POLITECNICO
MILANO 1863

Design Document

CodeKataBattle

Software Engineering 2 Project
Academic year 2023 - 2024

05 January 2024
Version 1.0

Authors:
Alzbeta Fekiacova
Federico Schermi
Diego Quattrone

Contents

Chapter 1

Introduction

The Design Document (DD) serves as a foundational document for project planning, providing a basis for estimating the size, cost, and schedule of the project. It comprehensively outlines the architecture of the new system. It also provides overview of offered functionality and user interfaces. This document is instrumental in supporting system testing, verification, and validation operations, ensuring the software meets specified criteria.

1.1 Purpose

CodeKataBattle (CKB) is a Platform that provides a unique opportunity for Educators to guide Students in their software development skills journey through Code Kata Tournaments. With CKB, Educators can create Code Kata Battles that challenge Students to compete against each other in a test-driven development environment. This Platform allows Educators to create a fun and engaging learning experience for their Students while also providing a way to measure their progress and skills. For Students, CKB provides a way to practice their coding skills in a collaborative and competitive environment, which can help them improve their skills and prepare for real-world coding challenges. Overall, CKB is an important tool for Educators and Students alike, providing a fun and effective way to learn and improve coding skills.
The whole software has a client-server architectural style with a three-tier architecture as explained in more detail in the following chapters.

1.2 Glossary

1.2.1 Definitions

Guest
Anyone that is not registered to the CKB Platform.

User
Anyone that is registered and logged in to the CKB Platform. It can be represented either by a Student or an Educator account.

Co-educator
An Educator who has been given rights to create and manage Battles he has created in a Tournament created by a different Educator from the same institution.

Tournament
A competition created by an Educator with specific topic. The Tournament can contain multiple Battles.

Tournament subscription deadline
Date specified by Educator during Tournament creation. Until this time, Students are able to express their want to take part in the Tournament.

Code Kata Battle
A programming exercise belonging to a specific Tournament. It can be solved by an individual Student or by a group of Students depending on the rules provided by Educator during the Battle creation. Sometimes, also called "Code Kata Challenge" or simply a "Challenge".

Battle registration deadline
Date specified by Educator. By this date, the Students can join a Battle on their own or by creating groups.

Battle solution deadline
Date specified by an Educator. Before this date occurs, Students solutions are being evaluated.

Badge
A specific award granted to a Student or Students (depending on the Badge specification by an Educator) when Badge requirements are met. Badges are created during Tournament creation, or Tournament editing. Each Badge is unique, therefore even if the the Badges have same name or rules, they are uniquely represented in the CKB Platform.

Battle score
Each Battle contains score table that can be viewed by the participants. The score is given every time code is submitted within given deadline. The score is natural number between 0

and 100. The Battle score is a composition of automated-based system and optional manual Educator evaluation. The Battle score is given to a team as whole, no individual score is considered.

Tournament score
Each Student subscribed to a Tournament has a Tournament score that can be viewed in the Tournament score table after the Tournament has been closed. The Tournament score is sum of all Battle scores received in that Tournament by an individual.

Personal score
Personal score is a sum of Student's Tournament scores.

GitHub repository
A place created for each participating unit in Battle created after the registration deadline has occurred. Each push to main branch triggers the Battle score calculation.

Student's profile
A Student's profile contains his/her:

- Name and surname
- E-mail
- Personal score
- participated Tournaments
- participated Battles
- Claimed Badges (if there are any)

The Student's profile can be viewed by any user registered and logged in the CKB Platform.

1.2.2 Acronyms

Acronym	Meaning
CKB	CodeKataBattle
RASD	Requirements Analysis and Specifications Document

1.2.3 Abbreviations

Abbreviations	Term
e.g.	Exempli gratia
i.e.	Id est
w.r.t.	With reference to
R	Requirement

1.3 Revision history

First release: CKB_DD 1.0 - 05/01/2024

1.4 Reference Documents

- Specification Document: Assignment RDD A.Y. 2023-2024.
- Prof. Matteo Camilli's Software Engineering 2 course.
- OpenSSH protocol, 7.6 version.

1.5 Document Structure

1. ***Introductory***: This part provides an initial understanding of the project's aims and extent. It includes the terms, abbreviations, and acronyms used in the document to enhance comprehension. Additionally, it covers the revision history and the reference materials, offering insights into the document's development process.
2. ***Architectural Design***: This section presents a high-level view of the main components and their interactions. It emphasizes both the static structure and dynamic behavior, using diagrams for illustration.
3. ***User Interface Design***: This describes the envisioned appearance of the User Interface.
4. ***Requirements Traceability***: This offers a detailed explanation of how the requirements specified in the RASD correspond to the design elements mentioned in this document.
5. ***Implementation, Integration and Test plan:*** an overview of the implementation, integration, and testing plans for the CKB platform.
6. ***Effort spent***: This part details the time invested by each team member in different sections of the project.

Chapter 2

Architectural design

2.1 Overview

The figure below represents a high-level description of the System's architecture. As mentioned in the 1st chapter, the System is organized in a client-server architecture and divided into three tiers: the presentation tier, the business tier, and the data tier.

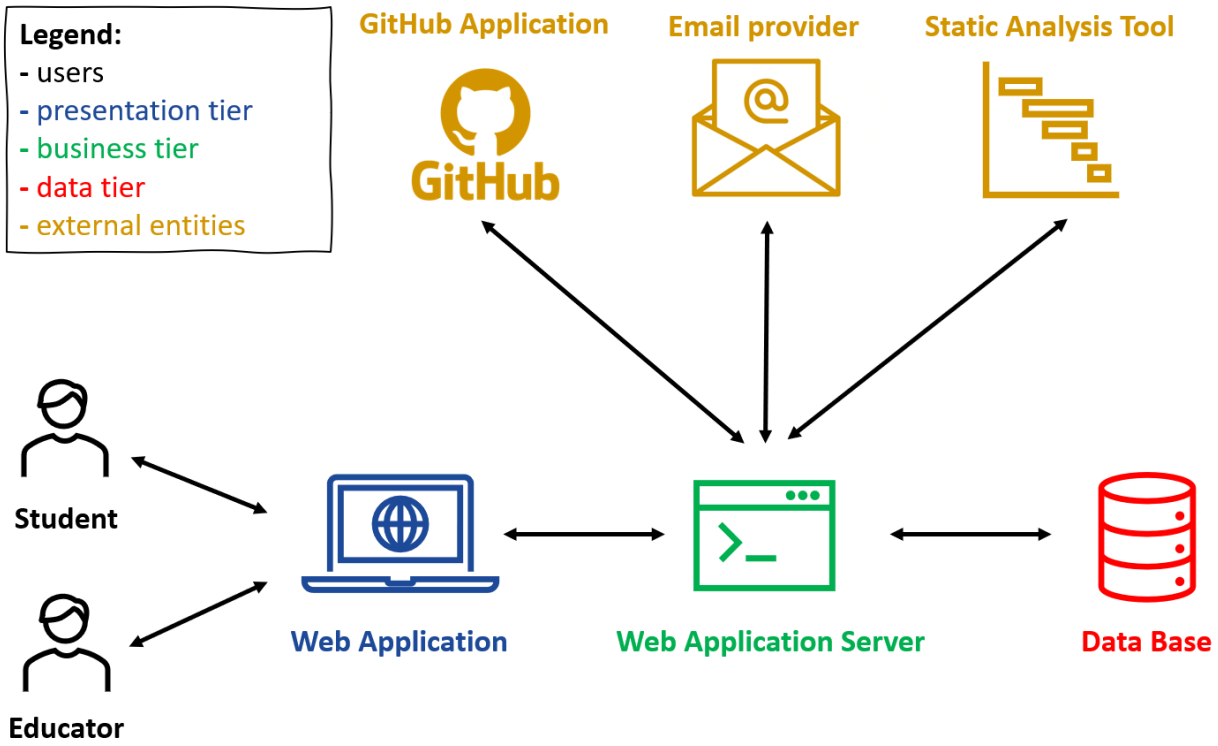


Figure 2.1: High-level description of the System's architecture

Users: interact with the Web Application of CKB. They can be:

- Student
- Educator

Presentation tier: the User Interface and communication tier of the System where User interacts with the application. Its main purpose is to display and collect information.

- Web Application: accessible through the User's browser. Allows him to have the functionalities of his role (either Student or Educator) inside the Platform by sending requests to the Web Server.

Business tier: provides the business logic for the application.

- Web Server: the back-end component of the Web Application that communicates with the User's browser and with the external entities.

Data tier: the tier whose resources are accessible only to the business tier.

- Database: responsible for the storage of all the CKB's data.

External entities: functionalities needed by CKB Platform that are provided by third parties and therefore not developed in the System.

- GitHub: the Platform interacts with GitHub to enable functionalities such as code repository access, version history retrieval, and collaboration on solving Code Kata Challenges.
- Static Analysis Tool: helps in the evaluation process by identifying potential vulnerabilities, and providing feedback to users.
- Email provider: an SMTP server to enable notification process via e-mail.

2.2 Component view

In this Section, we present and describe the System's high-level components. The color used in the component diagram corresponds to the colors used in the Overview 2.1.

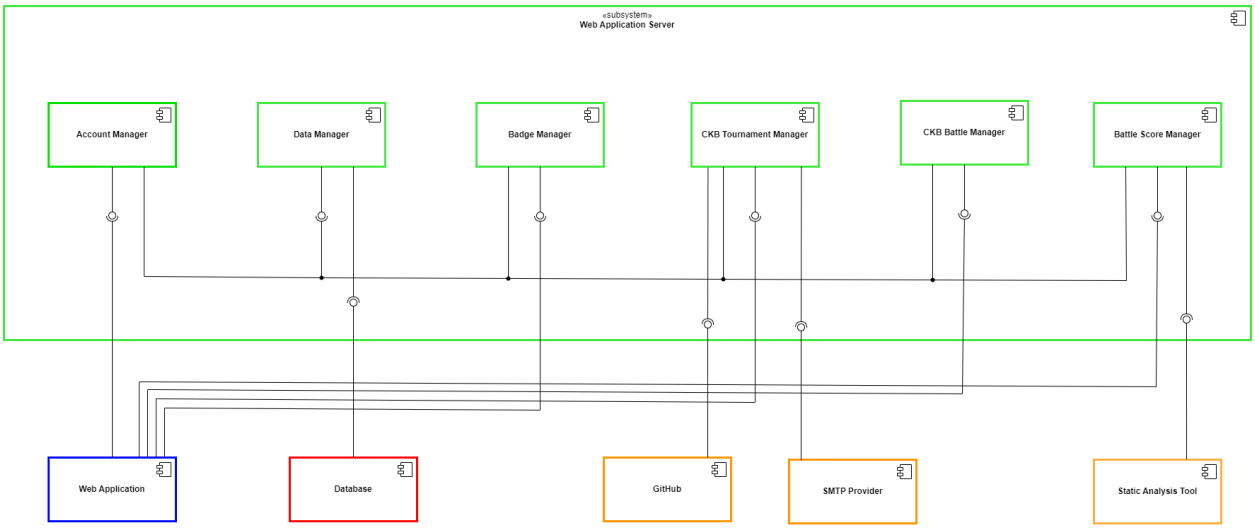


Figure 2.2: Component view of the System

Web Application: the front-end component of the System that allows the Users to interact with the CKB Platform. The only logic present in the component is the ability to perform basic checks, such as the detection of incomplete forms, while all the computational tasks are carried out by the application tier. For this reason, the application components must be able to communicate with it. Thanks to the Web Application, the Users can access the features of their roles. Students can subscribe to Tournaments, form Groups to join Battles, receive an evaluation of their solution, receive Badges. Educators can create and manage Tournaments and Battles, define Badges and Co-Educators, perform manual evaluation of the Students' solution.

Web Application Server Components: it represents the business logic, computes all the needed data, and coordinates the flow of information between the web application layer and the data layer. It is composed by:

- **Account Manager:** it handles all the operations related to the Users' accounts. It communicates with the Data Manager in order to verify, access store, and delete account information. Essentially, this component is responsible for account creation when the Users sign up and for authentication when the Users log in. Its sub-components are: **Student Registration Manager**, **Educator Registration Manager** and **Authenticator**

- **Data Manager:** it is the only component interacting with the data tier and therefore

it forwards all the requests coming from the other components to create, read, update, and delete data from the database.

- **CKB Tournament Manager:** it is responsible for all the operations related to the creation, management and subscription to a Tournament. It communicates with both Educator and Student.
- **Badge Manager:** it works in cooperation with the CKB Tournament Manager and permits an Educator to define Badges, alongside with their rule. It also assigns the Badge to all the Students that fulfill the requirement to claim the Badge.
- **CKB Battle Manager:** it is responsible for all the operations related to the creation, management and participation to a Battle. It communicates with both Educator and Student. It interacts with **GitHub**, through GitHub Actions interface, to track the push actions performed by a Student to the main branch of the repository associated with the Battle. This component needs to communicate with the **Battle Score Manager** to correctly assign the Battle Score to the Group participants .
- **Battle Score Manager:** when triggered by the **CKB Battle Manager**, it gives an automated solution evaluation, performed by the **Static Analysis Tool**. It also allows Educator to perform a manual evaluation of the given solution.

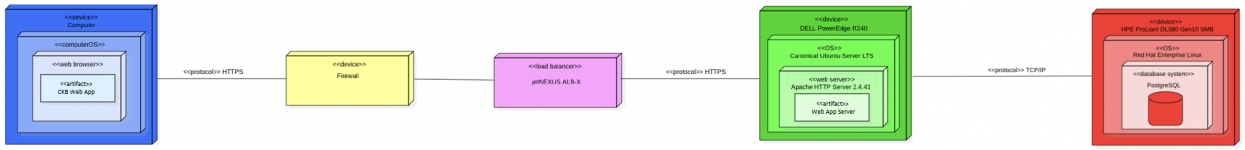
Each component within the Web Application Server interacts with a source monitoring system for real-time monitoring. This integration allows tracking of component-specific metrics and performance indicators.

Database: it is the third layer of our architecture, and it is responsible for storing and organizing all the System Data. A regular backup strategy is implemented, to ensure the integrity and availability of stored data. Automated backups can be scheduled by the Users, to not interfere with the usage of the application. Backup files are encrypted and protected.

External entities: as mentioned before, they offer all the functionalities needed by the CKB Platform. The entities are **GitHub**, **Static Analysis Tool** and **SMTP provider** to manage the sending of notifications via e-mails.

2.3 Deployment view

This section illustrates the distribution of components capturing the topology of the System by using a deployment diagram.



Computer
It is the device used by the Users. CKB web application is supposed to run in a supported browser and it uses the HTTPS protocol as well to communicate with the web server.

Firewall
It is a network security device that monitors both incoming and outgoing network traffic for security purposes. It filters the traffic coming from the computer since the device uses an untrusted network, which is the Internet, to communicate with the Web Application.

Load Balancer
It is a device that distributes the workload coming from the presentation tier among the available resources of the web application tier to increase capacity and reliability.

Web Application Server
It is the device where all the application logic resides and communicates with the presentation tier through the HTTPS protocol and with the data tier. The Web Application Server includes a monitoring node that communicates with the monitoring tool, collect performance metrics, error rates, and other relevant data to provide insights into the health of the server. Through the TCP/IP protocol, which provides end-to-end data communications specifying how data should be packetized, addressed, transmitted, routed, and received.

Database
It is the device where all the System's data are stored and communicates only with the application server through the aforementioned TCP/IP protocol. The Database server also includes a dedicated backup node, which is responsible for managing and executing the backup processes.

