CodeKataBattle (CKB) Design Documentation

Version: 1.0

Authors:
Rishabh Tiwari - rishabh.tiwari@mail.polimi.it
Adrian Valica - adrian.valica@mail.polimi.it
Pablo García Alvarado - pablo7.garcia@mail.polimi.it

Release Date: January 7, 2024

THE CODE KATA BATTLE

Contents

1 Introduction

1.1 Purpose

The Design Document serves as a detailed guide for the development, deployment, and maintenance of the CodeKata Battle (CKB) platform. This document elaborates on the system architecture, data models, interface designs, and interaction mechanisms with external APIs, specifically focusing on the requirements outlined in the RASD. Its primary purpose is to ensure that the development team has a clear, structured approach to creating a platform that is efficient, scalable, and user-friendly, targeting the educational needs of the I.I.S.S Bertrand Russell community.

1.2 Scope

This document details the architectural design of CKB, a dynamic, web-based coding challenge platform for educators and students. It includes the system's structural design, data handling strategies, user interaction flow, and integration with external services like GitHub. The scope extends to defining the software's behavior in various scenarios, ensuring compliance with performance and security standards, and presenting a roadmap for future enhancements and scalability.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Code Kata**: A programming exercise aimed at honing coding skills through practice and repetition.
- **Gamification**: The incorporation of game elements in a non-game context, used here to enhance student engagement in coding challenges.
- **Repository (Repo)**: A digital directory or storage space where the CKB platform's coding challenge projects are stored and managed.

1.3.2 Acronyms

- **CKB**: CodeKata Battle
- **RASD**: Requirements Analysis and Specification Document
- **API**: Application Programming Interface
- **GDPR**: General Data Protection Regulation

- **CKBDBMS:** CodeKataBattle DataBase Manager System

1.3.3 Abbreviations

- **Tourn.**: Tournament
- **Batt.**: Battle
- **DB**: Database
- **UI**: User Interface

1.4 Revision history

Version	Date	Description
1.0	2023-12-27	Initial creation of the Design Document, covering the basic architecture and design strategies for the CKB platform.
1.1	2024-01-07	Addition of detailed system architecture diagrams and descriptions of data flow.

1.5 Reference Documents

- **RASD Document for CKB**: The original document detailing the requirements and specifications for the CodeKata Battle platform.
- **GitHub API Documentation**: Reference for integrating GitHub functionalities within the CKB platform.
- **Alloy Modeling Language**: Used for formal analysis in the RASD document.
- **Educational Coding Platforms Study**: Comparative analysis of existing educational coding platforms.

1.6 Document Structure

The Design Document is structured as follows:

1. **System Architecture**: Detailed description of the system's architecture, including data flow and component interactions.
2. **Data Design**: Explanation of the database schema, data storage, and management strategies.

3. **User Interface Design**: Illustration and description of the platform's UI elements, layouts, and navigation flow.
4. **System Integration**: Details on how the platform integrates with external APIs and services.
5. **Security Protocols**: Outline of the security measures and data protection strategies.
6. **Performance and Scalability**: Discussion on system performance, scalability solutions, and maintenance plans.
7. **Appendices and References**: Additional resources, references, and supplementary materials.

2 Architectural design

2.1 Overview

This section presents a detailed overview of the CodeKataBattle system architecture, including the various elements that make up the system, how they interact, and an explanation of the chosen replication mechanism that enables the system to be distributed.

2.1.1 High level view

The CodeKataBattle platform was conceptualized with a clear separation of concerns. This division intervenes in the duality of the user interface (a web page for the visualization of information, and then a web app for the actual functions of the platform). On this same principle, the architecture is distributed into multiple main elements, divided into a Three-Tier Architecture, segmenting the system into three layers, each only capable of communicating between adjacent layers.

All the elements presented in this division (in the image) are elements that will be used by all types of users, no matter if they are Educators managing tournaments or Students taking part in challenges. Their functioning will become explicit in the following lines.

Presentation Layer: The first layer of the architecture is the Presentation one. It contains the Web App client and the Web Page Clients. The WebPage Client can be used on most of the devices, such as a smartphone or a computer. Its purpose is mainly for visualization, with little possible effect actions, both for the Educators and the Students. Users will be able to create an account or log in,

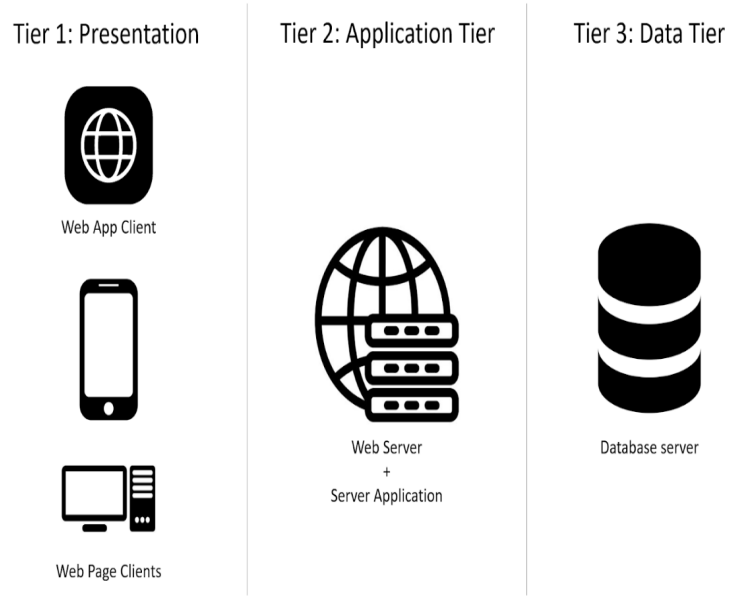


Figure 1: **High Level View.**

consult their profiles, edit the information, browse through the available challenges, and check their notifications. The Educators will also be able to see the state of the tournaments and challenges that they are currently managing; the Students will also be able to see the state of the tournaments they are taking part in, their scores, and submissions. On the other side, the Web App Client will be used for actually using the platform rather than visualizing data. Due to the nature of the operations that can be done on the Web App (managing the tournaments, or submitting and executing tests for challenges), it can only be accessed on a desktop. The Educators will this way be able to add or delete battles to their challenges, create new tournaments, create badges for their tournaments, add new educators as co-leaders, manually grade the submissions of the challenges they set the option to, and update the details of their tournaments or challenges. The Students will be able to manage their previous and current submissions, run the tests of the challenge and add members to their group.

Application Layer: The second layer of the architecture is the Application one. This is the layer containing the logic of the system. It is made of two elements: the server browser and the server application.
The server browser is mainly used for answering the queries that it will receive. This is the part of the application layer that will be used to transmit the information to display to the clients. This web server element would be able to process

different types of queries, depending on the type of account that is producing it (Students and Educators will have different displays).
On the other hand, the application server is used for all the computations that need to be done by the system. This will include running tests, managing Git Repositories, and computing scores on submissions.

Data Layer: The third and last layer is made of the database server. Its role is the data storage in the system. It will have a storage unit for the user accounts and all related information such as account data, tournaments and battles taken part in, submissions, and results in those; one storage unit for battles, tournaments, and related information, such as available badges and, participants, final leaderboards, requisites, submissions for each user.

Communication between the layers: The Presentation layer will only be able to communicate with the Application layer. The Web Page Client will mainly require the Web Server for queries related to the information to display on the web page depending on the user inputs, and some modification of basic data, such as account information. They will also have some limited access to the Application layer for some low-cost activities, such as user authentication, validation of some data updates, and basic actions (subscriptions, deletion, accept groups... non-related to creating or taking part in battles). The Web App Client on the other hand will need to access both the Web server and the Application server. The Web server will be accessed for the queries that need to be displayed, and the Application server to execute the required computations that the system would require while actively taking part in the battle (while managing the submissions and running the tests if it is a student, or while creating the tournaments and battles if it is an educator).

The Application Layer will need to communicate with the Data Layer to query the data needed by the user and to update or add fields to the stored types (for example when registering a new user, submitting a new submission, or changing the title of a tournament). The system will also need some additional services:

- An authentication service that will be connected to the Application server, used to create new accounts and ensure the security of the logins.
- A Github service that will be able to create git repositories and manage them (detect pulling, and execute tests on the code in the repository).
- An email provider to notify the users when they subscribe to an event.

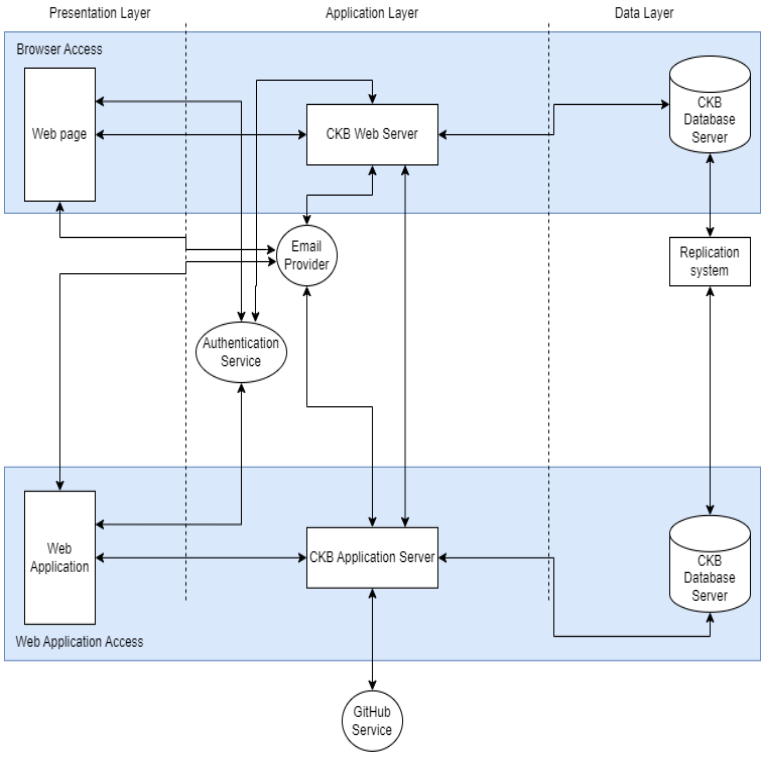


Figure 2: **Three tier architecture.**

2.1.2 Distributed View

In the CKB system we have one Database Cluster conformed by N servers that can be increased according to the needs of the moment and also an application server group that also has an N number of servers that could vary according to necessity. These two groups of servers can communicate between them. The system is provided with a firewall and a load balancer to increase its fault tolerance when the Web application tries to access the application servers group. Besides, the system also has an Authentication server for security reasons, an email provider and also a GitHub service for managing GitHub links and submissions for the tournaments. All these three components can communicate to the application servers group and vice versa. In addition, the email provider can communicate with the web application and vice versa while the authentication component can only access the web application. The CKB Database cluster can access both the Authentication service and the email provider.

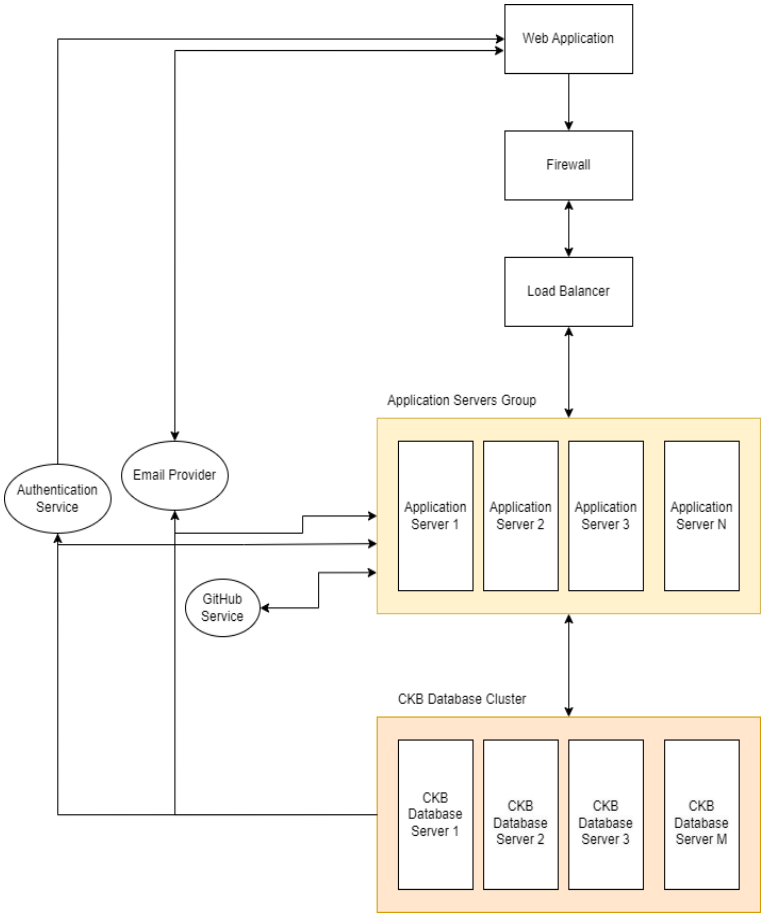


Figure 3: **Web Application Distributed View.**

The second image presents a similar yet distinct architecture for a web page system. It starts with a 'Web Page' component, followed by a 'Firewall' and a 'Load Balancer' that precedes a 'Web Servers Group'. This group contains several 'Web Servers', showing the system's capacity to handle web traffic by scaling horizontally. There is a separate 'Load Balancer' for an 'Application Servers Group', indicating a clear separation of concerns between serving web content and application logic. These application servers also scale from 1 to N. Below, like in the first image, there's a 'CKB Database Cluster'. To the side, the 'Authentication Service' communicates with an 'Email Provider', indicating this system also has email capabilities, which are essential for functions like user notifications or password resets.

