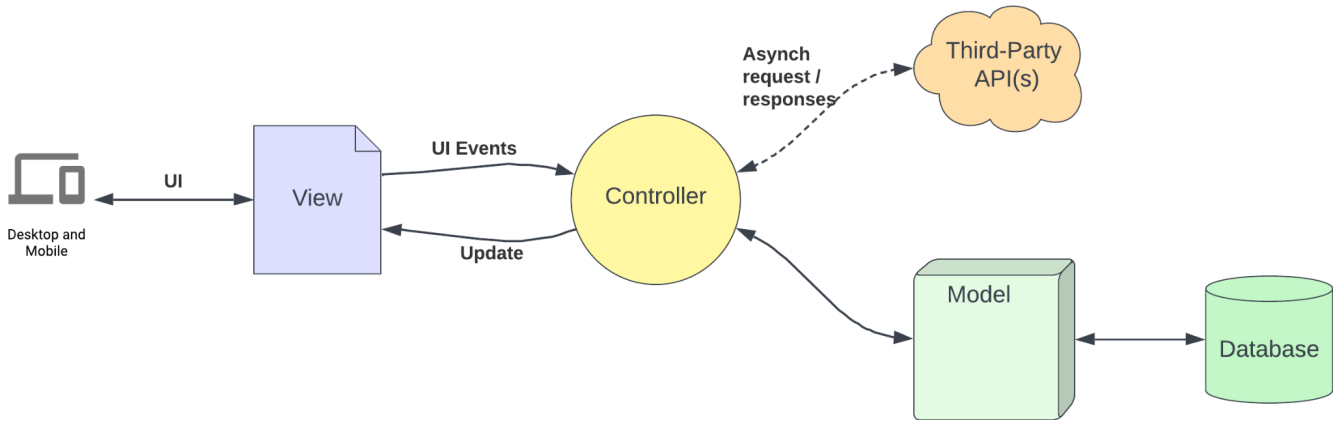
3. **User Interface Design:** Focusing on visual aspects, this section outlines the design considerations for the user interface, ensuring an intuitive experience for students and educators.
4. **Requirements Traceability:** Establishing a map between design decisions and the initial requirements, this section systematically aligns design elements to corresponding requirements from the RASD.
5. **Implementation, Integration and Test plan:** Outlining the plan for system implementation, integration, and testing, this section provides a roadmap for developers to bring the CKB system to fruition.
6. **Effort spent**
7. **References:** Compiling external resources, standards, or documentation referenced in the DD, this section ensures transparency and credibility.

2. ARCHITECTURAL DESIGN

2.1. Overview

The goal of this chapter is to define the architectural structure of the CodeKataBattle (CKB) platform, emphasizing the utilization of Django as the primary framework. It will outline the system's components, their interactions, and the rationale behind choosing Django for this application.

The proposed architecture for CKB is a Model-View-Controller (MVC) pattern (Figure 1), leveraging Django's inherent MVC-like architecture. Django's design aligns with the project's requirements, offering modularity, scalability, and extensibility, crucial for managing code kata battles, user interactions, and automated evaluations.



(Figure 2.1. MVC pattern)

- **Models:** Define the data structure and relationships. Models in Django will represent entities like Users, Tournaments, Battles, Teams, etc. Each battle will have associated models for storing metadata, submissions, and scores.

- **Views:** Handle user interactions and data presentation. Django views will process requests, render necessary templates, and manage the logic for battle creation, team formations, submissions, and score updates.

- **Controllers (or URLs/Endpoints):** Manage the flow of requests and connect views with models. Django's URL patterns will map incoming requests to specific views/controllers responsible for handling them.

- **Template Engine:** Utilize Django's template engine for rendering dynamic HTML content, enabling educators and students to interact with the platform through a user-friendly interface.

2.1.1. High-level components and their interaction

- **User Management Component:**
  - Responsible for user authentication, registration, and authorization.
  - Interacts with the database to store user information and permissions.
  - Interaction: Connected to various modules for access control and user-specific functionalities.

- **Tournament and Battle Management Component:**
  - Handles the creation, subscription, and management of tournaments and battles.
  - Interacts with the User Management Component for educator permissions and user subscriptions.
  - Interaction: Interfaces with the Team Formation and GitHub Integration Component to facilitate battle setup and notifications.

- **Team Formation and GitHub Integration Component:**
  - Enables students to form teams, join battles, and integrate with GitHub for code repository management.
  - Communicates with the Tournament and Battle Management Component for battle-specific information and deadlines.
  - Interaction: Communicates bidirectionally with the Automated Evaluation Component for code submission triggering and updates.

- **Automated Evaluation Component:**
  - Conducts automated evaluation of submitted code for functional aspects, timeliness, and code quality.
  - Utilizes static analysis tools and test case execution to assess code quality and functionality.
  - Interaction: Receives code submissions triggered by GitHub integration and updates the scoring system accordingly.

- **Scoring System Component:**
  - Calculates scores based on automated evaluation factors such as test case passes, timeliness, and code quality.
  - Allows for the incorporation of educator-assigned manual scores.
  - Interaction: Integrates with all evaluation components to compute battle scores and updates user profiles.

- **Notification Component:**
  - Manages notifications regarding tournament/battle updates, deadlines, and score availability.

- Sends notifications to subscribed users and educators based on system events.
- Interaction: Triggered by events from various components, particularly Tournament and Battle Management and Scoring System Components.

2.1.2. Interactions Overview:

- **User Actions:** Users interact with the User Management Component to register, authenticate, and access functionalities based on their roles (educator or student).

- **Educator Actions**: Educators utilize the Tournament and Battle Management Component to create tournaments, define battles, set deadlines, and configure evaluation criteria.

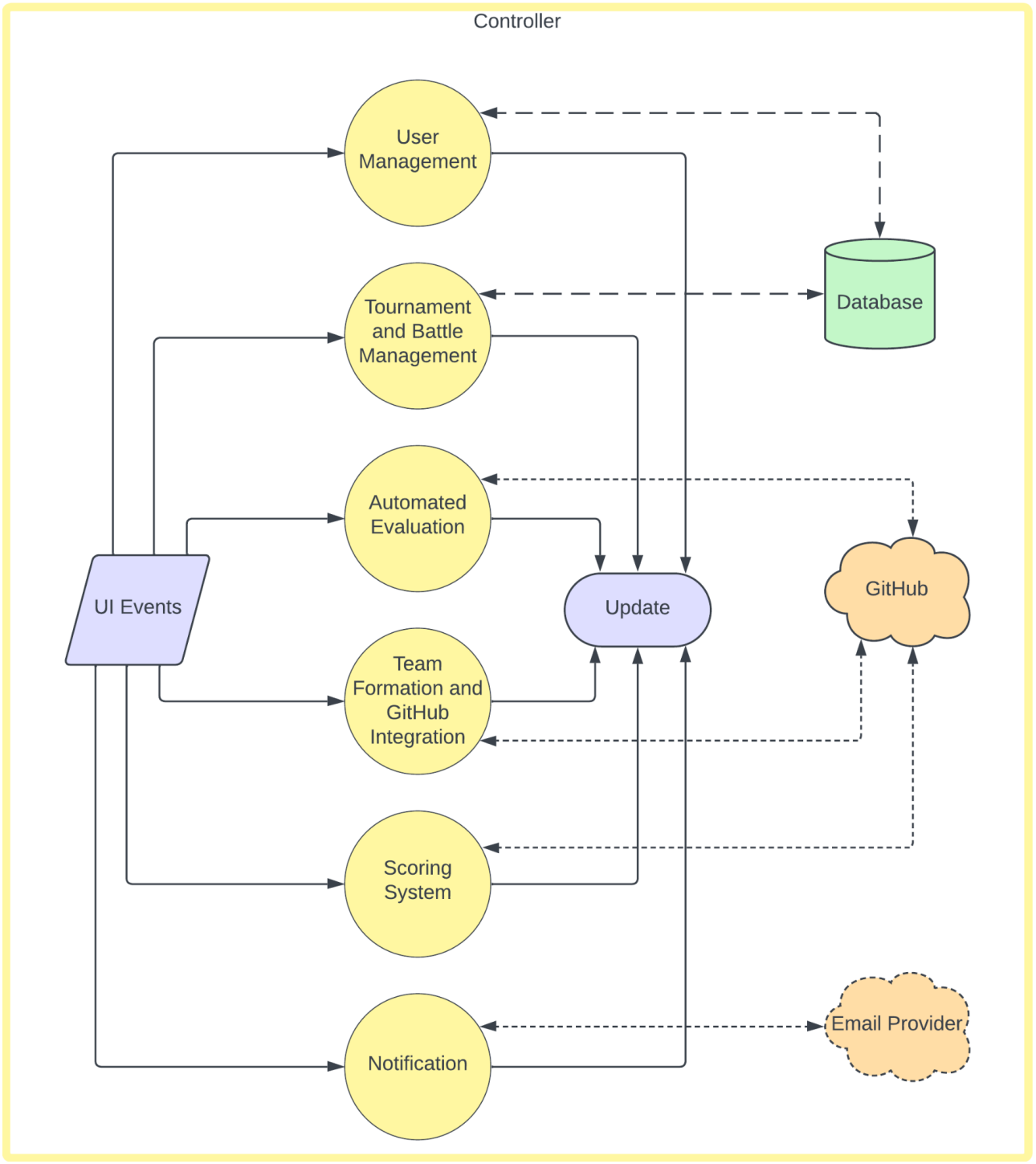
- **Student Actions:** Students interact with the Team Formation and GitHub Integration Component to form teams, join battles, and submit code via GitHub integration.

- **Automated Evaluation Flow:** Code submissions by students trigger the Automated Evaluation Component, which runs static analysis tools and test cases, updating scores based on criteria set by educators.

- **Scoring Updates:** The Scoring System Component receives updates from the Automated Evaluation Component and incorporates educator-assigned manual scores to calculate battle scores for each team.

- **Notifications:** The Notification Component sends notifications to users and educators regarding battle updates, deadlines, and score availability based on events triggered by various system components.

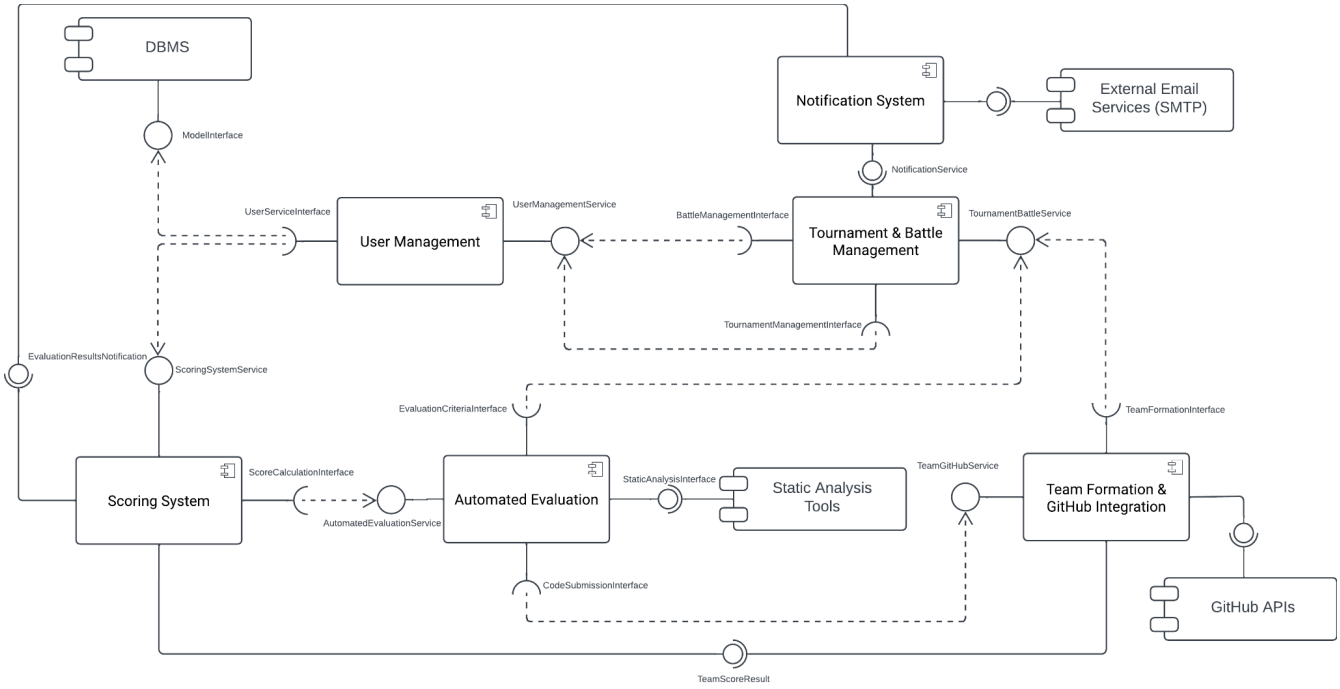
The interactions between these high-level components form the backbone of the CodeKataBattle (CKB) platform, enabling educators to manage competitions and students to participate in code kata battles while automating evaluation and scoring processes. Detailed diagrams and sequence charts will provide a visual representation of these interactions in subsequent sections of the Design Document.



(Figure 2.2. Controller High Level Architecture)

2.2. Component view

2.2.1. Component Diagram



(Figure 2.3. Component Diagram)

Components:

- **User Management:** This component serves as the backbone for user-related functionalities within the CodeKataBattle (CKB) platform. Responsible for user authentication, registration, and authorization, it ensures secure access for all users, managing their identities and permissions within the system.

- **Tournament & Battle Management:** Operating at the core of competitive interactions, this component orchestrates the creation, subscription, and oversight of tournaments and individual battles. It provides educators with tools to set up competitions while enabling students to engage and participate seamlessly.

- **Team Formation & GitHub Integration:** Empowering students to collaborate effectively, this component fosters team creation and battle involvement. It seamlessly integrates with GitHub, allowing for code repository management and providing a collaborative environment for students participating in battles.

- **Automated Evaluation:** Focused on maintaining fairness and quality, this component automates the evaluation of code submissions. It rigorously assesses functional aspects, timeliness, and code quality, ensuring adherence to predefined criteria.

- **Scoring System:** Vital in gauging and ranking performance, this component calculates scores based on a blend of automated evaluation results and manual assessments. It provides an accurate representation of participants' accomplishments in battles.

- **Notification System:** This component acts as the information hub, facilitating timely updates and announcements. It ensures participants are well-informed about tournament/battle progress, deadlines, and the availability of scores.

Interfaces:

- **User Management Interface:** Providing seamless user experiences, this interface manages the complex processes of user authentication, registration, and authorization within the CKB platform, ensuring secure and efficient user interactions.

- **Tournament & Battle Interface:** Offering educators and participants a user-friendly interaction, this interface facilitates the creation, subscription, and management of tournaments and individual battles, streamlining the process for both creators and participants.

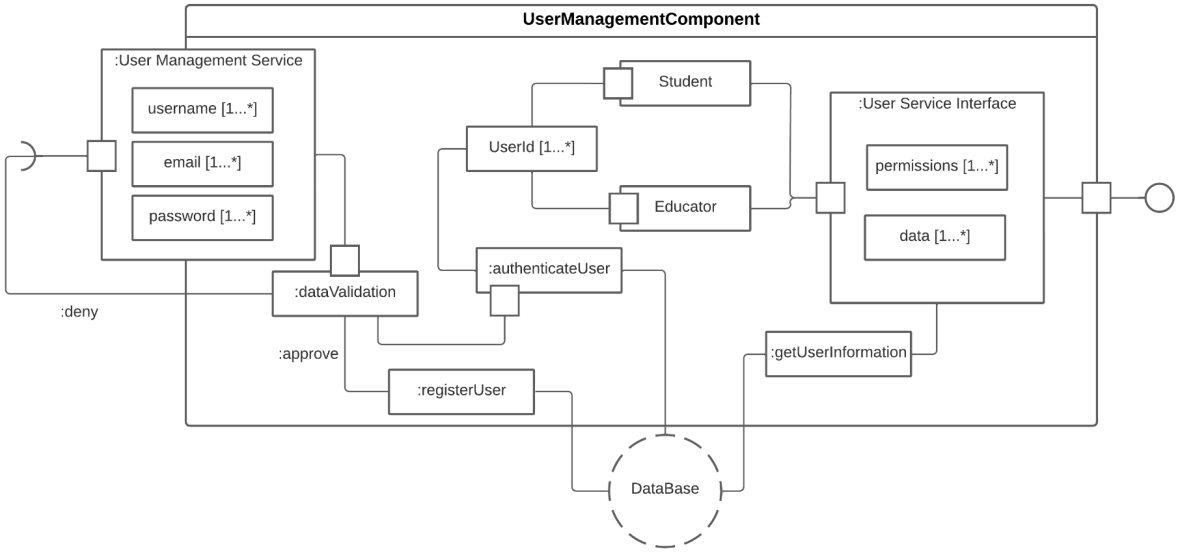
- **Team & GitHub Interface:** This interface acts as the bridge between students, their teams, and GitHub integration. It enables smooth team formation, battle participation, and code submissions while managing GitHub integration for repository access.

- **Automated Evaluation Interface:** Designed for efficiency, this interface conducts automated assessments of submitted code. It evaluates functional aspects, timeliness, and code quality, adhering to predefined evaluation criteria.

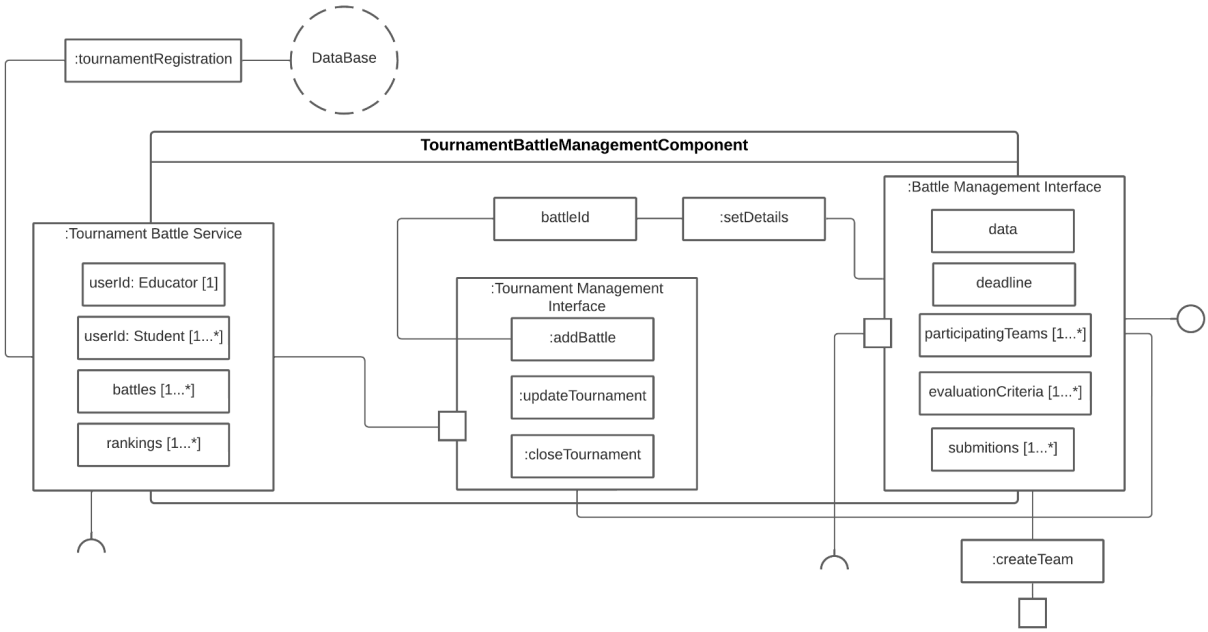
- **Scoring System Interface:** Managing the intricate scoring process, this interface oversees the calculation of scores based on evaluation results. It ensures fairness and accuracy by incorporating both automated and manual assessments.

- **Notification Interface:** Facilitating real-time updates, this interface handles notifications regarding tournament/battle progress, deadlines, and score availability, ensuring all involved parties stay informed and engaged throughout the competition.