2.4 Runtime view

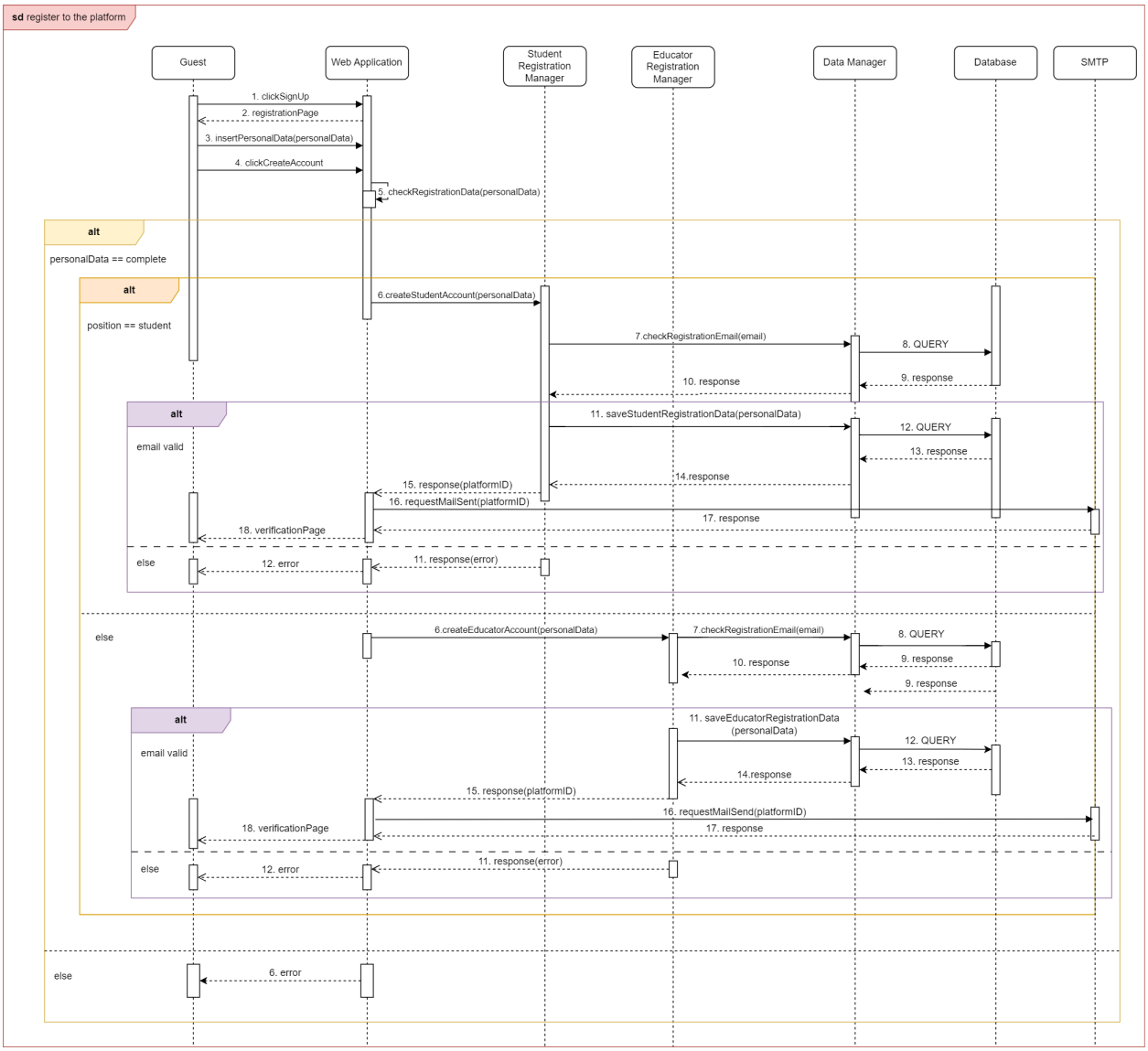


Figure 2.3: Use case 1: "Register to the Platform" - Sequence Diagram

Figure 2.4: Use case 2: "Login to the Platform" - Sequence Diagram

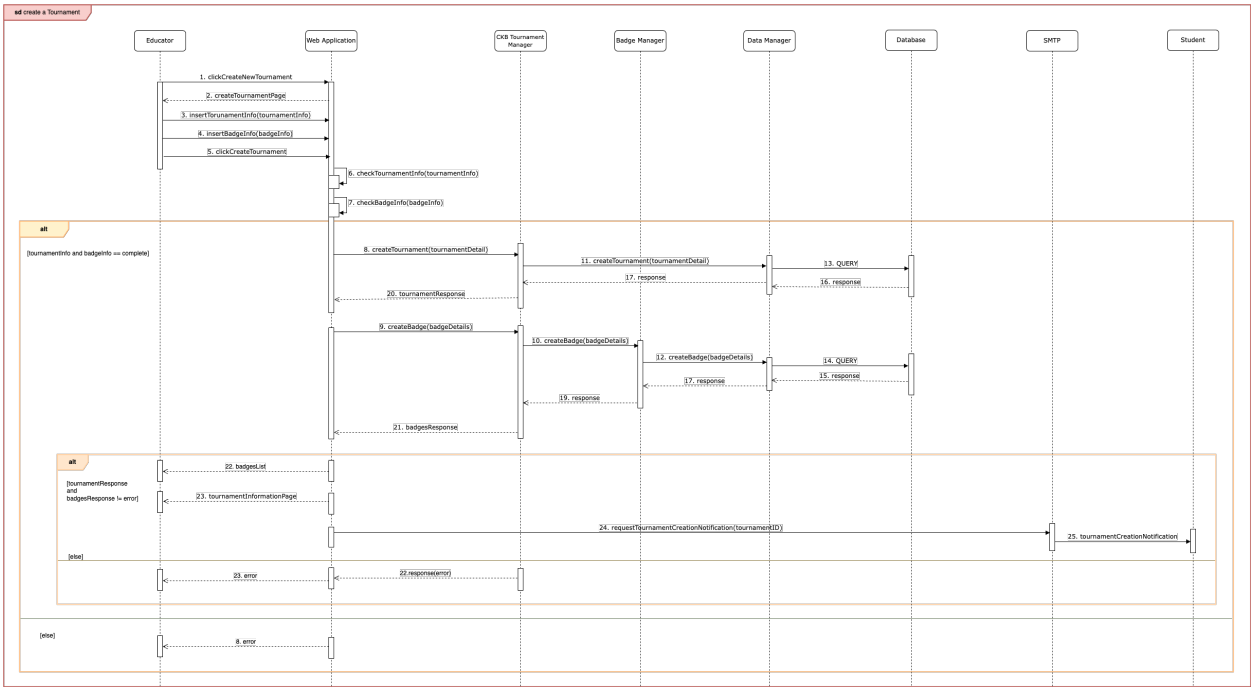


Figure 2.5: Use case 3: "Create a Tournament" - Sequence Diagram

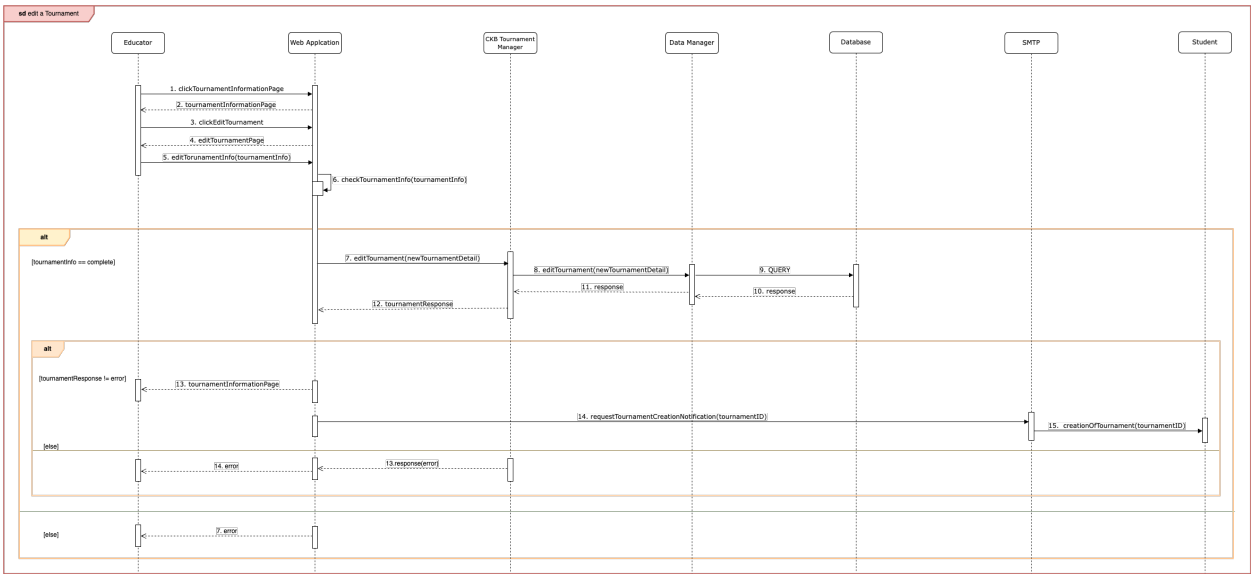


Figure 2.6: Use case 4: "Edit a Tournament" - Sequence Diagram

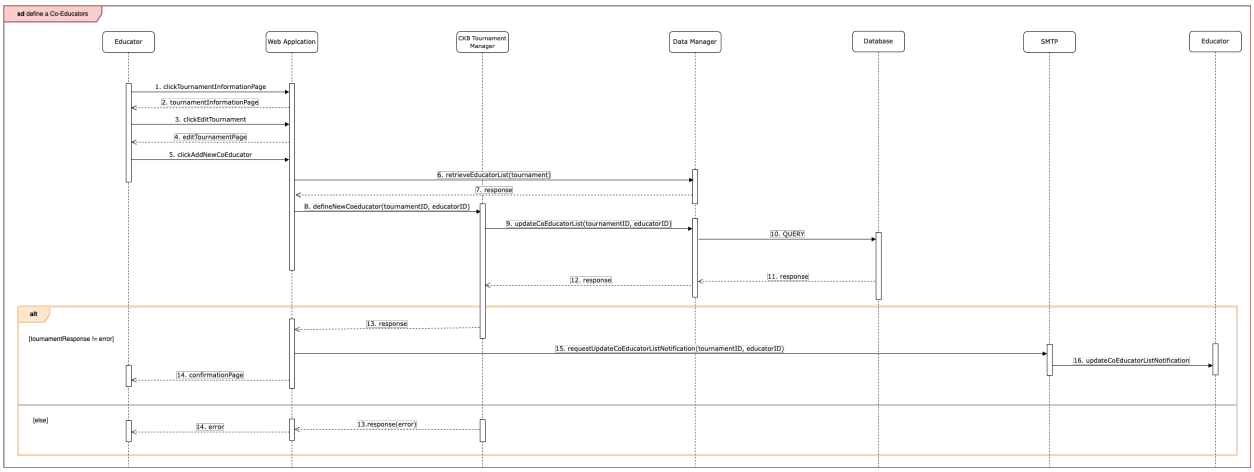


Figure 2.7: Use case 5: "Define a Co-Educators" - Sequence Diagram

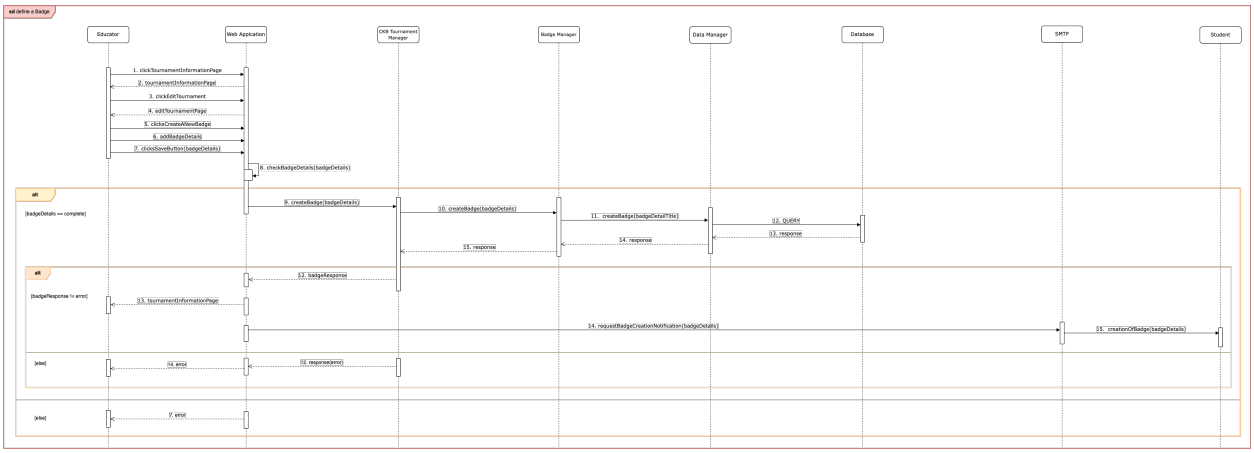


Figure 2.8: Use case 6: "Create a Badge" - Sequence Diagram

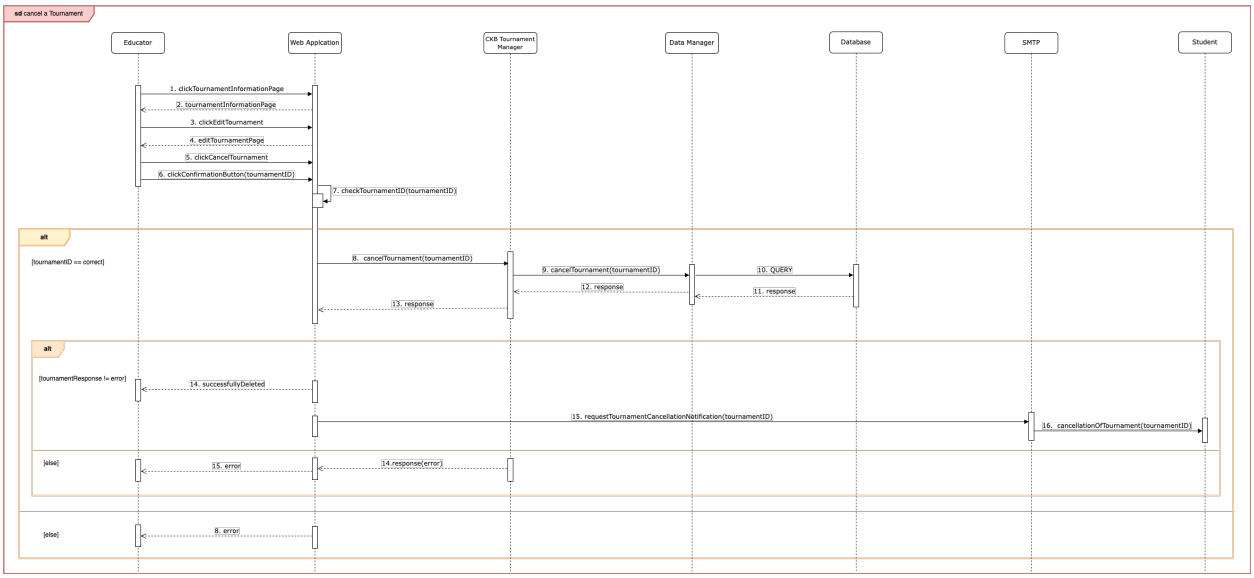


Figure 2.9: Use case 7: "Cancel a Tournament" - Sequence Diagram

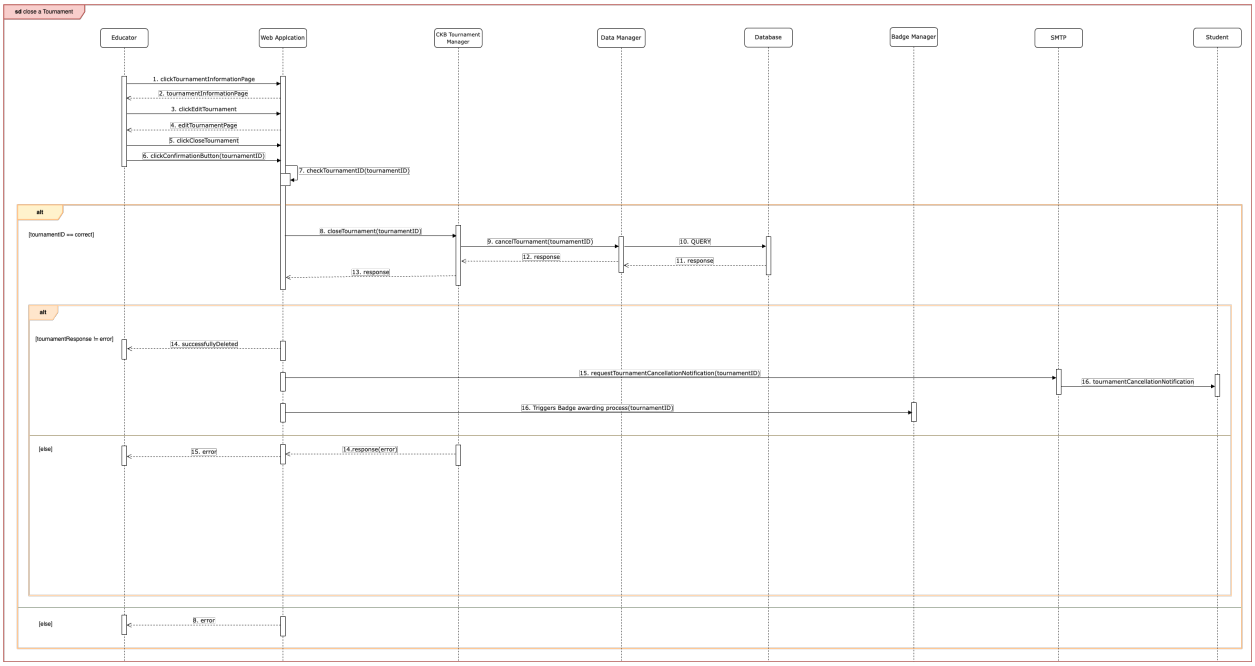


Figure 2.10: Use case 8: "Close a Tournament" - Sequence Diagram

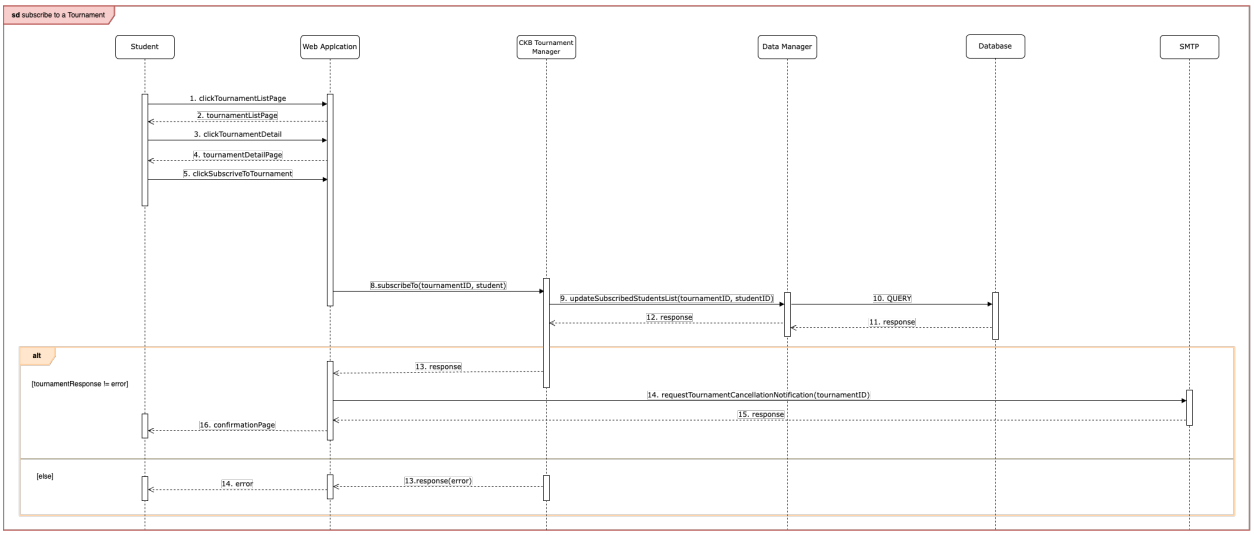