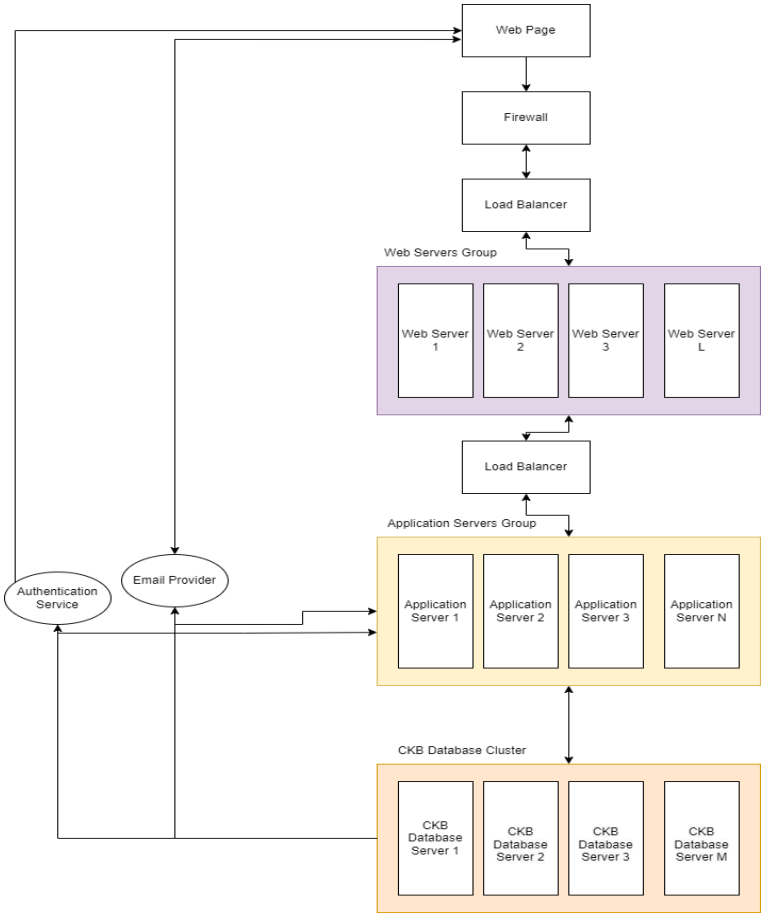


Figure 4: **Web Page Distributed View.**

2.2 Component view

Here come two different components diagrams, showing the connections between the system elements. We decided to provide two of them, one for each available flow (the Web App and the Web Page) for more readability, and less repetitive connections. These two flows are built in the same system, and all the connections shown in both diagrams will be present in the final system, but we included for each of them only the connections that are used, for more readability.
The Web App and the Web Page components are the only components in the presentation layer.
Then we have at the right the Model and CKBDBMS components which are the components from the data layer.
The Authentication Service, the GitHub Service, and the Email Provider Service are all external components, not implemented in our system. The rest of the

elements are part of the business logic layer, completely owned by the CKB system.

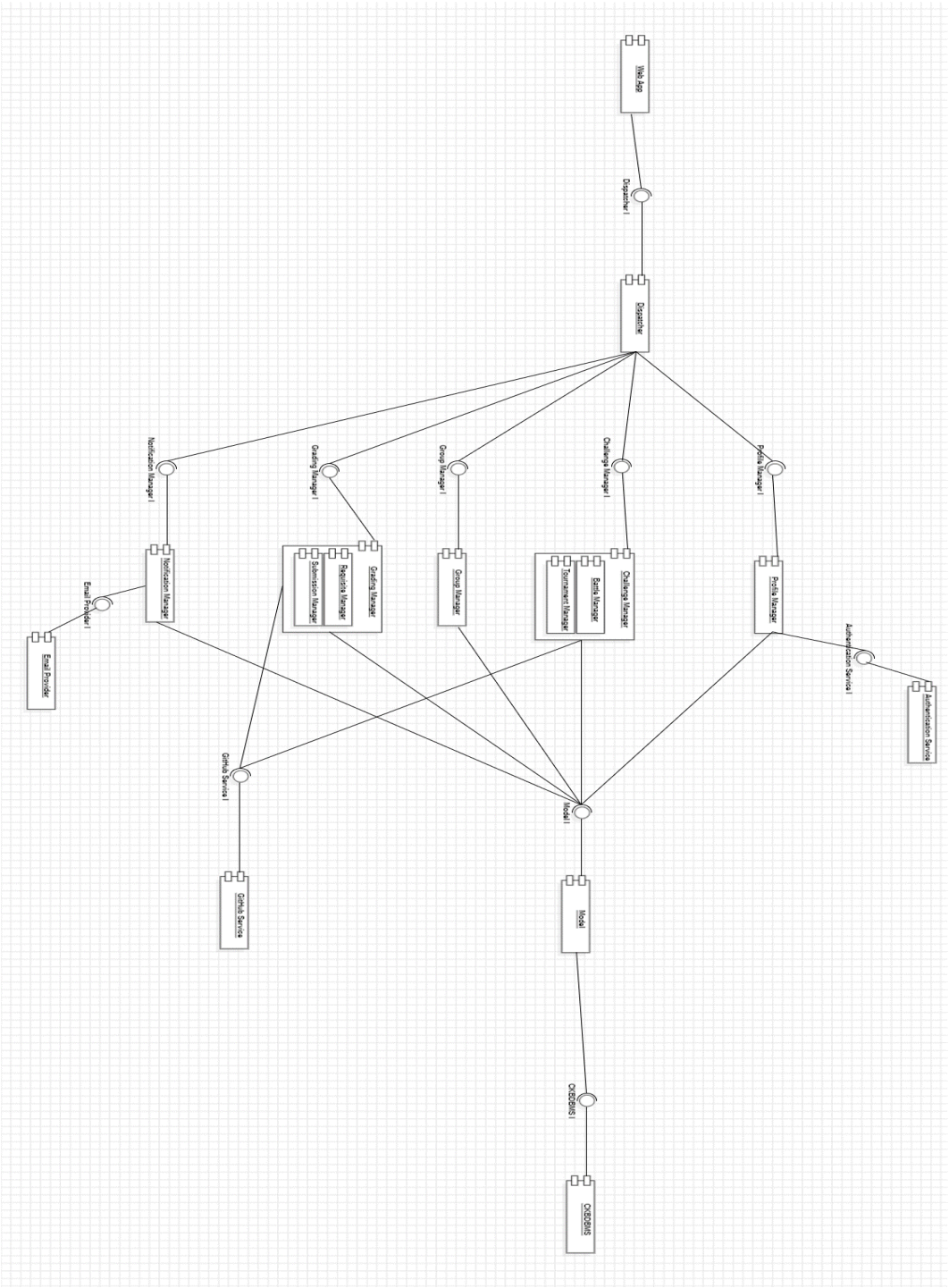


Figure 5: **Component View of Web Application.**

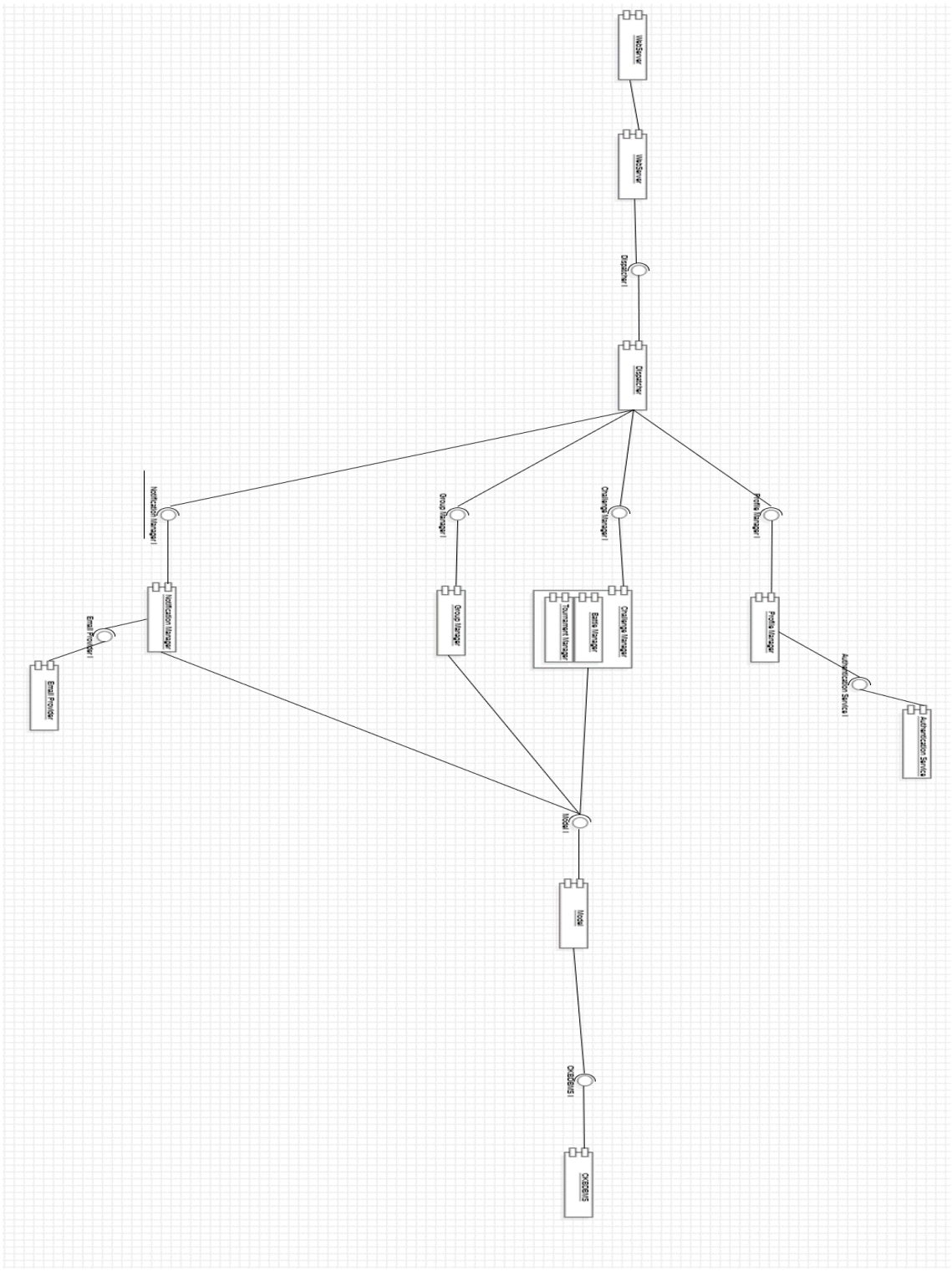


Figure 6: **Component View of Web Page.**

Here is an explanation of each of the system components that were presented in the previous section and the previous diagram.

Web App: It is the int erface used by educators when creating and updating tournaments and battles. It communicates with the web and application server by HTTP requests. It can only be executed on a computer.

Web App: The goal of this component is to manage all the requests coming from the browsers. Once it gets the request, it forwards it to the Dispatcher, which will then forward it to the right component for the task. Once the Web Server receives the result of the operation, it sends it back as an answer to the corresponding query.

Dispatcher: This component acts like a controller that directs incoming requests to the appropriate handlers or services within an application. It maps the requests from the server to the components that can execute the task. It then sends the results back to the component having requested it.

Group Manager: This component has the responsibility to manage the groups of students. It will send and delete the invitations for accounts to join the group, and update group information if the invitations are accepted or if members leave.

Tournament Manager: This component will take care of all the tournament-related aspects of the system. As for the user functionalities, it will be the component taking care of the user subscription or withdrawals to the tournaments and updating the members list accordingly, changing the states of the tournaments. Then, during the tournament, it will update the scores of the groups on the leaderboards, according to the Battle Managers' results. On the Educator's side, it will update the tournament structure according to the creation process done by the Educator. It will also take care of adding members to the team of leaders. At the end of the tournament, it will be the component computing the results and storing them.

Battle Manager: Similarly to the Tournament Manager, the battle manager will take care of all the battle-related aspects of the system. It will be the component taking care of the subscription or withdrawals to the battles updating the member list. It will also launch the testing on the new submissions, and update the score and leaderboards based on the results of the tests.

Requisite Manager: The requisite manager's role is to manage the requisite list for each tournament (adding or deleting requisites), and to keep track of their completion for each user. These requisite managers can be used for subscription conditions, battle submission checking, grading, and badge rewards.

Submission Manager: This component listens to the Git push events and as soon as one is detected, it pulls the code and launches the tests provided by the educators on the last submission. After auto-grading the results, it communicates the score to the Battle Manager.

Notification Manager: The notification manager is used by many of the other components. Its goal is to send notifications to users automatically and to listen to the user's responses to those.

Profile Manager: This component manages all changes related to the profile management. This could be the creation of an account, the deletion of an account, or the update of the information about an account and its visualization.

Model: The component that serves as the repository for the application's persistent data in the Model View Architecture, typically stored in a database. It's responsible for loading this data into the application and provides the necessary methods to fetch and manipulate this data. Essentially, when any part of the system needs data for processing, it calls on the Model component to obtain the required information. The Model retrieves and organizes the data within specialized data structures, preparing it for subsequent operations or computations. It is the sole part of the application that communicates directly with the database management system (DBMS) service.

CKBDBMS: This component is the database manager for the whole system. It stores the data for the user accounts and all related information such as account data, tournaments and battles taken part in, submissions, and results in those; one storage unit for battles, tournaments, and related information, such as available badges and, participants, final leaderboards, requisites, submissions for each user.