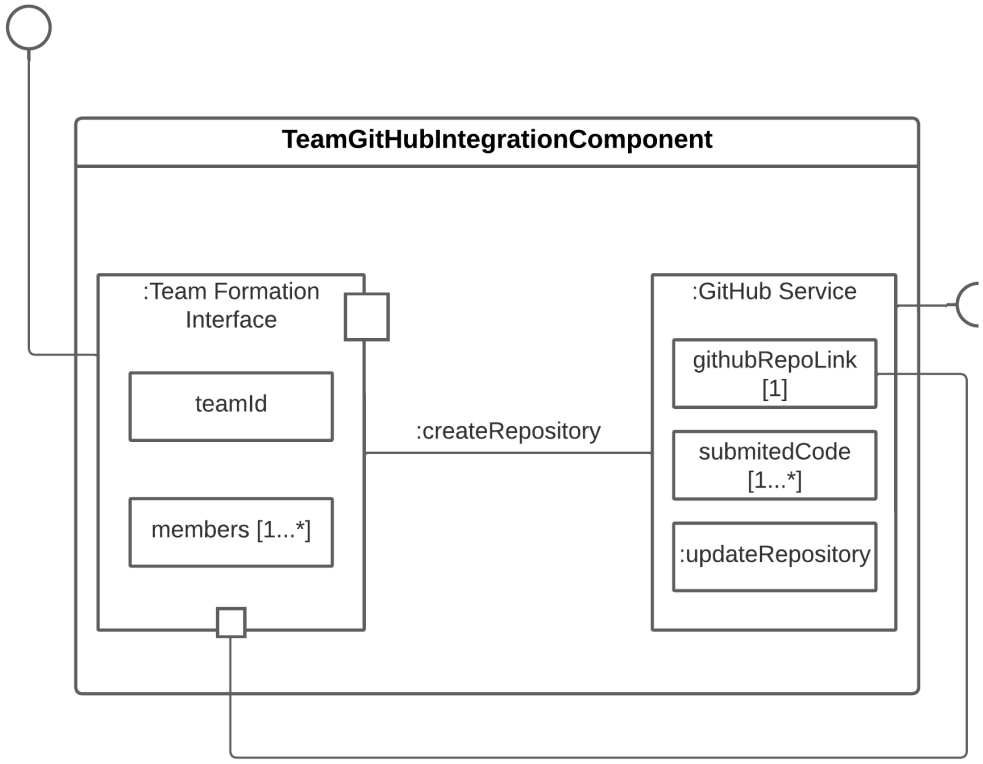
2.2.2. Composite Structure Diagram



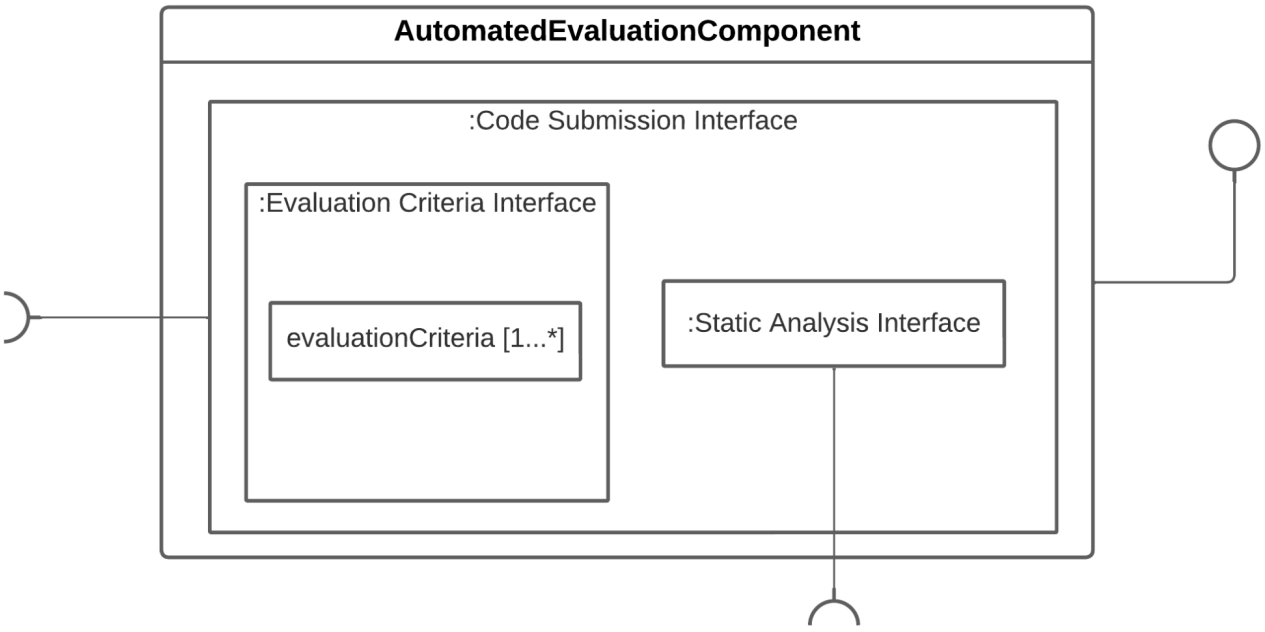
(Figure 2.4.1. Composite Diagram User Management)



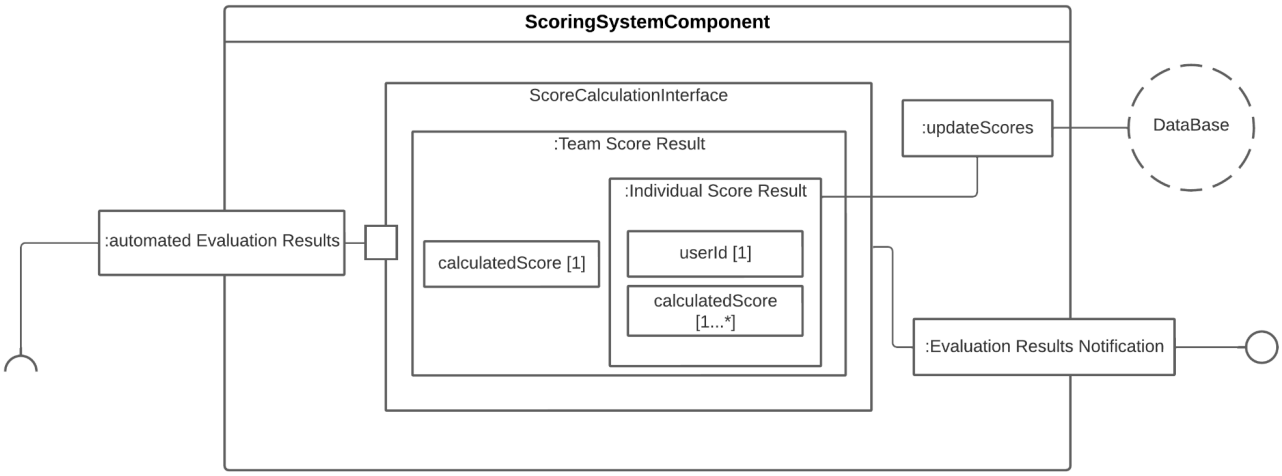
(Figure 2.4.2. Composite Diagram TBMS)



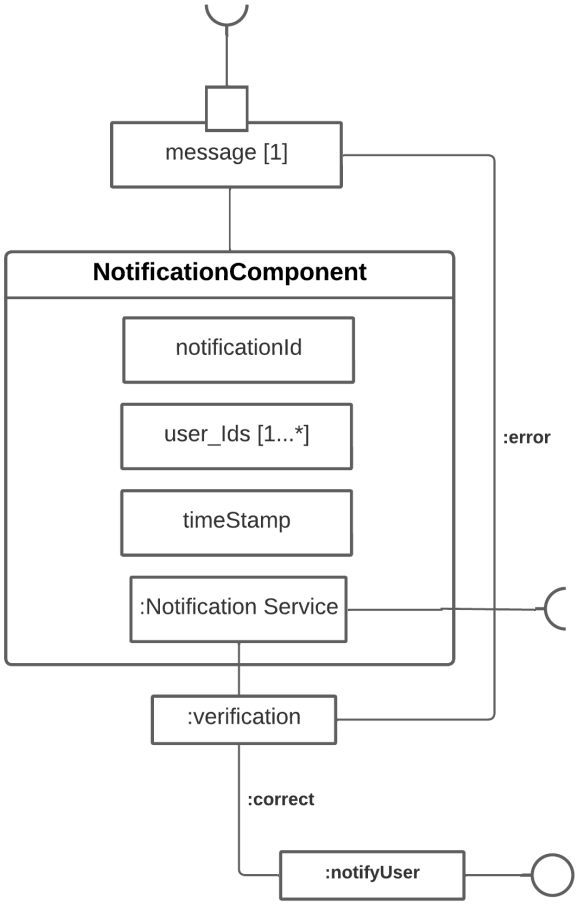
(Figure 2.4.3. Composite Diagram TGIS)



(Figure 2.4.4. Composite Diagram AES)

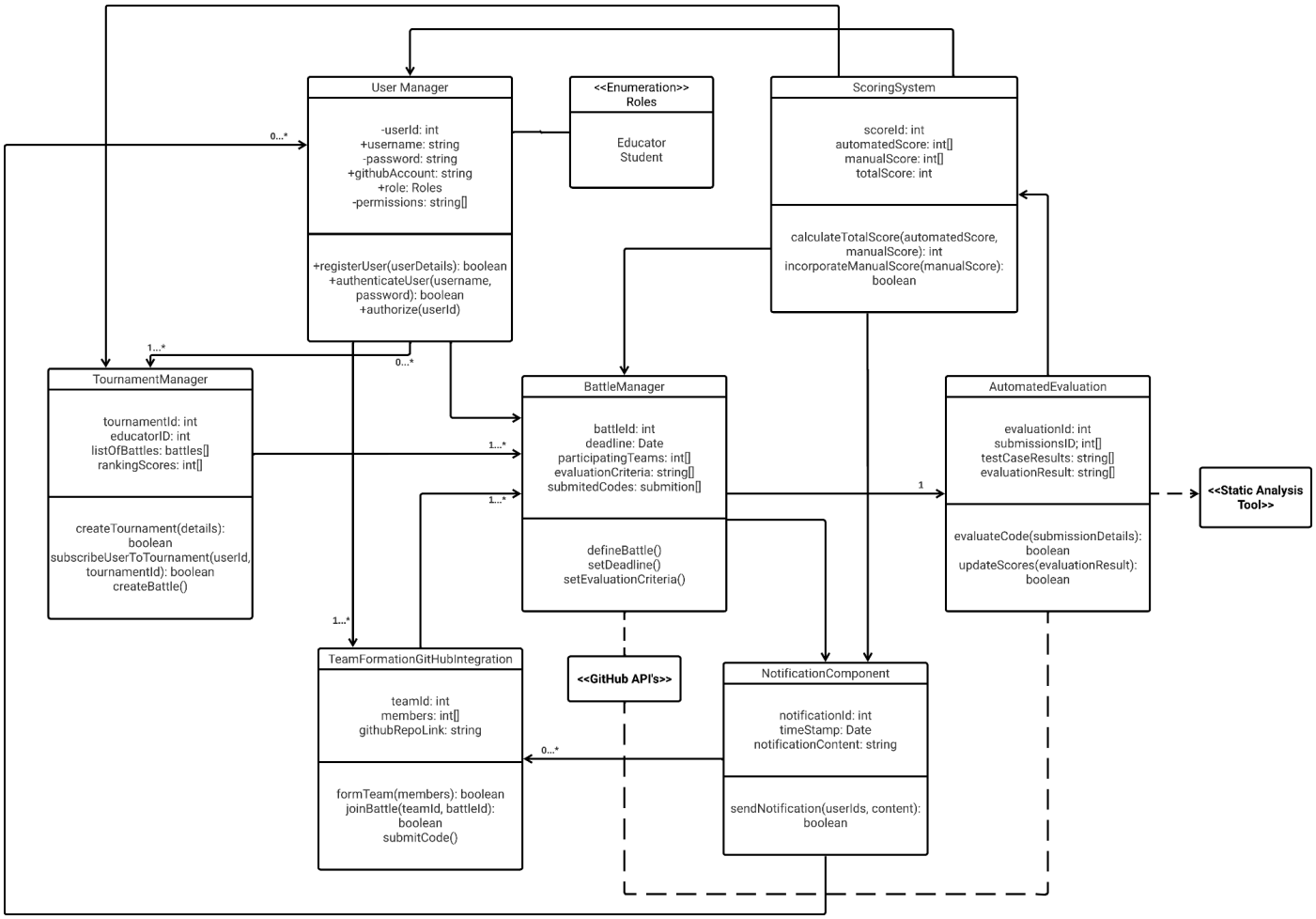


(Figure 2.4.5. Composite Diagram Scoring System)



(Figure 2.4.6. Composite Diagram Notification System)

2.2.3. Class Diagram



(Figure 2.5. Class Diagram)

This class diagram 2.5 for the CKB project will serve as the visual representation for the system and its architecture, decomposing each of the key components and their interactions and functions, with the purpose of providing an easier understanding, and illustrates the organization of the system and its relationships between the various elements. The class diagram its divided into the next 5 key components

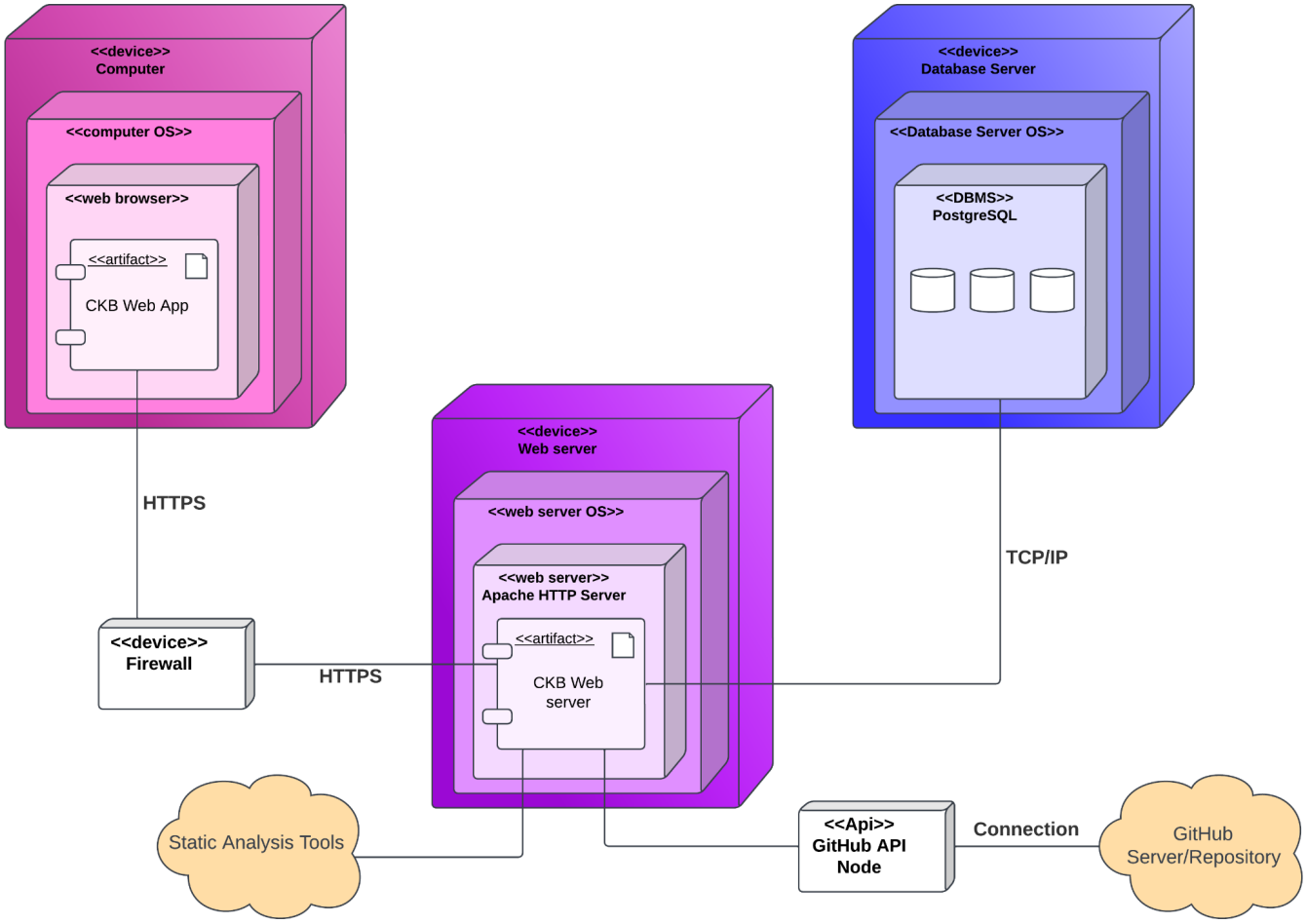
- **User Manager:**
  The User Manager component plays a pivotal role in handling user-related operations. This includes the registration, authentication, and authorization of users within the system. It encapsulates essential attributes and methods that manage user accounts and permissions, ensuring a secure and controlled access environment.
- **Tournament Manager:**
  At the core of the system's competitive structure, the Tournament Manager component oversees the creation, management, and execution of tournaments and battles. It orchestrates the lifecycle of these events, defining their parameters, deadlines, and overall configurations.

Additionally, it handles the computation of rankings, providing an integral function in the competitive aspect of the CKB system.

- **Battle Manager:**
  This component is dedicated to the management of the intricacies and details of each specific battle that is part of a tournament and allow the creation of the teams, its functions include setting the deadlines and the evaluating criterias for the code submissions of the different teams that may participate in the battle. This component is central to the execution of battles, ensuring that they adhere to predefined criteria and limits, facilitating the assessment of the student submissions.
- **Team Formation and Github Integration:**
  This component is responsible for the integration between team formation and GitHub functionalities. It facilitates the creation of the teams that participate in a battle, and the submission of code through the specific GitHub repository of each team. By connecting with GitHub, it streamlines the collaborative coding process, enhancing efficiency in team formation and code management.
- **Scoring System:**
  The ScoringSystem component calculates and manages the scores associated with battles. It integrates with other components, such as the TournamentManager and BattleManager, to ensure accurate and up-to-date scoring information. This is vital for providing feedback to users and determining the outcomes of individual battles and overall tournaments.
- **Automated Evaluation:**
  This component is another essential pivot for our system, providing the tools for an automated assessment of the code submission of each of the teams, following predefined code standards and goals, also following the additional evaluation criterias that may be set by the educator that organizes each specific battle.
- **Notification Component:**
  The Notification Component is crucial for keeping users informed and engaged. This component is responsible for sending notifications to users, ensuring they receive timely updates on various system events. It enhances user experience by providing real-time information about new tournaments, upcoming battles, and changes in rankings.

2.3. Deployment view

In this chapter, the deployment view of the CodeKataBattle (CKB) platform is presented, providing an overview of its execution environment and the hardware components involved in running the system.



(Figure 2.6. Deployment View of the CKB Platform)

This view delineates the infrastructure of the CKB platform and the distribution of hardware components facilitating its execution.

Web App (CKB Web Application):
The CKB Web Application serves as the user-facing interface accessible via web browsers. It facilitates interactions between students, educators, and the platform. Users utilize this interface to browse ongoing tournaments, create and join teams for battles, submit code, view scores, and receive notifications regarding their participation. It runs on the client-side, allowing seamless engagement with the CKB platform's functionalities.

Web Server:
The Web Server hosts and executes the CKB Web Application. This server handles incoming requests from users' web browsers and serves the application content. It ensures the availability and

responsiveness of the platform, managing user sessions, processing HTTP requests, and serving dynamic content. The Web Server is crucial in delivering the CKB application's functionalities to users in a secure and reliable manner.

Database Server:
The Database Server stores and manages the system's data, including user profiles, battle details, team information, scores, and tournament records. It serves as the central repository for persistent data storage, enabling efficient retrieval and management of information required for the functioning of the CKB platform. The database server ensures data integrity, security, and scalability of the system.

Firewall:
The Firewall represents the security measure employed to protect the servers and network infrastructure of the CKB platform. It acts as a barrier between the internal system and external threats, monitoring and controlling incoming and outgoing network traffic. The Firewall helps prevent unauthorized access, mitigates potential cyber threats, and ensures the confidentiality and integrity of the system's data and operations.

Static Analysis Tools:
External Static Analysis Tools are utilized by the CKB platform for code evaluation and quality assessment. These tools analyze the code submissions made by students, assessing various aspects such as security, reliability, and maintainability. They play a vital role in automatically evaluating the code solutions, determining compliance with predefined criteria, and providing feedback to participants and educators.

GitHub API Node:
The GitHub API Node represents the interface between the CKB platform and the GitHub service. It enables interactions between the CKB application and GitHub's repositories, allowing actions such as forking repositories, managing code submissions, and fetching code snippets. This integration facilitates seamless integration with GitHub, enhancing the platform's capabilities for code versioning and collaborative coding practices.

GitHub Server/Repository:
The GitHub Server/Repository symbolizes the external GitHub service where students fork repositories and submit their code for evaluation. It serves as the external storage for code submissions, allowing students to interact with version control features, collaborate on code projects, and integrate their work with the CKB platform for code evaluation and management.

2.4. Component interfaces

2.4.1. UserManagement

UserManagementService

- Signature:
  - create_user(username: str, email: str, password: str) -> User
  - authenticate_user(username: str, password: str) -> User
  - delete_user(user_id: int) -> bool
- Returned Objects: User

UserServiceInterface

- Signature:
  - get_user(user_id: int) -> User
  - update_user(user_id: int, data: dict) -> User
- Returned Objects: User

2.4.2. Tournament & Battle Management

TournamentBattleService

- Signature:
  - create_tournament(data: dict) -> Tournament
  - delete_tournament(tournament_id: int) -> bool
  - create_battle(data: dict) -> Battle
  - delete_battle(battle_id: int) -> bool
  - upload_kata(kata: dict) -> bool
- Returned Objects: Tournament, Battle

TournamentManagementInterface

- Signature:
  - get_tournament(tournament_id: int) -> Tournament
  - update_tournament(tournament_id: int, data: dict) -> Tournament
  - close_tournament(tournament_id: int) -> bool
- Returned Objects: Tournament

BattleManagementInterface

- Signature:
  - get_battle(battle_id: int) -> Battle
  - update_battle(battle_id: int, data: dict) -> Battle
  - withdraw_battle(battle_id: int) -> bool
  - close_battle(battle_id: int) -> bool
- Returned Objects: Battle

NotificationService

- Signature:
  - send_notification(user_ids: List[int], message: str) -> bool
- Returned Objects: bool (indicating successful delivery)

2.4.3. Scoring System

ScoringSystemService

- Signature:
  - get_score(data: dict) -> int
- Returned Objects: int (representing the score retrived)

ScoreCalculationInterface

- Signature:
  - calculate_score(data: dict) -> int
- Returned Objects: int (representing the calculated score)

EvaluationResultsNotification

- Signature:
  - notify_results(data: dict) -> bool
- Returned Objects: bool (indicating successful notification)

TeamScoreResult

- Signature:
  - get_team_score(team_id: int) -> int
  - set_team_score(team_id: int, score: int) -> bool
- Returned Objects: int, bool (indicating successful process)

2.4.4. Automated Evaluation

AutomatedEvaluationService

- Signature:
  - evaluate_submission(data: dict) -> dict
- Returned Objects: dict (containing evaluation results)

EvaluationCriteriaInterface

- Signature:
  - get_criteria(battle_id: int) -> dict

- Returned Objects: dict (containing evaluation criteria)

StaticAnalysisInterface

- Signature:
  - run_static_analysis(data: dict) -> dict
- Returned Objects: dict (containing static analysis results)

CodeSubmissionInterface

- Signature:
  - submit_code(data: dict) -> bool
- Returned Objects: bool (indicating successful submission)

2.4.5. Team Formation & GitHub Integration

TeamGitHubService

- Signature:
  - create_repository(team_id: int, data: dict) -> str
  - update_repository(team_id: int, data: dict) -> str
  - delete_repository(team_id: int) -> bool
- Returned Objects: str (GitHub repository URL), bool (indicating success)

TeamFormationInterface

- Signature:
  - create_team(data: dict) -> Team
  - get_team(team_id: int) -> Team
  - update_team(team_id: int, data: dict) -> Team
  - delete_team(team_id: int) -> bool
- Returned Objects: Team

2.4.6. Notification System

NotificationService

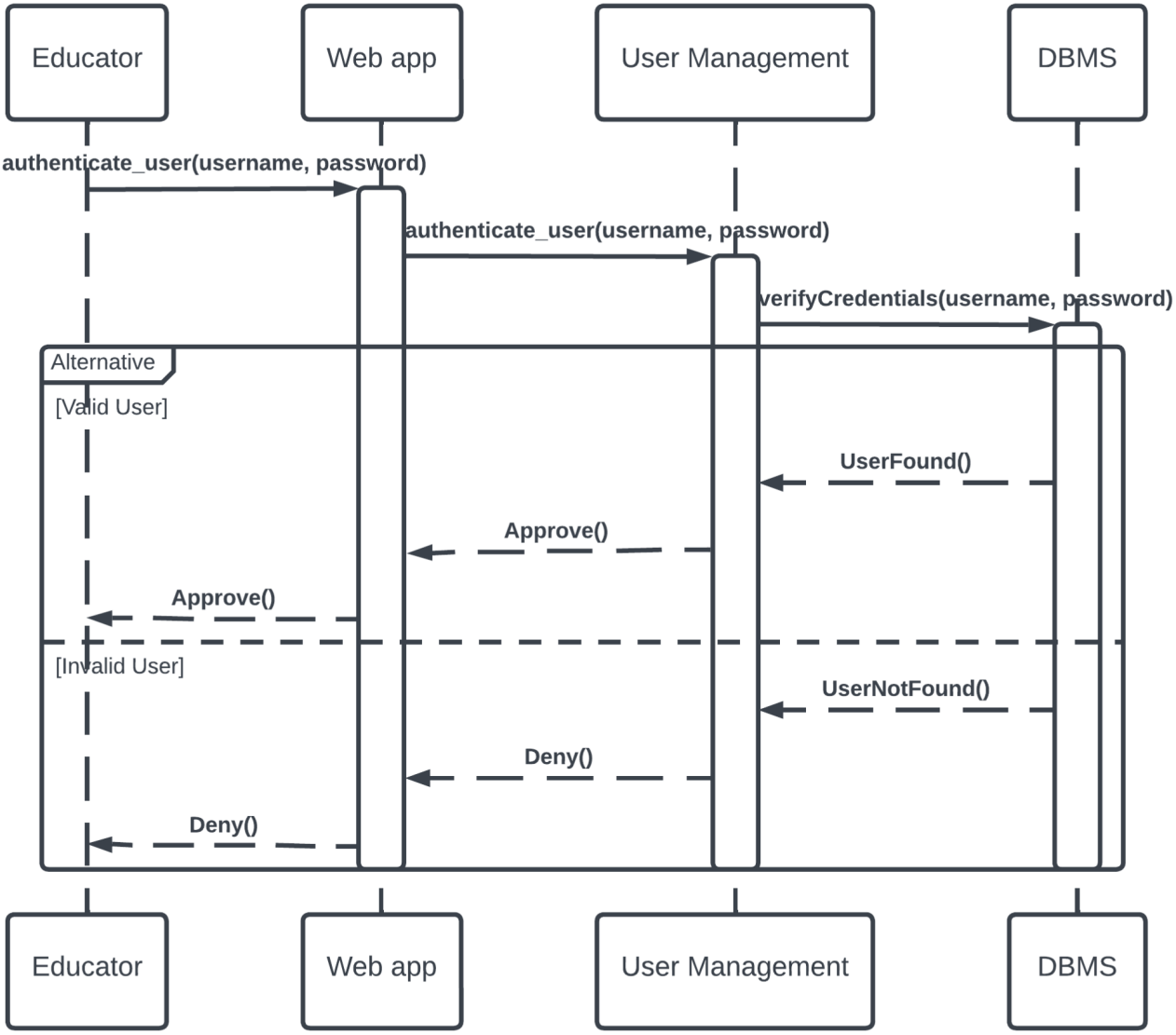
- Signature:
  - send_email(email: str, subject: str, message: str) -> bool
- Returned Objects: bool (indicating successful delivery)

2.5. Runtime view

This chapter offers an in-depth exploration of the runtime views intricately linked to the use cases outlined in the RASD (Requirements Analysis and Specification Document) for the CodeKataBattle (CKB) platform. These views serve as dynamic depictions showcasing the live operational scenarios where multiple components collaborate and interact to enable the diverse functionalities offered by CKB.