Figure 2.11: Use case 9: "Subscribe to a Tournament" - Sequence Diagram

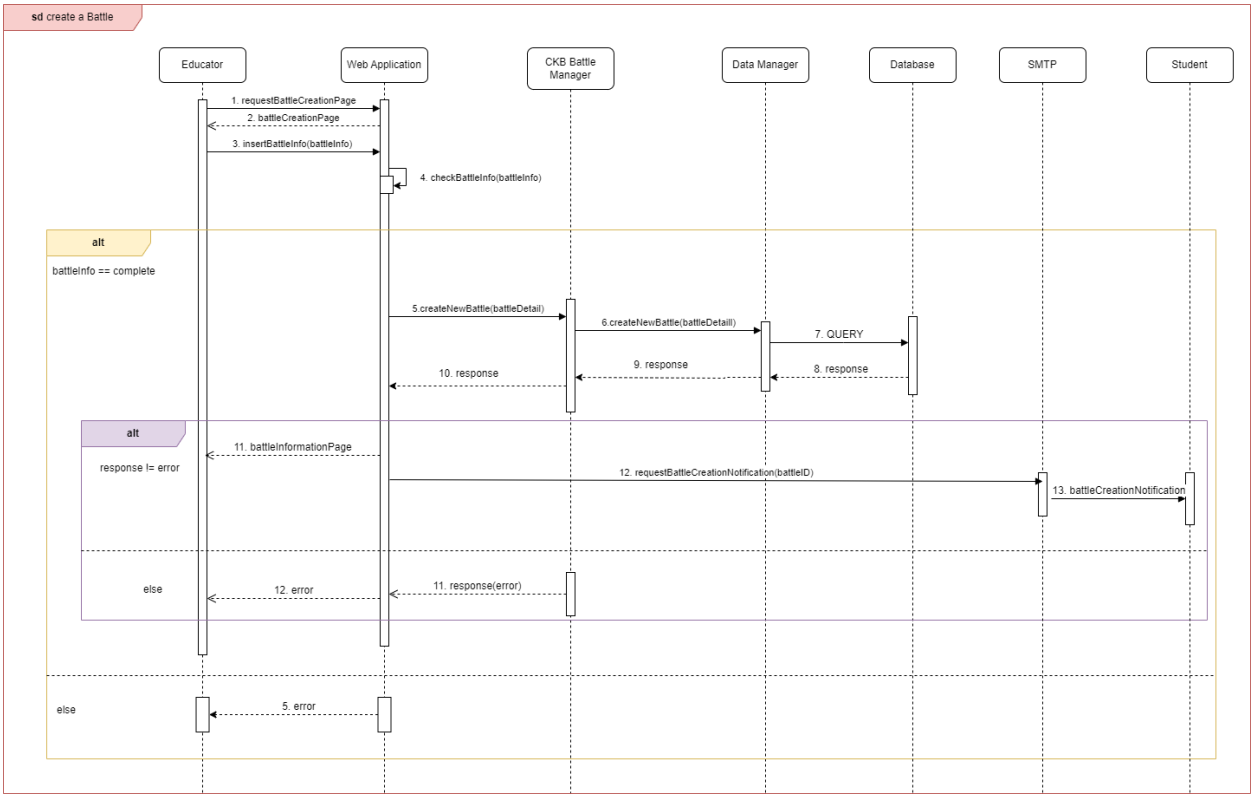


Figure 2.12: Use case 10: "Create a Battle" - Sequence Diagram

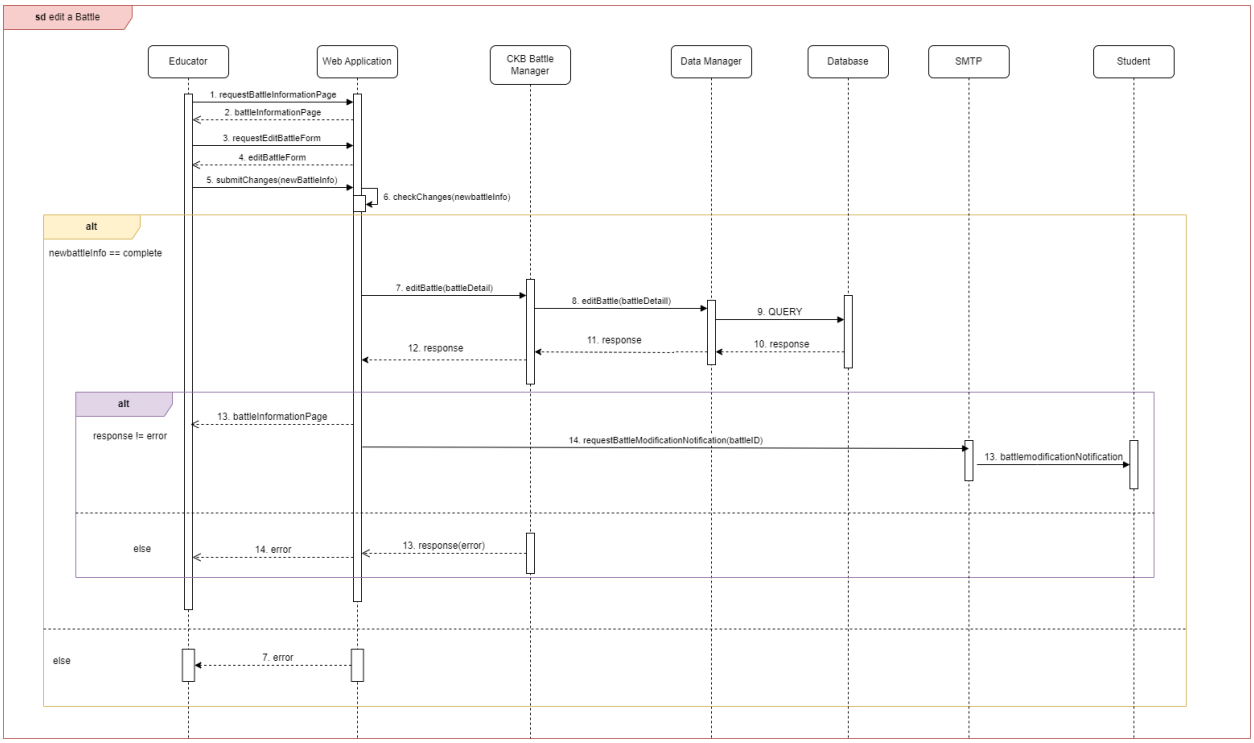


Figure 2.13: Use case 11: "Edit a Battle" - Sequence Diagram

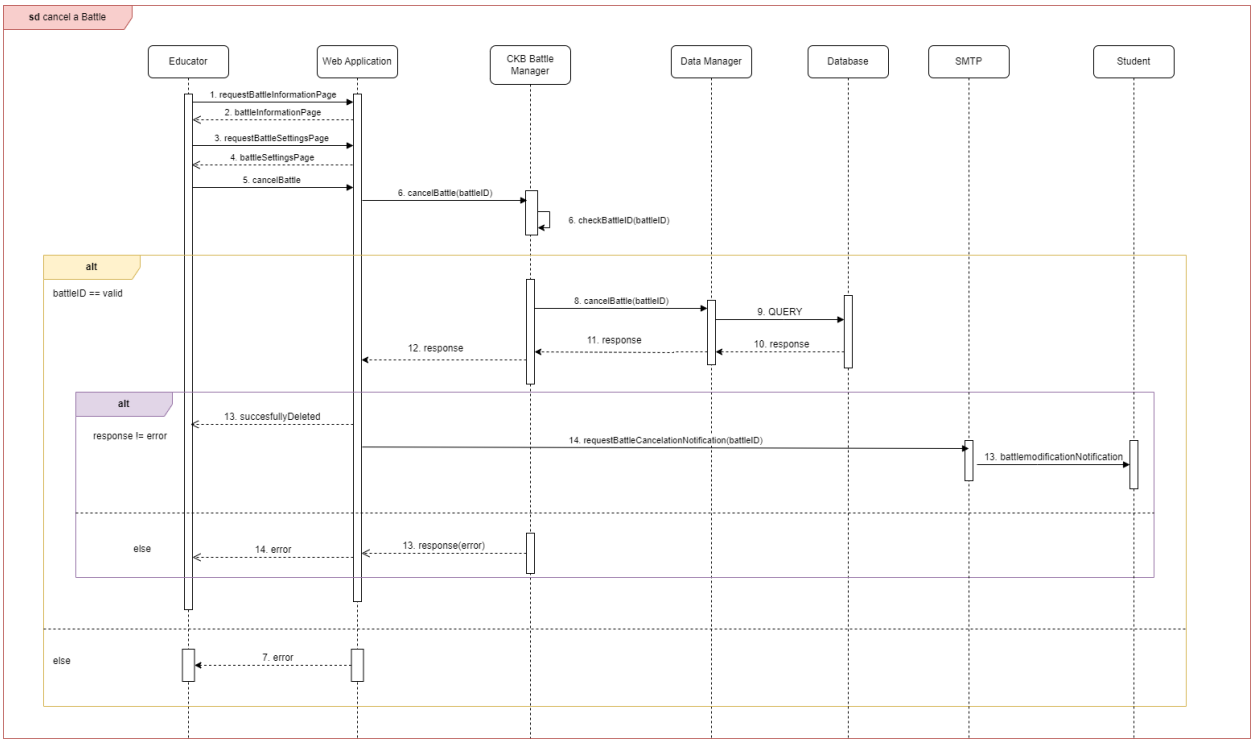


Figure 2.14: Use case 12: "Cancel a Battle" - Sequence Diagram

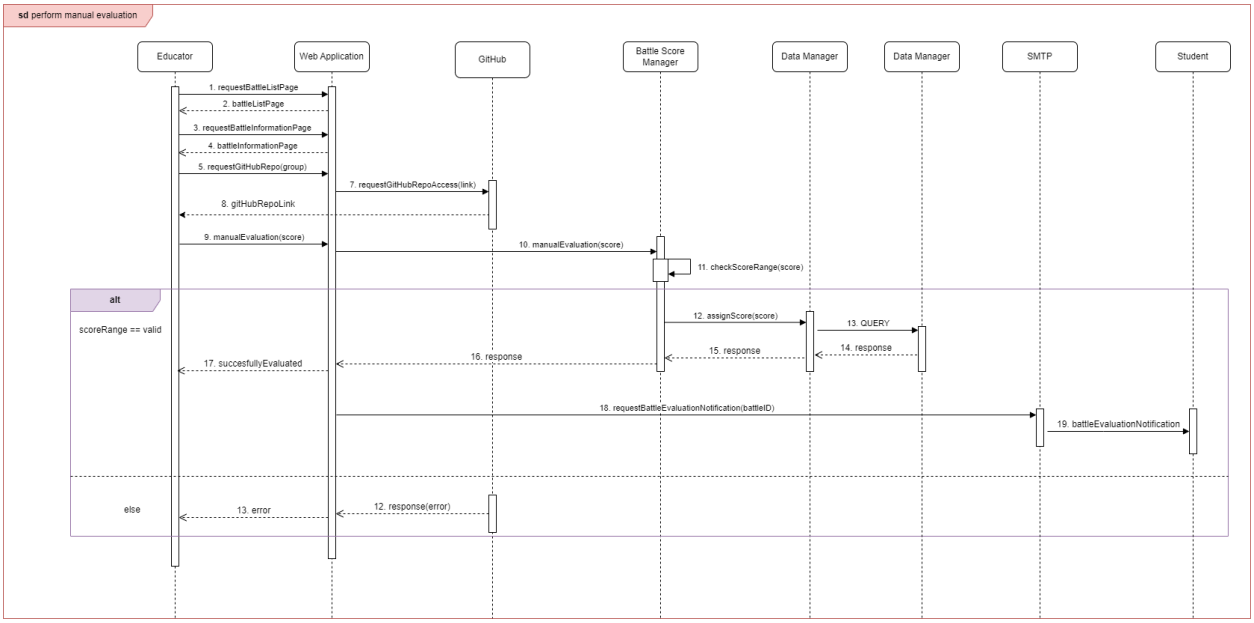


Figure 2.15: Use case 13: "Perform manual evaluation" - Sequence Diagram

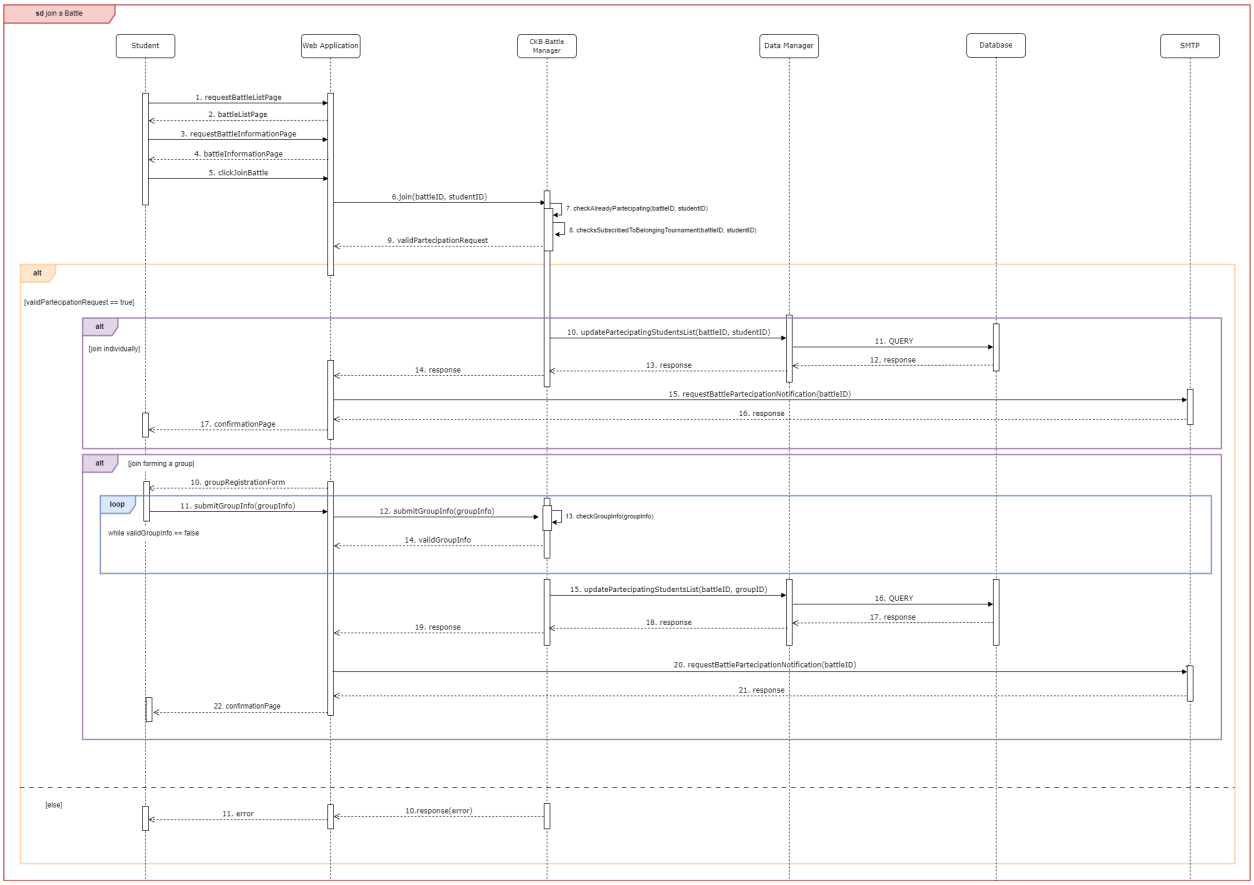


Figure 2.16: Use case 14: "Join a Battle" - Sequence Diagram

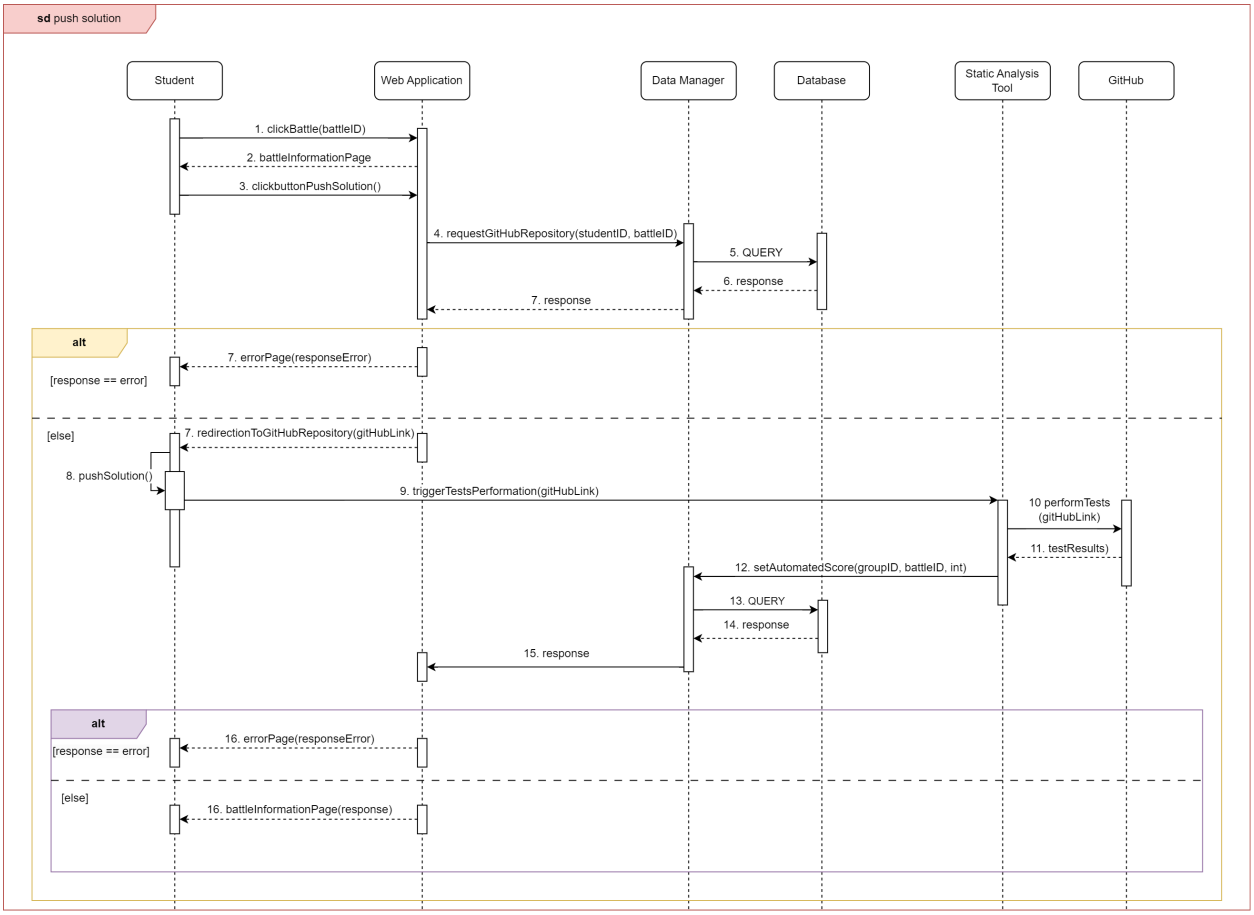


Figure 2.17: Use case 15: "Push solution" - Sequence Diagram

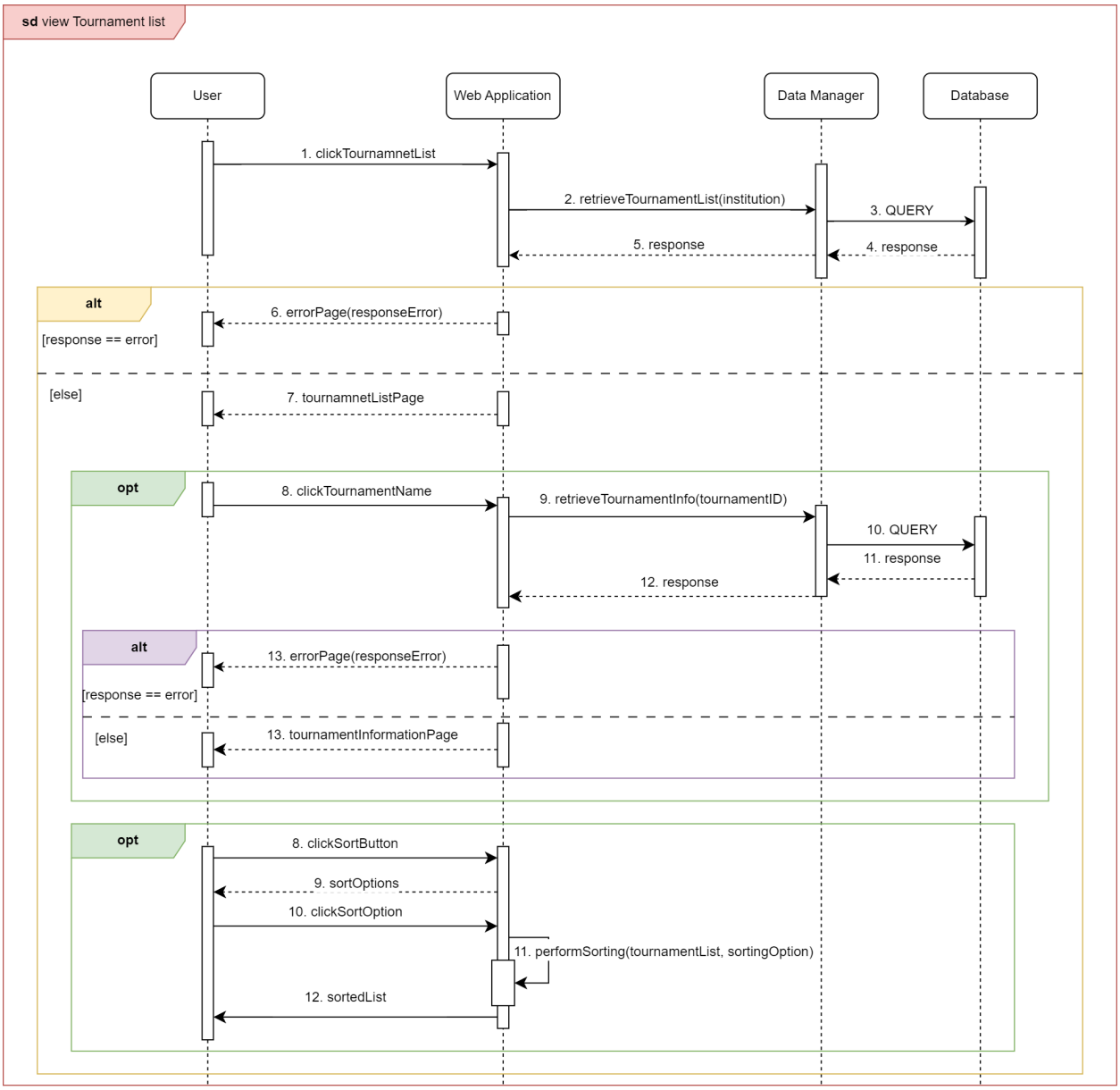


Figure 2.18: Use case 16: "View Tournament list" - Sequence Diagram

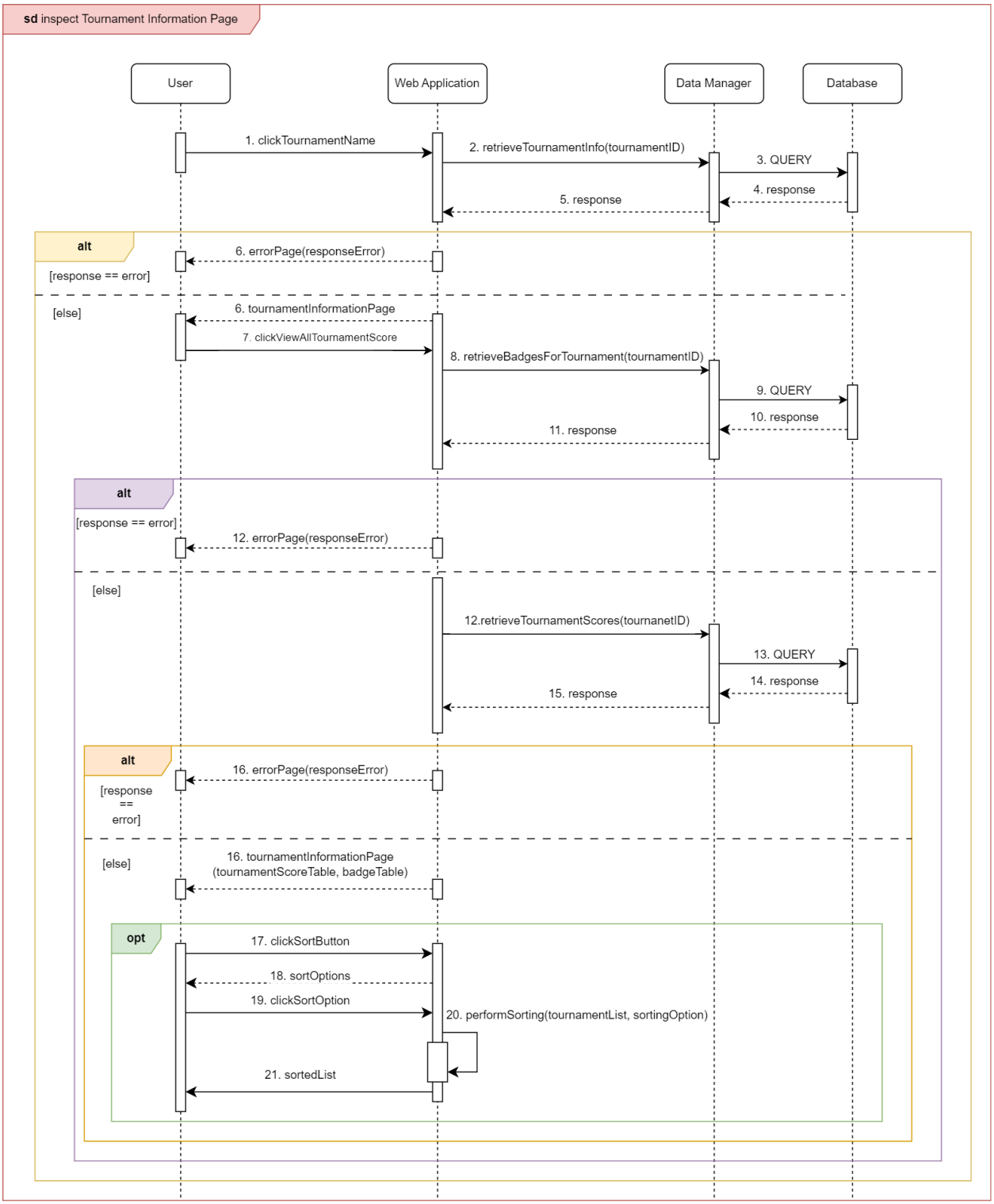


Figure 2.19: Use case 17: "Inspect Tournament Information Page" - Sequence Diagram