2.3 Deployment view

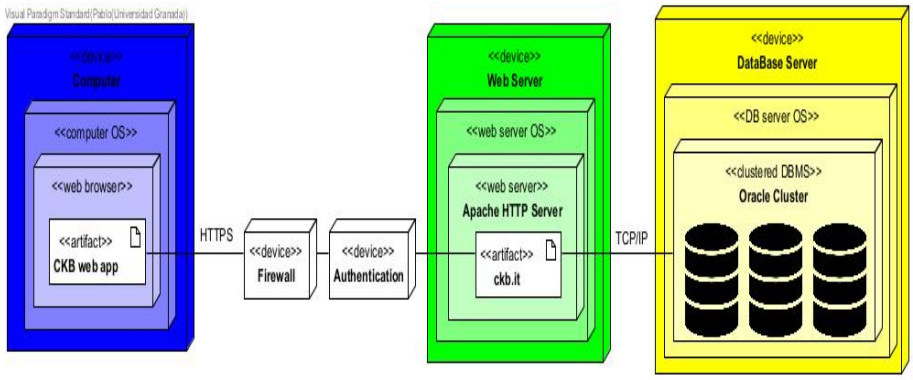


Figure 7: **Deployment View.**

- **Firewall:** The firewall is a device whose purpose is trying to identify all those packets that could jeopardize the data's integrity. Those potentially dangerous packets are then not forwarded, which means that only those that are safe will access the web server.
- **Authentication:** This is the device that will check whether the person who is trying to access his/her user account corresponds to that actual person. Otherwise, the authentication device will not let that person access the web server.
- **Computer:** As its name indicates, this is a normal PC with access to a web browser of preference. A stable internet connection is assumed so the CKB website can be accessed without problems. Through an HTTPS connection the ckb.it website is accessed after passing a firewall explained above.
- **Web Server:** The web server manages the functionalities of the system available to either students, educators, or both. It can communicate via a TCP/IP connection to a database of Oracle to easily access and check the data required to the correct performance of the system.
- **Database Server:** The database server is a cluster of several databases managed through an Oracle tool that will replicate the information so it can be easily recovered in case there is a fault that erased some data.

2.4 Runtime views

Now all the runtime views will be displayed. These show how the different components of the system communicate in order to achieve each use case that was

previously explained in the RASD document.

- **Sign Up:** The Sign Up use case is called from the WebApp and sends the information introduced by the potential user to create an account, that information is sent through the Web Server to the Profile Manager, this component then tries to check whether the information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will create an account, otherwise the system will send diverse messages of error depending on what was wrong.

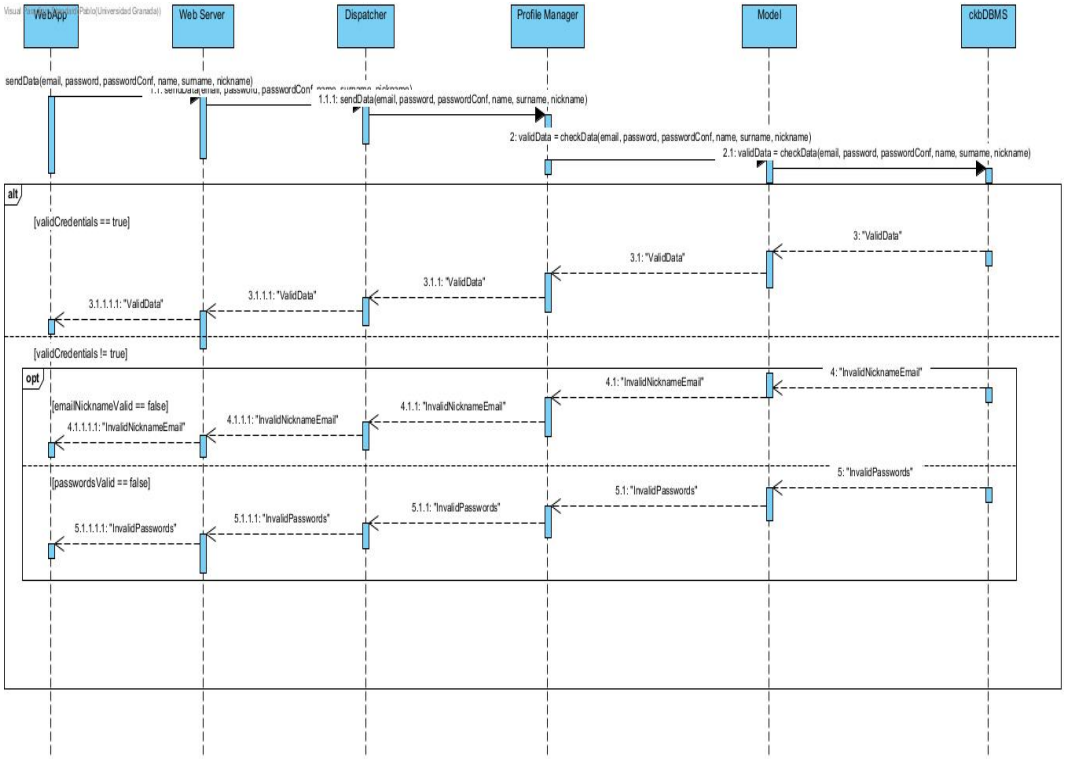


Figure 8: **Sign Up View.**

- **Log In:** The Log In use case follows a similar procedure to the Sign Up one since is called from the WebApp and sends the credentials introduced by the user to enter his/her account, those credentials are sent through the Web Server to the Profile Manager, this component then tries to check whether the credentials are correct by sending them to the Model and comparing to the credentials attached to that user in the CKB DBMS then if the credentials are in order the system will Log In the user, otherwise the system will send an error message.

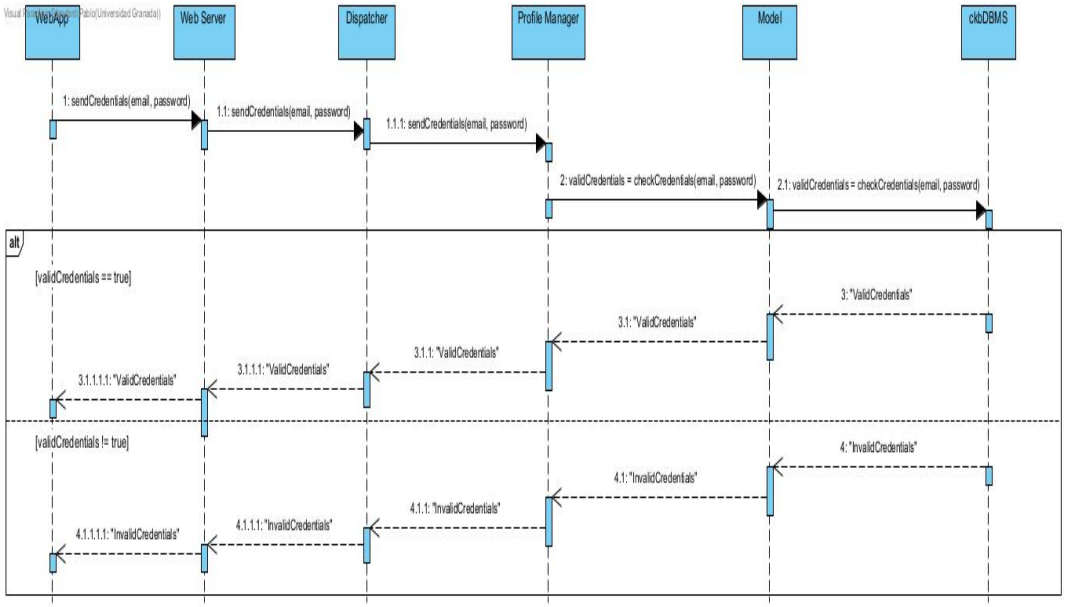


Figure 9: **Log In View.**

- **Log Out:** The Log Out use case is called from the WebApp and sends the answer introduced by the user to Log Out, the answer is then sent to the profile manager and checked by the model which will close the session if the answer was yes or cancel it otherwise and send the respective messages.

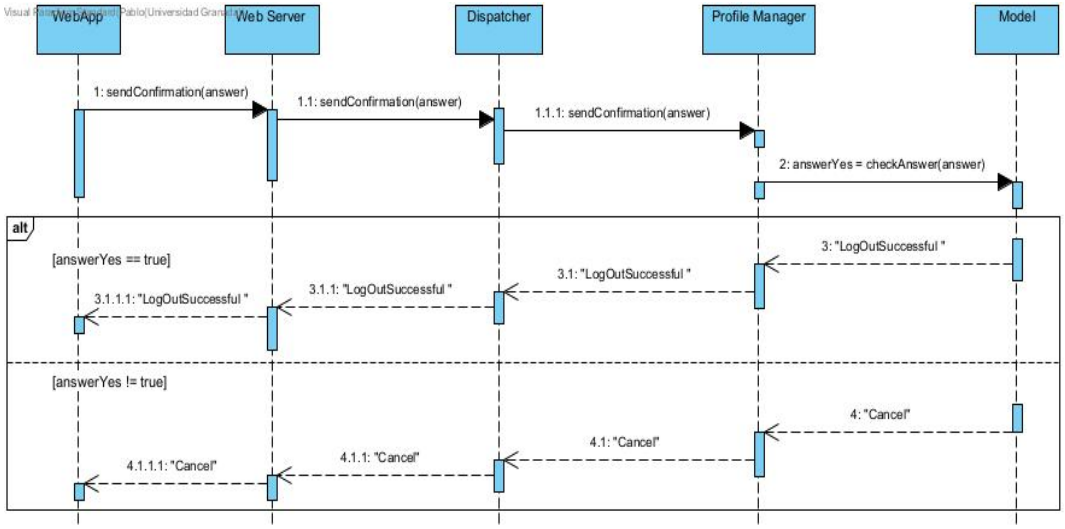


Figure 10: **Log Out View.**

- **Delete Account:** The Delete Account use case is called from the WebApp and sends the answer introduced by the user to Delete the Account, the answer is then sent to the profile manager and checked by the model and CKB DBMS which will delete the session if the answer was yes and the user is not a student that is enrolled in a tournament or cancel it otherwise and send the respective success or error messages. messages.

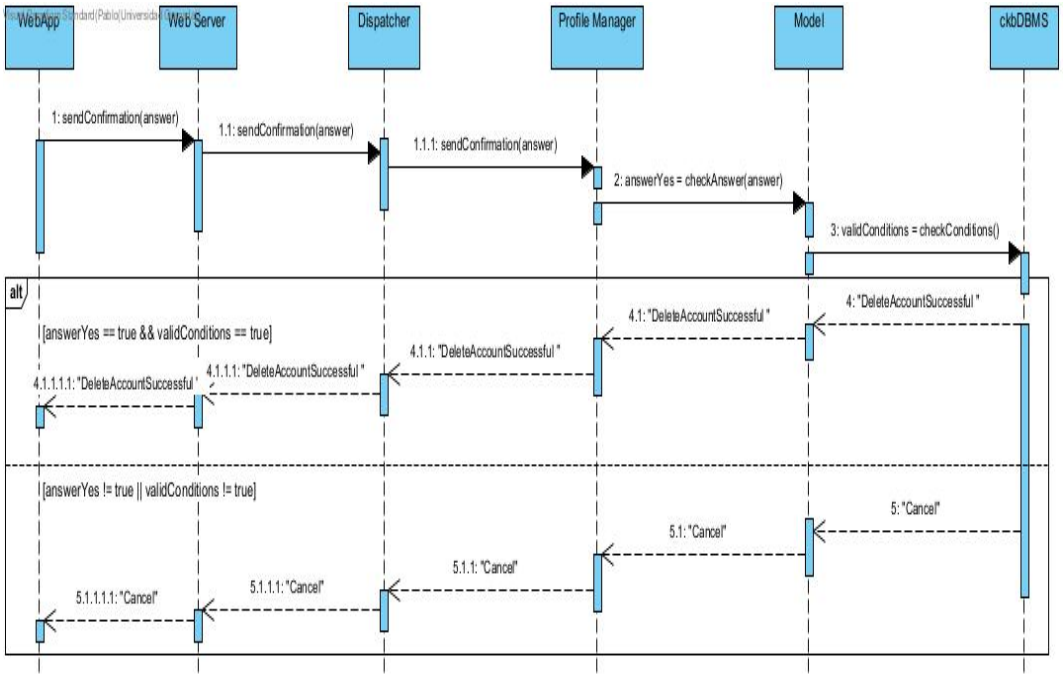


Figure 11: **Delete Account View.**

- **Create Tournament:** The Create Tournament use case is called from the WebApp and sends the information introduced by the educator to create a tournament, that information is sent through the Web Server to the Tournament Manager, this component then tries to check whether the information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will create a tournament with the information provided, otherwise the system will send an error message.

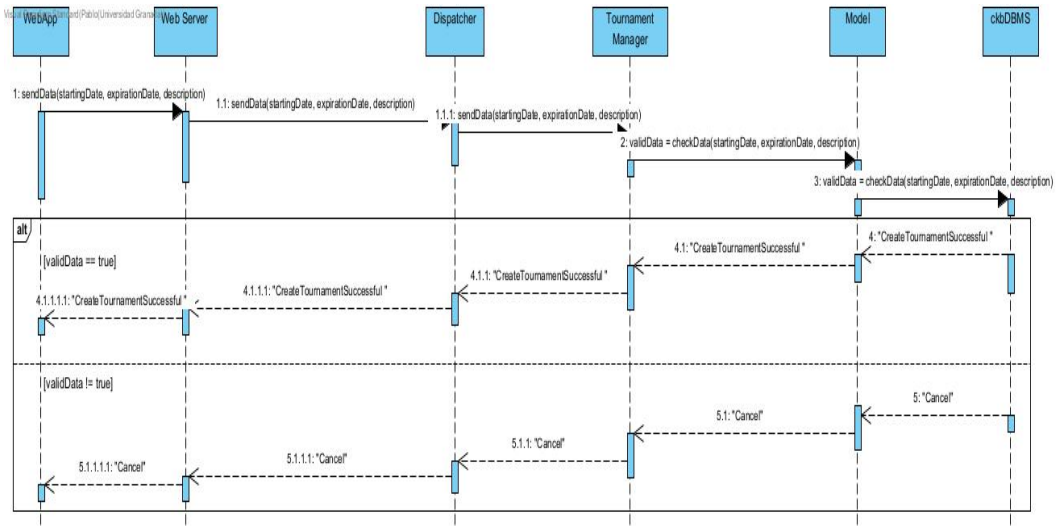


Figure 12: **Create Tournament View.**

- **Create Battle:** The Create Battle use case is called from the WebApp and sends the information introduced by the educator to create a battle, that information is sent through the Web Server to the Battle Manager, this component then tries to check whether the information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will create a battle with the information provided, otherwise the system will send an error message.