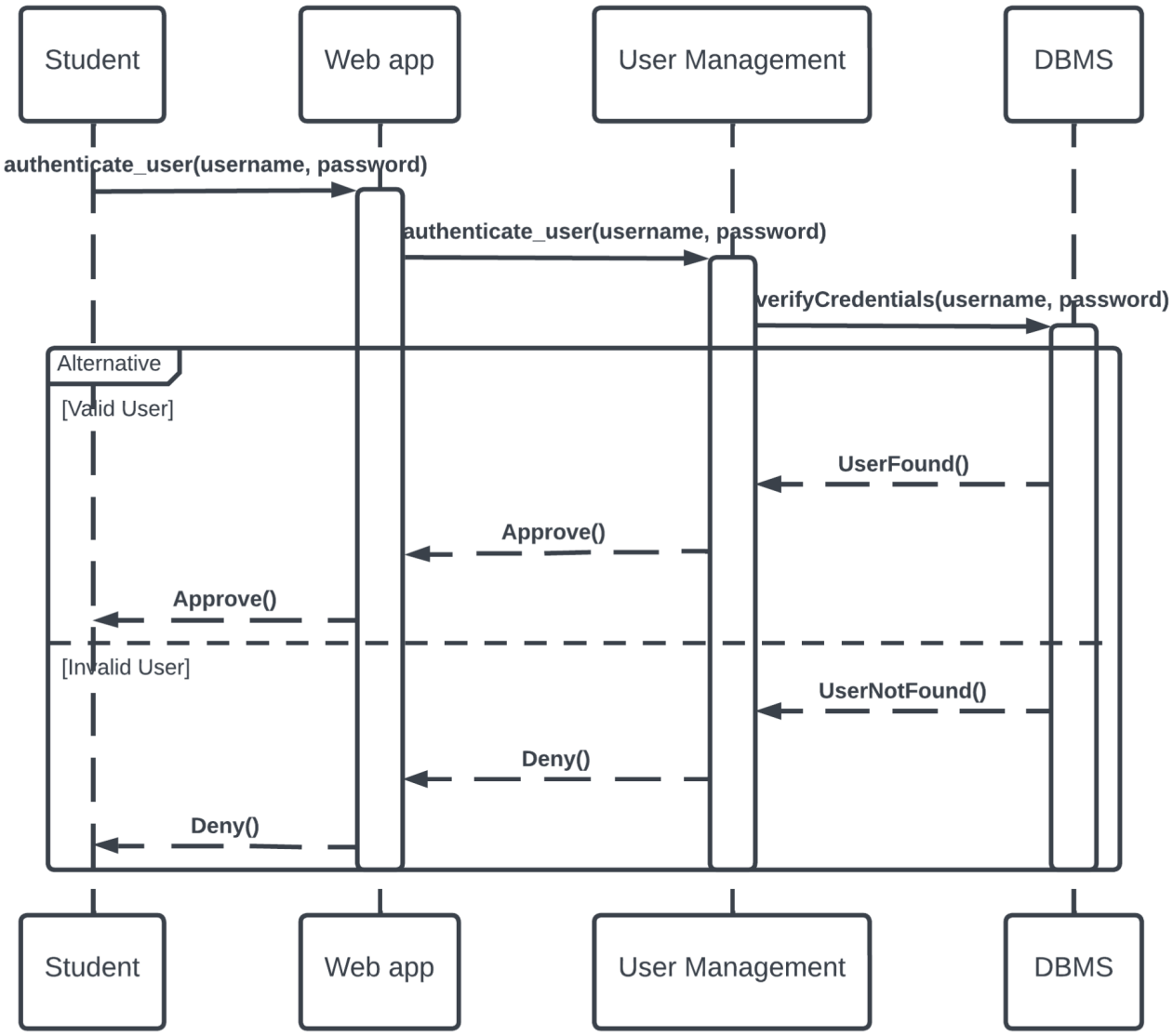
Every operational view within this chapter serves as a detailed snapshot, highlighting the dynamic nature of the interactions. They encapsulate the essence of how users, system modules, and external entities engage to fulfill the specific use cases defined in the RASD.

[UC1] - Educator Login



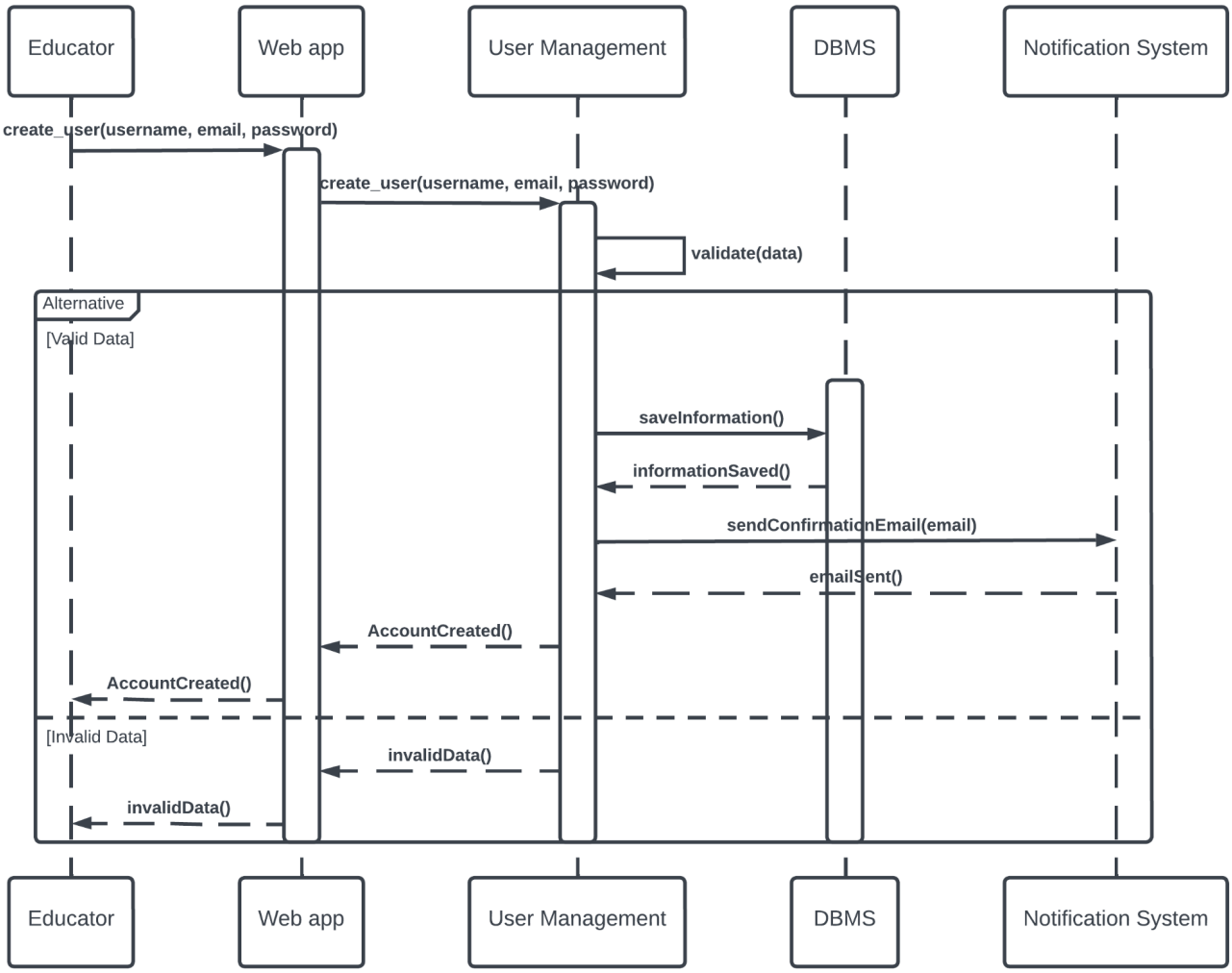
The sequence diagram demonstrates the user authentication of an Educator within the CodeKataBattle (CKB) platform. It commences when the educator initiates a login attempt through the Web App, which sends the educator's login credentials (username/password) to the User Management system (UMS) for verification. The UMS verifies the validity of the user and communicates the result to the DBMS. If the user exists, the UMS proceeds to send the credentials to the DBMS for validity confirmation. Upon successful validation by the DBMS, the user's access is approved, and the Web App authenticates the user, granting access to the platform. In case of invalid credentials, the UMS denies access, prompting the Web App to notify the user of the authentication failure. This triggers the user to re-enter valid credentials or seek assistance. The sequence diagram encompasses both the successful authentication flow and the handling of invalid user scenarios, outlining the step-by-step process for user access within the CKB platform.

[UC2] - Student Login



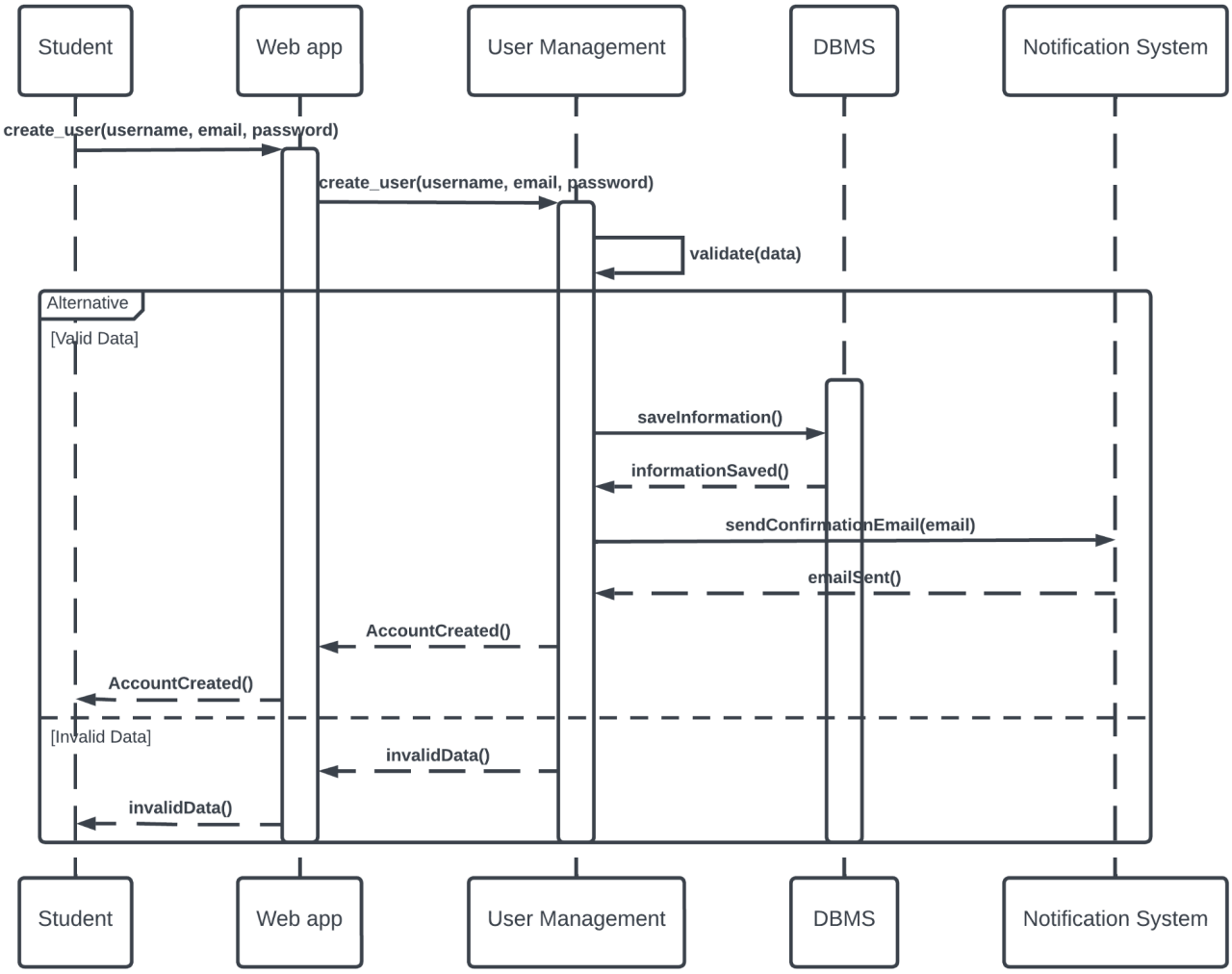
The sequence diagram demonstrates the user authentication of an Student within the CodeKataBattle (CKB) platform. It commences when the Student initiates a login attempt through the Web App, which sends the Student's login credentials (username/password) to the User Management system for verification. The UMS verifies the validity of the user and communicates the result to the DBMS. If the user exists, the UMS proceeds to send the credentials to the DBMS for validity confirmation. Upon successful validation by the DBMS, the user's access is approved, and the Web App authenticates the user, granting access to the platform. In case of invalid credentials, the UMS denies access, prompting the Web App to notify the user of the authentication failure. This triggers the user to re-enter valid credentials or seek assistance. The sequence diagram encompasses both the successful authentication flow and the handling of invalid user scenarios, outlining the step-by-step process for user access within the CKB platform.

[UC3] - Educator Registration



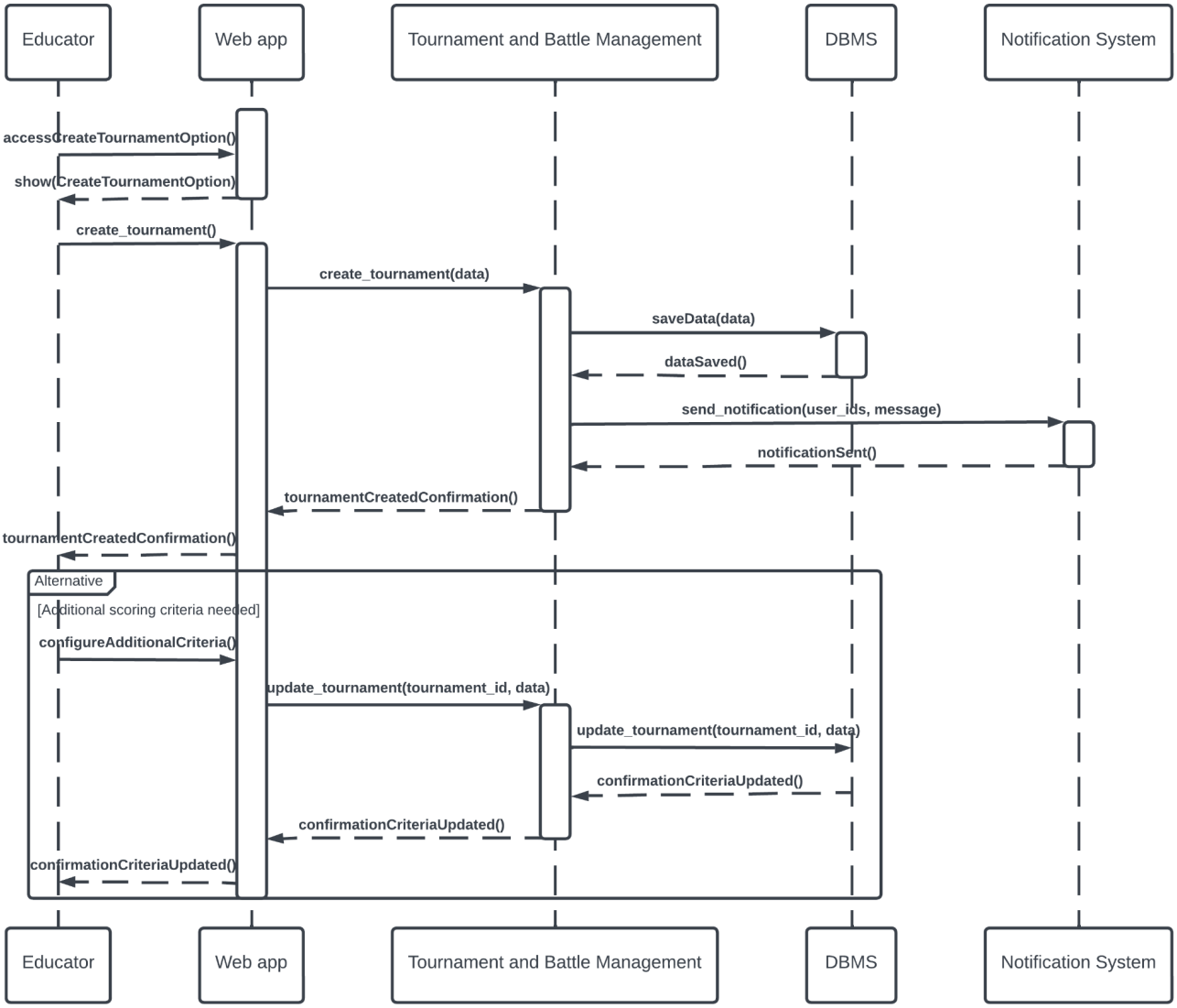
This sequence diagram delineates the user creation process initiated by an Educator within the CKB platform. Commencing with the Educator's input of essential data (username, email, password) through the Web App, this information is relayed to the User Management System (UMS) for validation. The UMS undertakes a validation check on the received data; if the data adheres to the defined criteria and is deemed valid, the UMS proceeds to store this information securely into the Database Management System (DBMS) for record-keeping purposes. Simultaneously, a confirmation email is dispatched to the Notification System, indicating the successful creation of the Educator's account. Additionally, the Web App promptly notifies the Educator of the account's successful creation, ensuring a seamless and informative user experience. Conversely, if the inputted data fails the validation process, the UMS communicates this issue back to the Web App, which promptly notifies the Educator that the provided information is invalid, prompting them to review and re-enter the correct details. This diagram encapsulates the end-to-end process, from data input by the Educator to the handling of valid and invalid data scenarios during account creation within the CKB platform.

[UC4] - Student Registration



This sequence diagram illustrates the process of creating a user account for a Student in the CKB platform. Initiated by the Student, the necessary data (username, email, password) is entered via the Web App, which forwards this information to the User Management System (UMS) for validation. The UMS rigorously validates the provided data, ensuring its compliance with defined criteria. Upon successful validation, signifying the data as accurate and complete, the UMS securely stores the Student's information into the Database Management System (DBMS) for future reference and access. Simultaneously, a confirmation email is generated and dispatched to the Notification System, signifying the successful creation of the Student's account. Concurrently, the Web App promptly notifies the Student of the successful account creation, offering a seamless and informative user experience. Conversely, if the entered data fails the validation process, the UMS communicates this issue back to the Web App, prompting the Student that the information provided is invalid. This immediate feedback loop allows the Student to rectify and re-enter the correct details, ensuring accuracy and completeness. This sequence diagram encapsulates the end-to-end process, from the Student's data input to the handling of valid and invalid data scenarios during account creation within the CKB platform.