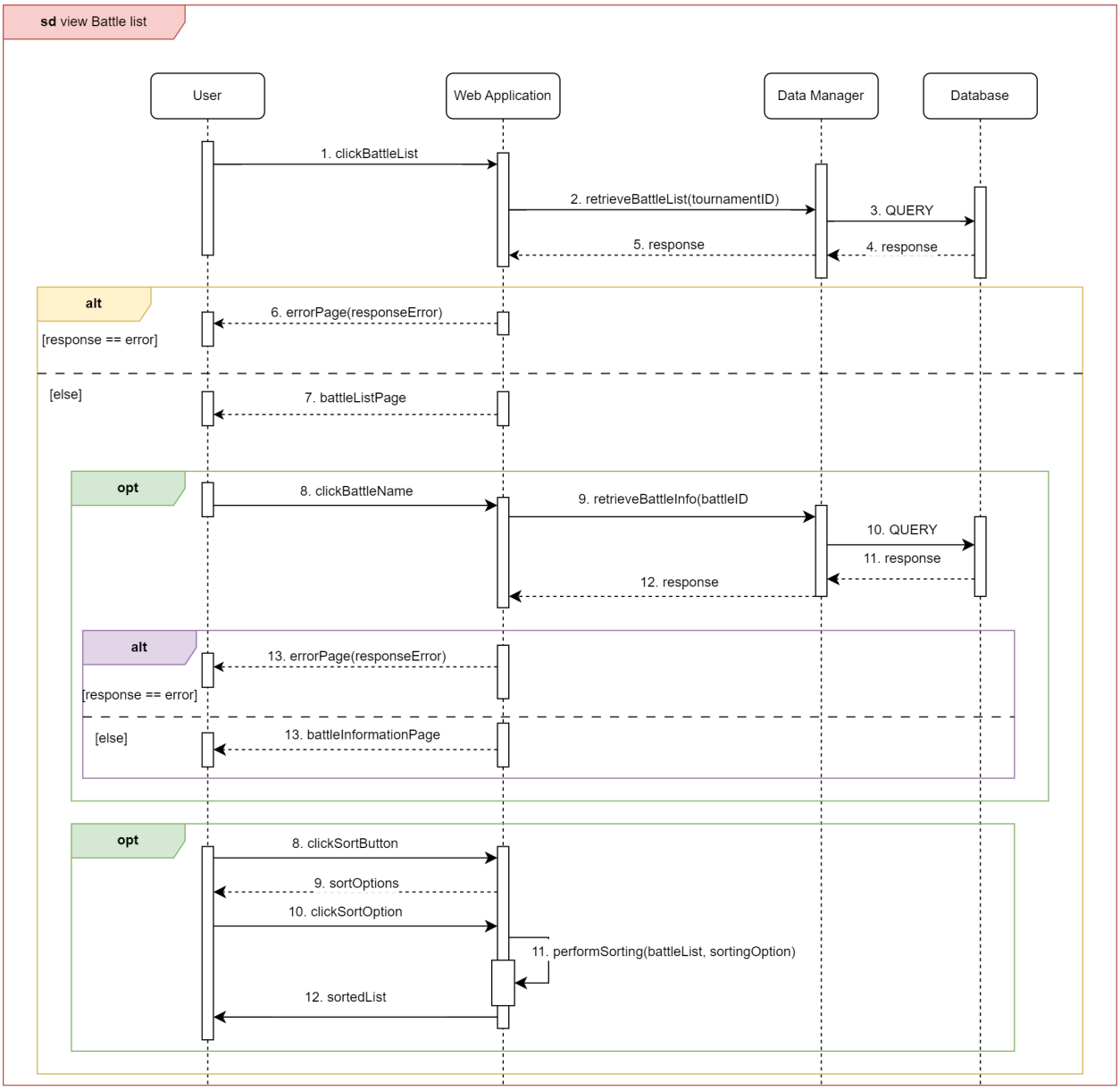


Figure 2.20: Use case 18: "view Battle list" - Sequence Diagram

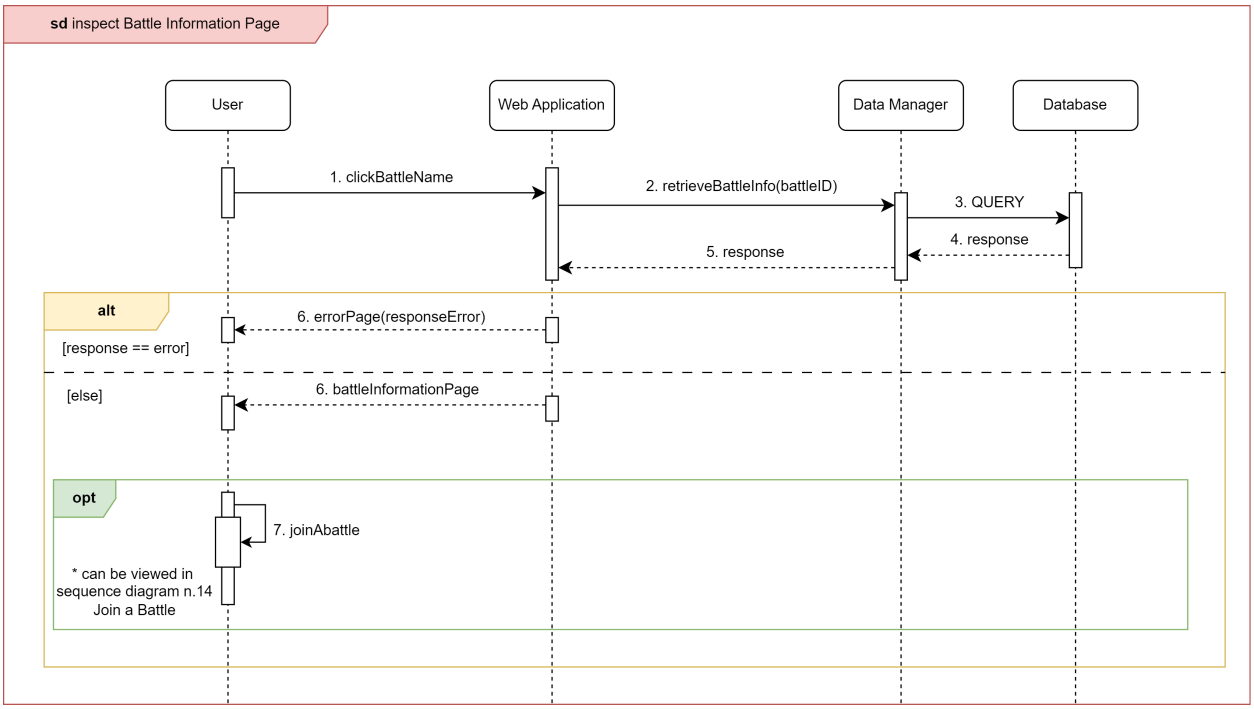


Figure 2.21: Use case 19: "Inspect Battle Information page" - Sequence Diagram

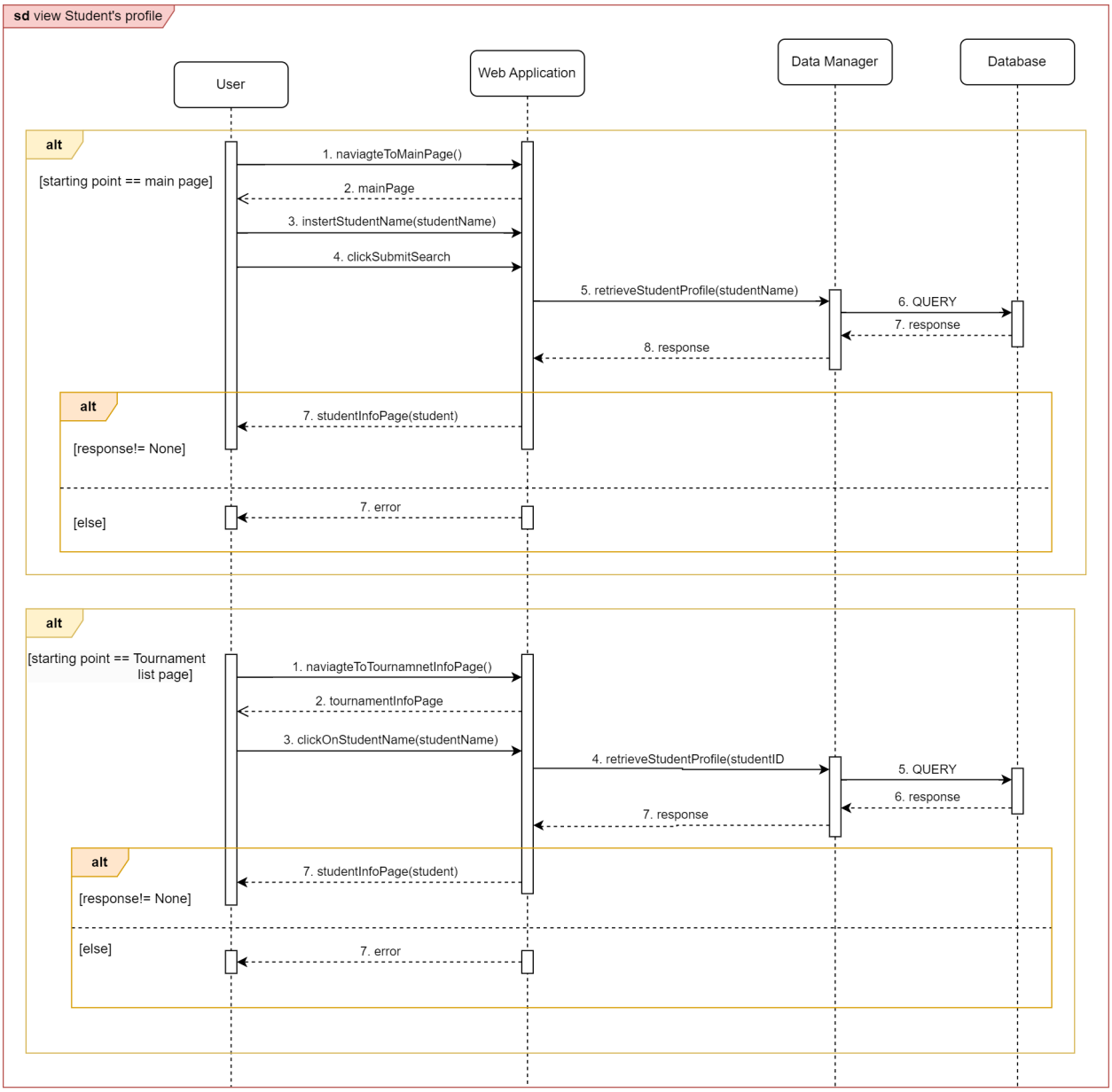


Figure 2.22: Use case 20: "View Student's profile" - Sequence Diagram

2.5 Component interfaces

This section provides a high-level description of the methods offered by the interfaces that the various system components expose.

Account Manager Interfaces: the Account Manager component exposes a single interface used by the web and mobile applications. Its main functions are the following:

- String: createStudentAccount(personalData) - a request to create a new Student account with the specified personal data. It returns Platform ID created for just registered Student.
- String: createEducatorAccount(personalData) - a request to create a new Educator account with the specified personal data. It returns Platform ID created for just registered Educator.
- Boolean: authenticateStudent(identifier, password) - it authenticates the specified credentials of a Student. It returns Boolean value, if the authentication was successful True, otherwise False.
- Boolean: authenticateEducator(identifier, password) - it authenticates the specified credentials of an Educator. It returns Boolean value, if the authentication was successful True, otherwise False.
- Boolean: requestMailSend(platformID) - it triggers the SMTP provider to send a verification e-mail. It returns Boolean value, if the e-mail sending was successful True, otherwise False.

Data Manager Interfaces: the Data Manager component exposes a single interface used by all the other application server components to interact with the database. Its main functions are the following:

- Boolean: checkRegistrationEmail(email) - it verifies that the specified e-mail is not already registered. It returns Boolean value, if the e-mail is not registered True, otherwise False.
- Boolean: saveStudentRegistrationData(personalData) - it saves the registration data of a new Student account. Return value is Boolean, if no error occurred True, otherwise False.
- Boolean: saveEducatorRegistrationData(personalData) - it saves the registration data of a new Educator account. Return value is Boolean, if no error occurred True, otherwise False.
- Boolean; checkStudentCredentials(email, password) - it checks the credentials submitted by the Student. Return value is Boolean, if no data is correct True, otherwise False.

- Boolean; checkStudentCredentials(studentID, password) - it checks the credentials submitted by the Student. Return value is Boolean, if no data is correct True, otherwise False.

- Boolean: checkEducatorCredentials(email, password) - it checks the credentials submitted by the Educator. Return value is Boolean, if no data is correct True, otherwise False.

- Boolean: checkEducatorCredentials(educatorID, password) - it checks the credentials submitted by the Educator. Return value is Boolean, if no data is correct True, otherwise False.

- Boolean: createStudentProfile(studentData) - it creates a Student's profile with the submitted data. Return value is Boolean, if no error occurred True, otherwise False.

- Student: retrieveStudentProfile(studentName) - it returns the Student's profile corresponding to the given Student's name. Return type Student, class containing all the information about retrieved student.

- Student: retrieveStudentProfile(studentID) - it returns the Student's profile corresponding to the given Student's ID. Return type Student, class containing all the information about retrieved student.

- List¡Educator¿: retrieveEducatorList(tournament) - it returns the list of Educators within a Tournament.

- List<Tournament> : retrieveTournamentList(institution) - it returns the Tournaments created by Educators working in the given institution.

- Tournament: createTournament(tournamentDetail) - it creates a Tournament with the specified details.

- Tournament: editTournament(newTournamentDetail) - it modifies the details of an already existing Tournament. It returns the modified Tournament.

- Tournament: updateCoEducatorList(tournamentID, educatorID) - it updates the Co-Educator list in the given Tournament. Returns updated Tournament.

- Tournament: setOpenSubscriptionStatus(tournamentID) - it sets the specified Tournament's status to "Open Subscription". Returns updated Tournament.

- Tournament: setOngoingStatus(tournamentID) - it sets the specified Tournament's status to "Ongoing". Returns updated Tournament.

- Tournament: setFinishedStatus(tournamentID) - it sets the specified Tournament's status to "Finished". Returns updated Tournament.

- Tournament: cancelTournament(tournamentID) - it deletes the specified Tournament. Returns updated Tournament.

- Tournament: setCanceledStatus(tournamentID) - set the specified Tournament's status to "Canceled". Returns updated Tournament.
- Tournament:retrieveTournamentInfo(tournamentID) - it returns the Tournament's info. Returns object of type Tournament.
- List<Student>: retrieveSubscribedStudentsList(tournamentID) - it returns the subscribed Students to the given Tournament.
- List<Student>: updateSubscribedStudentsList(tournamentID, studentID) - it updates the subscribed Student list with the given Student.
- Badge: createBadge(badgeDetails) - it creates a Badge with the specified details. Returns object of type Badge.
- Badge: retrieveBadgetInfo(badgeID) - it returns the Badge's info. Returns object of type Badge.
- Map<Badge, Student>:retrieveBadgesForTournament(tournamentID) - it returns a map containing Badges associated to Student who claimed the Badge.
- Badge: updateClaimedBadge(studentID) - it updates and returns updated claimed Badge by specified Student.
- List<Battle>: retrieveBattleList(tournamentID) - it returns the Battles associated with the given Tournament.
- Battle: createBattle(battleDetail) - it creates and returns a Battle with the specified details.
- Battle: editBattle(newBattleDetail) - it modifies the details of an already existing Battle and returns updated Battle.
- Battle: setOpenSubscriptionStatus(battleID) - set the specified Battle's status to "Open Subscription". Returns updated Battle.
- Battle: setOngoingStatus(battleID) - it sets the specified Battle's status to "Ongoing". Returns updated Battle.
- Battle: setFinishedStatus(battleID) - it sets the specified Battle's status to "Finished". Returns updated Battle.
- Battle: cancelBattle(battleID) - it deletes the specified Battle. Returns updated Battle.
- Battle: setCanceledStatus(battleID) - set the specified Battle's status to "Canceled". Returns updated Battle.
- Battle: setConsolidationStage(battleID) - set the specified Battle's status to "Consolidation Stage". Returns updated Battle.

- Battle: retrieveBattleInfo(battleID) - it returns object of class Battle with all the information.
- Group: createGroup(groupID, battleID) - it creates and returns a new Group registering to the specified Battle.
- Group: updateGroupMembers(studentID, groupID) - it updates the specified Group's members list with the given Student. Returs updated Group.
- List<Group>: retrieveParticipantList(battleID) - it returns the Groups participating to the given Battle.
- Map<Student, Integer>: retrieveTournamentScores(tournanetID) - it returns the Student' scores in the given Tournament.
- Integer: retrieveScore(groupID, battleID) - it returns the Group's score in the given Battle.
- Integer: setAutomatedScore(groupID, battleID, int) - it sets and returns the Group's Battle score to the one provided by the Static Analysis Tool.
- Integer: setManualScore(groupID, battleID, int) - it sets and reutrns the Group's Battle score to the one provided by the Educator.

CKB Tournament Manager

- List<Tournament>: getTournamentList(institution) - it returns the Tournaments created by Educators working in the given institution.
- Tournament: createTournament(tournamentDetail) - it creates and returns a Tournament with the specified details.
- Boolean: creationOfTournament(tournamentID) - it sends to the Students a notification about the creation of a new Tournament. It returns a Boolean value, True if no error occurred, False otherwise.
- Boolean: editTournament(newTournamentDetail) - it forwards the request to modify the details of an already existing Tournament. It returns a Boolean value, True if no edit is possible and valid, False otherwise.
- Boolean: modificationOfTournament(tournamentID) - it sends to the subscribed Students a notification about the modification of a Tournament's detail.It returns a Boolean value, True if no error occurred, False otherwise.
- Boolean: defineNewCoeducator(tournamentID, educatorID) - it defines a new Co-Educator in the given Tournament. It returns a Boolean value, True if no error occurred, False otherwise.

- Boolean: setOpenSubscriptionStatus(tournamentID) - it forwards the request to set the specified Tournament's status to "Open Subscription". It returns a Boolean value, True if no error occurred, False otherwise.

- Boolean: setOngoingStatus(tournamentID) - it forwards the request to set the specified Tournament's status to "Ongoing". It returns a Boolean value, True if no error occurred, False otherwise.

- Boolean: closeTournament(tournamentID) - it closes the specified Tournament. It returns a Boolean value, True if no error occurred, False otherwise.

- Boolean: clositionOfTournament(tournamentID) - it sends to the subscribed Students a notification about the closition of a Tournament. It returns a Boolean value, True if no error occurred, False otherwise.

- Boolean: setFinishedStatus(tournamentID) - it forwards the request to set the specified Tournament's status to "Finished". It returns a Boolean value, True if no error occurred, False otherwise.

- Boolean: cancelTournament(tournamentID) - it forwards the request to delete the specified Tournament. It returns a Boolean value, True if no error occurred, False otherwise.

- Boolean: cancellationOfTournament(tournamentID) - it send a notification about the cancellation of a Tournament to all the subscribed Students. It returns a Boolean value, True if no error occurred, False otherwise.

- Boolean: setCanceledStatus(tournamentID) - it forwards the request to set the specified Tournament's status to "Canceled". It returns a Boolean value, True if no error occurred, False otherwise.

- Tournament: getTournamentInfo(tournamentID) - it returns the object of class Tournament with all the information.

- Boolean: subscribeToTournament(tournamentID, studentID) - it subscribes the specified Student to the given Tournament. It returns a Boolean value, True if no error occurred, False otherwise.

- List<Student>: getSubscribedStudentList(tournamentID) - it forwards the request to return the Subscribed Students to the given Tournament. Return list of Students subscribed.

- Boolean: checkAlreadySubscribed(tournamenTitle, student) it checks if the specified Student is already subscribed to the given Tournament. Returns True if the Students is subscribed, False otherwise.

- Badge: createBadge(badgeDetails) - it forwards the request to create a Badge with the specified details. Returns new Badge.

- Badge: getBadgetInfo(badgeID) - it forwards the request to return the Badge's info. Returns object of type Badge.