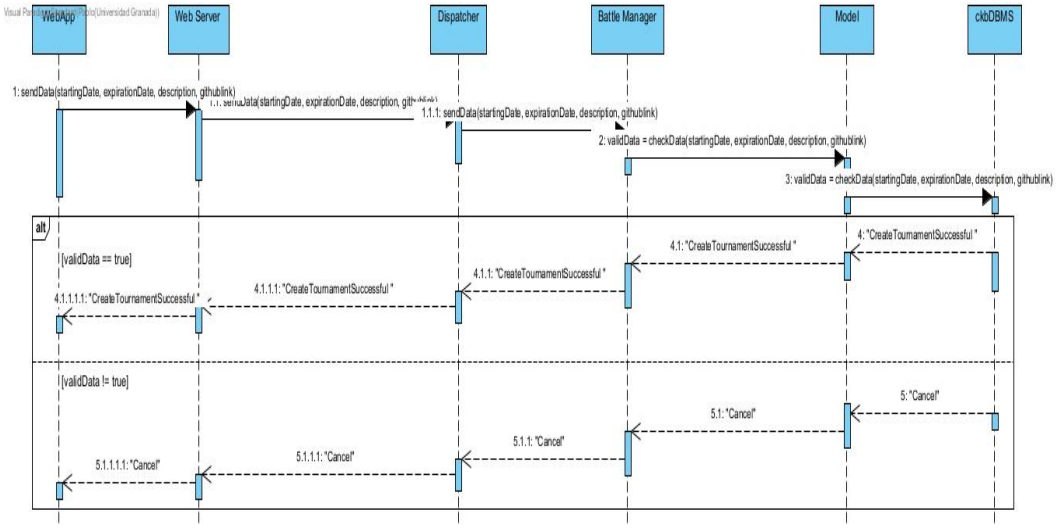


Figure 13: **Create Battle View.**

- **Delete Tournament:** The Delete Tournament use case is called from the WebApp and sends the answer introduced by the user to Delete the Tournament, the answer is then sent to the tournament manager and checked by the model and CKB DBMS which will delete the tournament if the answer was yes or cancel the operation otherwise while sending the respective success or error messages.

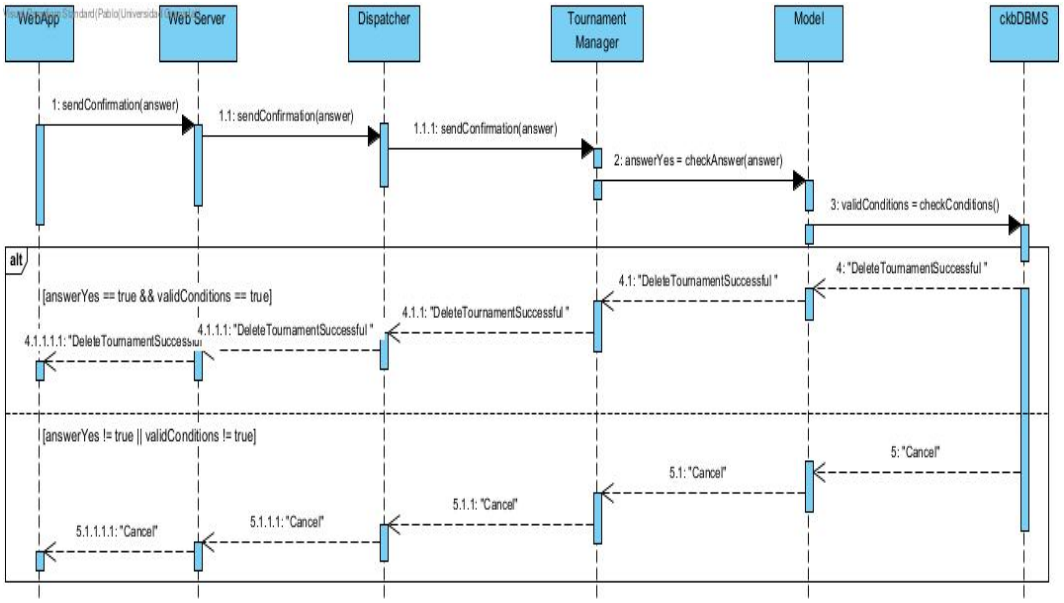


Figure 14: **Delete Tournament View.**

- **Create Badge:** The Create Badge use case is called from the WebApp and sends the information introduced by the educator to create a badge, that information is sent through the Web Server to the Badge Manager, this component then tries to check whether the information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will create a badge with the information provided, otherwise the system will send an error message.

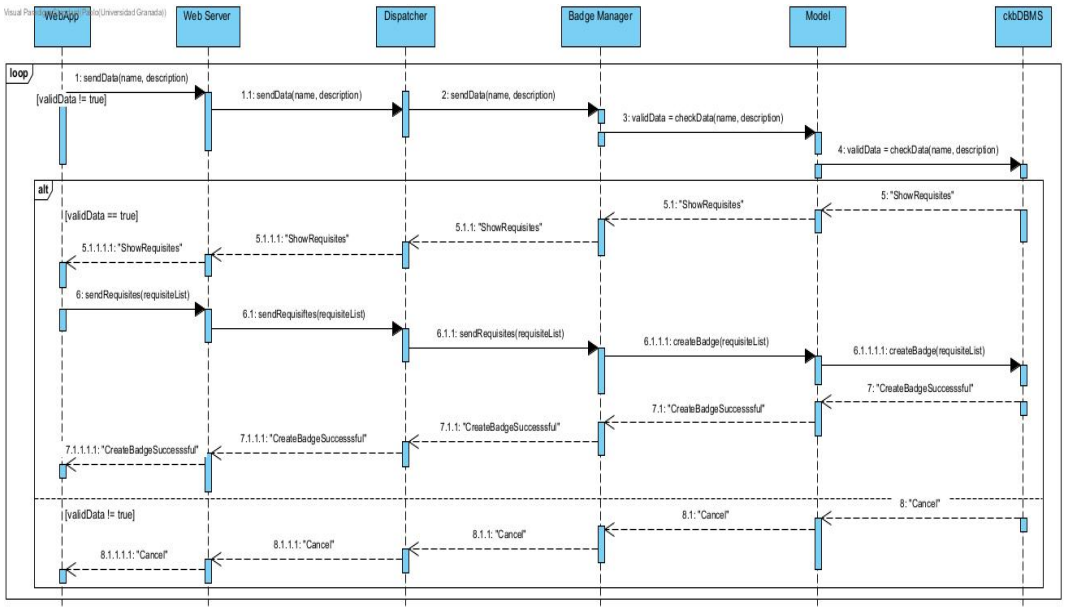


Figure 15: **Create Badge View.**

- **Create Requisite:** The Create Requisite use case is called from the WebApp and sends the information introduced by the educator to create a requisite, that information is sent through the Web Server to the Requisite Manager, this component then tries to check whether the information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will create a requisite with the information provided, otherwise the system will send an error message.

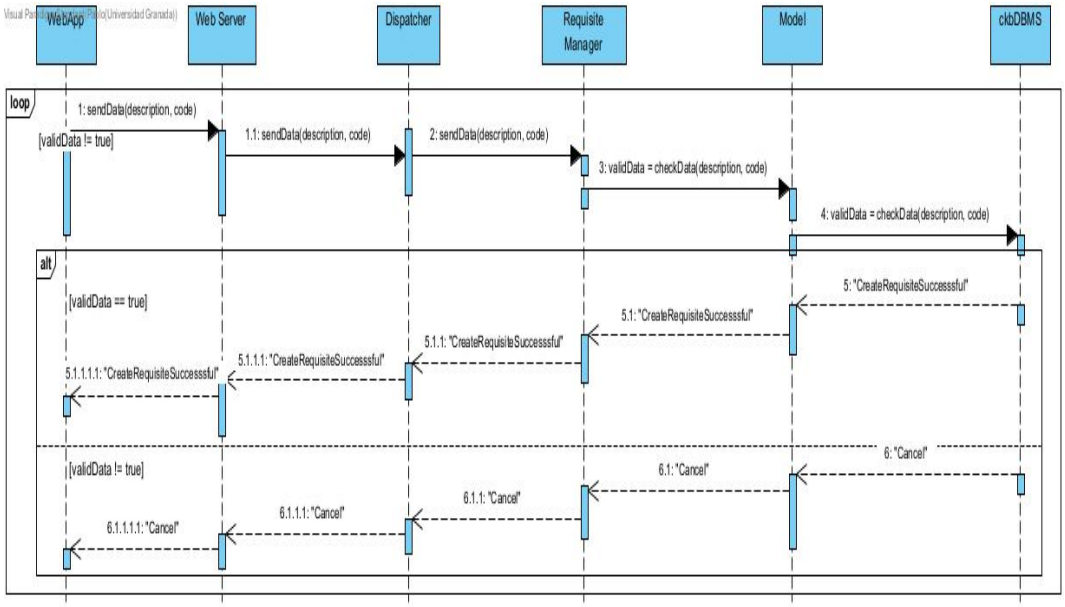


Figure 16: **Create Requisite View.**

- **Allow Other Educator to Create Battles:** The Allow Other Educator to Create Battle use case is called from the WebApp and sends the information introduced by the educator to allow one or several educators to create battles within a tournament, that information is sent through the Web Server to the Dispatcher to the Battle Manager, this component then tries to check whether the information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will allow the educators to create battles within that tournament.

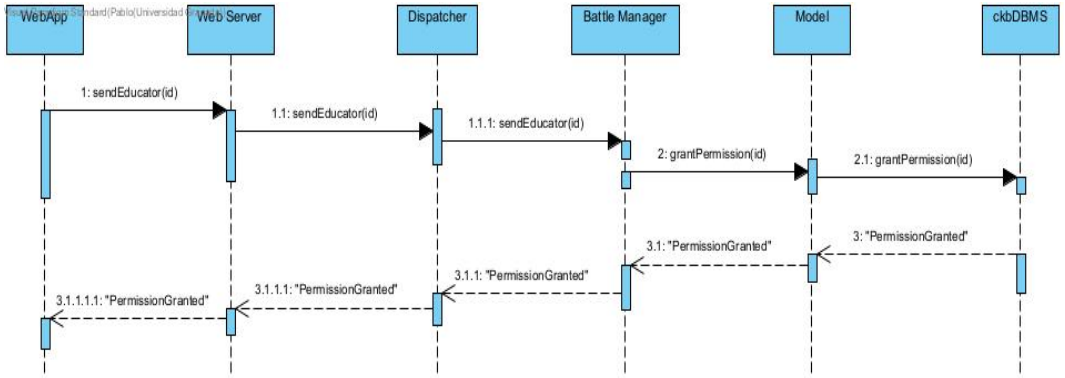


Figure 17: **Allow Other Educator to Create Battles View.**

- **Delete Badge:** The Delete Badge use case is called from the WebApp and sends the answer introduced by the user to Delete the Badge, the answer is then sent to the badge manager and checked by the model and CKB DBMS which will delete the badge if the answer was yes or cancel the operation otherwise while sending the respective success or error messages.

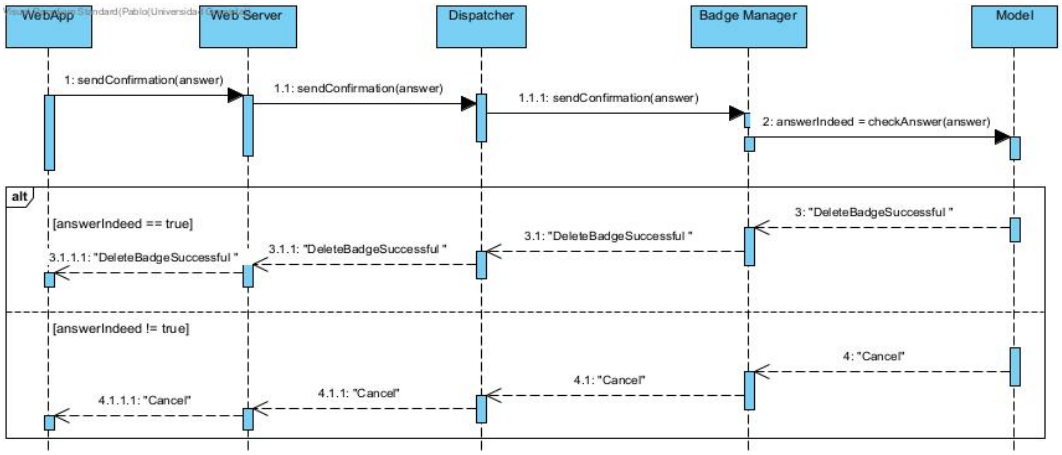


Figure 18: **Delete Badge View.**

- **Delete Requisite:** The Delete Requisite use case is called from the WebApp and sends the answer introduced by the user to Delete the Requisite, the answer is then sent to the requisite manager and checked by the model and CKB DBMS which will delete the requisite if the answer was yes or cancel the operation otherwise while sending the respective success or error messages.