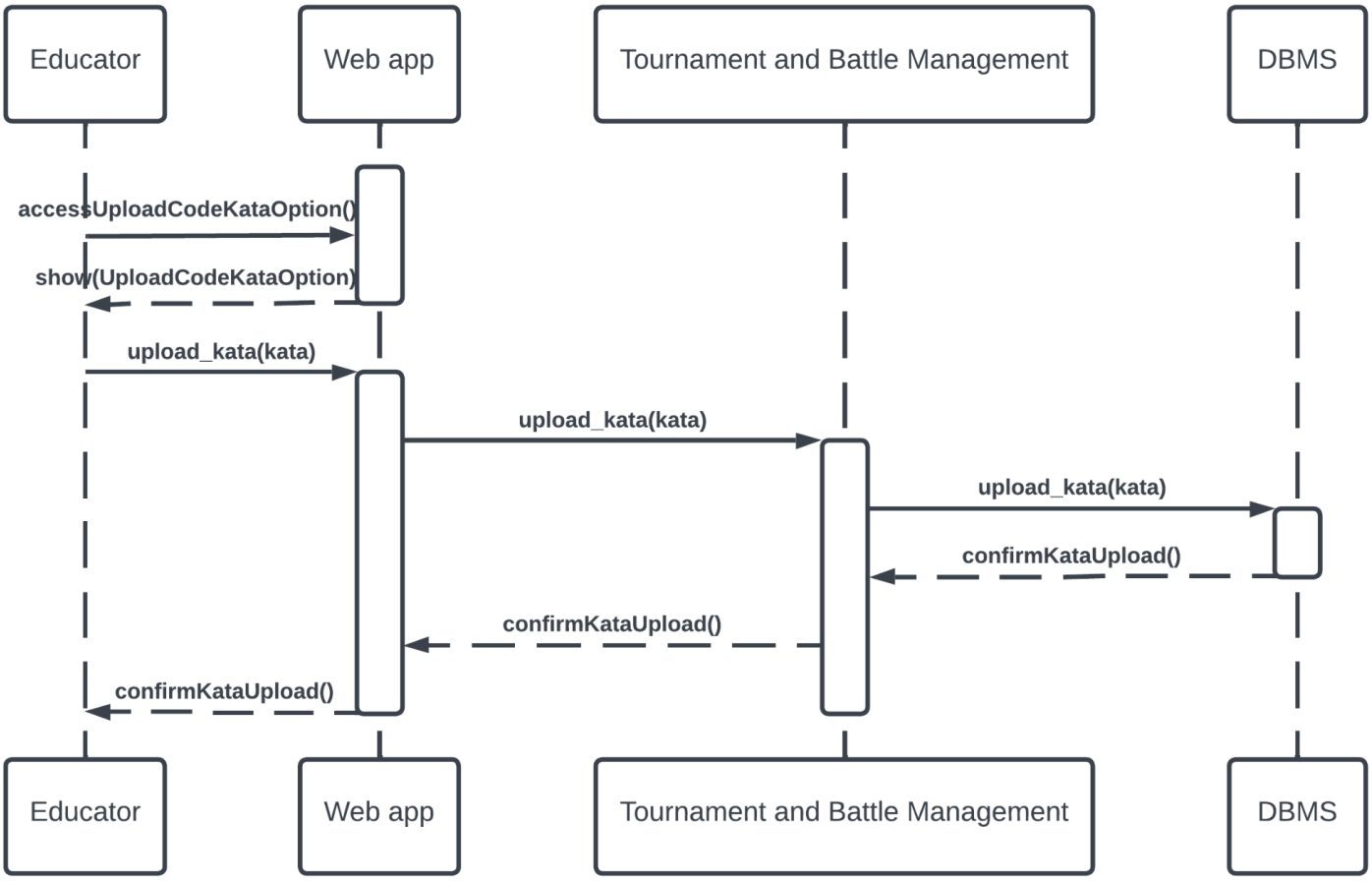
[UC5] - Educator Creates a Tournament



This sequence diagram meticulously delineates the sequence of actions when an Educator initiates the creation of a tournament within the CKB platform. Commencing with the Educator's request via the Web App to begin the creation process, the Web App triggers the tournament creation view. Once the Educator starts the creation process and submits the requested data, the signal is relayed to the Tournament & Battle Management System (TBMS) to commence processing. The TBMS, upon receiving the necessary data, securely saves this information into the Database Management System (DBMS) for systematic storage. Simultaneously, the TBMS activates the Notification System (NS), prompting it to broadcast notifications to all enrolled students, ensuring they are informed about the creation of the new tournament.

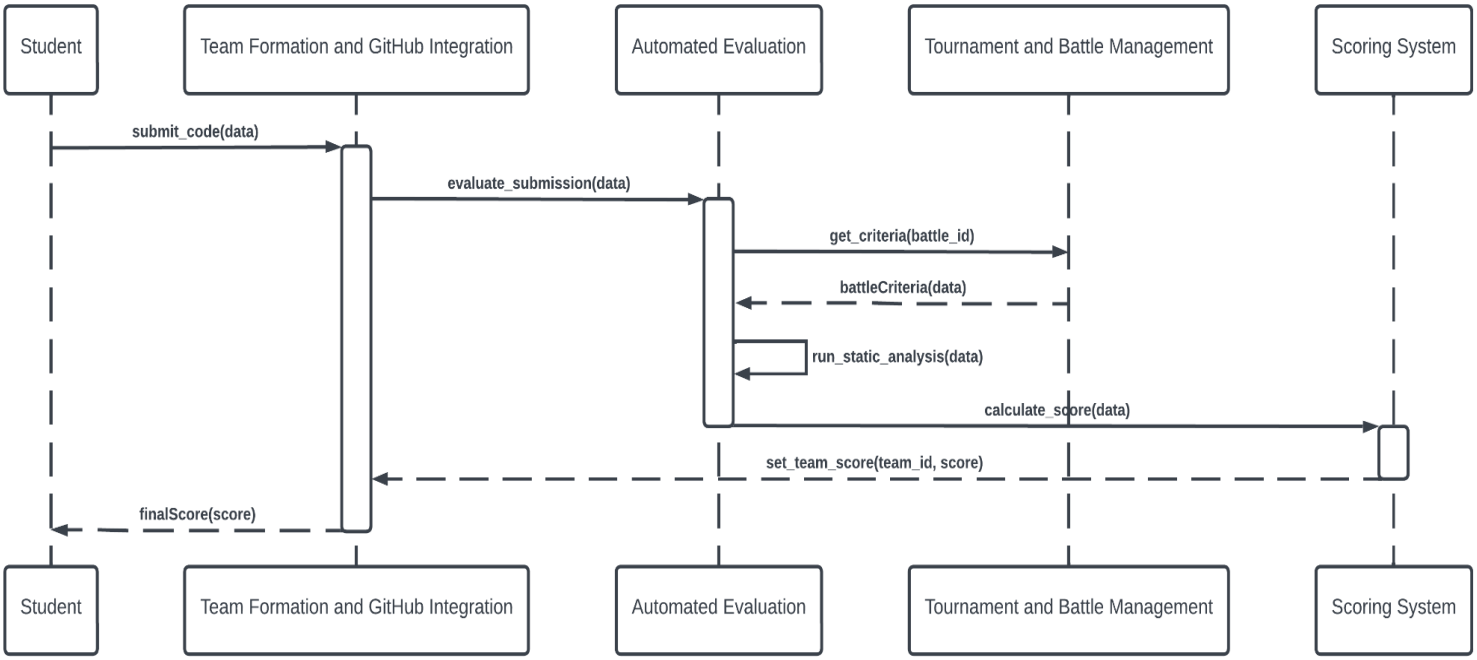
Additionally, in the event where the Educator desires to incorporate additional scoring criteria, they can send these criteria directly to the TBMS. The TBMS then updates this supplementary information in the DBMS, ensuring all relevant tournament data is up-to-date. Subsequently, the Educator is promptly prompted with confirmation of the successful tournament creation or the incorporation of additional scoring criteria.

[UC6] - Educator Uploads Code Kata for Battle



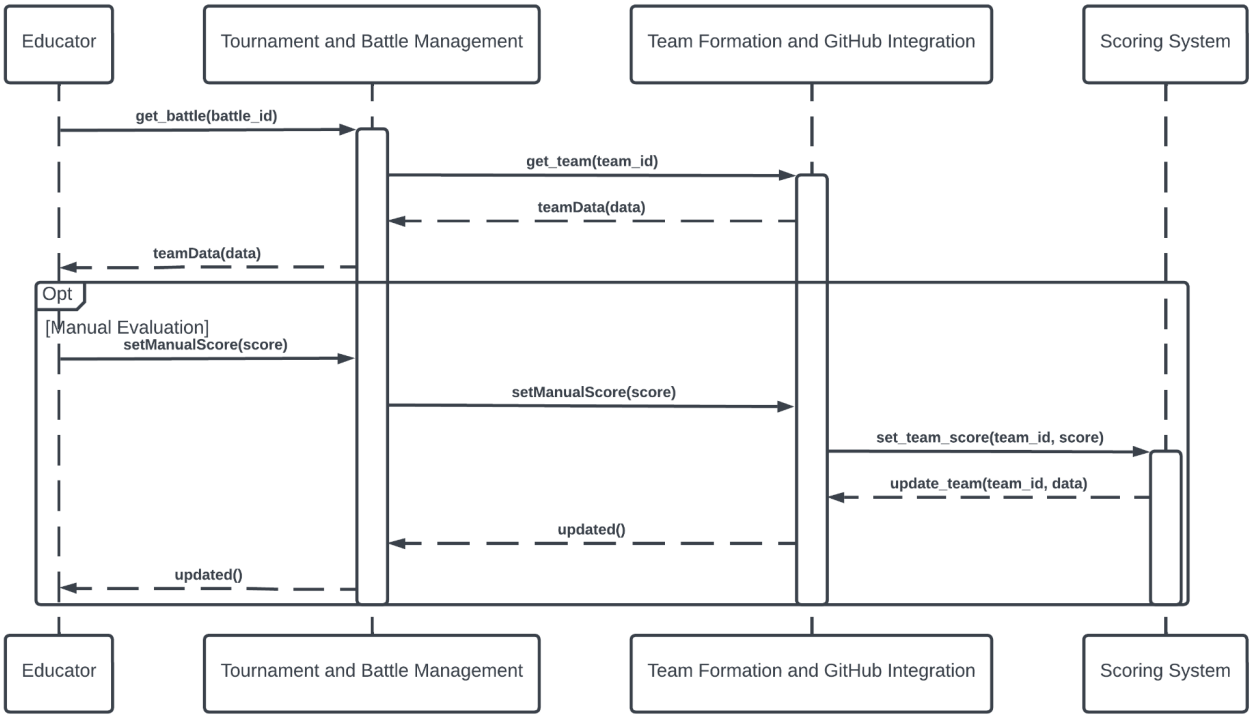
This sequence diagram outlines the process of an educator uploading a Kata to a battle in the CodeKataBattle (CKB) platform. It begins with the educator's request via the Web App for the upload view, triggering the specific interface needed for Kata upload. When the educator submits the Kata through the Web App, the Tournament & Battle Management System (TBMS) processes and uploads this information securely into the Database Management System (DBMS). Subsequently, upon successful upload, a confirmation signal is sent back to the educator through the Web App, indicating the successful completion of the Kata upload process.

[UC7] - Student Submits Code Solution



This sequence diagram illustrates the post-submission process when a student submits code to a battle. After code submission, the Team Formation and GitHub Integration System (TGIS) forwards the uploaded code to the Automated Evaluation System (AES). The AES, upon receiving the code, communicates with the Tournament & Battle Management System (TBMS) to retrieve the battle criteria necessary for evaluation. With the criteria in hand, AES initiates the code evaluation process, first performing evaluations based on the battle criteria and then conducting a static analysis of the code. Once the evaluation completes, AES sends the resulting score to the Scoring System (SS). The SS, upon receiving the score, sets the team score for that upload directly into the Team Formation and GitHub Integration System (TGIS). Subsequently, TGIS notifies the team of the assigned score, concluding the evaluation and scoring process for the submitted code.

[UC8] - Educator Evaluates Student Submissions



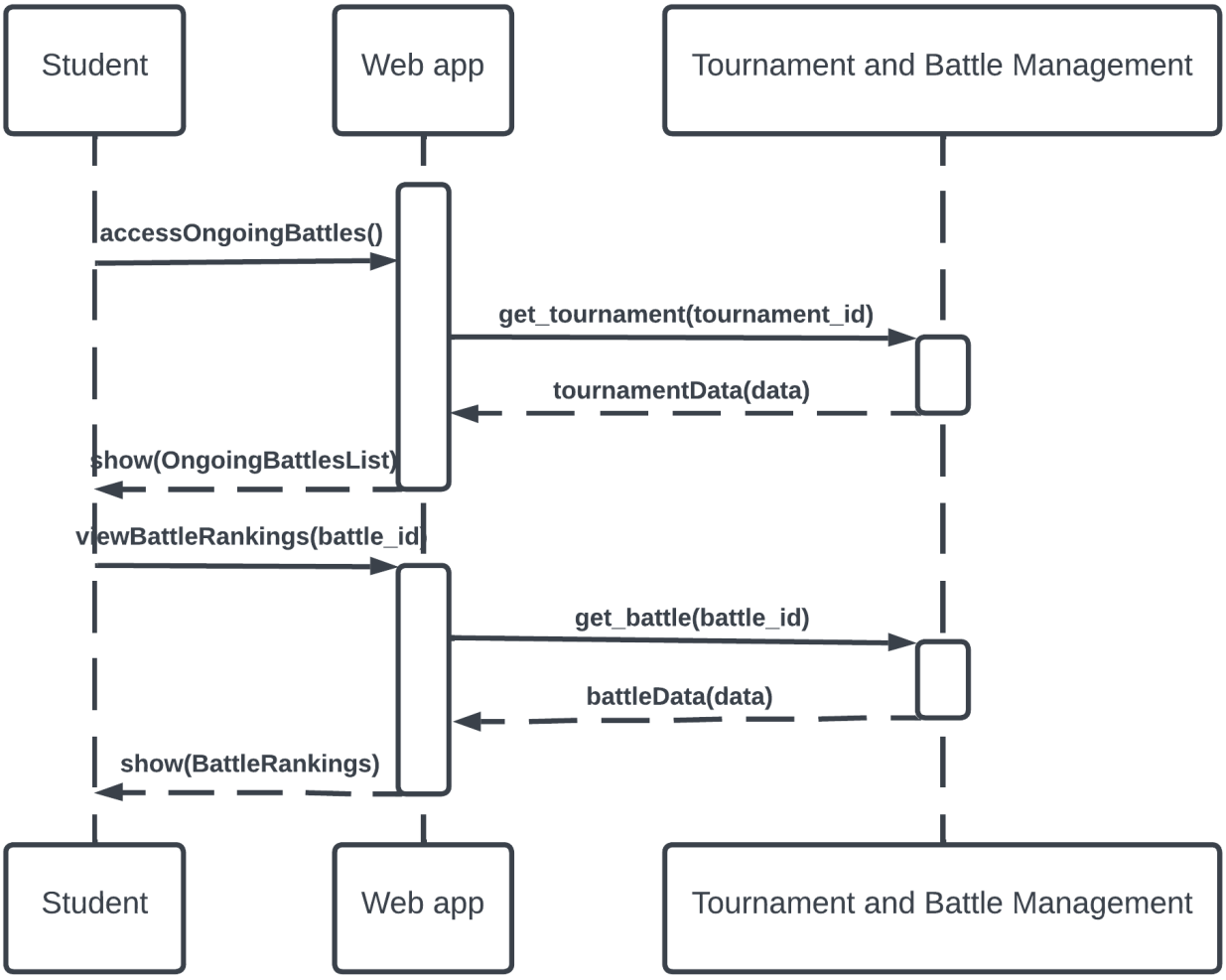
This sequence diagram illustrates the process initiated by an educator performing a manual evaluation for a team of students. The educator first contacts the Tournament & Battle Management System (TBMS) to retrieve the team details associated with that battle. The TBMS, upon receiving this request, contacts the Team Formation and GitHub Integration System (TGIS) to gather the required team data.

Subsequently, the TBMS relays this information back to the educator. If the educator decides to conduct a manual evaluation, they set the score through the TBMS. The TBMS then forwards this score to the TGIS, initiating the process of incorporating the manual score into the evaluation process.

Upon receiving the manual score, the Team Formation and GitHub Integration System (TGIS) forwards it to the Scoring System (SS). The SS calculates a new score by combining the automated score and the manual score. Once the new score is calculated, the SS sends this updated score to be written into the Team Formation and GitHub Integration System (TGIS).

Subsequently, the TGIS notifies the successful update to the Tournament & Battle Management System (TBMS), which in turn communicates this update to the educator. This sequence concludes the process of an educator performing a manual evaluation for a team of students within the CKB platform, ensuring the incorporation of manual scores into the overall assessment for the team's performance.

[UC9] - Student Views Battle Rankings

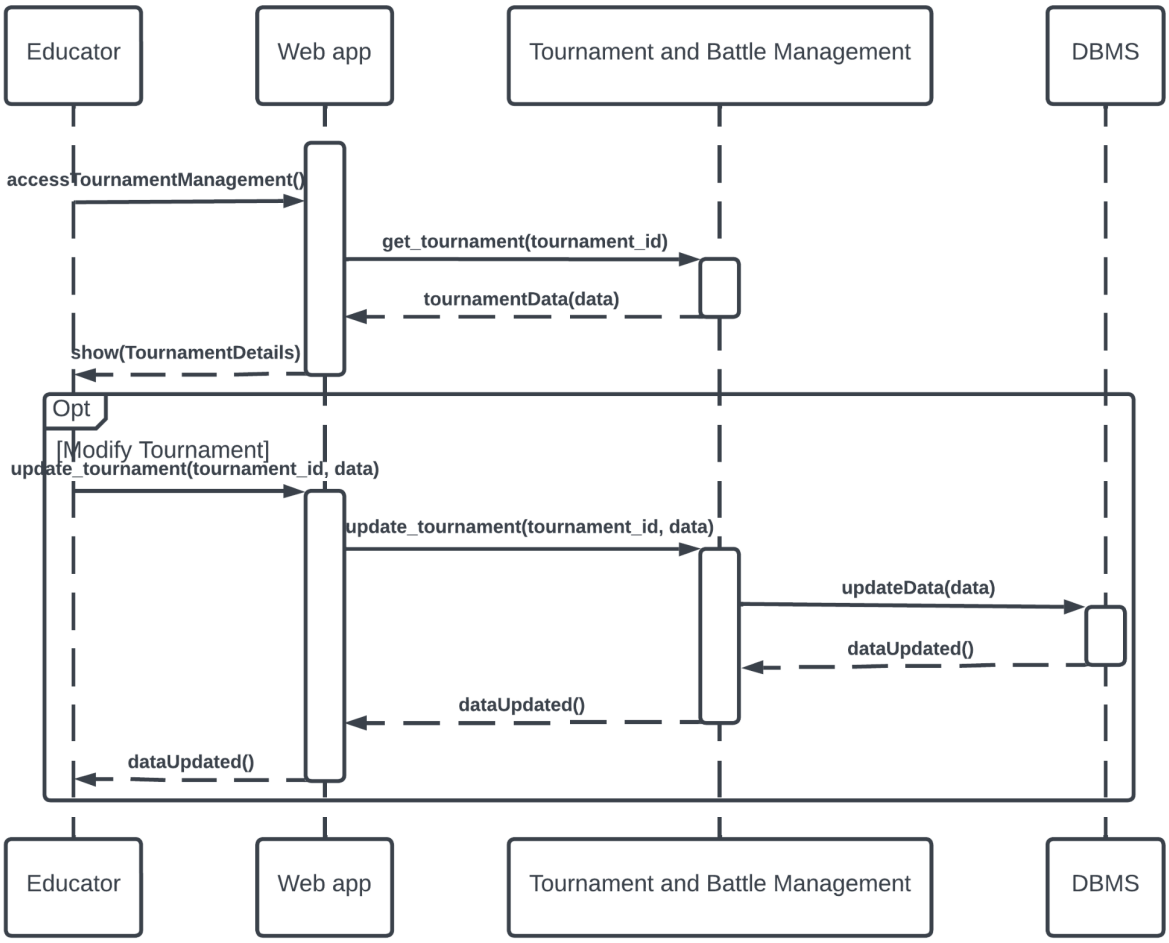


This sequence diagram outlines the steps when a student seeks to view the battle ranking. The student initiates this process through the Web App, communicating with the Tournament & Battle Management System (TBMS) to request the list of ongoing battles. Upon receiving this request, the TBMS provides the list of ongoing battles to the student via the Web App.

Subsequently, the student selects a specific battle of interest and communicates with the TBMS through the Web App, requesting the data related to that particular battle. The TBMS, upon receiving this request, promptly provides the relevant battle data to the student via the Web App.

Consequently, utilizing the Web App interface, the student gains access to view the ranking associated with the selected battle, providing a clear and informative overview of the battle standings. This sequence concludes the process where a student retrieves and views the battle ranking within the CKB platform, facilitating informed participation and engagement in ongoing battles.

[UC10] - Educator Manages Tournament



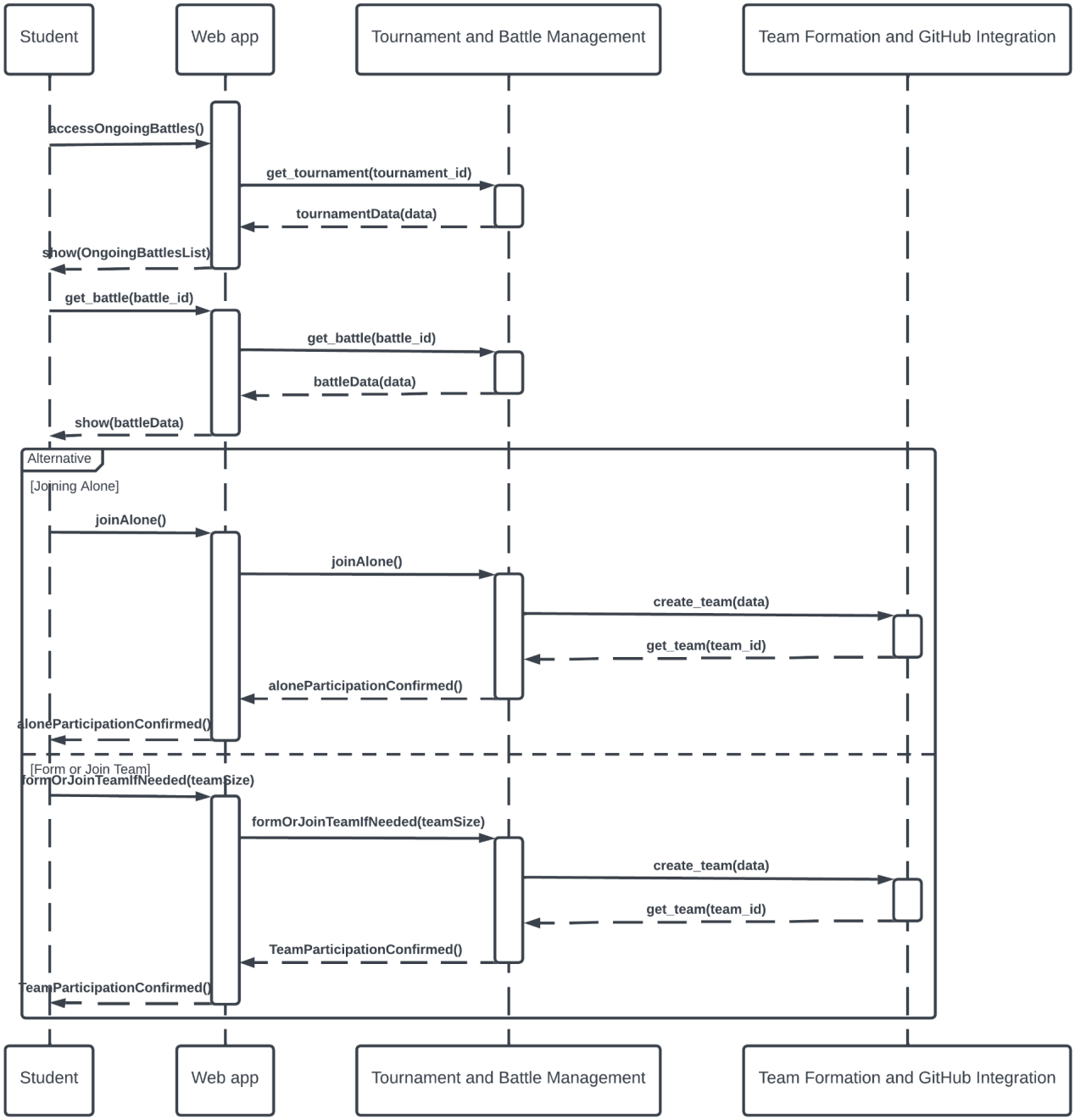
This sequence diagram illustrates the steps an educator takes to manage a tournament. Initially, the educator uses the Web App to request tournament details from the Tournament & Battle Management System (TBMS). Upon receiving this request, the TBMS promptly responds by providing the respective tournament information to the educator via the Web App.