Badge Manager

- Badge: createBadge(badgeDetail) - it creates and return a Badge with the specified details.
- Boolean: creationOfBadge(badgeID) - it sends a notification about the creation of a Badge to all the Students subscribed to the Badge's belonging Tournament. Returns a Boolean, if no error occurred True, otherwise False.
- Badge: getBadgetInfo(badgeID) - it returns object of type Badge with all the information.
- Student: assignBadgeTo(studentID, badgeID) - it assigns the given Badge to the specified Student. Returns object of type Student with updated Badges.
- Boolean: updateClaimedBadge(studentID) - it forwards the request to update the specified Student claimed Badges. Returns Boolean, True if no error occurred, False otherwise.

CKB Battle Manager

- List<Battle>: getBattleList(battleID) - it forwards the request to return the Battles associated with the given Tournament.
- Battle: createBattle(battleDetail) - it creates and returns a Battle with the specified details.
- Boolean: creationOfBattle(battleDetail) - it sends a notification about the creation of a new Battle to the Student subscribed to the Battle's belonging Tournament. Returns Boolean, if no error occurred True, otherwise False.
- Boolean: editBattle(newBattleDetail) - it forwards the request to modify the details of an already existing Battle. Returns Boolean, if no error occurred True, otherwise False.
- Boolean: modificationOfBattle: it sends to participating Students a notification about the modification of the Battle's detail.Returns Boolean, if no error occurred True, otherwise False.
- Boolean: setOpenSubscriptionStatus(battleID) - it forwards the request to set the specified Battle's status to "Open Subscription". Returns Boolean, if no error occurred True, otherwise False.
- Boolean: setOngoingStatus(battleID) - it forwards the request to set the specified Battle's status to "Ongoing". Returns Boolean, if no error occurred True, otherwise False.

- Boolean: setFinishedStatus(battleID) - it forwards the request to set the specified Battle's status to "Finished". Returns Boolean, if no error occurred True, otherwise False.
- Boolean: cancelBattle(battleID) - it forwards the request to delete the specified Battle. Returns Boolean, if no error occurred True, otherwise False.
- Boolean: cancellationOfBattle(battleID) - it sends a notification about the cancellation of a Battle to all the participating Students. Returns Boolean, if no error occurred True, otherwise False.
- Boolean: setCanceledStatus(battleID) - it forwards the request to set the specified Battle's status to "Canceled". Returns Boolean, if no error occurred True, otherwise False.
- Boolean: setConsolidationStage(battleID) - it forwards the request to set the specified Battle's status to "Consolidation Stage". Returns Boolean, if no error occurred True, otherwise False.
- Battle: getBattleInfo(battleID) - it returns object of class Battle with all the information.
- Group: createGroup(studentID, groupID, battleID) - it creates and returns a Group participating to the given Battle with the specified Student as member.
- Group: joinGroup(studentID, groupID) - it updates the specified Group's member list with the given Student. Returns updated Group.
- List<Group>: getParticipantList(battleID) - it forwards the request to return the Groups participating to the given Battle.
- Integer: getScore(groupID, battleID) - it returns the Group's score in the given Battle.
- Boolean: setManualScore(groupID, battleID, int) - it forwards the request to set the Group's Battle score to the one provided by the Educator. Returns Boolean, if no error occurred True, otherwise False.

Battle Score Manager

- Integer: setAutomatedScore(groupID, battleID, int) - it sets and returns the Group's Battle score to the one provided by the Static Analysis Tool.
- Integer: setManualScore(groupID, battleID, int) - it sets and returns the Group's Battle score to the one provided by the Educator.
- Boolean: checkScoreInValidRange(groupID, battleID, int) - it checks that the specified Group has a valid score. If the score is valid it returns True, otherwise False.

Static Analysis Tool

- Integer: requestAutomatedScore(groupID) - it allows the Tool the perform the code's analysis of the given groups and returns the Integer value.

SMTP

- sendTournamentCreationNotification(studentID) - it sends the Tournament creation notification to the Students.
- sendTournamentModificationNotification(studentID) - it sends the Tournament modification notification to the Students.
- sendNewCoEducatorNotification(educatorID) - it sends the new Co-Educator notification to the Educators.
- sendTournamentCancelationNotification(studentID) - it sends the Tournament cancelation notification to the Students.
- sendTournamentClositionNotification(studentID) - it sends the Tournament closition notification to the Students.
- sendTournamentSubscriptionNotification(studentID) - it sends the Tournament subscription notification to the Students.
- sendBattleCreationNotification(studentID) - it sends the Battle creation notification to the Students.
- sendBattleModificationNotification(studentID) - it sends the Battle modification notification to the Students.
- sendBattleCancelationNotification(studentID) - it sends the Battle cancelation notification to the Students.
- sendBattleEvaluationNotification(studentID) - it sends the Battle evaluation notification to the Students.
- sendBattlePartecipationNotification(studentID) - it sends the Battle partecipation notification to the Students.

GitHub

All the interactions with GitHub, such as creating repositories, retrieving the repositories links, and others, are handled through the GitHub Actions.

2.6 Selected architectural styles and patterns

The client-server three tier architecture adopted for our CKB Platform enhances maintainability, flexibility, and scalability. Each layer in the model serves a distinct purpose, allowing us to upgrade and improve individual components independently. In our CKB System, the presentation layer is responsible for displaying problem statements, receiving user input, and showcasing results. The business layer contains the logic needed to perform all the actions inside the Platform. The data layer manages any state or data relevant to the System. This three-tier model facilitates seamless integration of both internal services and third-party tools. It ensures that improvements in one layer don't disrupt the functionality of others, creating a robust and adaptable system. Upgrading the algorithmic logic in the business layer, for instance, doesn't impact the user interface or the data management layer.

Moreover, the chosen architecture expedites the implementation process: different teams can concurrently work on distinct tiers, encouraging parallel development and accelerating the overall progress of the CKB System. This not only enhances collaboration but also contributes to the agility and efficiency of the development process. In summary, the three-tier architecture provides a solid foundation for our CKB Platform, offering not only modular design and independence of components but also enabling rapid development and integration of various services. It ensures that our application remains maintainable, flexible, and scalable, satisfying needs of both users and developers."

2.7 Other design decisions

Monitoring: The System implements a robust monitoring solution to track the performance, health, and availability of components. A dedicated monitoring tool is utilized to collect metrics and generate alerts in case of anomalies or performance degradation. The monitoring system focuses on key metrics, including server resource utilization, response times, error rates, and network traffic. These metrics will be continuously monitored to ensure optimal system performance. Comprehensive logging mechanisms are implemented across all system components. This includes logging important events, errors, and user interactions. Log data are stored centrally for analysis and troubleshooting purposes.

Password storing and User authentication: login to the CKB Platform requires a personal identifier (either the registration email or the Platform ID) and a password. For each User, the passwords are first hashed using SHA512 algorithm with adding salt and pepper. The hash and salt are stored in the Database. No passwords are stored as clear text in the Database.

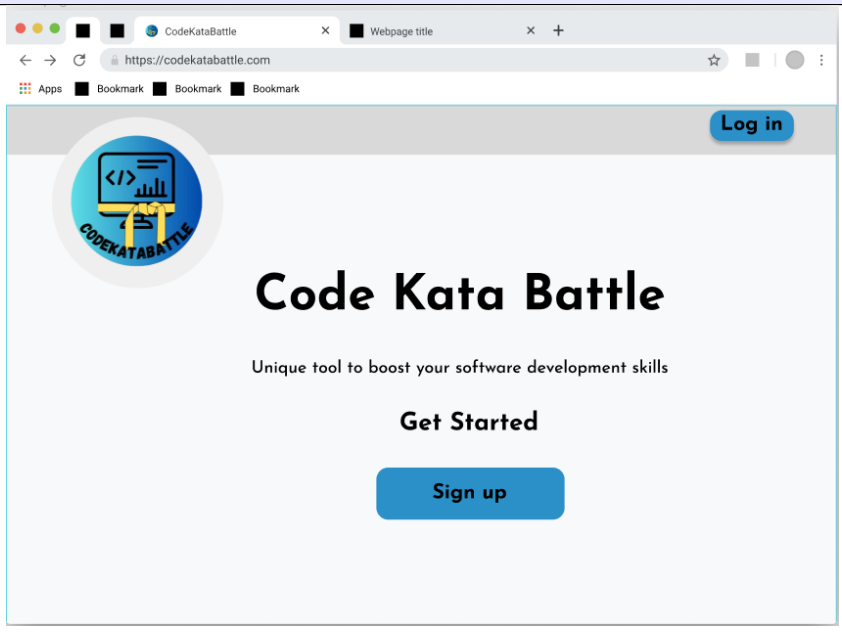
Relational Database: relational database management system (DBMS) will be used to manage the system's data. The following characteristics are guaranteed: atomicity, consistency, isolation, and durability. No partial execution is allowed, every state is consistent after a transaction execution and each transaction is isolated. Even in case of failures, changes in the database persist.

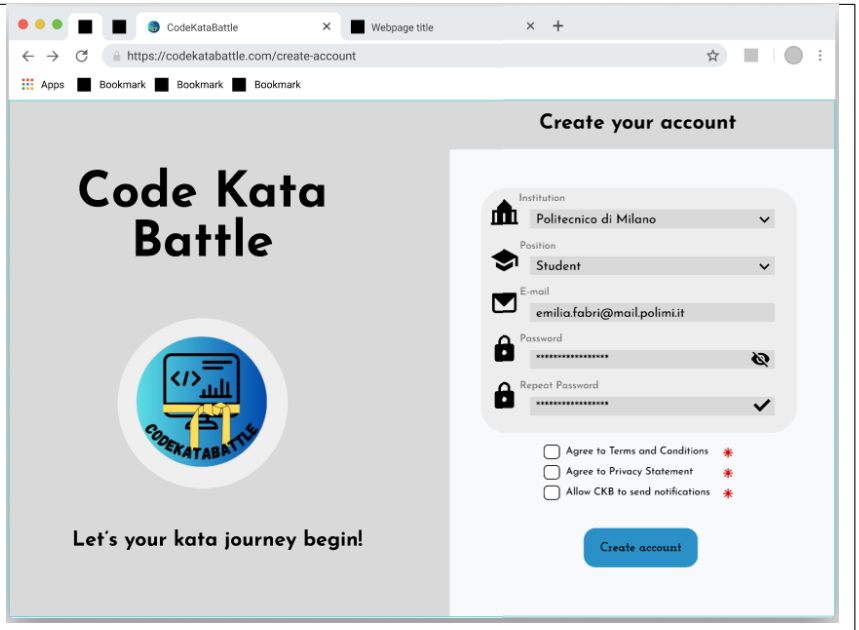
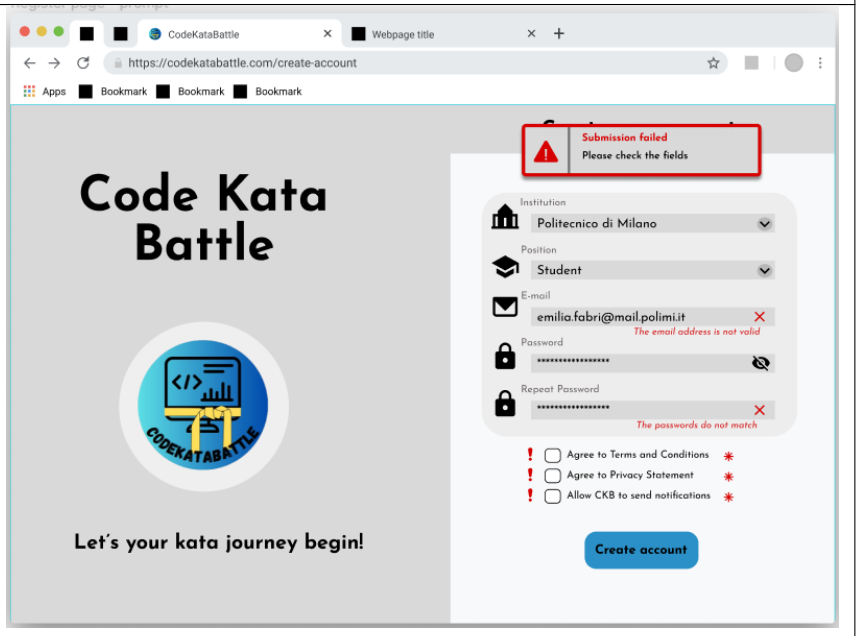
Backup Strategy: A robust backup strategy is implemented to safeguard critical data. Regular automated backups of the relational database will be performed to prevent data loss in the event of system failures or unpredictable issues. Backups are scheduled in regular intervals, considering the frequency of data updates and the criticality of the information. Backup files are encrypted to enhance security during both storage and transmission. Access controls are enforced to restrict unauthorized access to backup data.

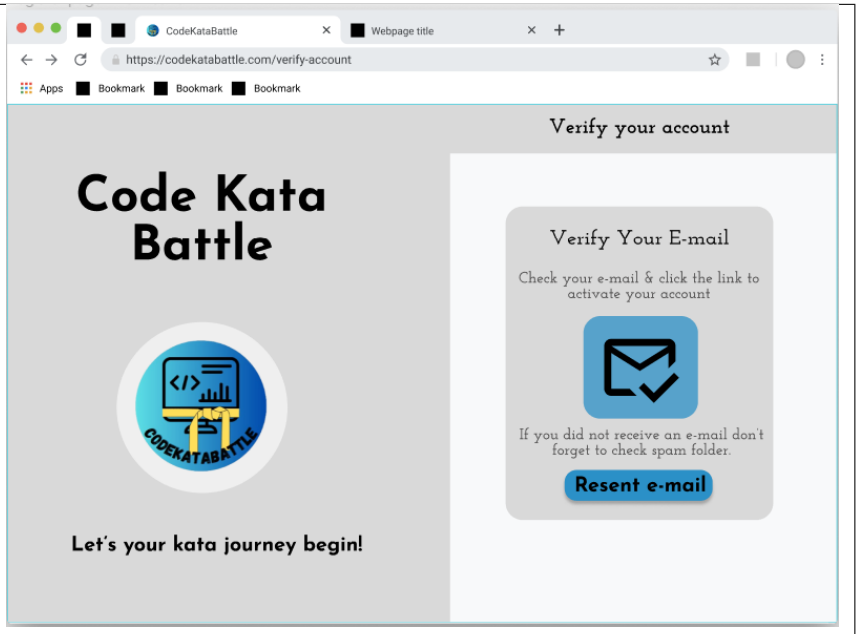
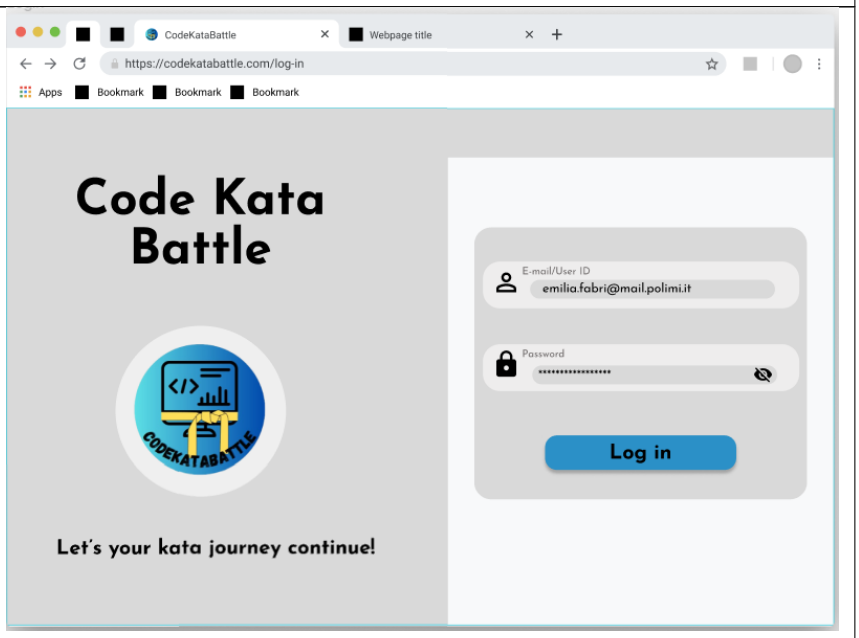
Chapter 3

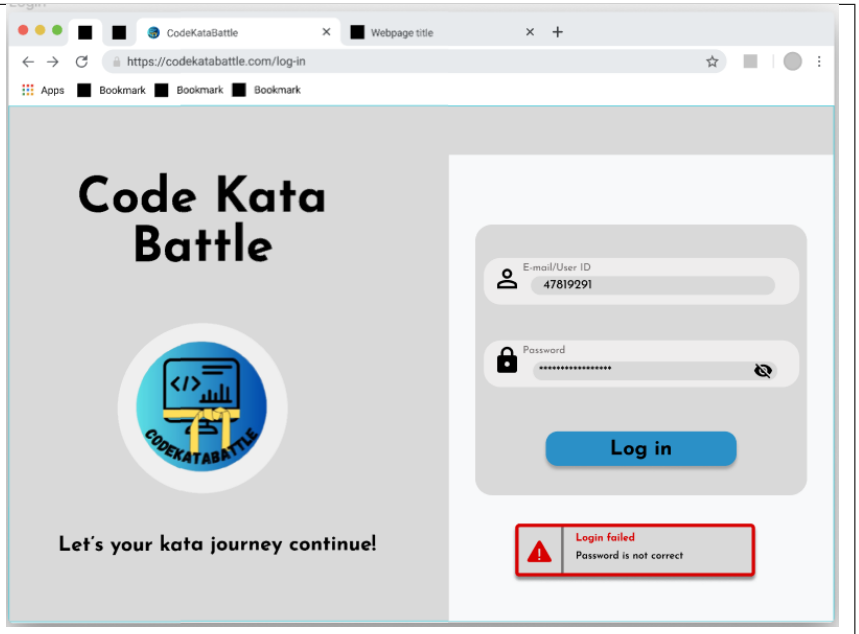
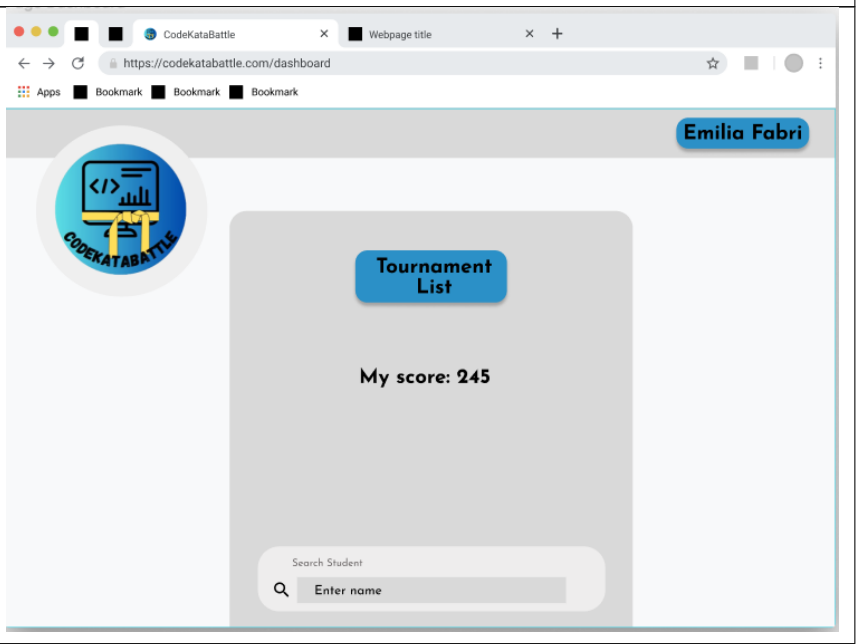
User Interface design

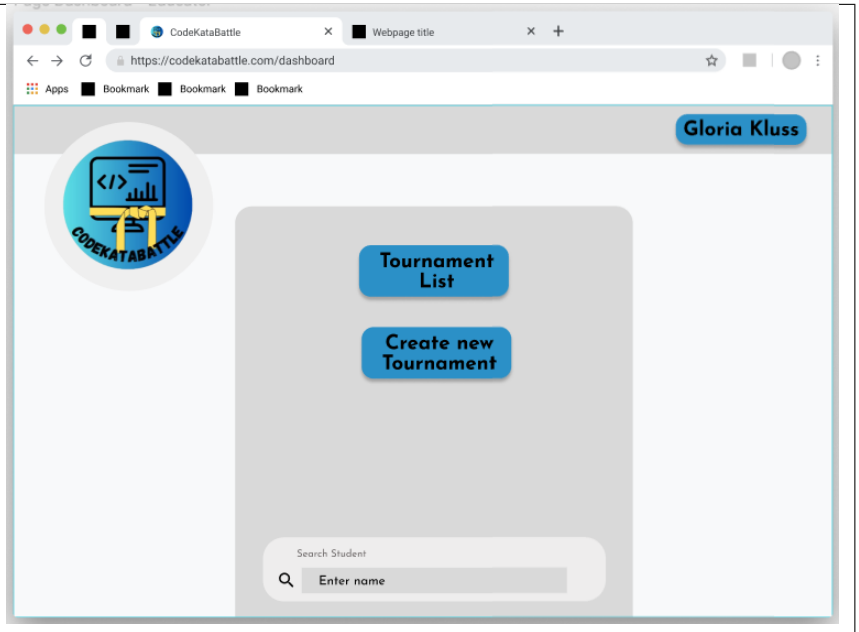
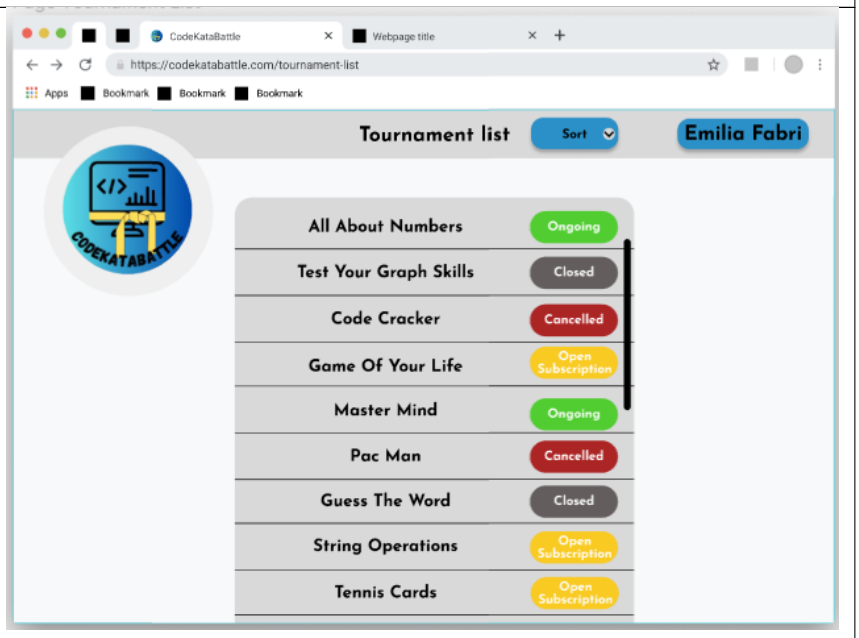
In this chapter we introduce the CKB platform design.