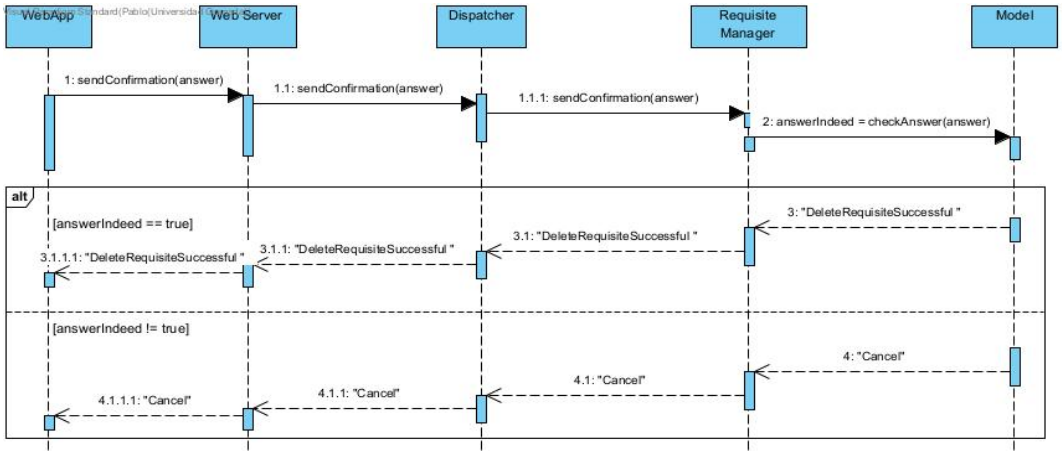


Figure 19: **Delete Requisite View.**

- **Create Group:** The Create Group use case is called from the WebApp and sends the information introduced by the student to create a group, that information is sent through the Web Server to the Group Manager, this component then tries to check whether the information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will create a student group with the information provided, otherwise the system will send an error message.

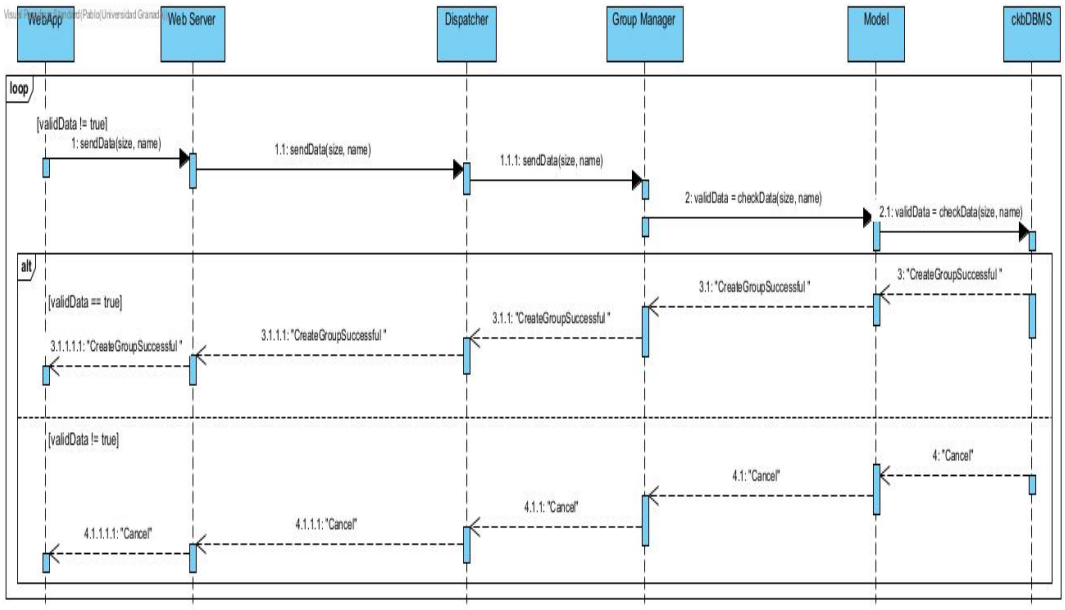


Figure 20: **Create Group View.**

- **Join Group:** The Join Group use case is called from the WebApp and sends the information introduced by the student to join one or several groups, that information is sent through the Web Server to the Group Manager which will reply by sending the requests to all the groups. Then there are two possible scenarios: either one group (the first to accept the request) or none of them reply/accept the request. In the first case the accepted request is sent to the group manager and from there the manager will join the student with the help of the Model and the CKB DBMS and send to the student the message telling him/her that he/she joined the group. In the second case the Group Manager will notify the student to try.

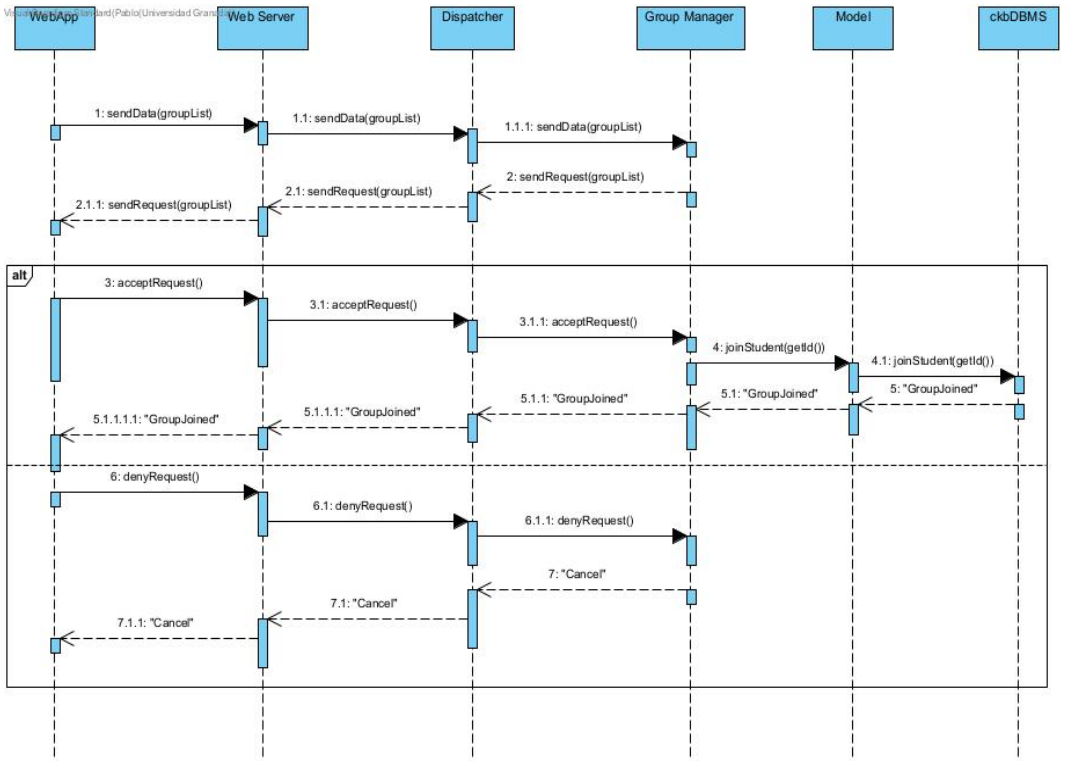


Figure 21: **Join Group View.**

- **Grant Badge:** The grant badge use case is called through the WebApp component every time a student fulfills one requisite and then the description of that message is sent through the Dispatcher to the Badge Manager that will check if that student has earned one of the badges that has that requisite (there are no more requisites to fulfil) and will grant it and notify the student.

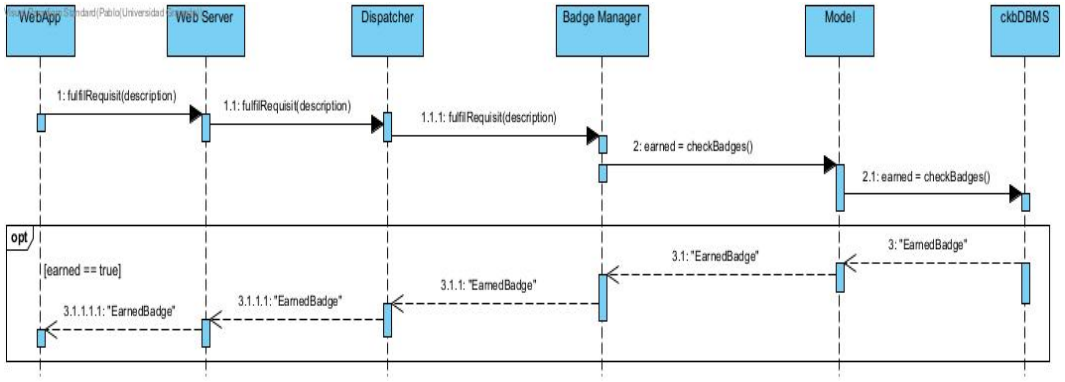


Figure 22: **Grant Badge View.**

- **Grant Badge:** The grant badge use case is called through the WebApp component every time a student fulfills one requisite and then the description of that message is sent through the Dispatcher to the Badge Manager that will check if that student has earned one of the badges that has that requisite (there are no more requisites to fulfil) and will grant it and notify the student.

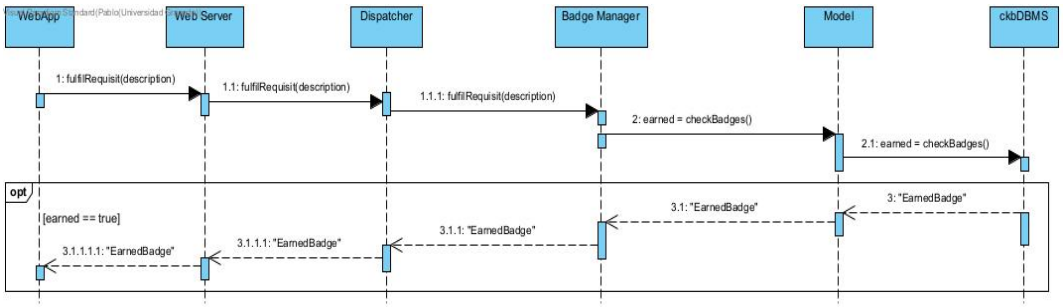


Figure 23: **Grant Badge View.**

- **Hand-In Submission:** The grant badge use case is called through the WebApp component every time a student fulfills one requisite and then the description of that message is sent through the Dispatcher to the Badge Manager that will check if that student has earned one of the badges that has that requisite (there are no more requisites to fulfil) and will grant it and notify the student.

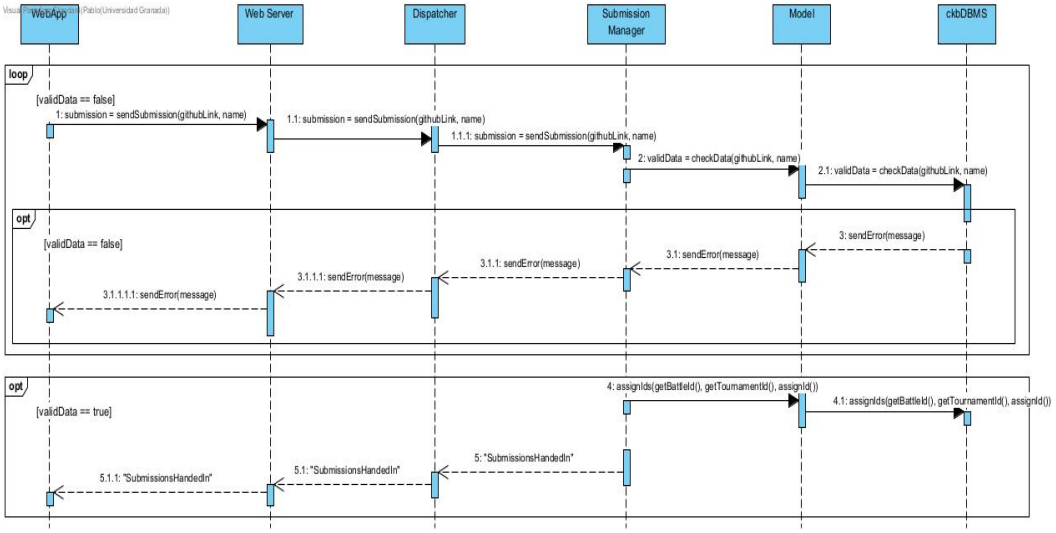


Figure 24: **Hand-In Submission View.**

- **Delete Submission:** The Delete Submission use case is called from the WebApp and sends the answer introduced by the user to Delete the Submission, the answer is then sent to the submission manager and checked by the model and CKB DBMS which will delete the submission/s while sending the respective success messages provided by the submission manager.

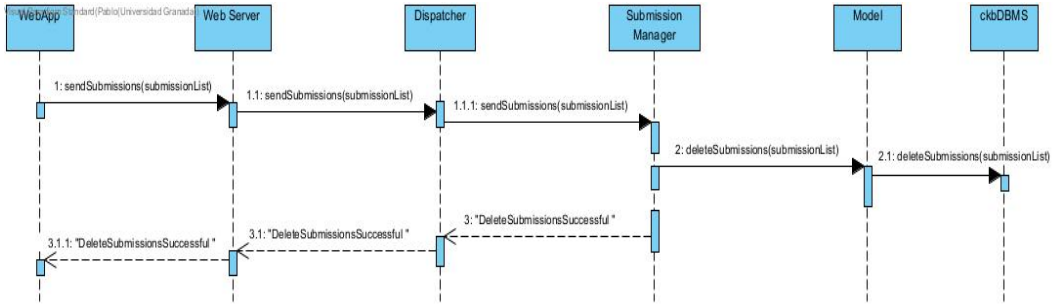


Figure 25: **Delete Submission View.**

- **Manually Score Submission:** The manually score use case is started by the WebApp which sends the id of the group to be scored through the Web Server and Dispatcher to the Submission Manager which searches for the group with the help of the Model and ckbDBMS which will send the group back to the educator in the WebApp. Then, the educator can score that group which will be sent to the submission manager. After that, the submission manager will update the score that will be stored in the CKB DBMS and a success message will be sent to the educator.