If the educator decides to update any tournament details, they communicate these changes through the Web App, which relays the information to the TBMS. Upon receiving the update request, the TBMS processes this information and ensures the necessary modifications are made within the Database Management System (DBMS) to reflect the changes to the tournament details.

Subsequently, upon successful updating of the tournament details in the DBMS, the TBMS sends a confirmation signal back to the educator through the Web App. This confirmation notifies the educator of the successful completion of the update process for the tournament details.

This sequence concludes the process where an educator manages tournament details within the CKB platform, allowing for efficient updates and modifications to tournament information through clear communication facilitated by the Web App and TBMS interaction.

[UC11] - Student Joins a Battle



This sequence diagram outlines the steps for a student to join a battle. Initially, the student communicates with the Tournament & Battle Management System (TBMS) to request the list of ongoing battles. Upon receiving this request, the TBMS promptly provides the student with the list of ongoing battles.

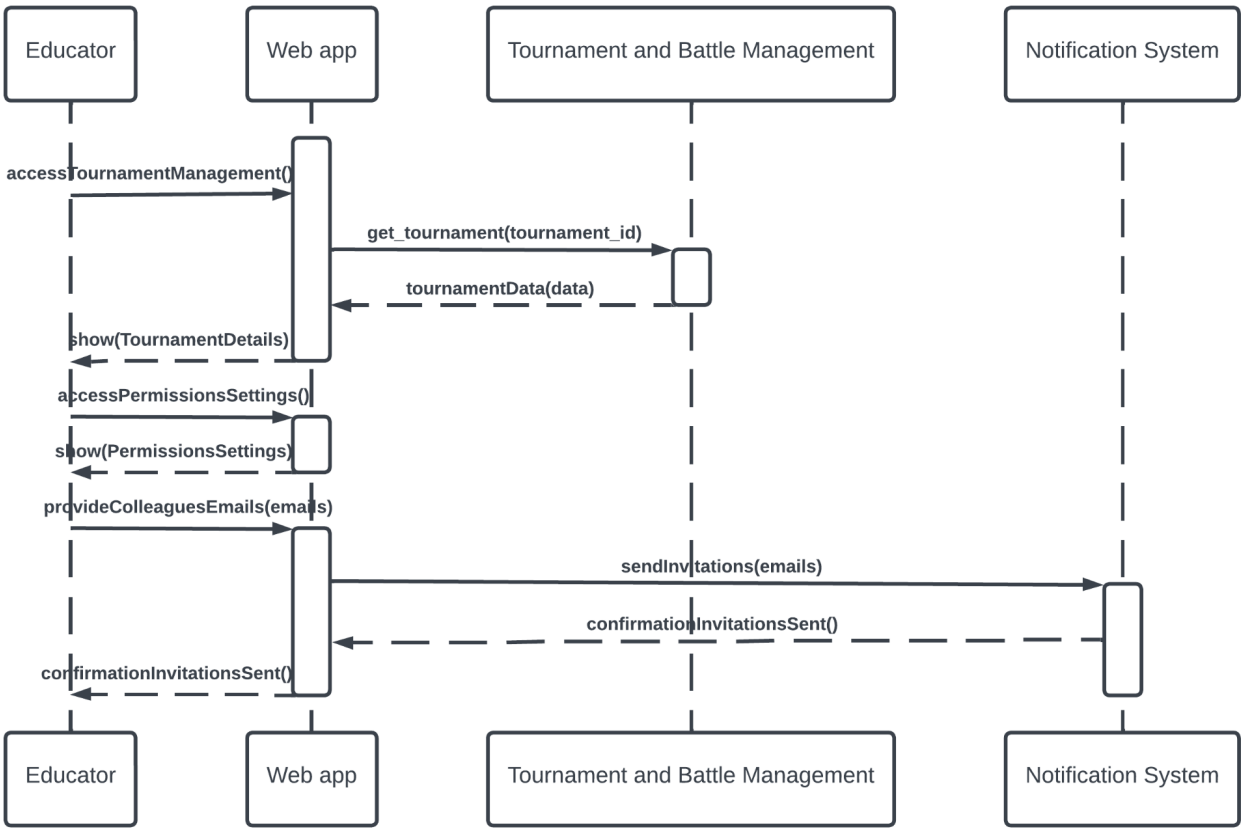
Subsequently, the student selects a specific battle of interest and requests the battle's data from the TBMS. Upon receiving this request, the TBMS sends the relevant battle information to the student.

If the student intends to join the battle individually, they convey this message to the TBMS. Upon receiving this signal, the TBMS communicates with the Team Formation and GitHub Integration System (TGIS) to create a team for the student. The TGIS confirms the successful creation of the team to the student.

Alternatively, if the student wishes to join an existing team or form a new team, they send a signal to the TBMS indicating the team size and preference. Upon receiving this message, the TBMS communicates with the TGIS to create the specified team. The TGIS generates the team as requested, confirming the successful team formation to the student.

This sequence illustrates the process by which a student joins a battle within the CKB platform, allowing for individual participation or team formation through seamless interaction facilitated by the TBMS and TGIS components.

[UC12] - Educator Grants Permissions



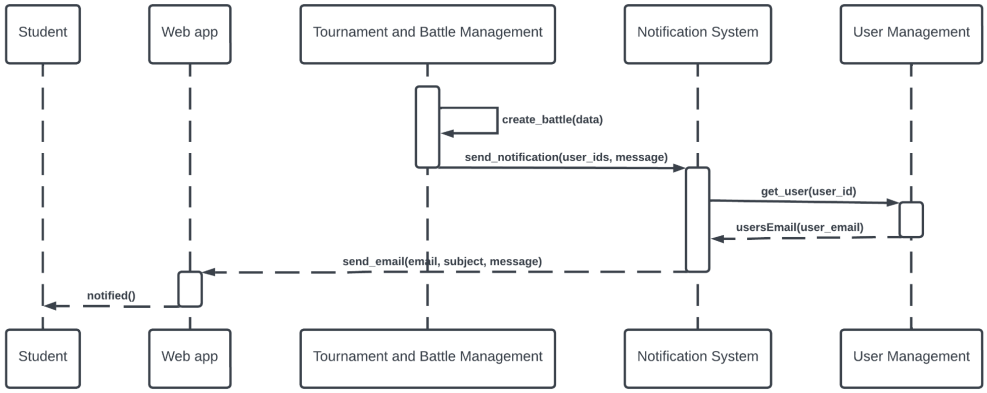
This sequence diagram outlines the steps when an educator invites colleagues to a tournament. Initially, the educator requests tournament details from the Tournament & Battle Management System (TBMS). Upon receiving this request, the TBMS promptly provides the educator with the relevant tournament details.

Subsequently, the educator seeks permission settings for the tournament via the Web App. The Web App communicates this request to the TBMS, which in turn provides the permission settings information to the educator.

Upon obtaining the necessary details and settings, the educator provides the emails of the colleagues via the Web App. The Web App interacts with the Notification System (NS) to send invitation emails to the specified colleagues. The NS promptly sends out the invitation emails to the colleagues for the tournament invitation.

Finally, upon successful distribution of the invitations, the NS sends a confirmation signal back to the educator via the Web App. This confirmation assures the educator of the successful completion of the invitation process, facilitating the efficient and comprehensive invitation of colleagues to the tournament within the CKB platform.

[UC13] - Student Receives Battle Notification

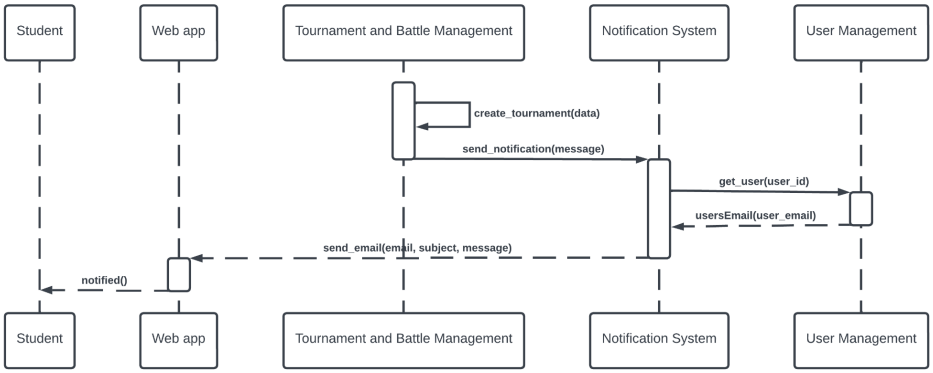


This sequence diagram illustrates the process of notifying a student about a battle. Initially, the Tournament & Battle Management System (TBMS) creates a battle. Upon successful creation, the TBMS notifies the Notification System (NS) about the newly created battle.

Upon receiving the notification, the NS requests the necessary user data from the User Management System (UMS). The UMS promptly provides the required user data to the NS.

Subsequently, armed with the user data, the NS sends out notification emails to the students, informing them about the newly created battle. This process ensures that students are promptly and effectively notified about the existence of the battle, enabling their participation within the CKB platform.

[UC14] - Student Receives Tournament Notification

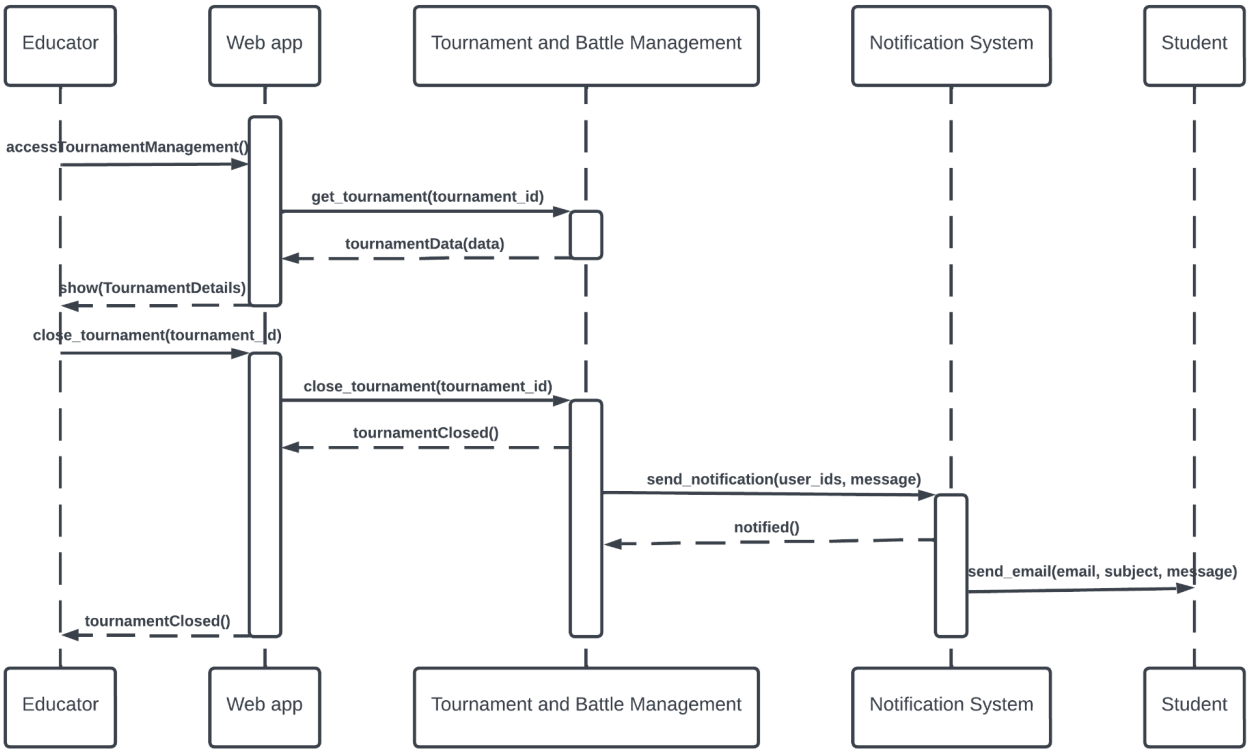


This sequence diagram delineates the process of notifying a student about a tournament. Initially, the Tournament & Battle Management System (TBMS) creates a tournament. Upon successful creation, the TBMS notifies the Notification System (NS) about the newly created tournament.

Upon receiving the notification, the NS requests the necessary user data from the User Management System (UMS). The UMS promptly provides the required user data to the NS.

Subsequently, armed with the user data, the NS sends out notification emails to the students, informing them about the newly created tournament. This process ensures that students are promptly and effectively notified about the existence of the tournament, facilitating their participation within the CKB platform.

[UC15] - Educator Closes Tournament



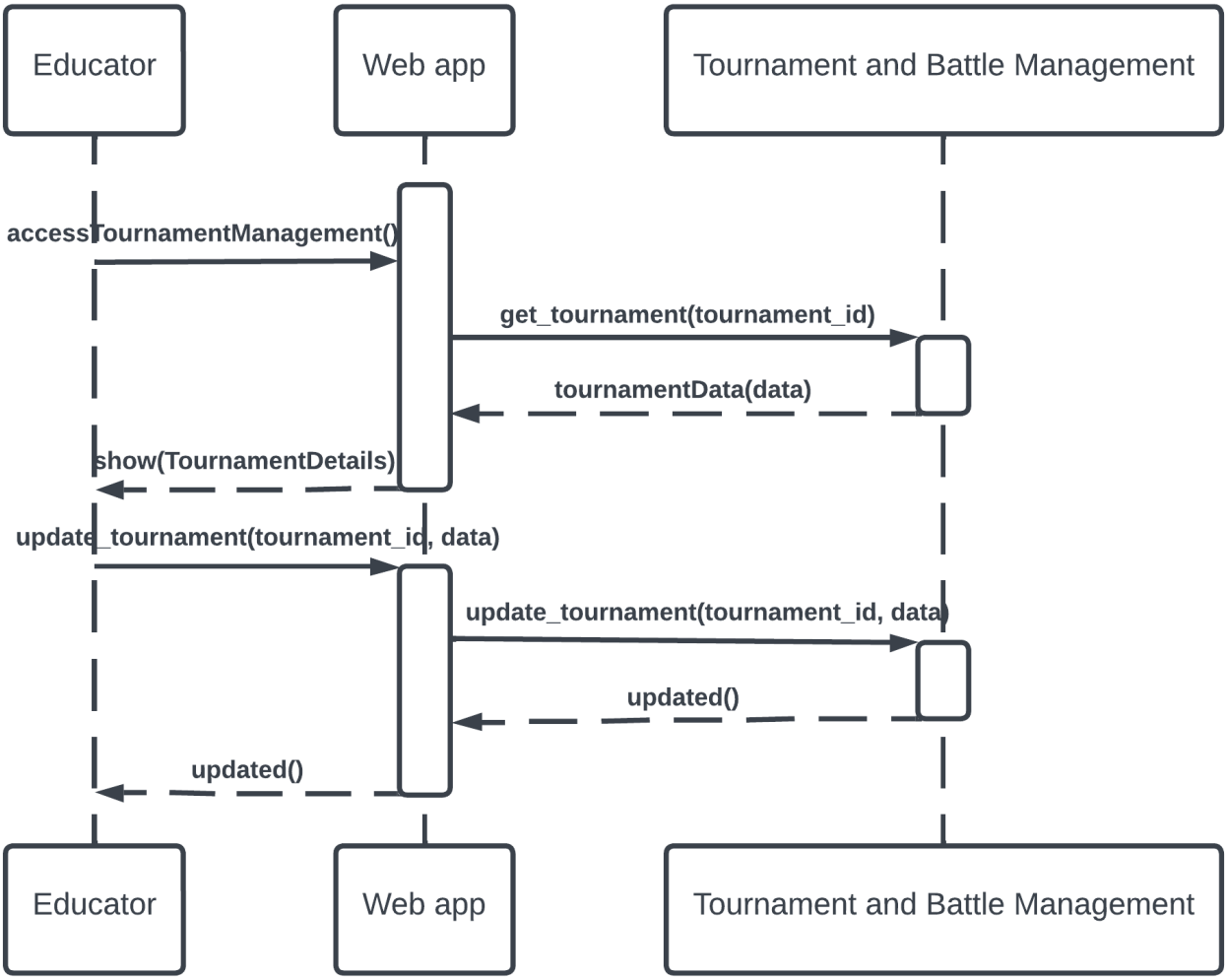
This sequence diagram outlines the steps for an educator to close a tournament. Initially, the educator requests tournament information from the Tournament & Battle Management System (TBMS). Upon receiving this information, the educator initiates a request through the Web App to close the tournament.

The request to close the tournament is relayed successively through the Web App and then the TBMS. Upon receiving this request, the TBMS proceeds with the closing process.

Subsequently, the TBMS notifies the Notification System (NS) to inform the students about the closure of the tournament. The NS acknowledges this request and sends confirmation back to the TBMS.

Simultaneously, the TBMS confirms the closure of the tournament to the educator via the Web App. This sequence ensures a systematic and comprehensive process for the educator to close a tournament, informing the students and ensuring clarity about the tournament's status within the CKB platform.

[UC16] - Educator Defines Scoring Criteria

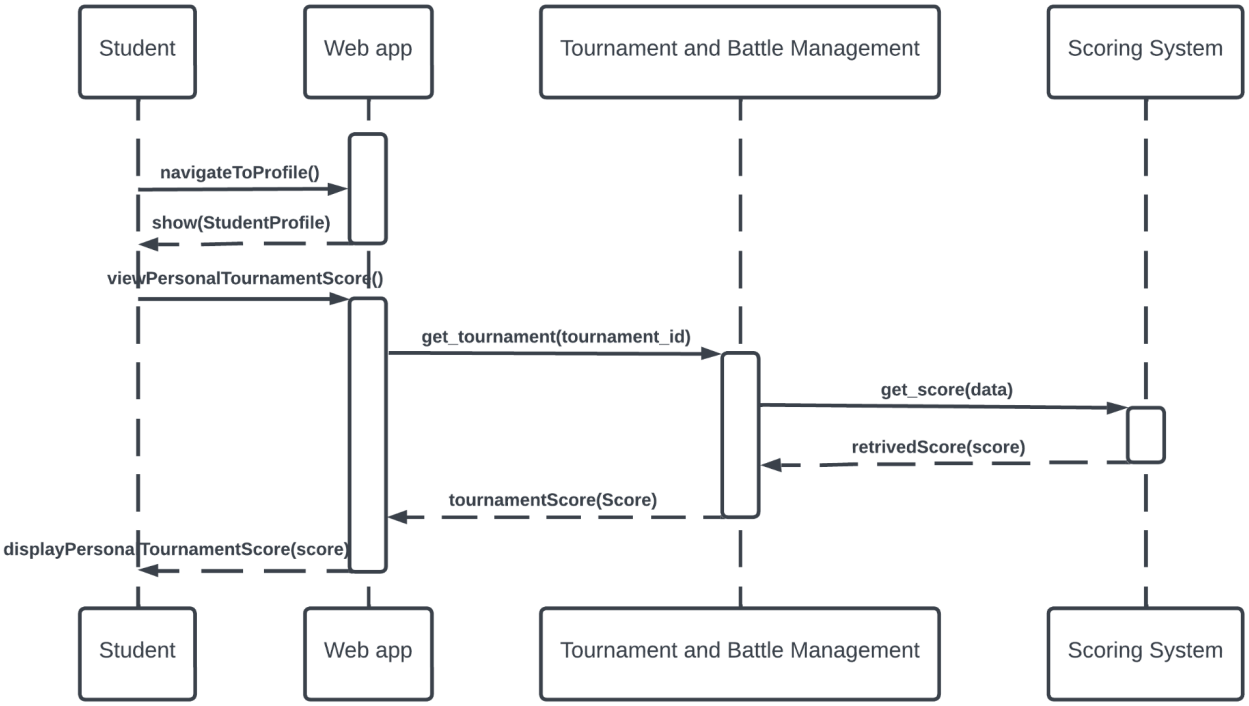


This sequence diagram illustrates the steps an educator takes to define scoring criteria. Initially, the educator requests tournament details from the Tournament & Battle Management System (TBMS) through the Web App. Upon receiving this information, the educator updates the scoring criteria data and sends the modified information via the Web App back to the TBMS.

Upon receiving the updated information, the TBMS processes these modifications, ensuring the scoring criteria are appropriately updated. Subsequently, the TBMS confirms the successful update of the scoring criteria and sends a confirmation signal back to the educator through the Web App.

This sequence ensures an efficient and systematic process for educators to define and update scoring criteria within tournaments, facilitating clearer guidelines for assessments within the CKB platform.