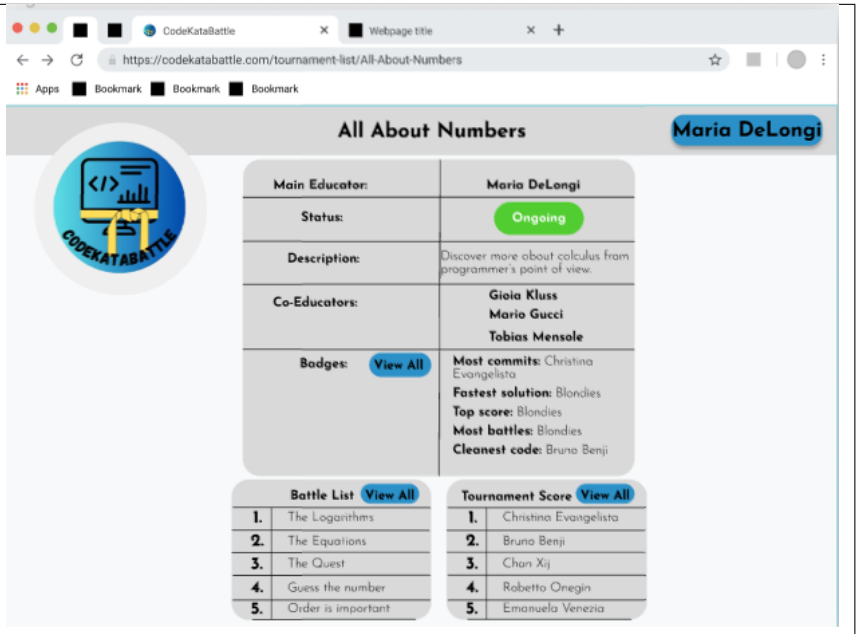
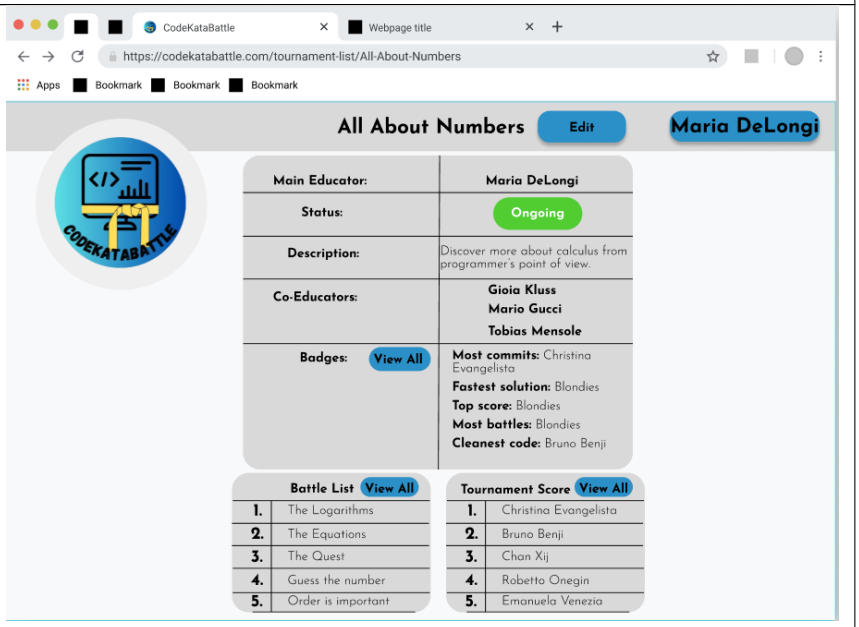
Interface name & Description	Design
Home: the landing page of the CKB Platform. It allows the Guest to create a new account and the Users to Log in with their credentials.	

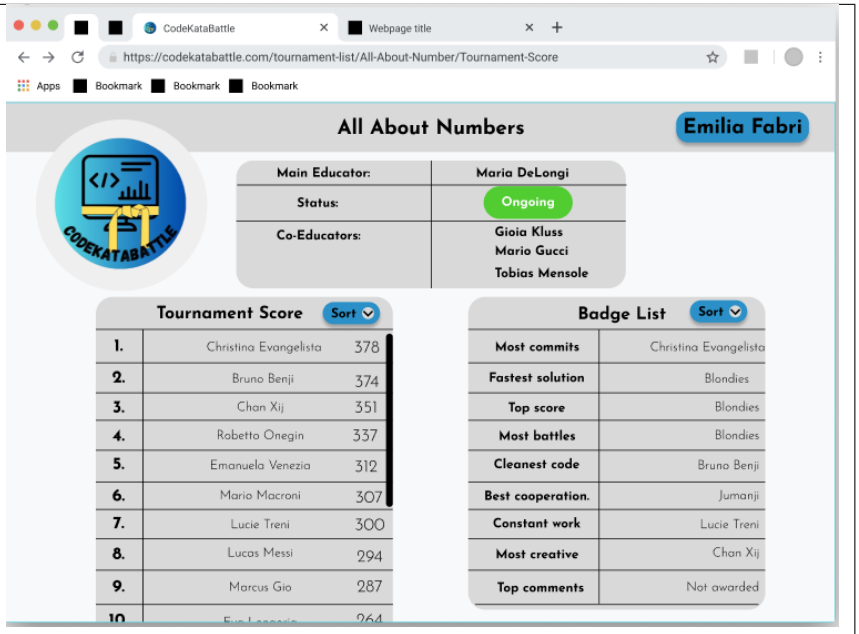
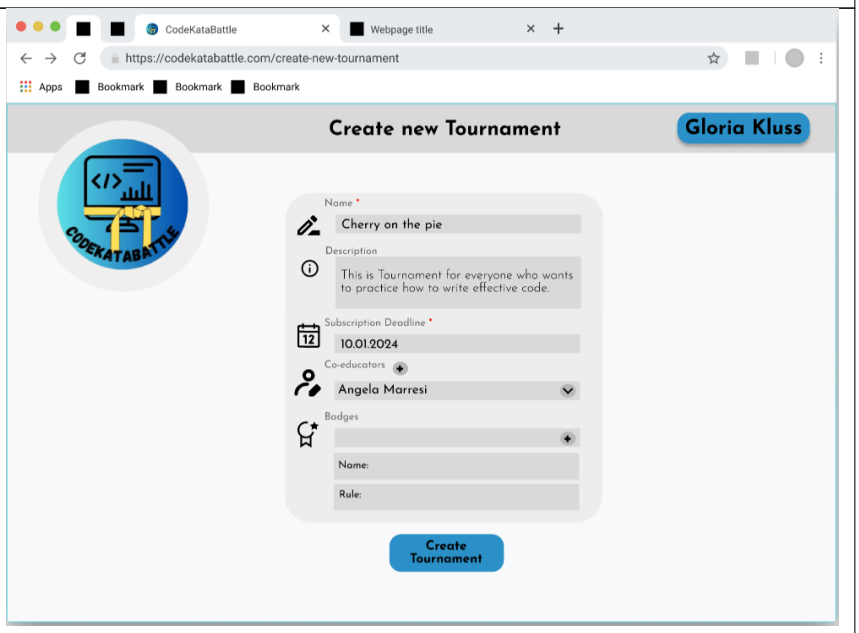
Register: after choosing Sign up from the landing page, the Guests are redirected to page where they can create their account. It allows them to choose an institution, which is using CKB platform, and their position, student or educator. All the fields are mandatory.	
Register - error shown: when a Guest tries to Create an account with incorrect data, he is informed about the missing or wrong input by prompts.	

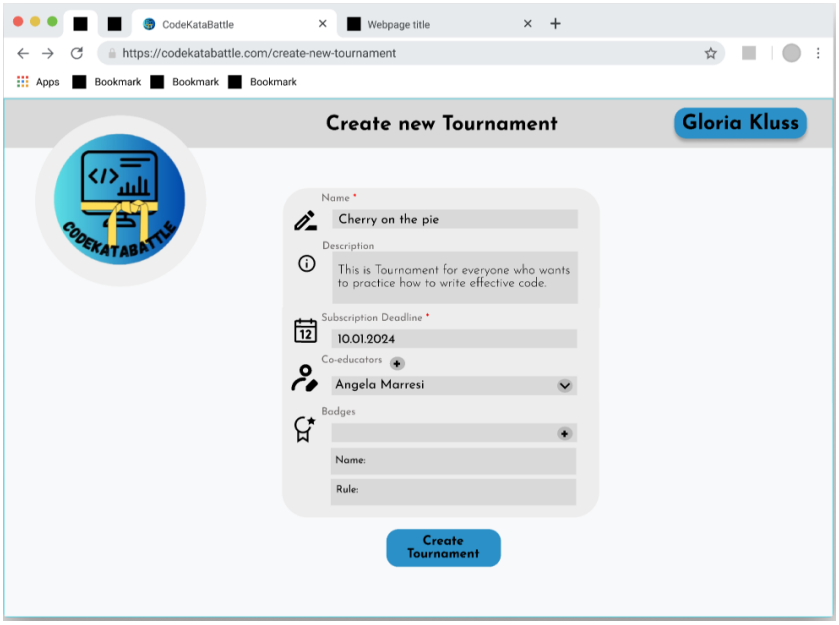
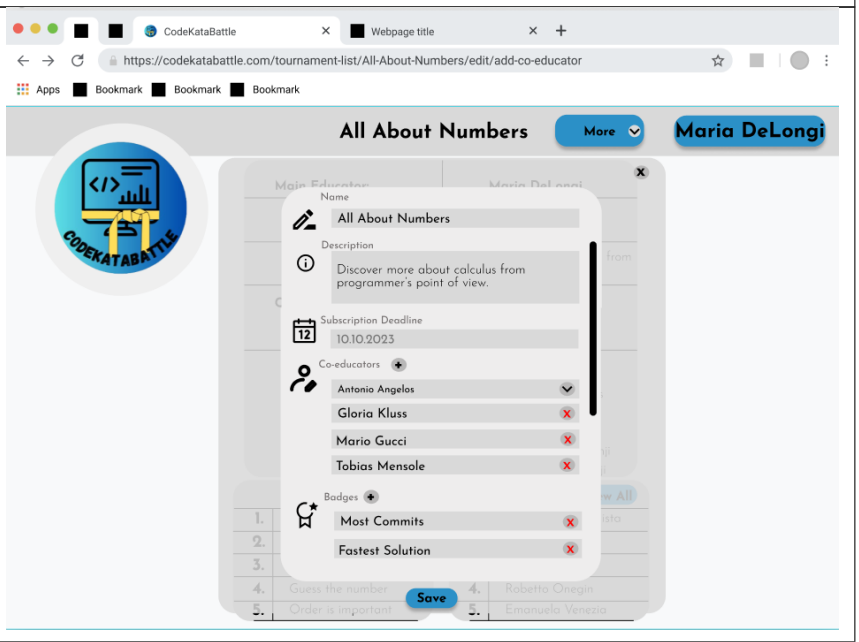
Verify Account: when a Guest filled in the information correctly and pressed Create account, he is redirected to page informing about account verification by email.	
Login: when User pressed Log in button on the Home Page, he is redirected to the Login page, where he can log in either by e-mail or Platform ID, he has received in the verification e-mail.	

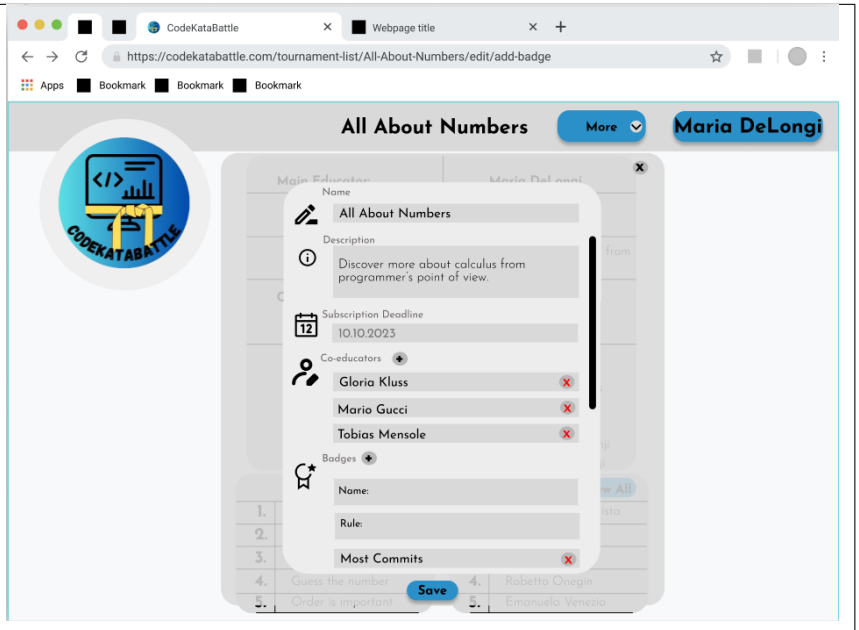
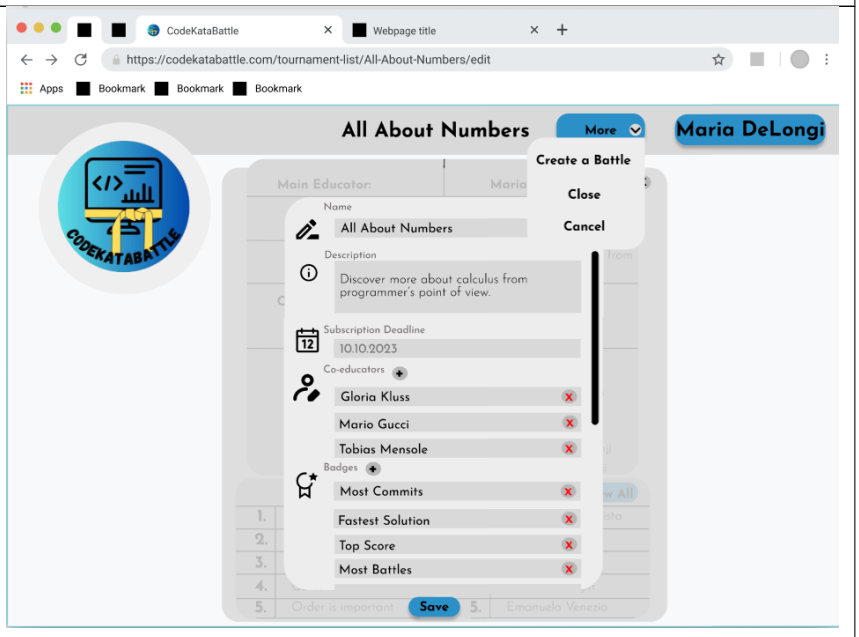
Login - error shown: when User is on Login page, and submitted incorrect password, he is informed about wrongness of the password. Similar prompt is shown, when e-mail/Platform ID is wrong.	
Main Page - Student: when Student has successfully logged in, he is redirected to Main page, when he can see his name, score, search for other Students, or button to be redirected to the Tournament list.	

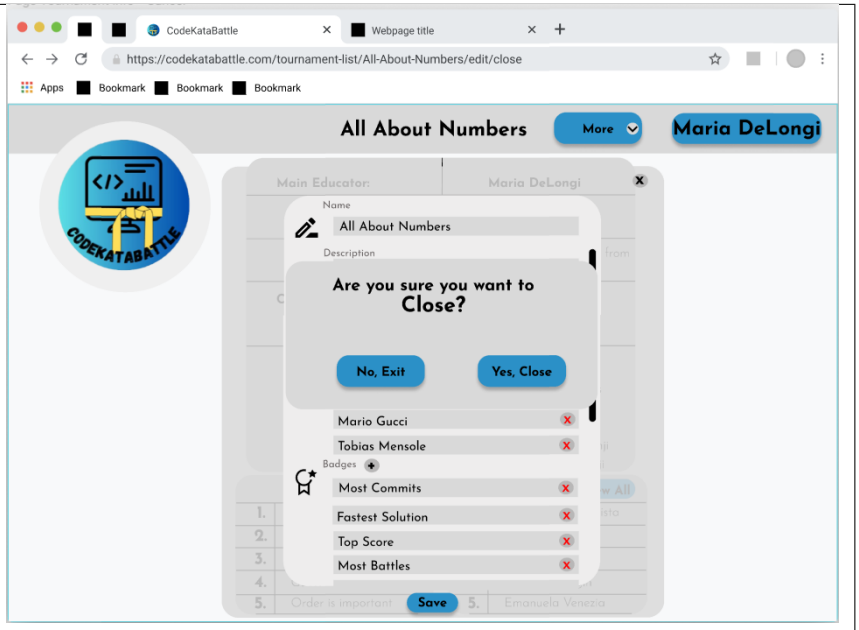
Main Page - Educator: when Educator has successfully logged in, he is redirected to Main page. The content of the page is same as for Students, but the Educator instead of seeing his score, can see a button "Create new Tournament".	
Tournament List: when both Student and Educator selects "Tournament List" button, they are redirected to Tournament List Page. They can use Sort button, to change the order in the list. Each Tournament entry is clickable. Click redirects the User to the Tournament Information page of selected Tournament.	