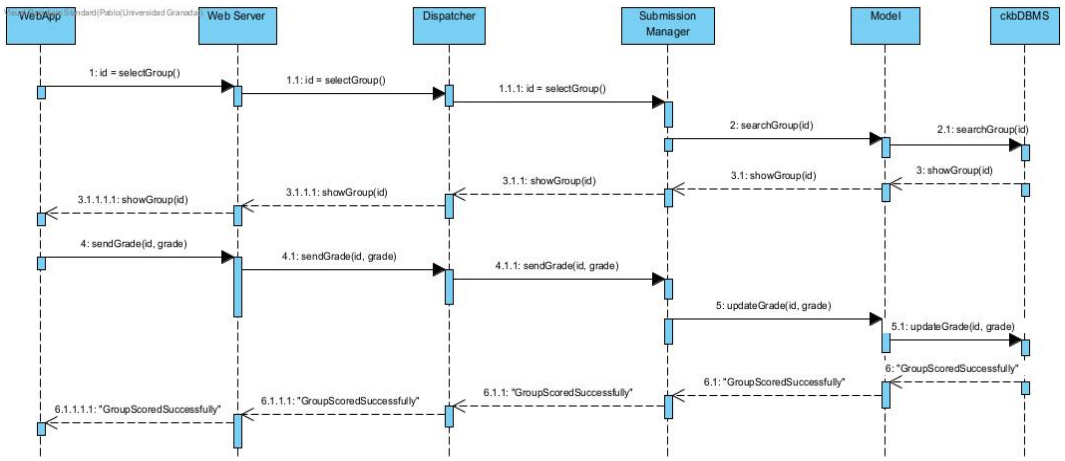


Figure 26: **Manually Score Submission View.**

- **Automatically Score Submission:** The automatically score use case is started by the Submission Manager which gets the deadline date of a submission. Then the model gets the actual date and sends it to the submission manager that compares both dates until the actual date is greater than the deadline date and scores the group submission and sends it to the CKB DBMS through the Model. The CKB DBMS will send the respective success message all the way back to the WebApp so that the educator in charge is aware.

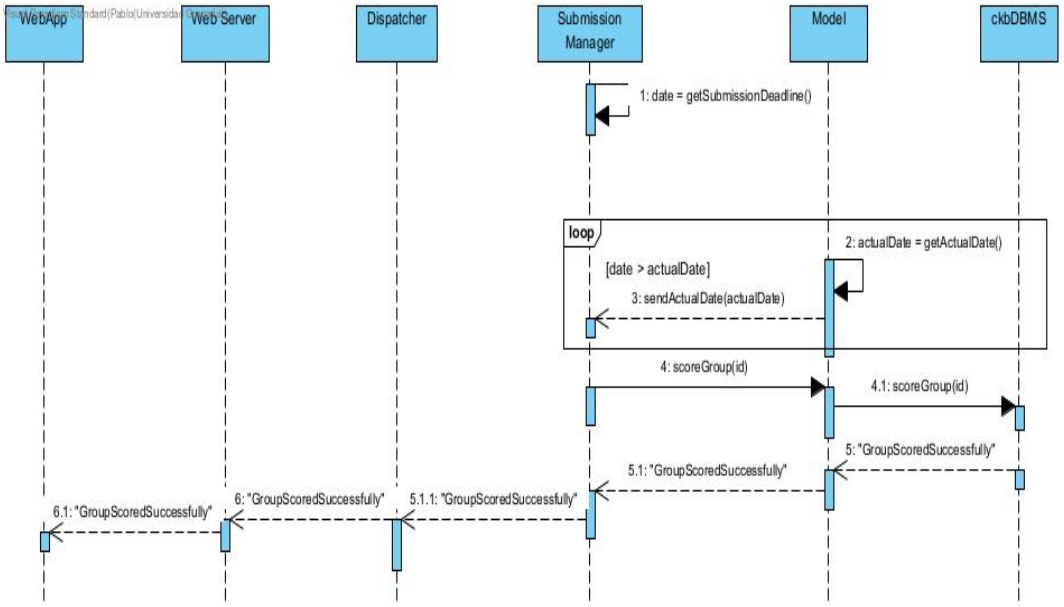


Figure 27: **Automatically Score Submission View.**

- **Show FAQs:** The Show FAQs use case is called through the WebApp to the Web Server which will reply by showing the FAQs screen.

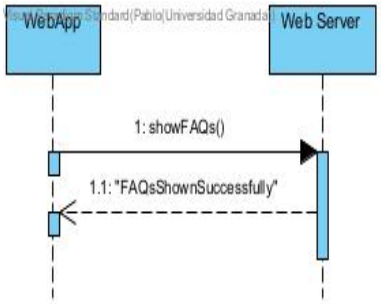


Figure 28: **Show FAQs View.**

- **Send Direct Message:** The send direct message use case is called through the WebApp component which sends the message and the user id through the Web Server to the Dispatcher to the Notifications Manager which if everything goes well will send the message to the respective user or an error message to the sender otherwise.

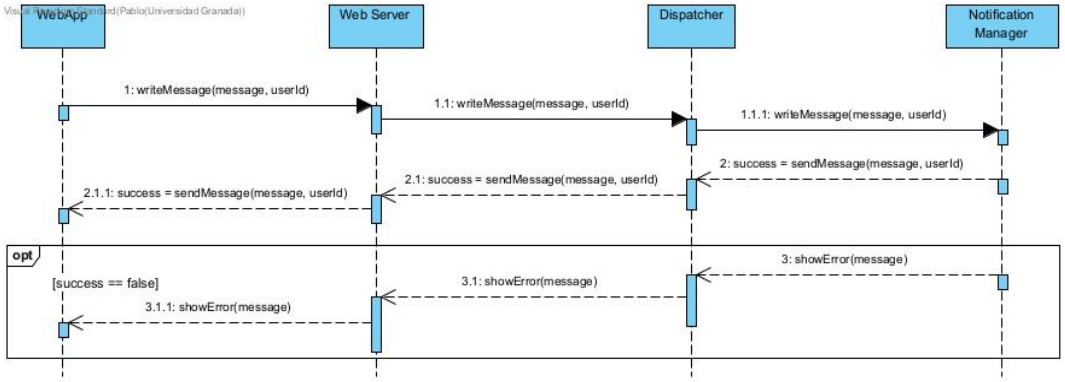


Figure 29: **Send Direct Message View.**

- **Update Profile:** The Update Profile use case is called from the WebApp and sends the updated data introduced by the user to update a profile's data, that data is sent through the Web Server to the Profile Manager, this component then tries to check whether the new data is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the data is in order the system will update the tournament with the information provided and a success message, otherwise the system will send an error message.

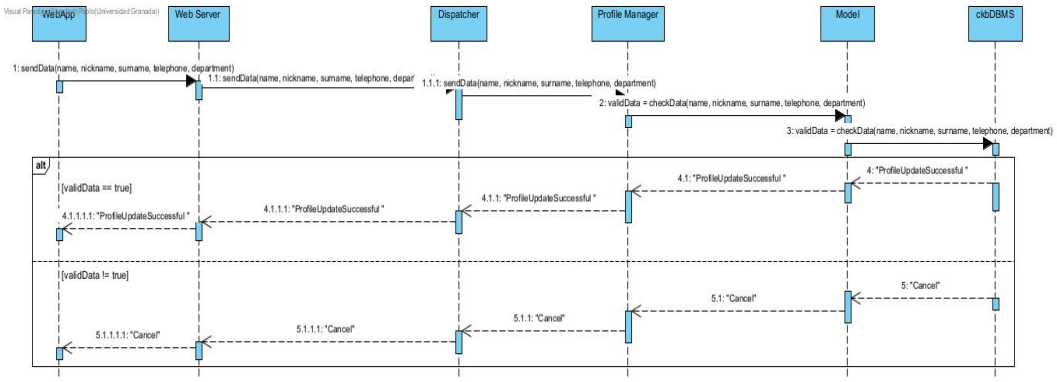


Figure 30: **Update Profile View.**

- **Expel Student From a Group:** The Expel Student From a Group use case is started by the WebApp where the educator selects the group where he/she wants to expel a student from, then the information is sent to the Group Manager through the Web Server and Dispatcher. The Group Manager sends the group information to the educator who will then ask to expel a student. The Group Manager will show to the educator the list of students who will choose the student to expel. That information will be received by the Group Manager who will expel the student with the help of the model and the CKB DBMS, the latter sending the corresponding success message.

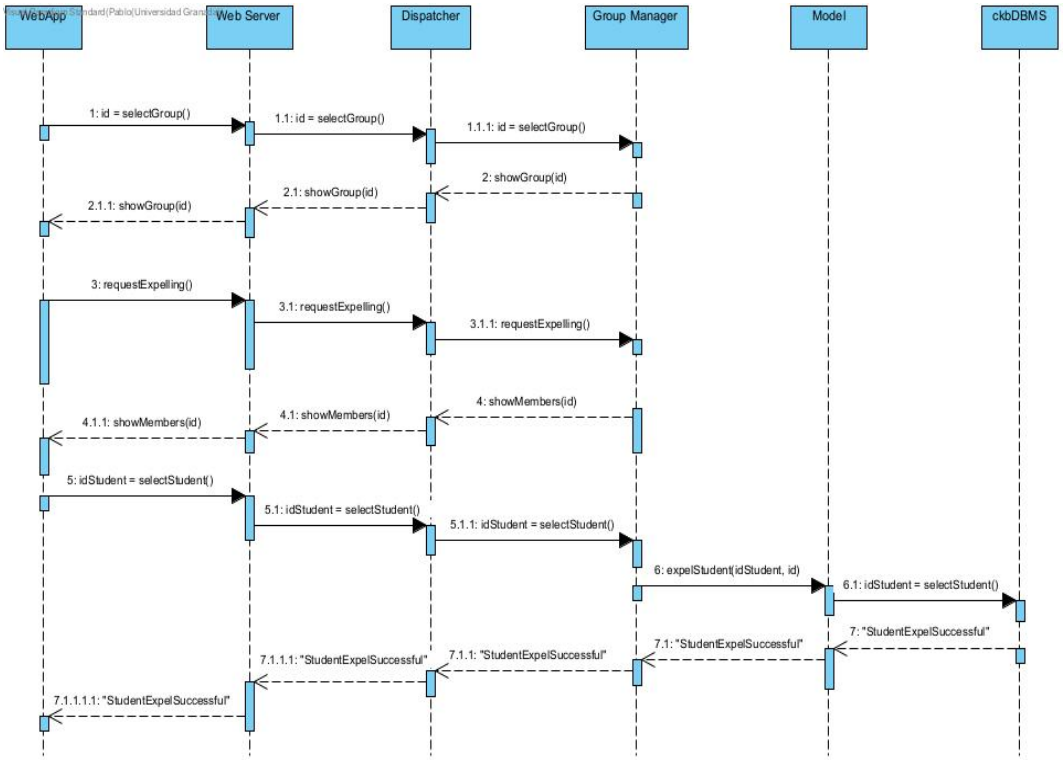


Figure 31: **Expel Student From a Group View.**

- **Update Battle Information:** The Update Battle Information use case is called from the WebApp and sends the updated information introduced by the educator to update a battle's information, that information is sent through the Web Server to the Battle Manager, this component then tries to check whether the new information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will update the battle with the information provided and a success message, otherwise the system will send an error message.

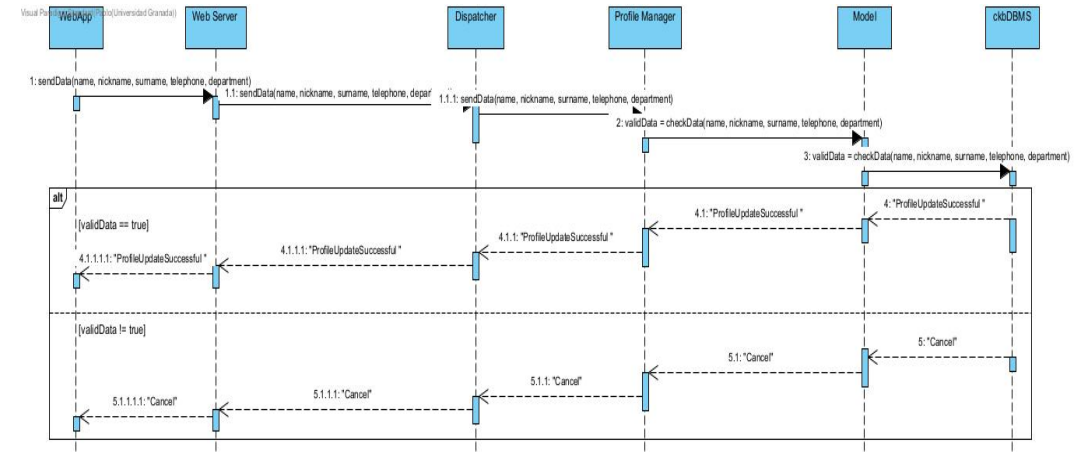


Figure 32: **Update Battle Information.**

- **Update Tournament Information:** The Update Tournament Information use case is called from the WebApp and sends the updated information introduced by the educator to update a tournament's information, that information is sent through the Web Server to the Tournament Manager, this component then tries to check whether the new information is correct by sending it to the Model and comparing it to the information in the CKB DBMS then if the information is in order the system will update the tournament with the information provided and a success message, otherwise the system will send an error message.