[UC17] - Student Views Personal Tournament Score



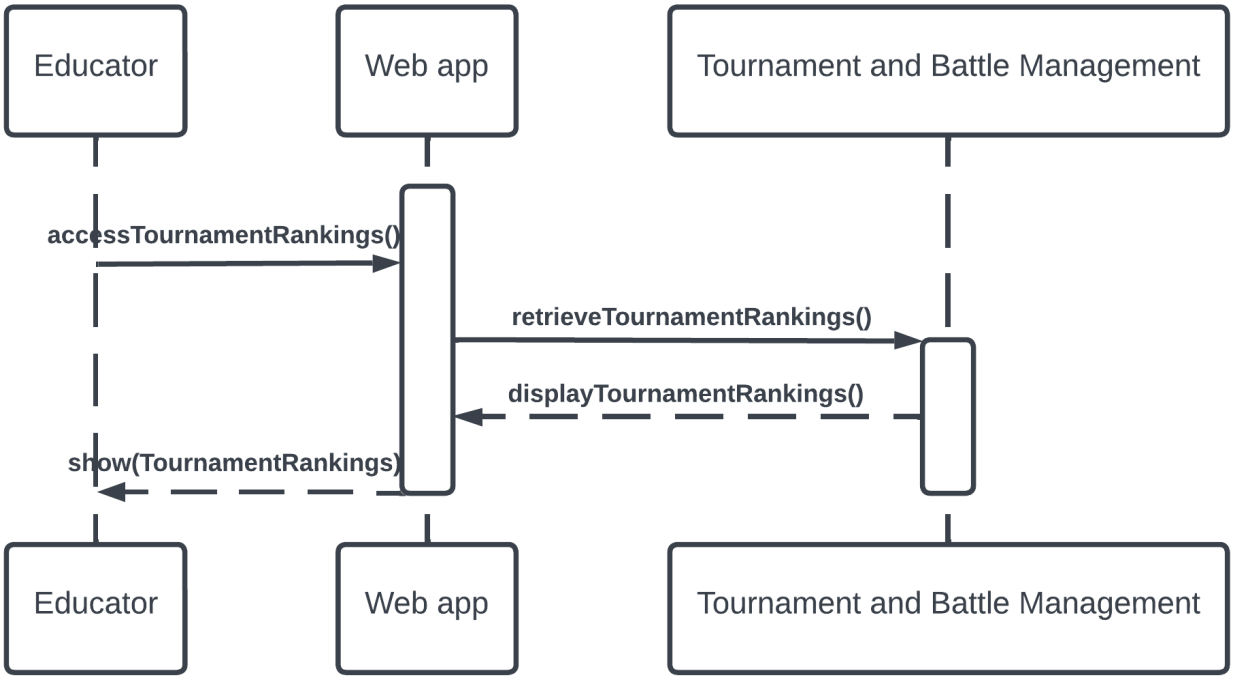
This sequence diagram outlines the steps for a student to view their personal tournament score. Initially, the student navigates to their profile and requests personal tournament score information through the Web App.

Upon receiving this request, the Web App communicates with the Tournament & Battle Management System (TBMS) to retrieve information on the tournaments associated with the student.

Subsequently, the TBMS, after obtaining tournament details, requests the personal score from the Scoring System (SS). The SS processes this request and replies with the student's personal tournament score to the TBMS.

Following this, the TBMS forwards the received score back to the Web App. Finally, the Web App presents the personal tournament score information to the student, enabling them to view their performance within tournaments on the CKB platform.

[UC18] - Educator Reviews Tournament Rankings

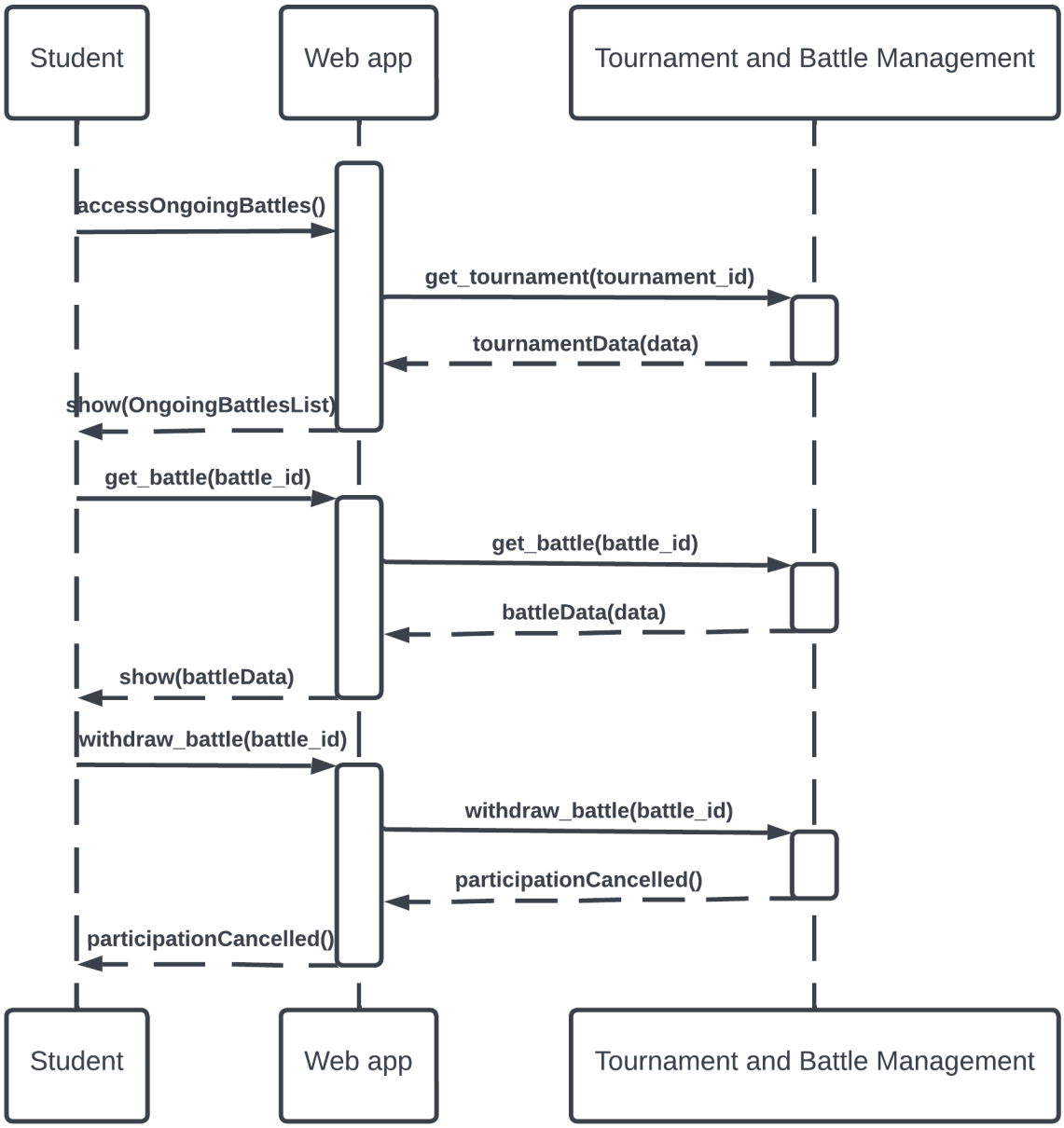


This sequence diagram outlines the steps for an educator to review tournament rankings. Initially, the educator contacts the Tournament & Battle Management System (TBMS) through the Web App to request the rankings of a specific tournament.

Upon receiving this request, the TBMS processes the inquiry and gathers the necessary ranking information for the requested tournament. The TBMS promptly replies with the tournament rankings to the Web App, fulfilling the educator's request.

This sequence ensures a straightforward process for educators to access and review tournament rankings within the CKB platform, allowing them to gauge participant performance and standings effectively.

[UC19] - Student Cancels Battle Participation

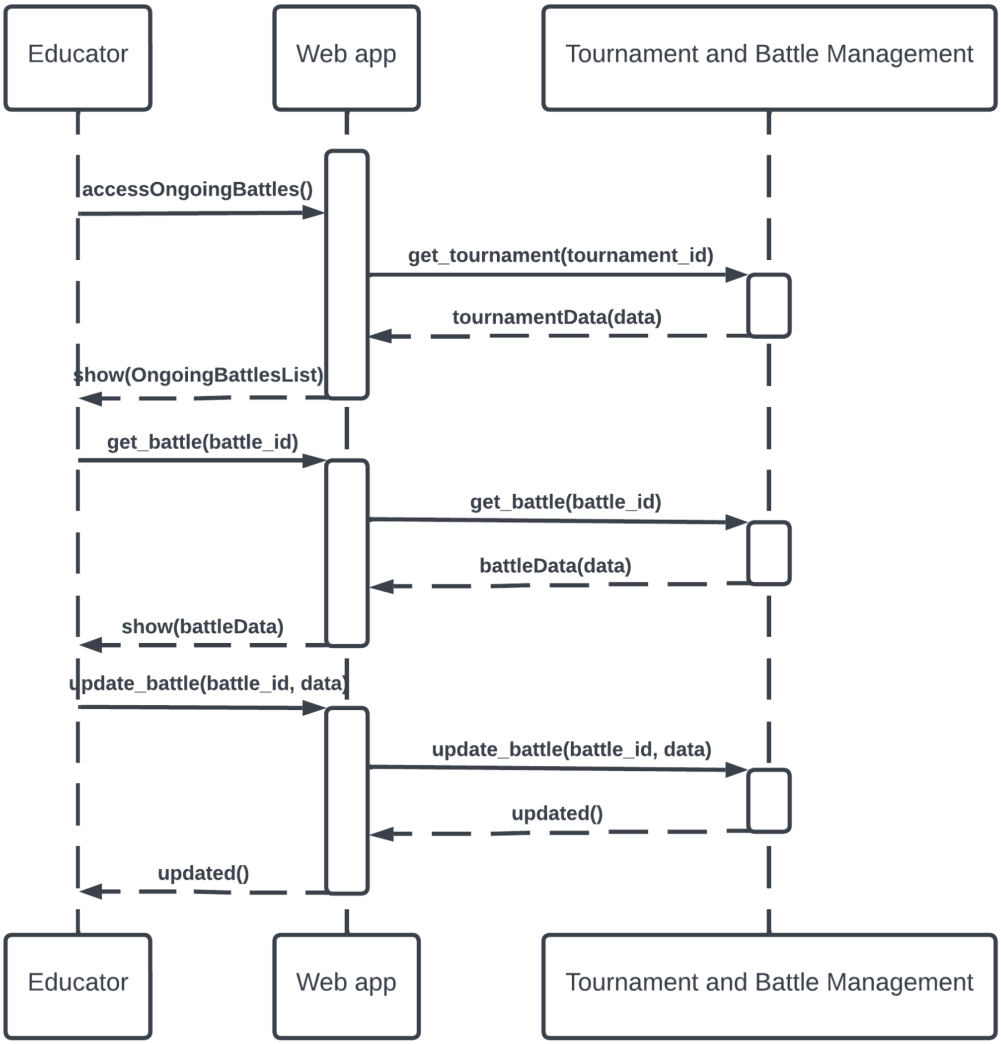


This sequence diagram illustrates the steps for a student to cancel their participation in a battle. Initially, the student requests tournament details through the Web App to the Tournament & Battle Management System (TBMS). Subsequently, after obtaining tournament information, the student requests specific battle details from the TBMS through the Web App.

Upon receiving the battle information, the student decides to withdraw from the battle and sends a withdrawal request to the TBMS through the Web App. The TBMS processes this request and confirms the successful withdrawal from the battle to the student via the Web App.

This sequence outlines a streamlined process for students to cancel their participation in a battle within the CKB platform, ensuring efficient handling of withdrawal requests and clear communication of the withdrawal confirmation.

[UC20] - Educator Extends Battle Submission Deadline



This sequence diagram outlines the steps for an educator to extend the submission deadline for a battle. Initially, the educator requests tournament details via the Web App to the Tournament & Battle Management System (TBMS). Following this, the educator requests specific battle details from the TBMS through the Web App.

Upon receiving the battle information, the educator decides to update the battle, specifically extending the submission deadline. The educator sends the updated information to the TBMS through the Web App.

Upon receiving the updated information, the TBMS processes this request and updates the battle information to reflect the extended submission deadline. Subsequently, the TBMS replies with a confirmation of the successful extension of the battle submission deadline to the educator via the Web App.

This sequence delineates a clear and systematic process for educators to extend the submission deadline for battles within the CKB platform, ensuring effective communication and management of deadline extensions.

2.6. Selected architectural styles and patterns

2.6.1. Model-View-Controller (MVC) Pattern

The MVC pattern divides the application into three interconnected components: Model, View, and Controller. The Model represents the data and business logic, storing and managing information independently of the user interface. The View represents the presentation layer, responsible for rendering the user interface elements. The Controller acts as an intermediary, handling user input, orchestrating actions, and updating the Model and View accordingly. This pattern ensures separation of concerns, making the system more maintainable, testable, and scalable. For instance, in Django, Models represent database tables, Views render HTML templates, and Controllers (Views in Django) handle user requests, processing them, and returning appropriate responses. (See more under Overview 2.1)

2.6.2. Client-Server Architecture

The client-server model divides the system into two main components: the Client (user's web browser) and the Server (hosting the CKB Web Application). The client sends requests to the server, and the server processes these requests, performs necessary computations or data retrieval, and sends back responses to the client. This architecture ensures centralized data management, enabling better control over access, security, and scalability. The web server, as part of the server-side, hosts and serves the CKB application to multiple clients (web browsers), managing interactions and ensuring consistent delivery of content and functionality.

2.6.3. RESTful API Design

RESTful API design follows Representational State Transfer (REST) principles, emphasizing a stateless, uniform, and standardized approach to communication between components. It focuses on resource-based interactions via well-defined endpoints, utilizing standard HTTP methods (GET, POST, PUT, DELETE). RESTful APIs ensure clear and predictable communication between the CKB application and external services like GitHub, promoting scalability, modifiability, and ease of integration. For example, the Django REST Framework provides tools for building RESTful APIs, defining endpoints for actions like retrieving battles, submitting code, or fetching user information.

2.6.4. Observer Pattern (Event-Driven Architecture)

The Observer pattern fosters an event-driven architecture, where components subscribe to and receive notifications about specific events or changes. In the CKB platform, events like submission deadlines, score updates, or battle results trigger notifications to inform users. This pattern decouples components, allowing asynchronous processing and reducing dependencies between system elements. Messaging services or event listeners are employed to broadcast notifications based on specific events, ensuring users stay updated on relevant information without blocking other system operations.

2.6.5. Microservices (Partial Implementation)

Microservices architecture involves breaking down a system into smaller, independent, and loosely coupled services. While not fully implemented in CKB, some components might be designed as microservices for modularity, flexibility, and scalability. For instance, code evaluation using external static analysis tools might operate as a separate microservice, allowing independent scaling and updates without affecting the entire system. This modular approach supports future enhancements and aligns with evolving scalability needs.

2.7. Other design decisions

In the design of the CodeKataBattle (CKB) platform, several other critical design decisions were made to ensure functionality, scalability, and usability. Here are some notable design decisions and their rationale

2.7.1. GitHub Integration

Integration with GitHub was chosen to leverage version control features, facilitating code management and collaboration among students. GitHub provides robust versioning, forking, pull request, and code review functionalities, enhancing the platform's capabilities for code submission, evaluation, and team collaboration.

2.7.2. Automated Workflow (GitHub Actions)

The decision to employ GitHub Actions for automated workflows was made to streamline code evaluation processes. GitHub Actions allow for setting up automated processes triggered by events (like code submission). This automated workflow integrates with the CKB platform, initiating code analysis and evaluation upon code submission, providing real-time feedback to students and educators.

2.7.3. Use of Static Analysis Tools

Integrating external static analysis tools enables automated code evaluation. These tools assess code quality based on predefined criteria, analyzing aspects like security, reliability, and maintainability. This decision aims to provide objective assessment metrics to students, enhancing the educational value of the platform and promoting best coding practices.

2.7.4. Notification Mechanism

Implementing a notification mechanism within the platform ensures timely communication with users. Notifications inform users about upcoming battles, submission deadlines, and updates on battle results. This design decision aims to improve user engagement, keep participants informed, and enhance the overall user experience.

2.7.5. Scalability Considerations

Design decisions were made with scalability in mind to accommodate potential growth in user base and platform usage. The architecture and components were designed to be scalable, allowing for increased loads, concurrent users, and future expansion. Utilizing cloud-based services or scalable infrastructure supports handling increased traffic and user demands.

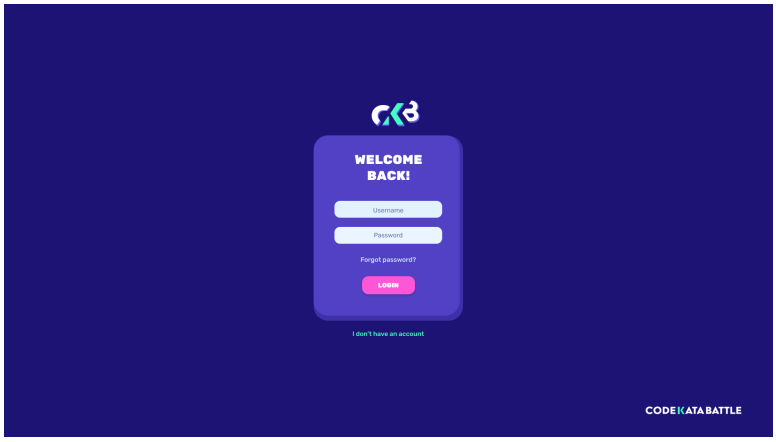
2.7.6. User-Friendly Interface (UI/UX)

Prioritizing a user-friendly interface was a crucial design decision to enhance usability. The UI/UX design focuses on intuitive navigation, clear instructions, and an engaging layout, ensuring ease of use for both educators and students navigating through battles, submissions, and scores.

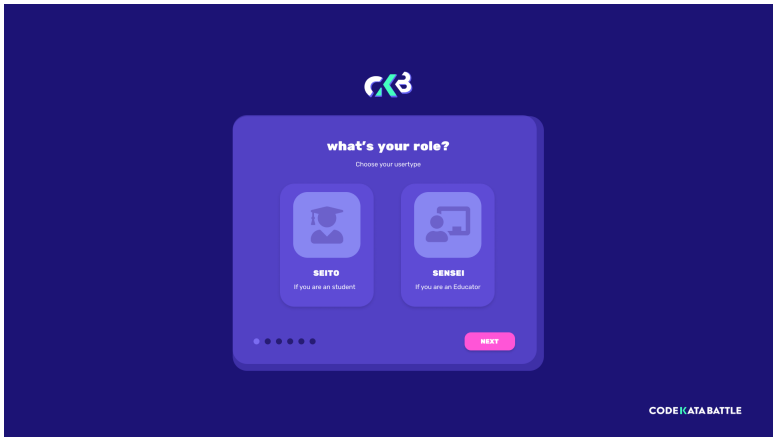
3. USER INTERFACE DESIGN

This chapter serves as a visual gateway into the user experience envisioned for the CodeKataBattle (CKB) platform. Through a collection of mockups and graphical representations, it showcases the intuitive and interactive interface designed to facilitate seamless user interaction.

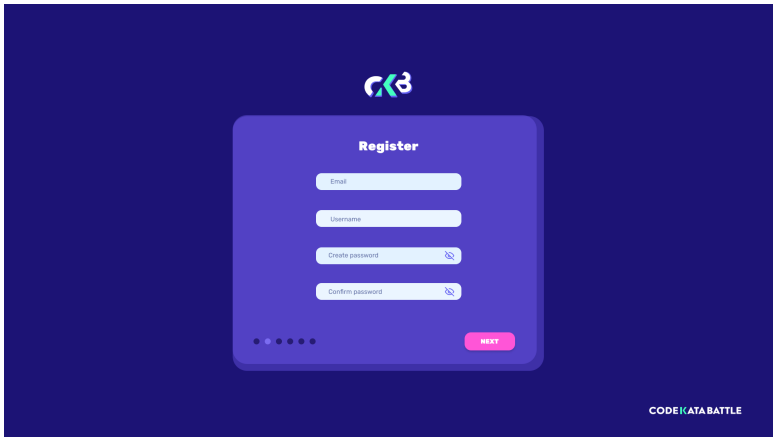
The primary goal is to present a comprehensive depiction of the platform's user interface, encompassing various screens, layouts, and functionalities envisioned within the CKB ecosystem. Each mockup elucidates the visual aesthetics, navigational pathways, and functionalities, offering stakeholders a preview of the platform's look and feel.