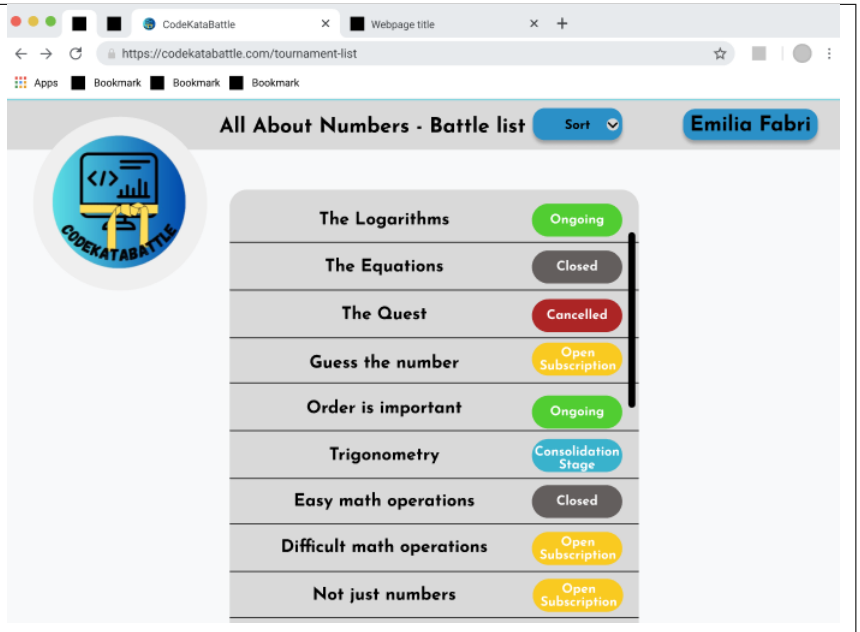
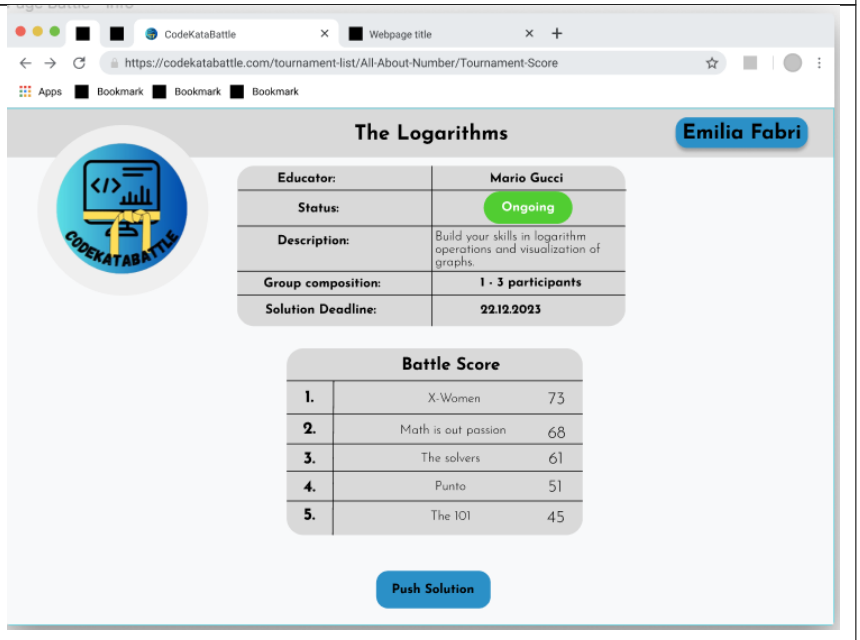
Tournament Information: on this page User can see information about the Tournament. This is view for a Student and an Educator who is not Tournament creator.	
Tournament Information - Creator: view for the Educator who is the Tournament creator. The view has has an extra "Edit" button if the status permits editing.	

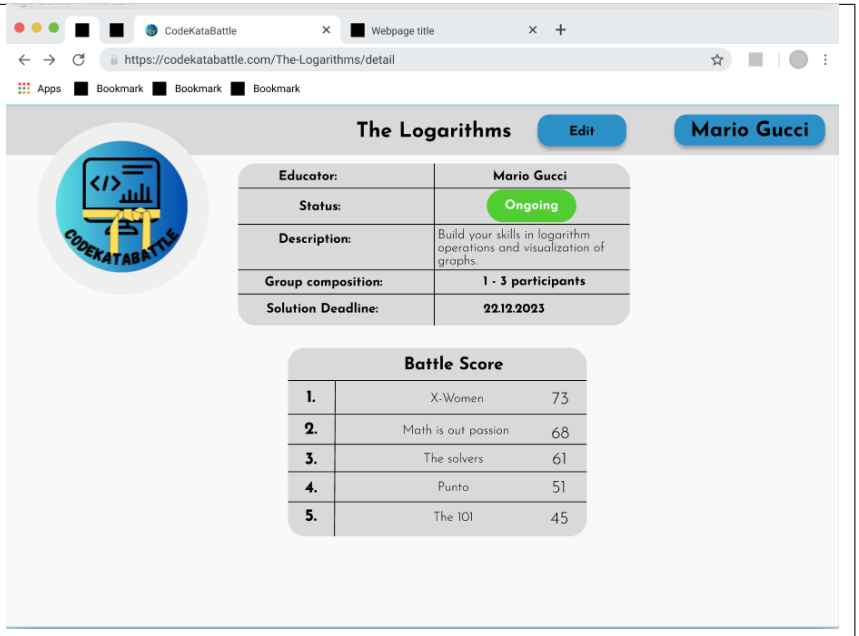
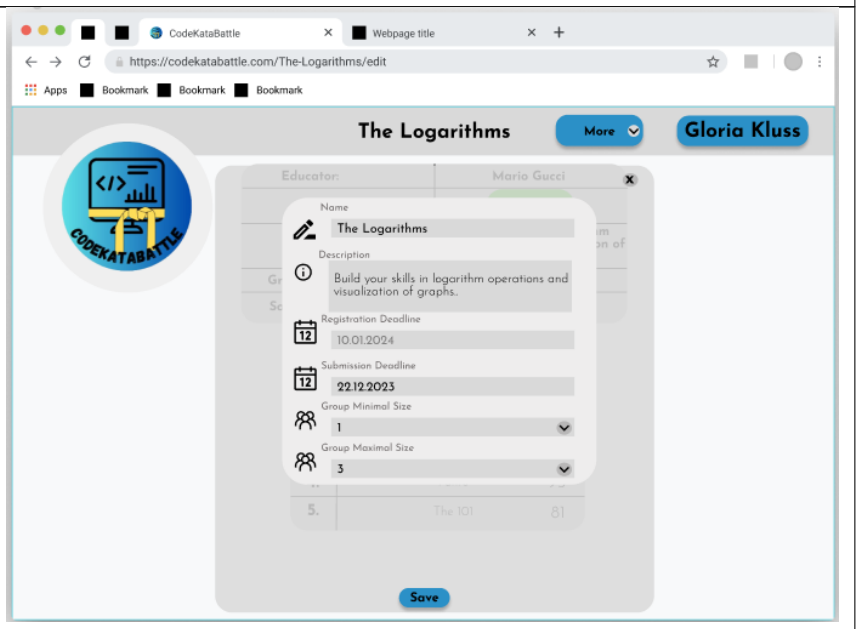
Tournament Score: view for displaying all the Tournament Badges and Tournament score in more detail. User is redirect to this page when he clicks "View all" in the Tournament Score table or Badges.	
Create a Tournament: when Educator clicks "Create new Tournament" from the Dashboard, he is redirected to this page. Tournament name and Subscription Deadline are mandatory fields, others are optional.	

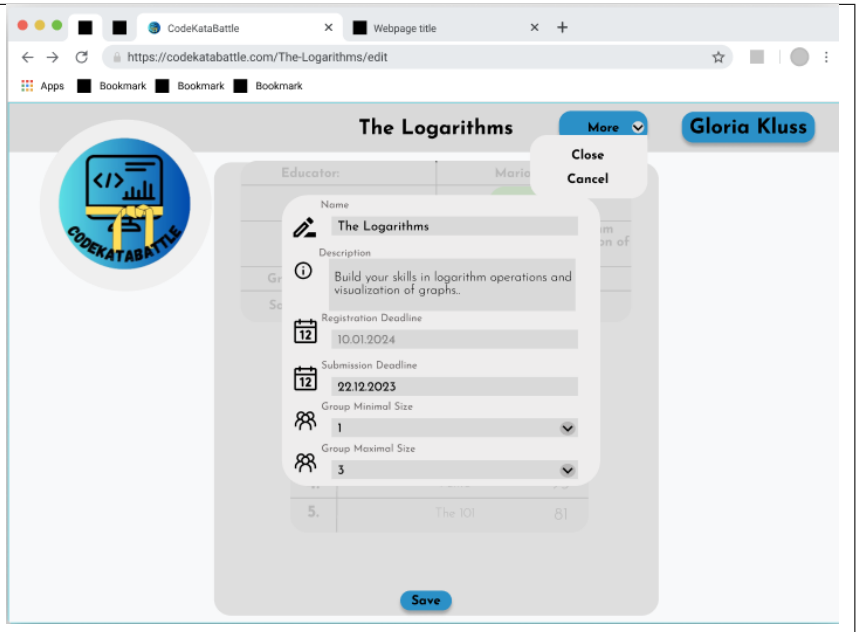
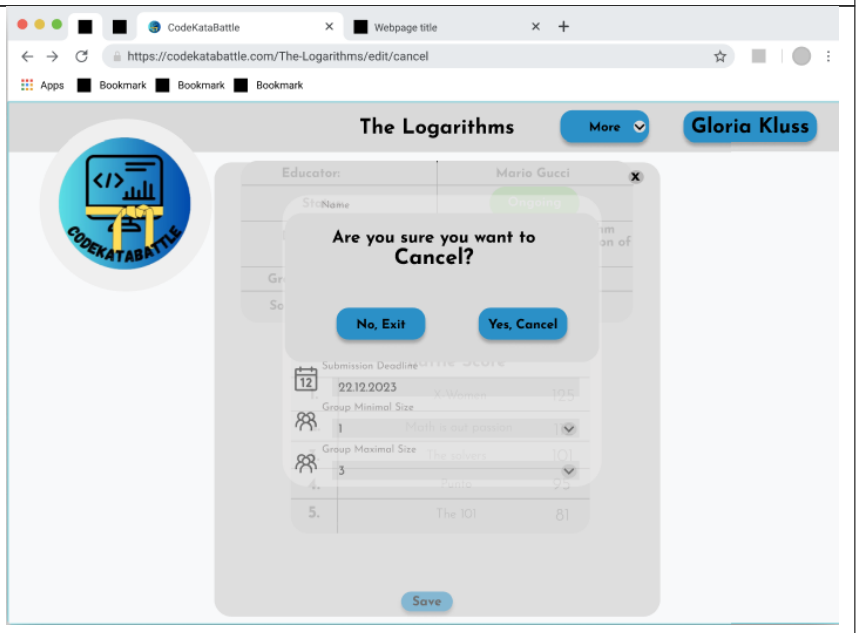
Edit a Tournament: when Tournament Creator clicks "Edit" on the Tournament Information page, he is presented with Edit interface. Tournament's name and description can be modified easily. If the Subscription deadline has already occurred, it cannot be modified. Also the badges, if earned, and Co-educators, if already created a battle, cannot be deleted and should be unclickable.	
Add a Co-educator: when Educator on Edit Tournament Page clicks "+" in the Co-educators section, new field with drop-down appears. Educator is able to choose a new co-educator.	

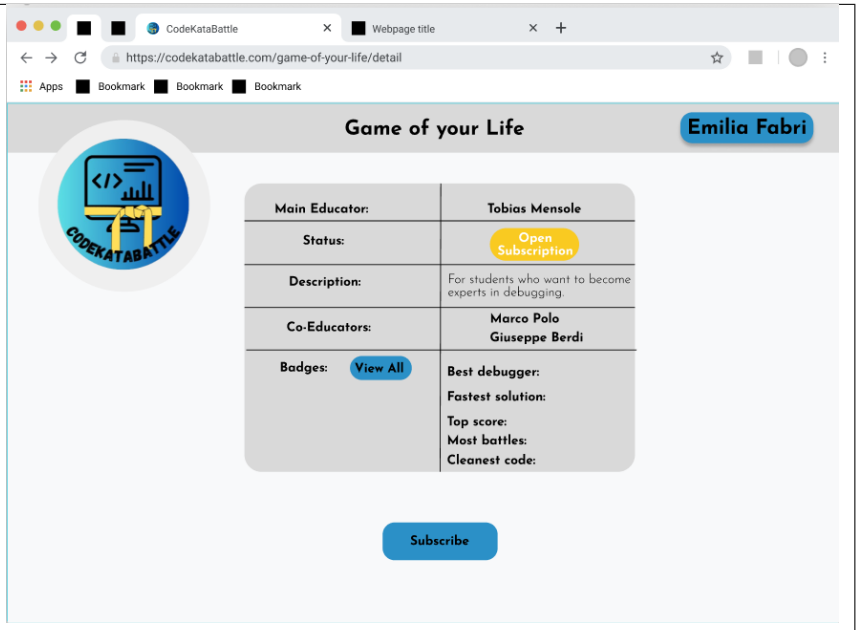
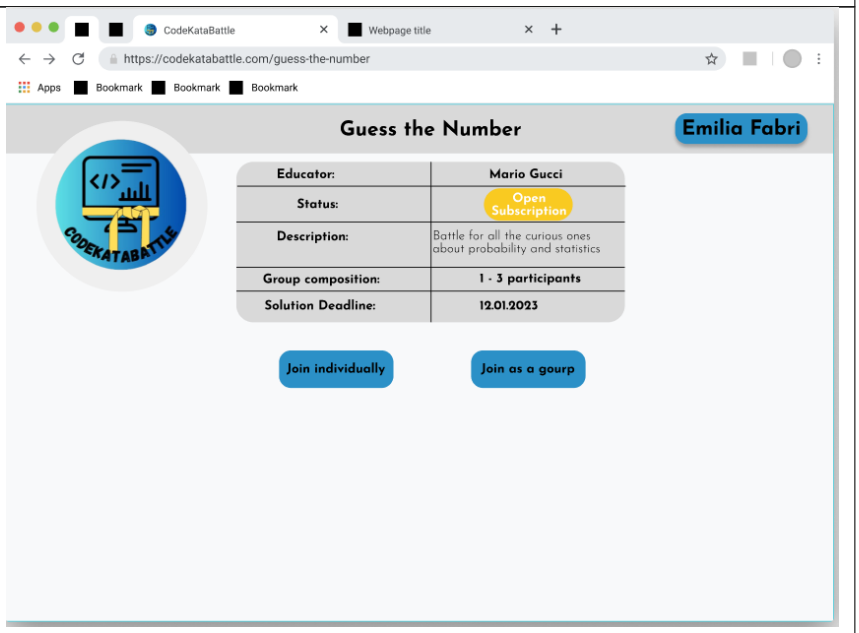
Add a Badge: when Educator on Edit Tournament Page clicks "+" in the Badges section, new fields are open. Educator is able to fill in name and rule of the Badge.	
Edit a Tournament - More: when Educator on Edit Tournament Page clicks "More" following options are opened: Create a Battle, Close (if possible to close the Tournament) and Cancel (if possible to cancel the Tournament).	

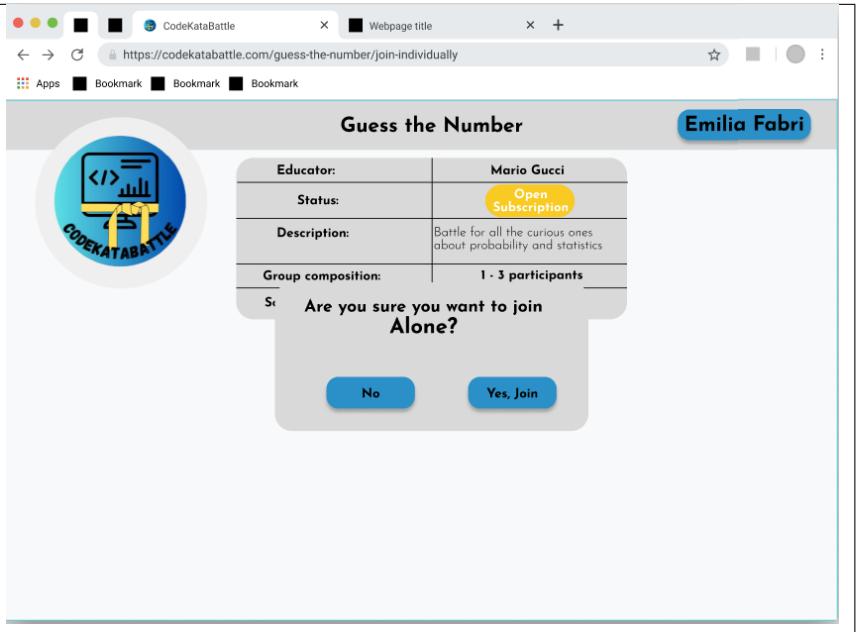
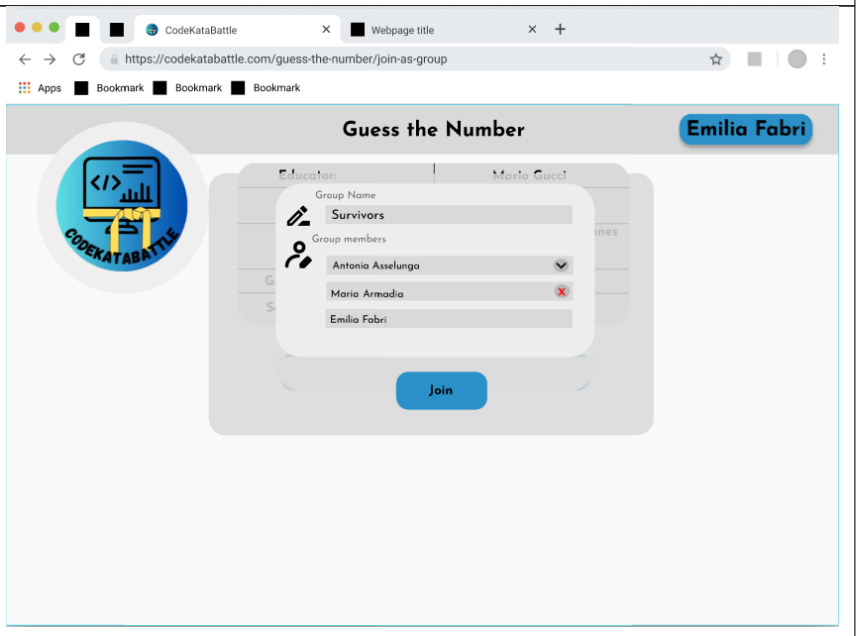
Close a Tournament: when Educator selects "Close" option form to Close a Tournament is opened.	
Cancel a Tournament: when Educator selects "Cancel" option form to Cancel a Tournament is opened.	

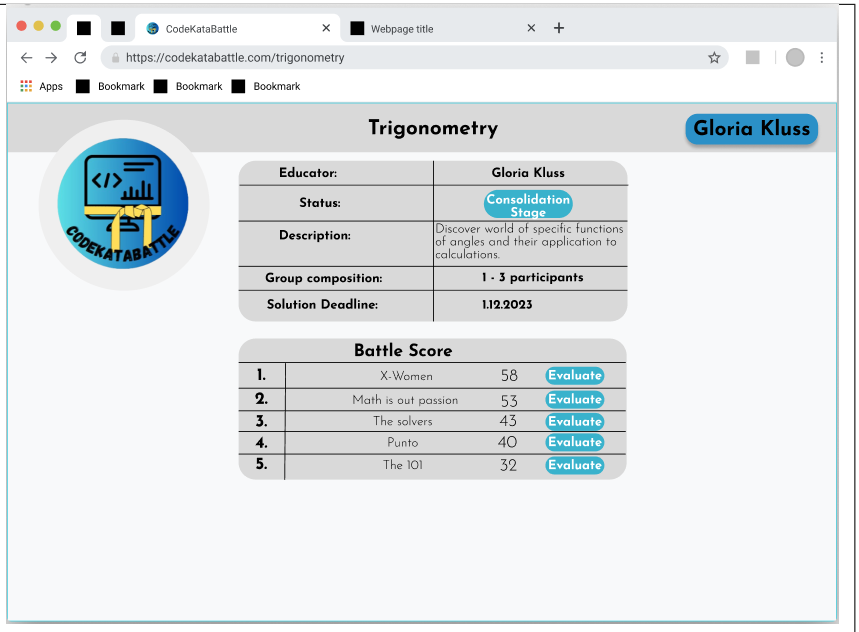
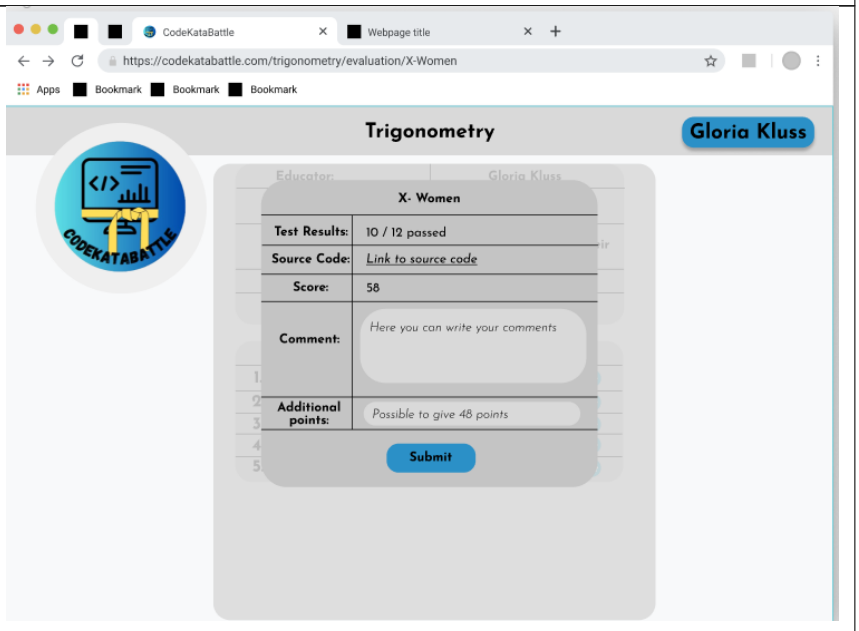
Battle List: when both Student and Educator selects "View All" button, in the Battle List preview table on the Tournament Information page, they are redirected to Battle List Page. They can use Sort button, to change the order in the list. Each Battle entry is clickable. Click redirects User to Battle Information page.	
Battle Information: on this page User can see information about specific Battle. This is view for a Student and an Educator who is not Battle creator. If it is a Student viewing, and the Battle is Ongoing, there is button "Push Solution" present.	