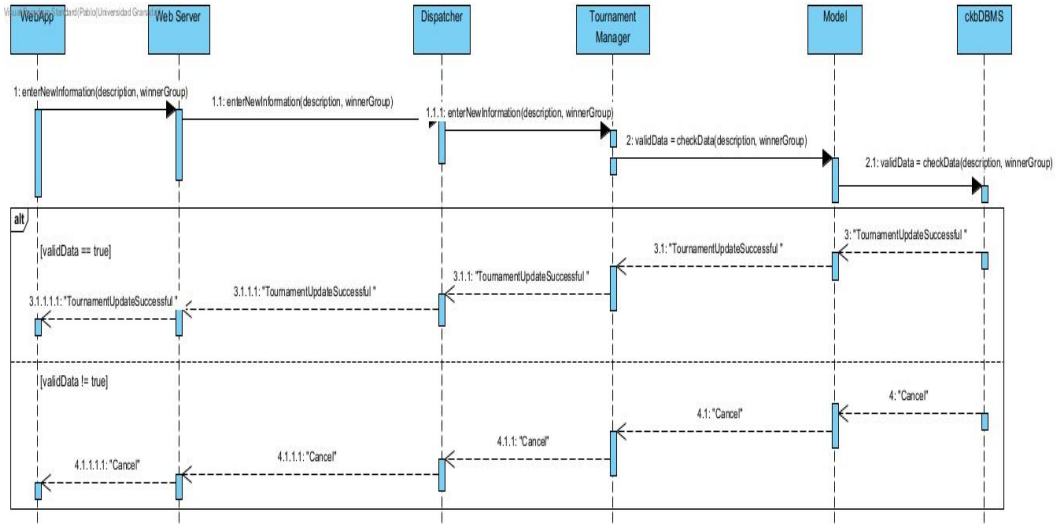


Figure 33: **Update Tournament Information View.**

- **Delete Group:** The Delete Group use case is called from the WebApp and sends the list of groups introduced by the educator to delete the respective groups, the list is then sent to the group manager and checked by the model and CKB DBMS which will delete the groups while the group manager sends the respective success message.

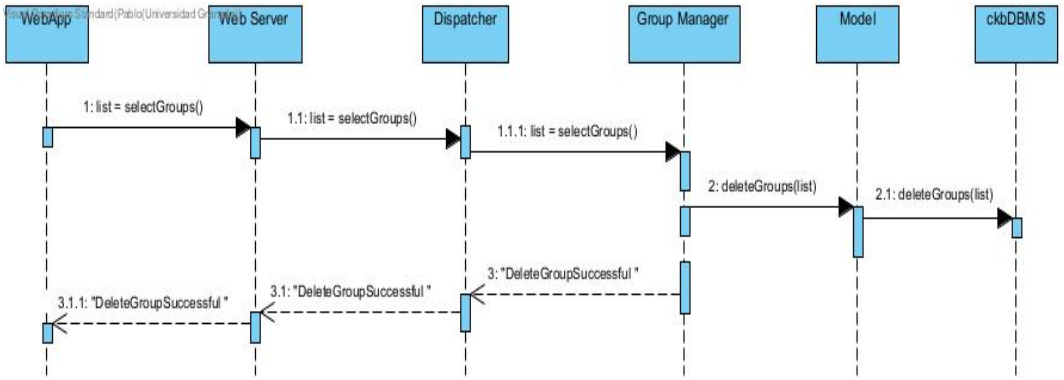


Figure 34: **Delete Group View.**

- **Abandon Group:** The Abandon Group use case is called from the WebApp and sends the answer introduced by the user to Abandon the Groups that he/she is an actual member, the list is then sent to the group manager and checked by the model and CKB DBMS which will drop out the student from the groups that he/she requested and where is possible to do so while sending the respective success message.

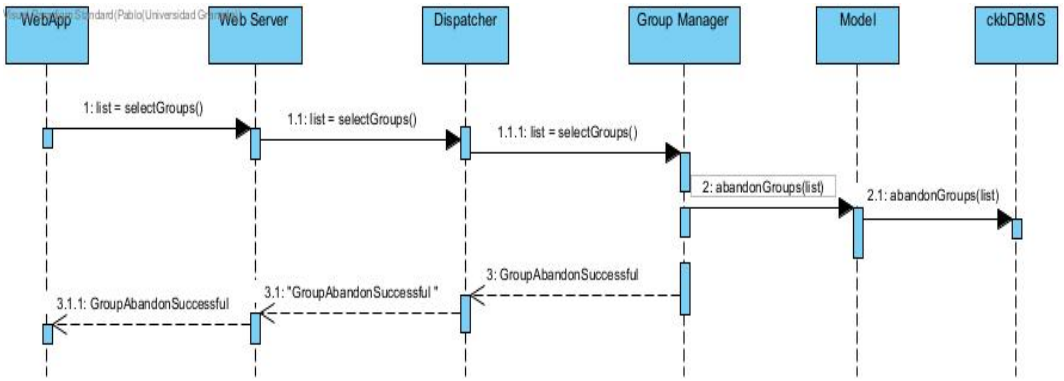


Figure 35: **Abandon Group View.**

2.5 Component interfaces

- **WebApp**
  - acceptRequest()
  - denyRequest()
  - fulfilRequisit(description)
  - sendEducator(id)
  - sendData(email, passwordConf, name, surname, nickname)
  - sendData(startingDate, expirationDate, description)
  - sendData(startingDate, expirationDate, description, githubLink)
  - sendData(name, description)
  - sendData(description, code)
  - sendData(size, name)
  - sendData(groupList)
  - sendData(name, nickname, surname, telephone, department)
  - sendCredentials(email, password)
  - sendConfirmation(answer)
  - sendRequisites(requisiteList)
  - sendSubmission(githubLink, name)
  - sendSubmissions(submissionList)
  - selectGroup()
  - selectGroups()
  - sendGrade(id, grade)
  - showFAQs()
  - writeMessage(message, userId)
  - requestExpelling()
  - selectStudent()
  - enterNewInformation(githubLink, description, winnerGroup)
  - enterNewInformation(description, winnerGroup)
- **Web Server**
  - acceptRequest()

  - denyRequest()
  - fulfilRequisit(description)
  - sendEducator(id)
  - sendData(email, passwordConf, name, surname, nickname)
  - sendData(startingDate, expirationDate, description)
  - sendData(startingDate, expirationDate, description, githubLink)
  - sendData(name, description)
  - sendData(description, code)
  - sendData(size, name)
  - sendData(groupList)
  - sendData(name, nickname, surname, telephone, department)
  - sendCredentials(email, password)
  - sendConfirmation(answer)
  - sendRequisites(requisiteList)
  - sendSubmission(githubLink, name)
  - sendSubmissions(submissionList)
  - selectGroup()
  - selectGroups()
  - sendGrade(id, grade)
  - showFAQs()
  - writeMessage(message, userId)
  - requestExpelling()
  - selectStudent()
  - enterNewInformation(githubLink, description, winnerGroup)
  - enterNewInformation(description, winnerGroup)
  - sendRequest(groupList)
  - sendError(message)
  - showGroup(id)
  - showMembers(id)
  - sendMessage(message, userId)

- **Dispatcher**

- acceptRequest()
- denyRequest()
- fulfilRequisit(description)
- sendEducator(id)
- sendData(email, passwordConf, name, surname, nickname)
- sendData(startingDate, expirationDate, description)
- sendData(startingDate, expirationDate, description, githubLink)
- sendData(name, description)
- sendData(description, code)
- sendData(size, name)
- sendData(groupList)
- sendData(name, nickname, surname, telephone, department)
- sendCredentials(email, password)
- sendConfirmation(answer)
- sendRequisites(requisiteList)
- sendSubmission(githubLink, name)
- sendSubmissions(submissionList)
- selectGroup()
- selectGroups()
- sendGrade(id, grade)
- showFAQs()
- writeMessage(message, userId)
- requestExpelling()
- selectStudent()
- enterNewInformation(githubLink, description, winnerGroup)
- enterNewInformation(description, winnerGroup)
- sendRequest(groupList)
- sendError(message)
- showGroup(id)
- showMembers(id)
- sendMessage(message, userId)

- **Group Manager**
  - getName()
  - getSize()
  - checkData(size, name)
  - sendRequest(groupList)
  - joinStudent(id)
  - getId()
  - showGroup(id)
  - showMembers(id)
  - expelStudent(idStudent, id)
  - deleteGroups(list)
  - abandonGroup(list)
- **Tournament Manager**
  - getDescription()
  - getWinnerGroup()
  - getStartingDate()
  - getExpirationDate()
  - checkData(startingDate, expirationDate, description)
  - checkAnswer(answer)
  - checkData(description, winnerGroup)
- **Battle Manager**
  - getDescription()
  - getGithubLink()
  - getWinnerGroup()
  - getStartingDate()
  - getExpirationDate()
  - checkData(startingDate, expirationDate, description, githubLink)
  - checkData(githubLink, description, winnerGroup)
  - checkAnswer(answer)

  - grantPermission(id)

- **Badge Manager**
  - getName()
  - getDescription()
  - checkData(name, description)
  - checkAnswer(answer)
  - checkBadges()
  - createBadge(requisiteList)

- **Requisite Manager**
  - getDescription()
  - getCode()
  - checkData(description, code)
  - checkAnswer(answer)

- **Profile Manager**
  - checkData(email, password, passwordConf, name, surname, nickname)
  - checkData(name, nickname, surname, telephone, department)
  - checkCredentials(email, password)
  - checkAnswer(answer)

- **Submission Manager**
  - getSubmissionDeadline()
  - checkData(githubLink, name)
  - sendError(message)
  - scoreGroup(id)
  - searchGroup(id)
  - showGroup(id)
  - updateGrade(id, grade)
  - deleteSubmissions(submissionList)
  - assignIds(idbattle, idtournament, idsubmission)
  - getBattleId()

  - getTournamentId()
  - assignId()

- **Notification Manager**
  - showError(message)
  - sendMessage(message, userId)

- **Model**
  - checkData(email, password, passwordConf, name, surname, nickname)
  - checkData(name, nickname, surname, telephone, department)
  - checkData(startingDate, expirationDate, description)
  - checkData(startingDate, expirationDate, description, githubLink)
  - checkData(name, description)
  - checkData(description, code)
  - checkData(size, name)
  - checkData(githubLink, name)
  - checkData(githubLink, description, winnerGroup)
  - checkData(description, winnerGroup)
  - sendError(message)
  - deleteSubmissions(submissionList)
  - searchGroup(id)
  - showGroup(id)
  - checkCredentials(email, password)
  - checkConditions()
  - updateGrade(id, grade)
  - createBadge(requisiteList)
  - grantPermission(id)
  - joinStudent(id)
  - getId()
  - checkBadges()
  - sendActualDate(date)
  - getActualDate()

  - scoreGroup(id)
  - selectStudent()
  - deleteGroups(list)
  - abandonGroups(list)

- **ckbDBMS**
  - getName(id)
  - getSurname(id)
  - getDepartment(id)
  - getTournament(startingDate)
  - getBarttle(startingDate)
  - showGroup(id)
  - sendError(message)

2.6 Selected architectural Styles and patterns

2.6.1 3-tier Architecture

CodeKataBattle will be developed using a three-tier architecture, which offers numerous advantages through the system's division into three distinct, independent layers:

- **Presentation Tier:** Presentation Tier: It is the first layer of the system, presenting the user interface. It gets its information from the Application Tier, interprets it, and displays it in a user-friendly and understandable way.
- **Application Tier:** It is the second layer of the system, composed of the business logic components. Its goal is to use the data provided by the data layer and do the necessary computations before sending them to the Presentation Layer to be displayed.
- **Data Tier:** It is the third layer of the system. It is made of the database system data, for data storage, access, and manipulation.

This architecture enables easier development, with a separation of concerns, reading to an isolated component implementation. Moreover, the presence of an intermediary tier between the client and the data server adds an extra layer of security, as data is retrieved via the Application Tier rather than being directly accessed by the client.

2.6.2 Model View Controller pattern

Both the web page and the web application for the CodeKataBattle system will be developed according to the Model View Controller design pattern. It is characterized by three core components:

- **Model:** Holds the application's data and the methods required to manipulate this data.
- **View:** Concerns all the various ways in which the data can be visually presented to the user.
- **Controller:** Acts as an intermediary between the Model and the View. It responds to user inputs, such as button clicks, by executing predefined actions, which often involve modifying the Model. These modifications are then propagated back to the View.

By clearly dividing responsibilities among these three components, the MVC pattern enhances the separation of concerns, resulting in improved component modularity. This separation allows for benefits like increased reusability and simplified implementation.

2.6.3 Facade pattern

The facade pattern is employed, by using a Dispatcher component. This component makes it easy for the client to call functions, by presenting a complete and as simple as possible interface. This enhances the encapsulation since the clients do not need to know anything about which components are supposed to execute the current task. The Dispatcher takes care of this internally.

2.6.4 Mediator pattern

At several moments in the design of CodeKataBattle, we used the Mediator design pattern. Since we know that the different components of a system must have the lowest coupling possible, and therefore not depend on the other components' implementations, we decided to include managers between actual component implementations and real implementations. This was important to be able to take care of the logic before sending it to the next component. This way changing the implementation or the services will not impact the rest of the system because the interface of the manager will remain the same. All that will need to be changed is the internal manager logic, which shows the reduced coupling and impact of modifications. The managers that we included are:

- The ProfileManager is the mediator linking the components of the system to the Authentication Service. This way, if the service we chose was meant to change throughout the life of the system, the ProfileManager would be the only component to be changed.