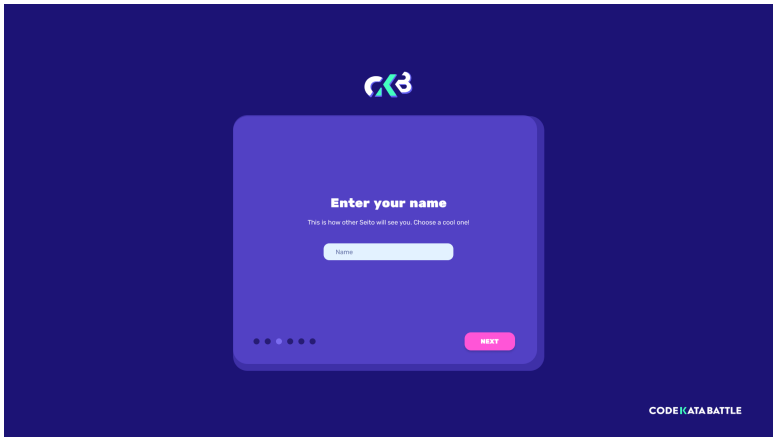
(Figure 3.1. Login Screen)



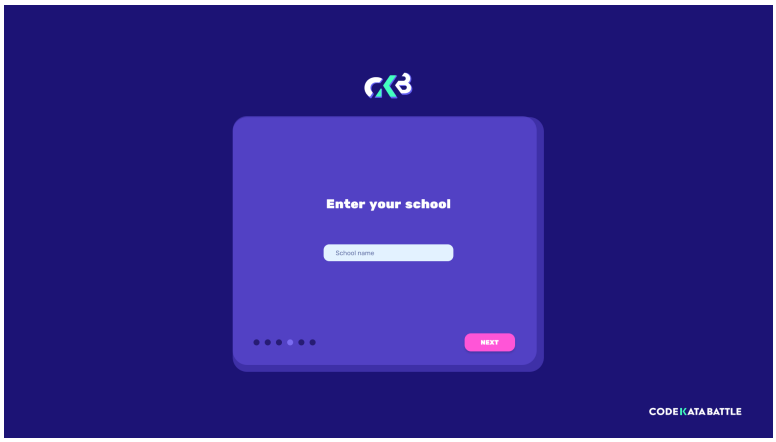
(Figure 3.2.1. UserType Screen)



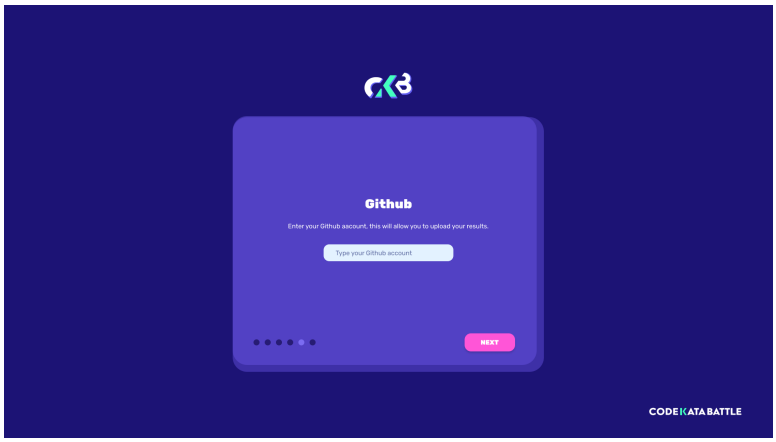
(Figure 3.2.2. Sign-up Screen)



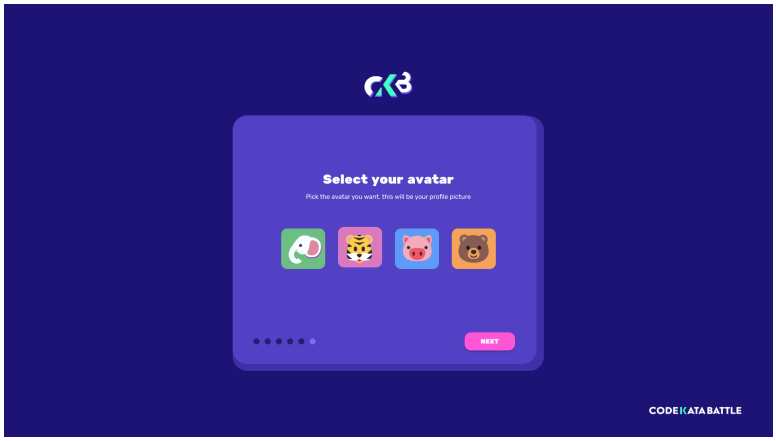
(Figure 3.2.3. Sign-up Name Screen)



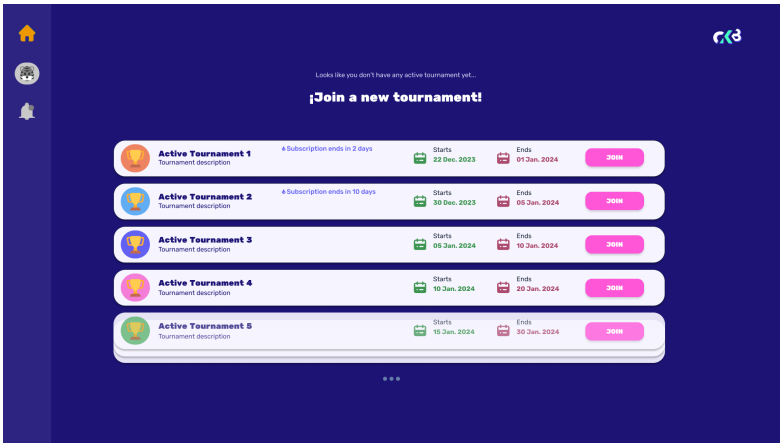
(Figure 3.2.4. Sign-up School Screen)



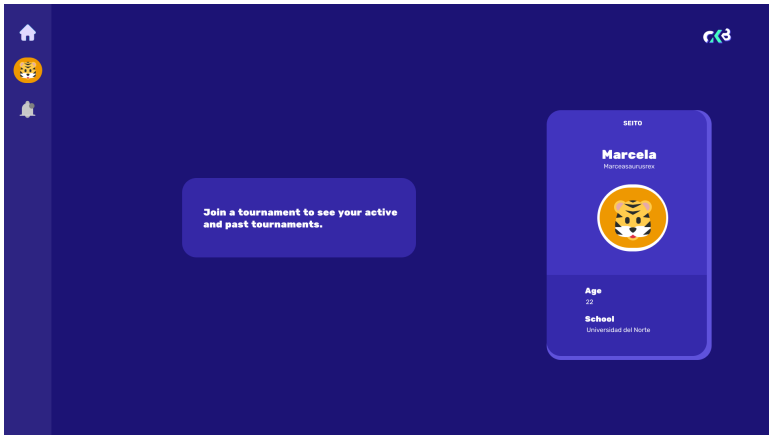
(Figure 3.2.5. Sign-up Github Screen)



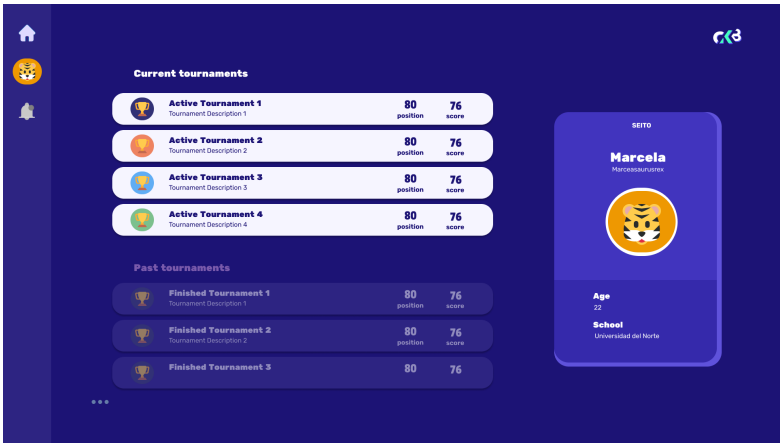
(Figure 3.2.6. Sign-up Avatar Screen)



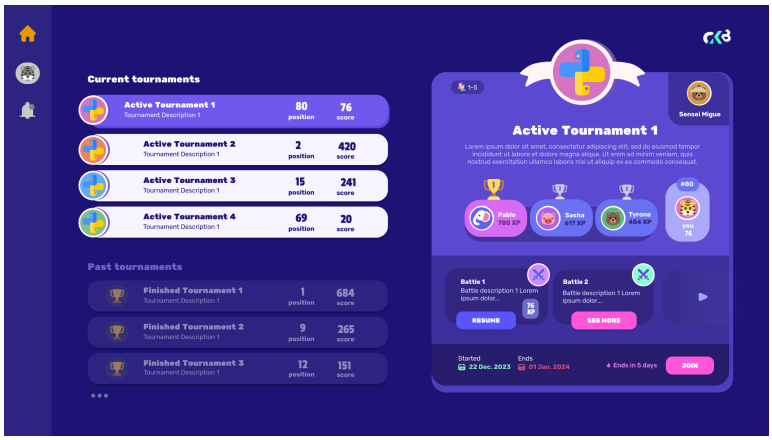
(Figure 3.2.7. Sign-up First Tournament Screen)



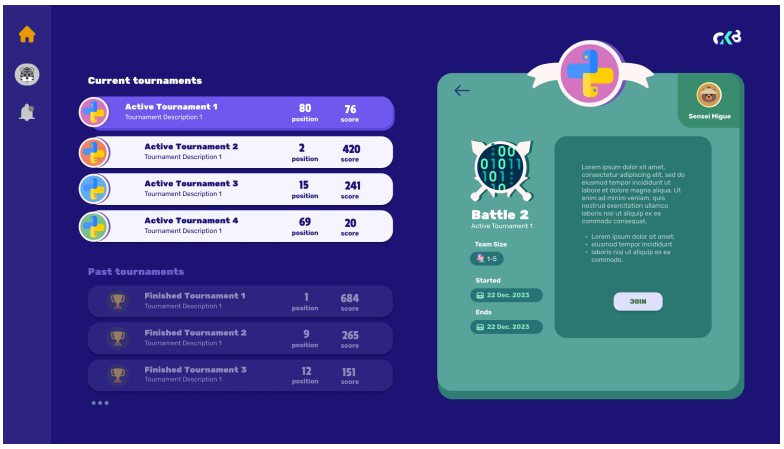
(Figure 3.3.1. Student Profile Screen)



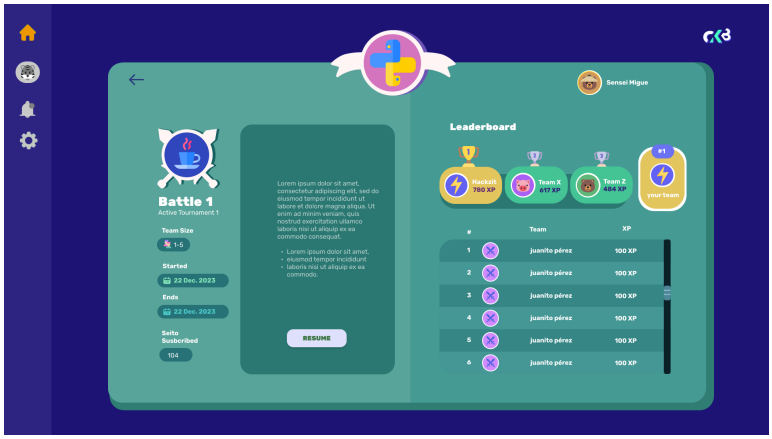
(Figure 3.3.2. Student Profile Screen)



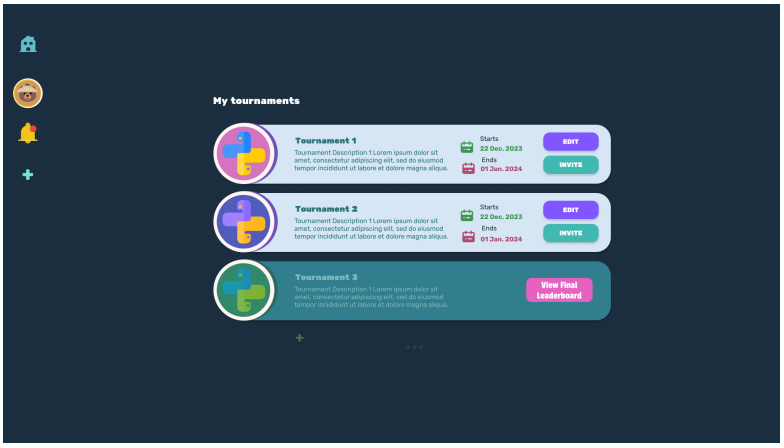
(Figure 3.4. Student dashboard Screen)



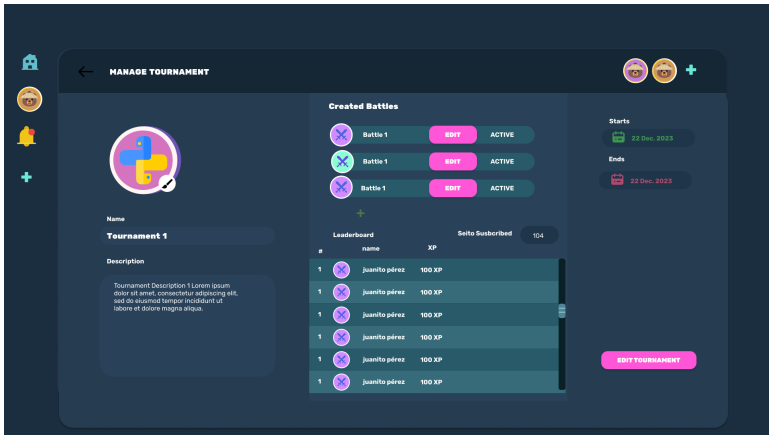
(Figure 3.5. Student battle Join Screen)



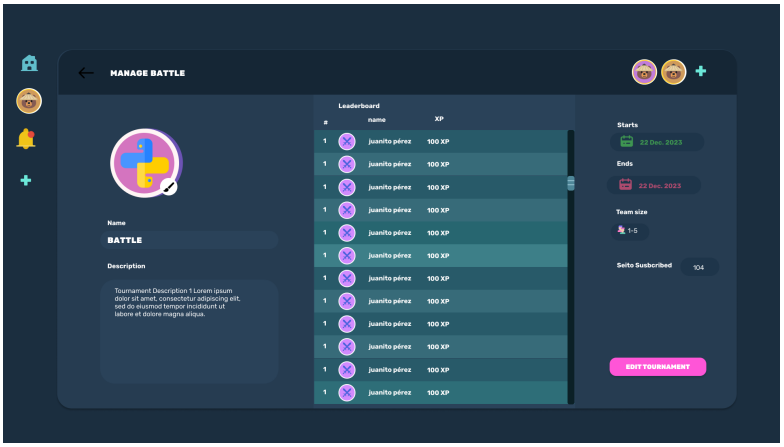
(Figure 3.6. Student battle Screen)



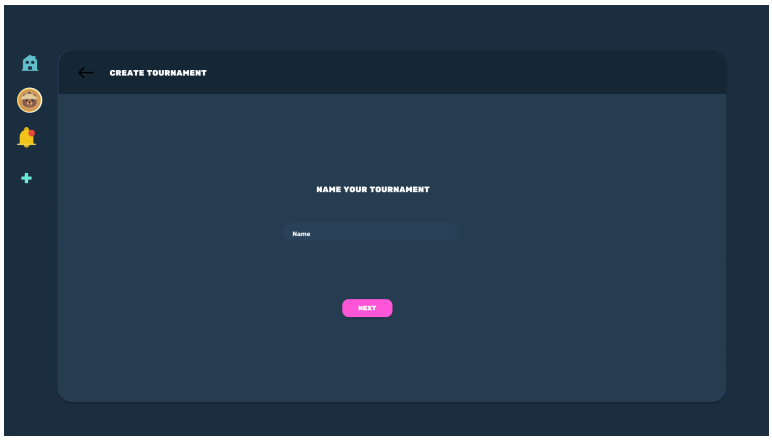
(Figure 3.7. Educator Dashboard Screen)



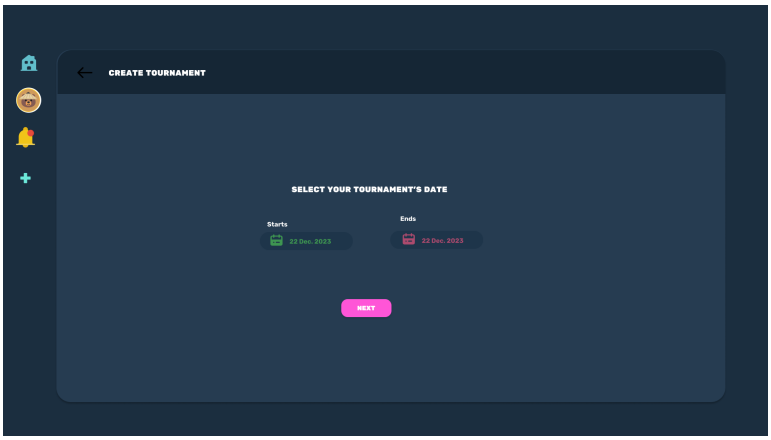
(Figure 3.8. Educator tournament management Screen)



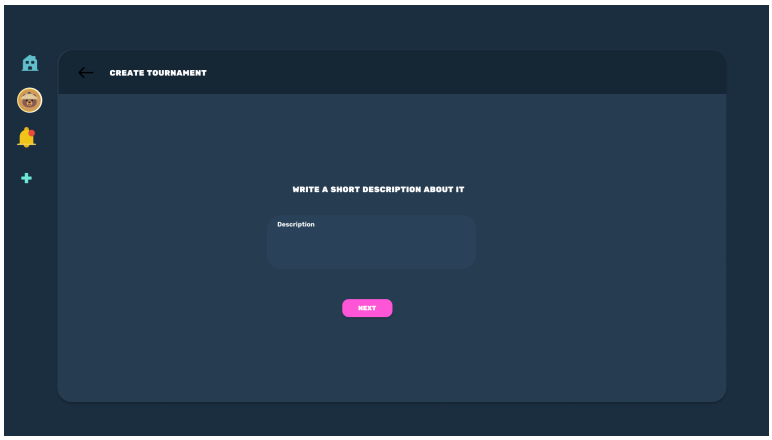
(Figure 3.9. Educator battle management Screen)



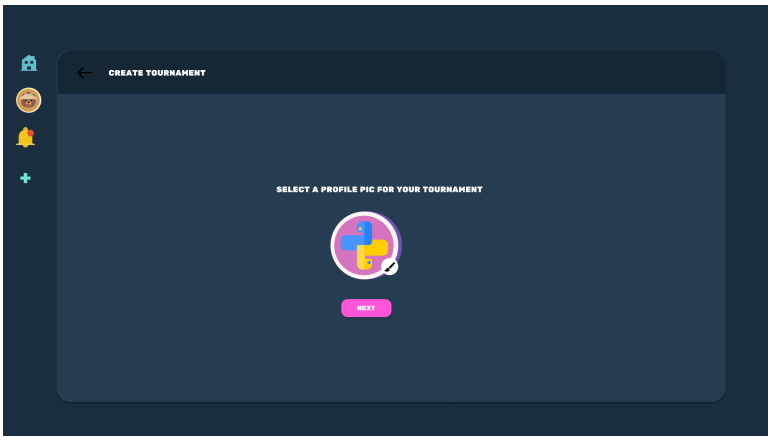
(Figure 3.10.1 Educator Torunament creation Screen)



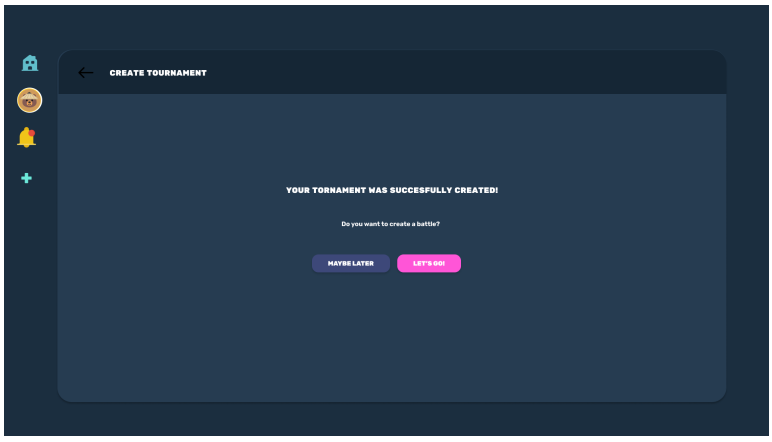
(Figure 3.10.2 Educator Torunament creation Screen)



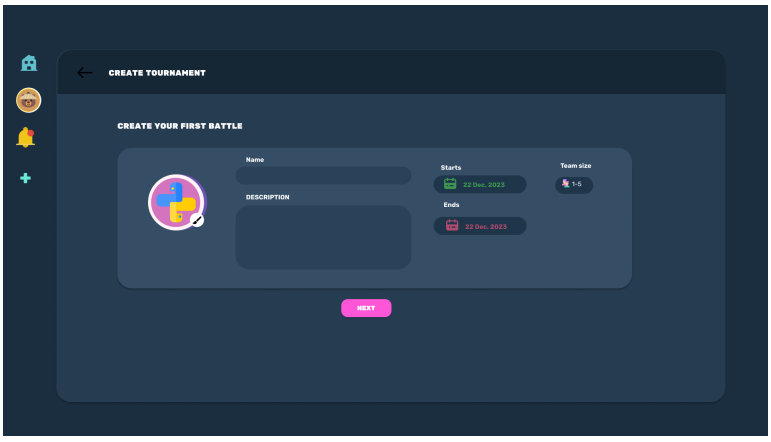
(Figure 3.10.3 Educator Torunament creation Screen)



(Figure 3.10.4 Educator Torunament creation Screen)



(Figure 3.10.4 Educator Torunament creation Screen)



(Figure 3.10.5 Educator Torunament creation Screen)

4. REQUIREMENTS TRACEABILITY

Requirements Traceability stands as a fundamental aspect of the CodeKataBattle (CKB) platform's development lifecycle, ensuring the alignment and coherence between the initial project requirements and the eventual system design, implementation, and testing phases. The objective is to establish a clear mapping, illustrating how each requirement in the RASD aligns with specific design components.

Through concise tables, this chapter outlines the relationships between the identified requirements and the corresponding design elements. These mappings serve as a guide, showcasing how each design element contributes to fulfilling the defined functionalities and objectives set forth in the initial requirements.

Requirements	[R1] The system allows students to sign up and log in.
Components	● User Management ○ Register User ○ Authenticate User ● WebApp ● DBMS ● Notification System ○ notifyUsers

Requirements	[R2] The system allows educators to sign up and log in.
Components	● User Management ○ Register User ○ Authenticate User ● WebApp ● DBMS ● Notification System

Requirements	[R3] Educators can create new tournaments, defining code battles with descriptions, deadlines, and scoring configurations. [R4] Educators can set minimum and maximum team sizes for battles within a tournament. [R5] Educators can manage and view the list of battles they've created within a tournament. [R6] Educators can grant permission to colleagues to create battles within a specific tournament.
Components	● WebApp ● Tournament and Battle Management ○ Create Tournament ○ Create Battle ● DBMS ● Notification System ○ notifyUsers

Requirements	[R7] Educators can upload the code kata with the respective description and software project, including test cases and build automation scripts. [R9] Educators can set a registration deadline. [R10] Educators can set a final submission deadline. [R24] Implement role-based permissions, allowing educators administrative access to manage tournaments, battles, and evaluations, while students have limited access as participants.
Components	● WebApp ● Tournament and Battle Management ○ setDetails ● DBMS ● Notification System ○ notifyUsers

Requirements	[R12] Students can form teams for battles, adhering to set team size limits. [R13] The platform automatically generates GitHub repositories for each battle upon creation.
Components	● WebApp ● Tournament and Battle Management ● Team Formation and Github Integration ● DBMS

Requirements	[R11] Educators can set additional configurations for scoring. [R14] The platform performs automated evaluation of submitted code based on test cases, calculating the functionality score. [R15] Integration with static analysis tools for quality evaluation, considering security, reliability, and maintainability of code.
Components	● WebApp ● Tournament and Battle Management ● Automated Evaluation ● Scoring System ● DBMS

Requirements	[R16] Educators can manually assess and assign personal scores to student submissions. [R17] After the submission deadline, educators review and provide manual evaluations based on code quality and adherence to the test-first approach.
Components	● WebApp ● Tournament and Battle Management ● Team Formation and Github Integration ● Scoring System ● DBMS ● Notification System

Requirements	[R18] The system continuously updates battle scores during the battle period, displaying evolving rankings to students and educators. [R20] The platform updates personal tournament scores for each student, presenting the cumulative performance across battles within a tournament.
Components	● WebApp ● User Management ● Tournament and Battle Management ● Scoring System ○ updateScores ● DBMS ● Notification System

Requirements	[R21] Automated notifications inform students about new tournaments and upcoming battles within subscribed tournaments. [R22] Real-time updates notify users (students and educators) about rank changes, deadlines, and significant tournament-related information.
Components	● WebApp ● Tournament and Battle Management ● DBMS ● Notification System ○ notifyUser

Requirements	[R23] Email notifications alert users about critical updates, including new tournaments, battle results, and personal scores.
Components	● WebApp ● Tournament and Battle Management ● Scoring System ● DBMS ● Notification System ○ notifyUser

Requirements	[R25] Enable users (students and educators) to view their profiles, displaying personal information, ongoing tournaments, past performances, and badges earned.
Components	● User Management ○ getUserInformation ● WebApp ● DBMS

5. IMPLEMENTATION, INTEGRATION AND TEST PLAN

5.1. Overview

The Implementation, Integration, and Test Plan chapter provides a structured approach to realize the CodeKataBattle (CKB) platform by detailing the systematic implementation of subcomponents, their integration, and the subsequent testing procedures. This section is crucial in delineating the roadmap for development, ensuring a systematic and controlled process from the initial implementation of individual subcomponents to their seamless integration and rigorous testing. By establishing a clear plan for development, integration, and testing, this chapter aims to ensure the reliability, functionality, and cohesiveness of the CKB platform. The orderly implementation and integration of subcomponents, followed by comprehensive testing, are critical to achieving a robust and functional system that fulfills the requirements and expectations of educators and students engaged in the platform.