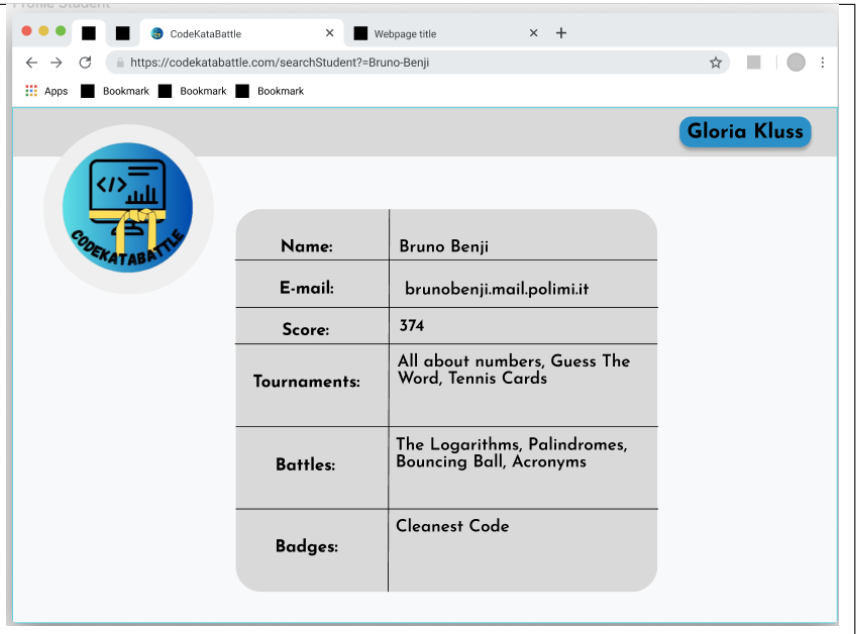
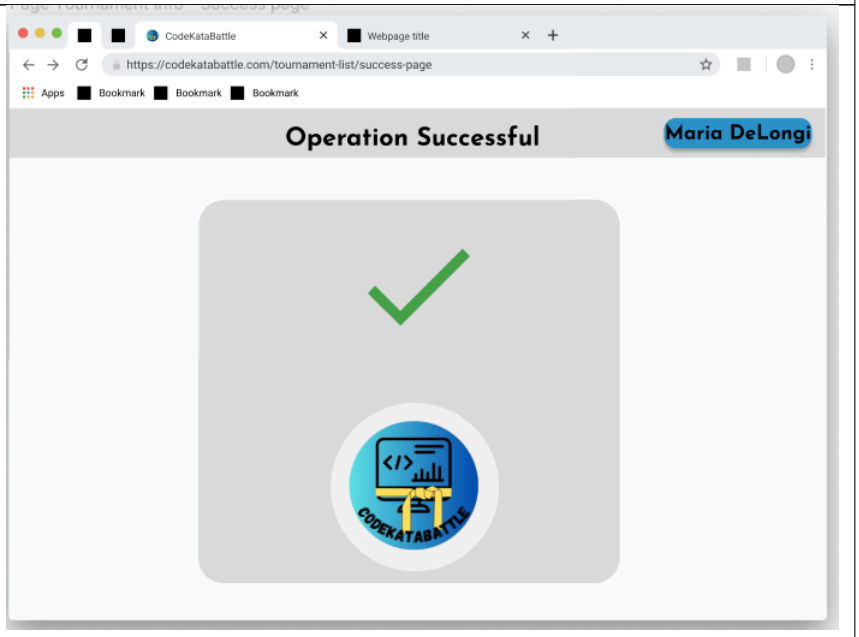
Battle Information - Creator: view for the Educator who is the Battle creator. The view has has an extra "Edit" button, if the status of the Tournament allows Edit.	
Edit a Battle: when Battle Creator clicks "Edit" on the Battle Information page, he is presented Edit interface. Tournament's name and description can be modified easily. If the Registration or Submission deadline have already occurred, they cannot be modified, otherwise, yes. Group size can be chosen from a drop-down.	

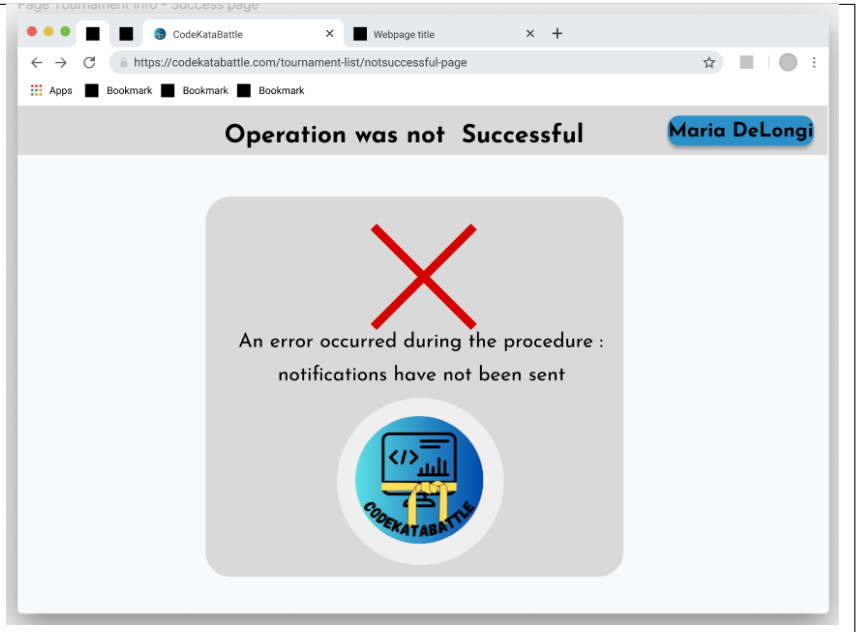
Edit a Battle - More: when Educator on Edit Battle Page clicks "More", Close (if possible to close the Tournament) and Cancel (if possible to cancel the Tournament). If none of them is possible, button "More" is not present.	
Cancel a Battle : when Educator selects "Cancel" option form to Cancel a Battle page is opened.	

Subscribe to a Tournament: view, when it is possible to Subscribe, the Students' view contains an extra "Subscribe" button.	
Join a Battle: view, when is possible to Join a Battle for the Students, the page contains extra buttons: "Join individually" and "Join as a group".	

Join individually: view, when a Student has chosen to join a Battle individually.	
Join as a group: view, when a Student has chosen to join a Battle as a group. There are as many fields as maximal capacity of the group is.	

Battle Information - Consolidation Stage: view for the Educator who is the Battle creator during consolidation stage. The are extra buttons for each group to give optional evaluation.	
Manual evaluation info: view for the Educator who has clicked on a button presented in a row of the group which is above to be evaluated.	

Student's profile view for any User who has clicked on a Student's name or search for the Student in a search field.	
Success page a page that is shown to User who performed an action and the action was successful. The word operation would change according to the performed operation, e.g: Tournament Closure successful, Tournament Creation Successful, etc.	

Failure page a page that is shown to User who performed an action and the action was not successful. The word operation would change according to the performed operation, e.g: Tournament Closure successful, Tournament Creation Successful, etc. The error message also changes according to what error has occurred.	

Pages are responsive, so they are adapted to size of the screens. In case of forms, if there is a required field missing, or an error occurs during performing of the action, a prompt opens, similar to the one that has been shown on the Registration error page.

Chapter 4

Requirements traceability

Requirements	R1: The Platform must allow a Guest to register as an Educator by filling a form. R2: The Platform must allow a Guest to register as a Student by filling a form. R6: The Platform must allow a Student to login by entering his credentials, that are academic email or Platform ID and password. R7: The Platform must allow an Educator to login by entering his credentials, that are working email or Platform ID and password. R35: The Platform must show a correctly updated Student's profile page. R37: The Platform must allow Users to update their profile information. R38: The Platform must allow Users to reset their password through a secure and user-friendly process.
Components	Web Application Account Manager Data Manager Database

Requirements	R3: The Platform must store the registration data of the Users.
Components	Web Application Data Manager Database

Requirements	R4: The Platform must allow a User to read and accept the Terms and Conditions and the Privacy Statement. R5: The Platform must ask Users the permission to send him notification.
Components	Web Application Account Manager Database

Requirements	R36: The Platform must allow interaction only between Users from the same institution.
Components	Web Application Account Manager CKB Tournament Manager CKB Battle Manager Data Manager Database

Requirements	R8: The Platform must allow an Educator to create a CKB Tournament. R9: The Platform must send to Students a notification about the creation of a new Tournament. R10: The Platform must allow an Educator to edit an ongoing CKB Tournament that he has created. R12: The Platform must allow an Educator, that has already created a Tournament, to grant permission to other Educators to create Battle in that Tournament. R13: he Platform must allow an Educator to cancel a CKB Tournament that he has created. R14: The Platform must allow an Educator to close a CKB Tournament that he has created. R15: The Platform must send a notification about the closure of a CKB Tournament to the subscribed Student. R18: The Platform must allow a Student to subscribe to a CKB Tournament. R29: The Platform must show a correctly updated Tournament list page. R30: The Platform must allow the Users to filter the Tournament list page by various criteria. R31: The Platform must allow Students to inspect Tournament information page of any state.
Components	Web Application Account Manager Data Manager CKB Tournament Manager Database

Requirements	R11: The Platform must allow an Educator, that has already created a Tournament, to define new Badges within that Tournament. R16: The Platform must triggers the Badge awarding process, after the closure of a Tournament. R17: The Platform must show a correctly updated Tournament information page.
Components	Web Application Account Manager Data Manager CKB Tournament Manager Badge Manager Database

Requirements	R20: The Platform must allow an Educator to create a CKB Battle. R21 :The Platform must send to Students a notification about the creation of a new Battle in a Tournament they are currently subscribed to. R22: The Platform must allow an Educator to edit a CKB Battle that he/his co-Educator has created. R23: The Platform must allow an Educator to cancel a CKB Battle that he/his co-Educator has created. R25: The Platform must allow a Student, forming a Group (even as the only member) to join a CKB Battle. R26:The Platform must store the information about participating Groups to a specific Battle. R33: The Platform must allow the Users to filter the Battle list page by various criteria. R34: The Platform must allow Students to inspect Battle information page of any state.
Components	Web Application Account Manager Data Manager CKB Battle Manager Database

Requirements	R24: The Platform must allow an Educator give a manual evaluation to a CKB Battle, that he/his co-Educator has created, respecting the 100 points threshold. R27: The Platform must be triggered by each push action on the main branch of the GitHub Repository and must upgrade the Battle score. R28: The Platform must show a correctly updated Battle information page. R32: The Platform must show a correctly updated Battle list page. R26: The Platform must store the information about participating Groups to a specific Battle. R15: The Platform must send a notification about the closure of a CKB Tournament to the subscribed Student. R18: The Platform must allow a Student to subscribe to a CKB Tournament. R19: The Platform must store the information about subscribed Students to the specific Tournament.
Components	Web Application Account Manager Data Manager CKB Battle Manager Static Analysis Tool GitHub Database

Chapter 5

Implementation, Integration and Test plan

5.1 Overview

In this section we provide an overview of the implementation, integration, and testing plans for the CKB platform. We present not just the plans, but also justify the decisions why such plans were chosen.

5.2 Implementation Plan

The CKB platform is composed from various components, that have been described in the previous sections.
Implementation has to start from the Data Manager, as the functionality provided by this manager is crucial for the rest of the components.
After implementing database related part, Account Manager can follow, because creating the necessary infrastructure for user registration, authentication, and account management is needed for remaining elements.
Afterwards, CKB Tournament Manager and Badge Manager might be implemented, followed by the CKB Battle management.
The final step of the implementation contains including external entities as **GitHub**, **Static Analysis Tool**, and the **SMTP provider**. The order of integrating of the external components can vary, or even they can be implemented simultaneously as they do not rely on each other.
By following this Implementation Plan, starting from low-level components we ensure a systematic and coherent development process, leading to the successful integration of all components within the CKB Platform. Also, thanks to the chosen architecture, the Web Application can be developed independently, even along the implantation of the other components, ensuring effective process.

5.3 Integration plan

The Integration plan ensures a smooth merging of all the components. The process is aligned with the Implementation plan, prioritizing components interacting with the database, allowing for a efficient and coherent development workflow.
After implementing the Data Manager interaction between database components and verification of the data integrity must be performed.

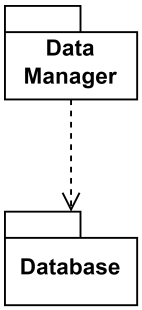


Figure 5.1: Integration of Data Manager

The Account Manager implementation must contain integration of all the managers responsible for User management, including Student Registration Manager, Educator Registration Manager, and Authenticator.

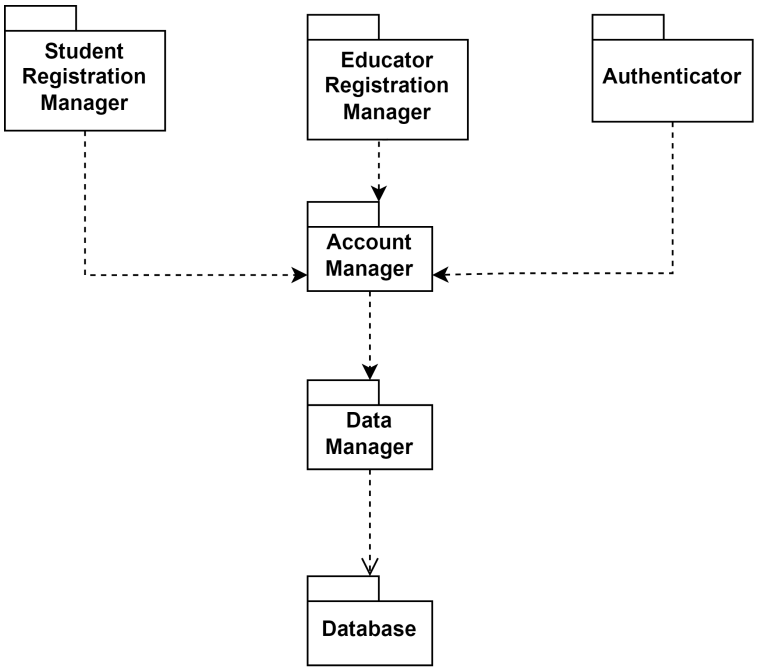


Figure 5.2: Integration of Account Manager and the sub-managers

Afterwards, integration of the CKB Tournament Manager and Badge Manager providing creation, management for both, Tournament subscription processes, and Badge awarding process may be performed.

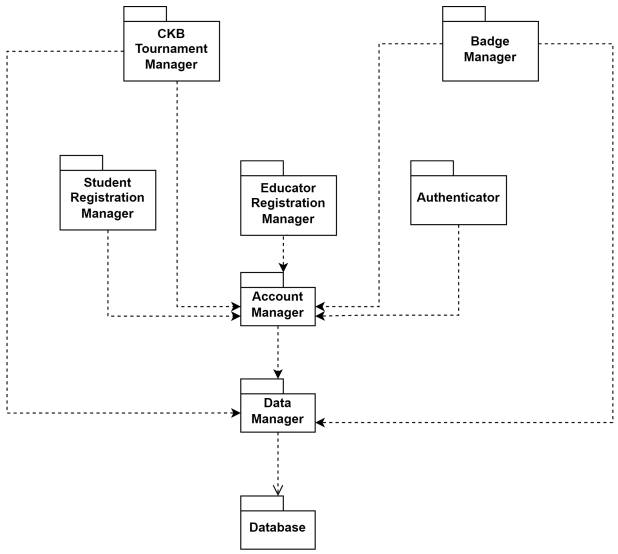


Figure 5.3: Integration of CKB Tournament Manager and Badge Manager

Followed by the Battle creation, management, and participation processes should be carried out.

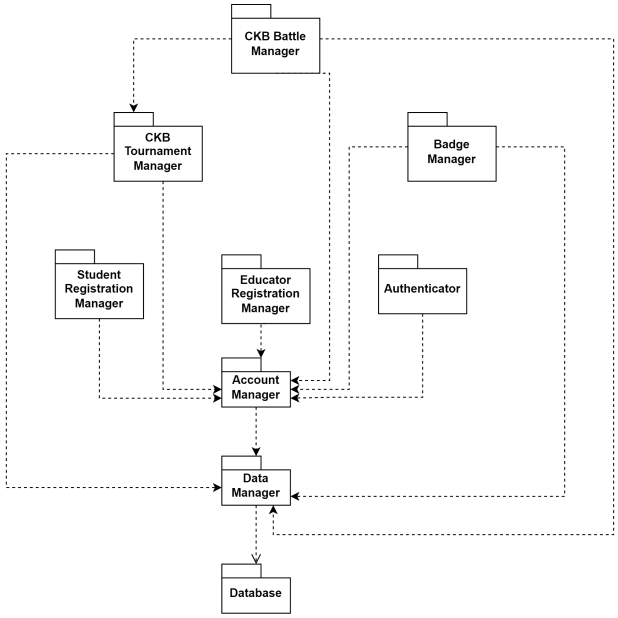


Figure 5.4: Integration of CKB Battle Manager

Last but not least, integration of external tools can be executed.

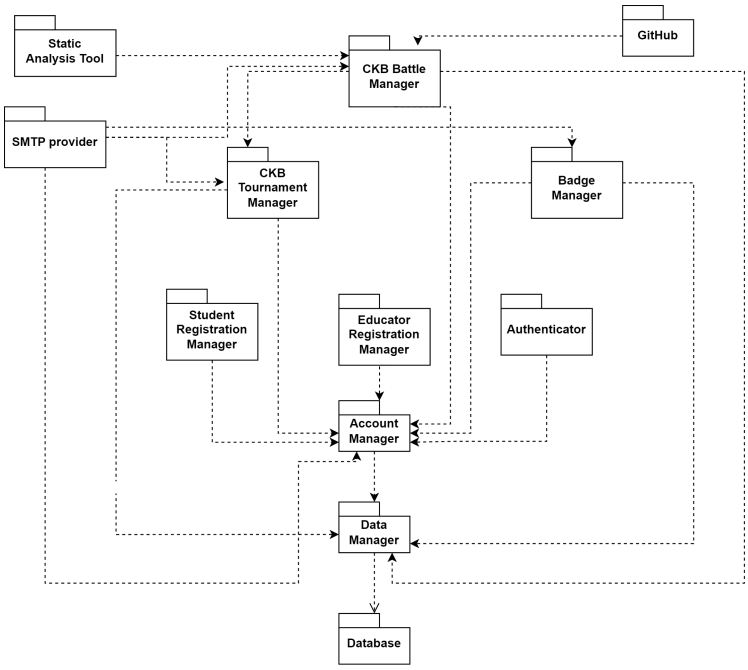


Figure 5.5: Integration of External Tools

Aiming for this Integration plan we ensure the reliability and functionality of each component within the overall system. The Web Application Server Components have to moreover successfully and flawlessly communicate with the Web Application.

5.4 Test Plan

Testing must be done in various stages of the CKB platform implementation. All the components have to be tested independently, in the end, also the Web Application as whole must be tested to ensure the CKB platform works as designed.

1. Unit Testing - to ensure functionality of all the individual components
2. Integration Testing - to verify components communication and cooperation is flawless
3. System Testing - to confirm the CKB platform works as a complete and unified whole
4. User Acceptance Testing - to ensure the platform meets the Users expectations
5. Security Testing - to identify and address potential security vulnerabilities
6. Performance Testing - to assess system performance under different conditions
7. Deployment Testing - to validate the deployment process and verify system stability
8. Regression Testing - to check whether any update did not negatively effected the platform

Chapter 6

Effort spent

6.1 Alzbeta Fekiacova

Section	Hours
Introduction	3
Architectural Design	14
User Interface design	30
Requirements traceability	2
Implementation, Integration and Test plan	6

6.2 Federico Schermi

Section	Hours
Introduction	2
Architectural Design	24
User Interface design	3
Requirements traceability	3
Implementation, Integration and Test plan	2

6.3 Diego Quattrone

Section	Hours
Introduction	2
Architectural Design	25
User Interface design	3
Requirements traceability	6
Implementation, Integration and Test plan	2

Chapter 7

References

- Runtime view diagrams made with DrawIO
- User interface design made with Figma
- Software Engineering 2 course material - Politecnico di Milano 2023-2024