5.2. Implementation plan

The Implementation Plan for the CodeKataBattle (CKB) platform delineates a strategic roadmap for building a robust and functional system. This section outlines a comprehensive approach, leveraging both bottom-up and threaded strategies to ensure an efficient and cohesive development process.

Bottom-Up Strategy:
The bottom-up strategy focuses on laying a solid foundation by commencing with the implementation of fundamental and foundational components. This approach entails initiating the development process by first addressing core building blocks such as User Authentication, and Database Integration. By solidifying these foundational elements, the strategy aims to establish a sturdy framework that forms the backbone of the CKB platform.

Threaded Approach:
In conjunction with the bottom-up strategy, a threaded approach will be employed to address various aspects of development concurrently. This method involves parallel development threads where multiple teams or developers work on distinct components simultaneously. The threaded approach aims to maximize visible progress for users or stakeholders by concurrently developing portions of several modules that collectively contribute to user-visible program features. This method focuses on integrating functionalities in a way that ensures tangible progress in components that directly impact the end-user experience.
By developing and integrating modules in parallel threads, the strategy ensures that each integrated portion contributes meaningfully to user-facing functionalities. This approach fosters a steady and tangible evolution of the program features, maximizing the perceivable progress for users or stakeholders at each stage of the platform's development.

5.2.1. Features Identification

The identification of features within the CodeKataBattle (CKB) platform is a pivotal step in understanding the functional requirements that drive its development. Each feature

encapsulates a specific functionality or user-visible aspect essential to the platform's operation. Identifying and defining these features provide a structured framework for development, ensuring that the system fulfills the needs of educators and students while delivering a seamless and comprehensive user experience.

[F1] Sign in and sign up

The Sign-in and Sign-up feature serves as a core foundational component pivotal for user interaction. It ensures secure access to the platform, validating user identities during authentication processes. While seemingly a singular feature, it can be subdivided into multiple components such as user registration, password management, and authentication protocols. While not directly visible, its reliability and security establish the groundwork for user interaction, making it a foundational element crucial for the platform's integrity and user trust.

[F2] Tournament Creation

Tournament Creation stands as a core feature enabling educators to design and structure coding challenges. It allows customization of competitions, setting parameters, and deadlines. This feature is integral, forming the backbone for educators to define diverse challenges and learning experiences. While not subdivided, its functionalities may include tournament setup, rule establishment, and submission guidelines, serving as a key driver for student engagement and skill development.

[F3] Battle Creation

Battle Creation is fundamental for educators to craft specific coding scenarios within tournaments. It acts as a core feature influencing the learning objectives by tailoring challenges. It could be divided into sub-features like task creation, test case setting, and deadline management. Its significance lies in providing precise and varied coding challenges, essential for students to enhance their problem-solving skills.

[F4] GitHub Integration

GitHub Integration is pivotal for seamless code submission and management. While seemingly independent, it plays a critical role in code version control and submission, making it an integral part of the platform. It may encompass sub-features like repository linking, version control setup, and automated code retrieval. Though not directly visible, its implementation significantly impacts user experience and collaborative coding practices.

[F5] Team Management

Team Management fosters collaboration among students participating in battles. It plays a crucial role in enhancing peer learning and teamwork. Sub-features might include team creation, member invitation, and collaborative tools. Its significance lies in fostering a

supportive environment, augmenting learning through teamwork and collective problem-solving.

[F6] Tournament management
Tournament Management serves as a foundational feature for overseeing coding competitions. It allows educators to administer tournaments effectively. While not subdivided, its functionalities might encompass tournament settings, participant monitoring, and progress tracking. Its significance lies in providing educators with control and oversight, ensuring organized and impactful coding events.

[F7] Battle management
Battle Management ensures precise oversight of individual coding challenges within tournaments. It might include sub-features like task-specific configurations and submission deadlines. Its significance lies in tailoring and monitoring coding challenges, ensuring alignment with learning objectives and assessment precision.

[F8] Model Implementation
Model Implementation forms the backbone of the platform architecture. This foundational feature establishes core structural models and functionalities essential for scalability, stability, and seamless integration of platform components. While not explicitly subdivided, its robust development is crucial for the overall stability and performance of the CodeKataBattle platform.

[F9] Automated evaluation
Automated Evaluation expedites the assessment process for submitted code solutions. This feature automates grading, executes test cases, analyzes code quality, and generates automated scores. While not subdivided, its role in providing timely feedback and code quality improvement among participants.

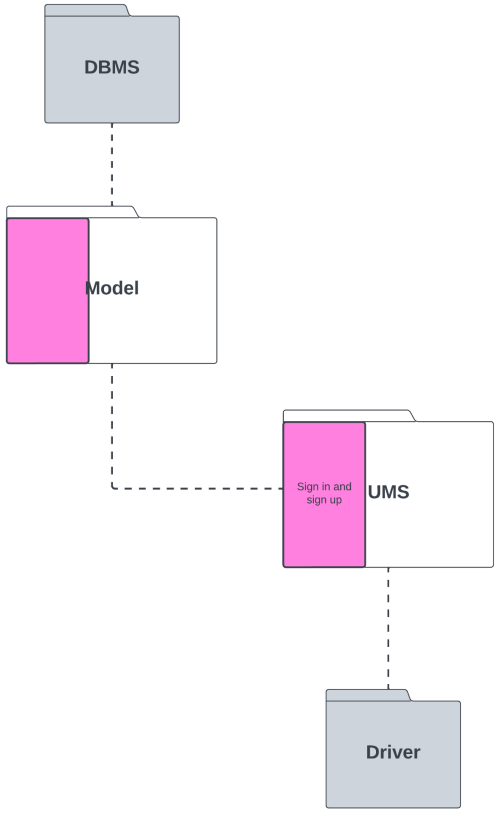
[F10] Scoring system
The Scoring System quantifies performance and progress within coding challenges. It calculates and assigns scores based on automated evaluations and potential manual scores provided by educators. While not divided, its implementation ensures fair and transparent evaluation.

[F11] Notification system
The Notification System manages timely communication across the platform. It disseminates updates, deadlines, and critical information to users involved in tournaments and battles. While not subdivided, its implementation ensures effective information flow, enhancing user engagement and platform responsiveness.

5.3. Component Integration and testing

This section delineates the staged integration and testing plan for the CodeKataBattle (CKB) platform. Each stage involves the implementation of specific components followed by integration procedures and testing methodologies to ensure the platform's functionality, reliability, and interoperability.

Stage 1: Foundational Component Implementation



(Figure 5.1. Stage 1 Implementation)

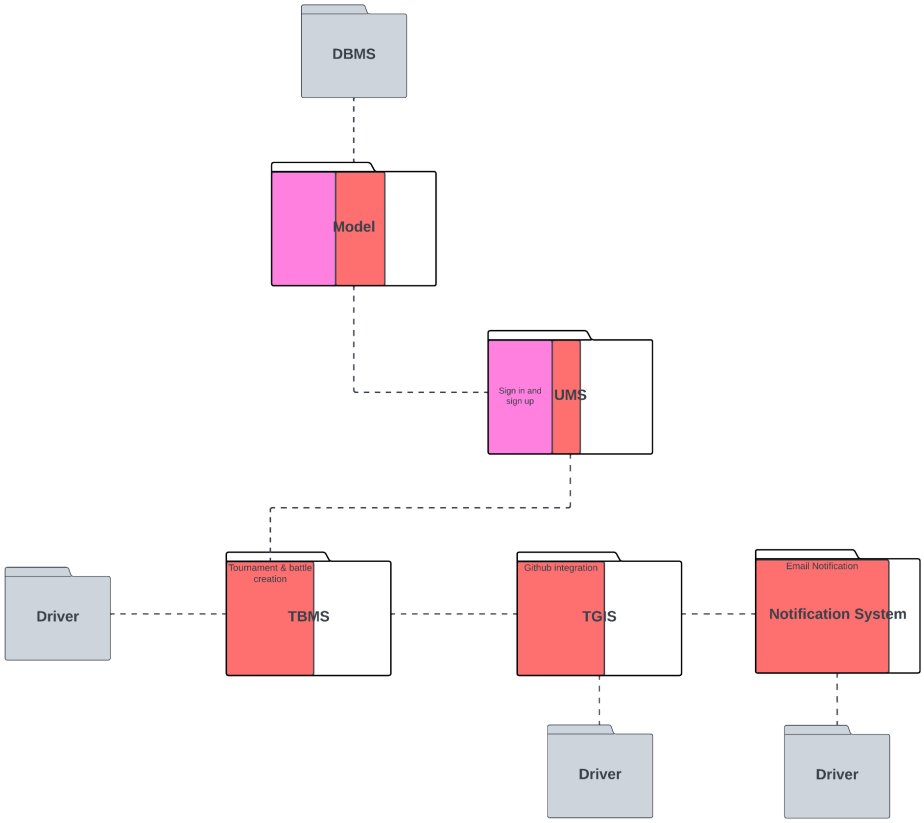
Components Implemented:

- Model Implementation (F8)
- Sign in and sign up (F1)

Integration Approach:

- Model Implementation Integration: Integrate foundational models with authentication functionalities into the database management system.
- User Management system: Sign in and sign up functionalities integrated with the model
- Authentication Workflow Testing: Verify user authentication, registration, and data storage functionality.

Stage 2: User Engagement Features Implementation



(Figure 5.2. Stage 2 Implementation)

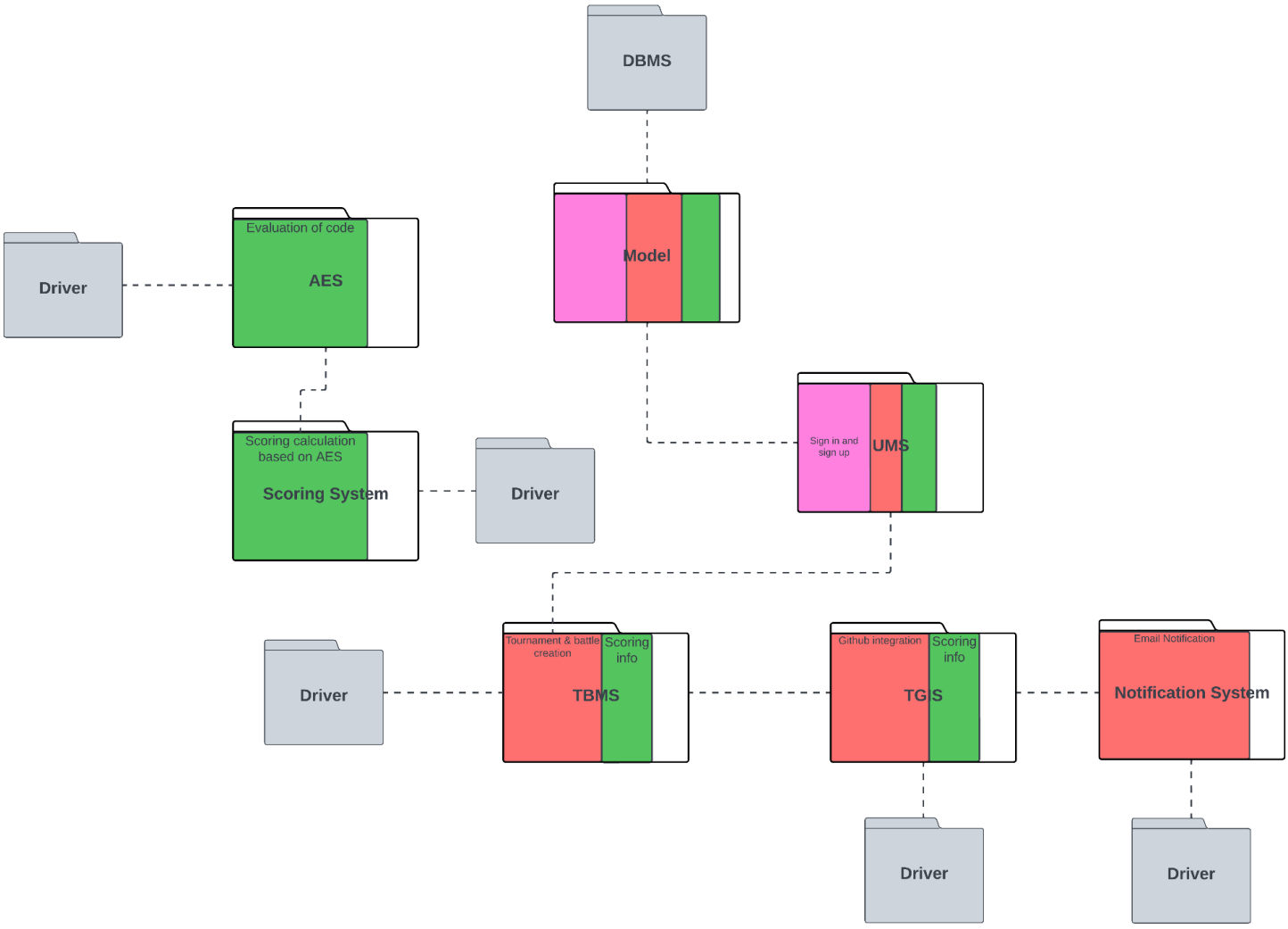
Components Implemented:

- Tournament Creation (F2)
- Battle Creation (F3)
- GitHub Integration (F4)
- Notification System (F11)

Integration Approach:

- Tournament and Battle Management Integration: Integrate tournament and battle creation functionalities.
- GitHub Integration and Notifications: Integrate GitHub repository management with notifications for user engagement.
- End-to-End Workflow Testing: Validate tournament creation, battle setup, GitHub linkage, and notification system functionality.

Stage 3: Evaluation and Scoring Mechanisms Implementation



(Figure 5.3. Stage 3 Implementation)

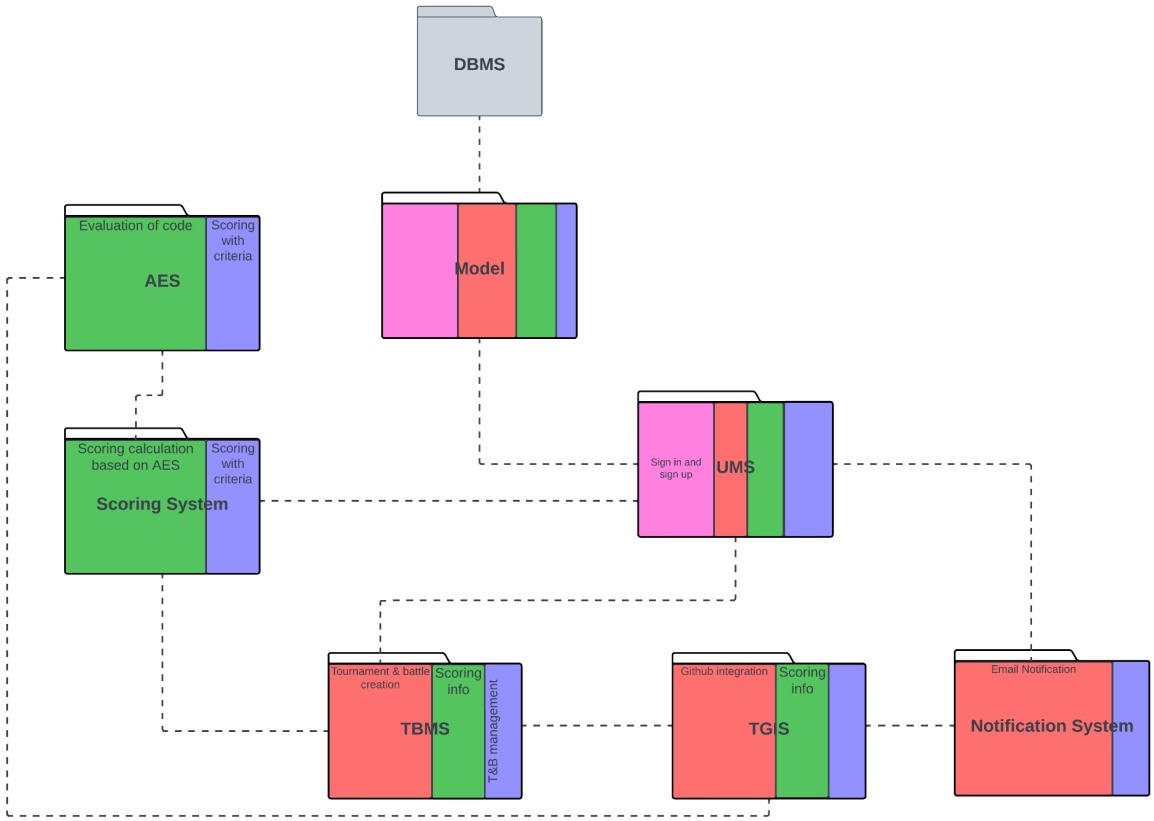
Components Implemented:

- Automated Evaluation (F9)
- Scoring System (F10)

Integration Approach:

- Automated Evaluation and Scoring Integration: Integrate automated evaluation results with the scoring system.
- Criteria-Based Scoring Testing: Validate scoring calculations based on automated evaluations.

Stage 4: Management and Ranking Features Implementation



(Figure 5.4. Stage 4 Implementation)

Components Implemented:

- Tournament Management (F6)
- Battle Management (F7)

Integration Approach:

- Tournament and Battle Oversight Integration: Integrate management functionalities with ranking calculations.
- Finishing details: Connection of the Notification system and the TGIS with the AES
- Performance Metrics Testing: Validate tournament and battle management alongside ranking generation.

This structure outlines the staged implementation, integration procedures, and testing methodologies for each phase of the CKB platform development, emphasizing the specific components involved and their interconnection throughout the integration process. Integration diagrams will visually represent the linkage between components at each stage.

5.4. System Testing

System testing within the development of the CodeKataBattle (CKB) platform is imperative to ensure the robustness, reliability, and seamless functionality of the entire system. This phase aims to evaluate the integrated system as a whole, validating its compliance with specified requirements, assessing its performance under varying conditions, and ensuring its ability to meet user expectations. Through a combination of diverse testing methodologies, this phase aims to identify and rectify potential issues, thereby guaranteeing a stable and user-friendly CKB platform.

To achieve comprehensive system testing, a range of methodologies, including functional, non-functional, and performance testing, will be employed. Each method plays a crucial role in assessing different aspects of the system, from its core functionalities to its responsiveness, security, and scalability.

5.4.1. Testing Methodologies

The system testing phase encompasses several testing methodologies:

- Functional Testing:
  - **Description:** Evaluates system functions against specified requirements.
  - **Approach:** Conducts validation of features like user authentication, tournament and battle creation, GitHub integration, scoring, and notifications.

- Non-functional Testing:
  - **Description:** Assesses system characteristics beyond functional aspects.
  - **Approach:** Includes security testing, usability assessment, reliability checks, and compatibility testing to ensure the system's robustness under diverse conditions.

- Performance Testing:
  - **Description:** Measures system performance under varying load conditions.
  - **Approach:** Conducts stress testing, load testing, and scalability checks to assess system responsiveness and stability.

- Usability Testing:
  - **Description:** Evaluates user interface intuitiveness and user experience.
  - **Approach:** Conducts user-centric tests to ensure the platform's ease of use and accessibility for diverse users.

- Compatibility Testing:
  - **Description:** Verifies the system's compatibility across various devices and platforms.

- **Approach:** Tests the platform's functionality and appearance across different browsers, devices, and operating systems.

This diverse set of testing methodologies aims to comprehensively assess the CKB platform, ensuring its functionality, security, performance, and user experience across varied scenarios and use cases.

5.5. Additional specifications on testing

These additional specifications detail the focused testing approaches for each methodology, aiming to comprehensively evaluate the CodeKataBattle platform. The diverse set of tests ensures adherence to functional requirements, security standards, performance benchmarks, and user-centric design principles.

5.5.1. Functional Testing

- User Authentication Testing: Validate login, registration, password recovery, and role-based access control functionalities.
- Tournament and Battle Functionality Testing: Verify creation, subscription, deadline management, and task setup features.
- GitHub Integration Testing: Ensure seamless code repository linking, commit tracking, and submission functionalities.
- Notification System Testing: Validate notification delivery for updates, deadlines, and score availability.

5.5.2. Non-functional Testing

- Security Testing Approach: Conduct vulnerability scans, SQL injection, cross-site scripting tests, and encryption assessments to fortify the system against potential threats.
- Usability Assessment: Gather user feedback through surveys, observing user interactions, and heuristic evaluations to enhance platform usability.
- Reliability Checks: Perform system stability tests, fault tolerance assessments, and error handling validations to ensure system reliability.

5.5.3. Performance Testing

- Stress Testing: Evaluate system response under maximum load conditions to identify performance bottlenecks.
- Load Testing: Assess system performance under normal and peak load scenarios to ensure consistent performance.
- Scalability Checks: Validate the system's ability to handle increased workload and user concurrency.

5.5.4. Usability and Compatibility Testing

- User Interface Testing: Assess ease of navigation, layout consistency, and accessibility compliance for users.
- Cross-Browser and Device Testing: Verify platform functionality across multiple browsers (Chrome, Firefox, Safari) and devices (desktop, mobile).

6. EFFORT SPENT

Daniel Mauricio Ruiz Suarez

Chapter	Effort(in hours)
1	8
2	25
3	2
4	20
5	2

Sebastian Enrique Perea Lopez

Chapter	Effort(in hours)
1	3
2	33
3	30
4	2
5	20

7. REFERENCE

Diagrams:

Lucidchart. (2023). Diagrams. Retrieved from https://www.lucidchart.com

Mockups:

Figma. (2023). Mockups. Retrieved from https://www.figma.com

IEEE Standards:

IEEE. IEEE Standards. Retrieved from https://grouper.ieee.org/groups/802/22/Documentation/format-rules.html#Document

Software Engineering 2 course material:

Politecnico di Milano 2023-2024