- The ChallengeManager and the GradeManager are the mediators respectively in the first case for the BattleManager and the Tournament Manager, and in the second case for the RequisiteManager and the SumissionManager. They group both of their components and take care of the logical demands, calling one or another. This also applies the facade pattern in this case, making a simpler interface for users of these components. We can add to this the fact that they also serve as mediators for the GitHubService, which might evolve through time.

- The NotificationManager which also is a mediator for the email provider. In the same way as the ProfileManager, it links the EmailProviderService to the rest of the system and makes it usable, regardless of its implementation.

2.7 Other design decisions

2.7.1 Dual Interface for Performance Optimization

As we presented in the previous sections, we implemented two interfaces for the presentation layer: the web page and the web app. The goal is to enhance the performance of the system, providing more resources to the users actively taking part in a battle or creating/updating a tournament/battle, while still allowing browsing through the platform. The web app serves the first purpose and therefore is a component connected to most of the business logic and services components as shown in the component diagrams. The web page serves the second purpose and is lightly connected to business logic, mostly using the webserver to make queries. This way we enhance the performance and service availability to more users at the same time, by scaling the resources to the need of the task.

2.7.2 Data storage

We decided to include a replication mechanism, having the data stored in multiple database units, with different database managers. The data will be the same in both units. Having two units and two managers allows to separate the concerns between the two flows stated earlier. It adds system resilience: in case one of the systems breaks, the other flow is still able to provide the required data.

3 User Interface Design

In the first figure we can see the profile of a certain user. He/She can easily see his/her personal information such as the nickname, name, surname, etc. Besides, the user can click on the options "Log out" or "Delete Account" or click in other sections like the user's enrollments or the profile settings, and so on and so forth.

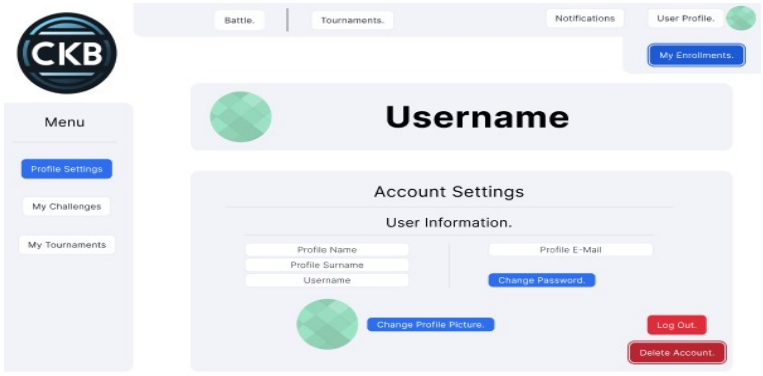


Figure 36: **Profile Interface.**

If the user clicks on "Log Out", the system will show the following screen asking for confirmation. If the user confirms it, then the system will close the current session, in the second case, the screen will be closed and the system will look like it did in the previous figure.

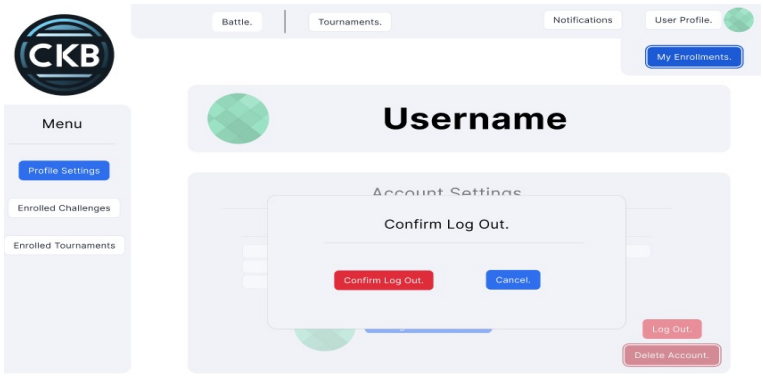


Figure 37: **Log Out Confirmation Interface.**

If the user clicked "Delete Account" the the following screen will pop up and the user will be kindly asked to either agree or disagree with deleting the account by clicking on "Yes, delete everything" (the former) or "Cancel" (the latter) which will close the screen and the system will look like in the first image.

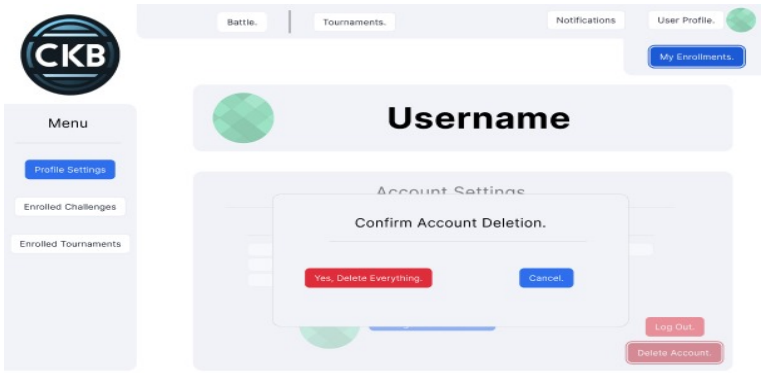


Figure 38: **Delete Account Confirmation Interface.**

When anyone opens the CKB website, the first thing that appears is the log in screen where the user is required to insert his/her email and password, that is, his/her credentials in order to have access to the profile if the credentials are correct (authentication).

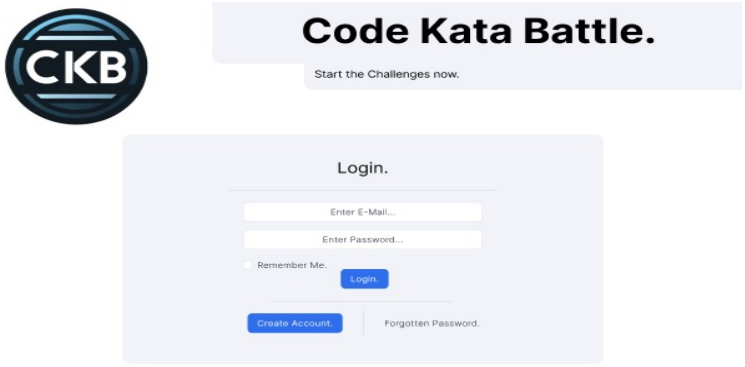


Figure 39: **Log In Interface.**

If the person that wants to use the website does not have an account, she can decide to create an account by clicking on the "Create Account" button which will open the following screen where the potential user is now required to enter her name, nickname, surname, email address and write the password twice. Once she is finished, clicking in the "Sign Up" button will create the account if everything is okay.

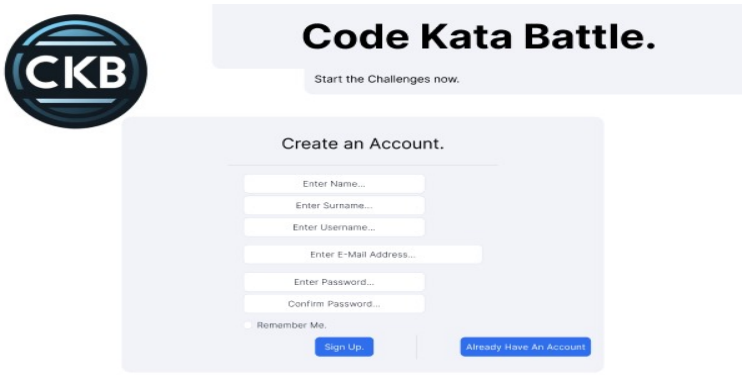


Figure 40: **Create Account Interface.**

An educator can create a tournament by clicking on the "Tournaments" button. He/She then will be asked to enter the tournament title, description and both starting and expiration dates. Not only can he/she create new tournaments but also check the state of current and past ones.

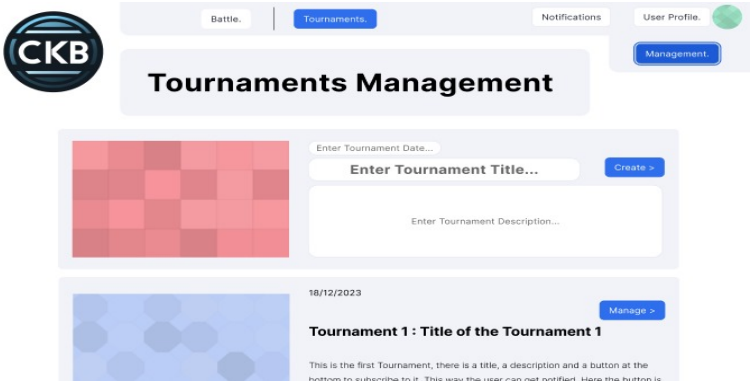


Figure 41: **Create Tournament Interface.**

The educator that created a tournament can allow other educators to create battles within that tournament. In order to do so, he/she has to click on the "Educators" button in a specific tournament and the enter the email of those educators that will be granted access. And also the educator can check the current educators that can create battles and can decide to remove that privilege by clicking on the "Delete Educator" button.

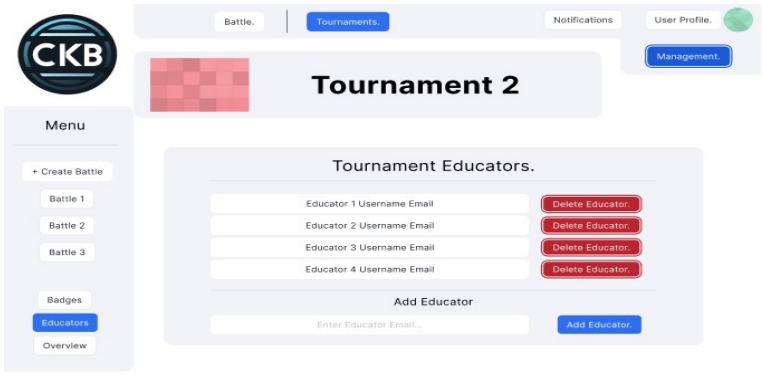


Figure 42: **Add Educator Interface.**

Each tournament has an overview screen where an educator can see a graph with several statistics and decide to manually score all the groups that are participating there.

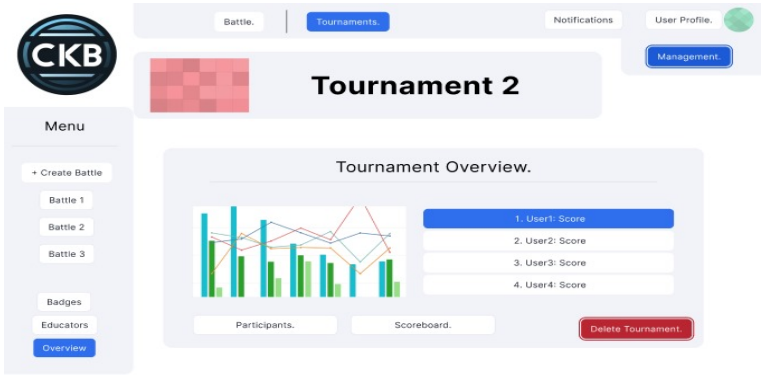


Figure 43: **Tournament Overview Interface.**

The educator that created that tournament will also have the "Delete Tournament" button. As the name indicates, if the educator clicks on it, the system will ask if he/she is sure to do so. If the educator clicks on "Confirm Deletion" the tournament will be deleted and nothing will happen if the educator finally decides to click on "Cancel".

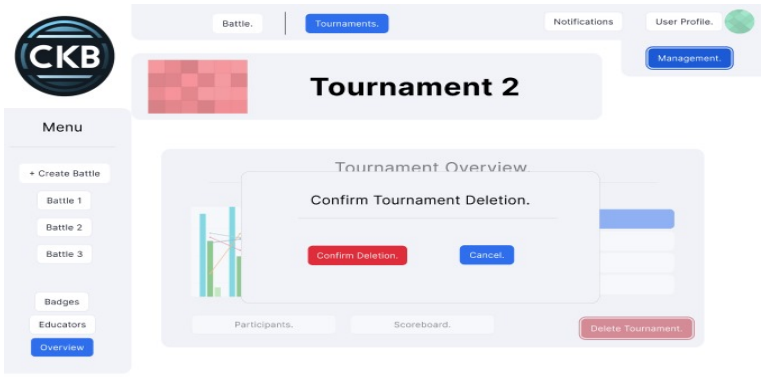


Figure 44: **Delete Tournament Confirmation Interface.**

Any educator that can create battles in a specific tournament can do so by clicking on the "Create Battle" button which will display the following screen where a battle can be created by introducing a description, name, expiration date, starting date and a github link and finally clicking on the "Proceed" button.

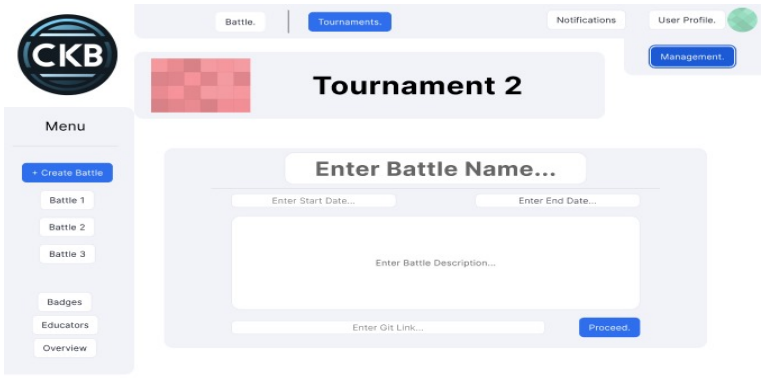


Figure 45: **Create Battle Interface.**

A student group can hand in a submission and check the status, whether it passes the several tests provided and look at previous submissions. In addition, a group can decide to delete a submission by clicking on the "Delete Submission" button.

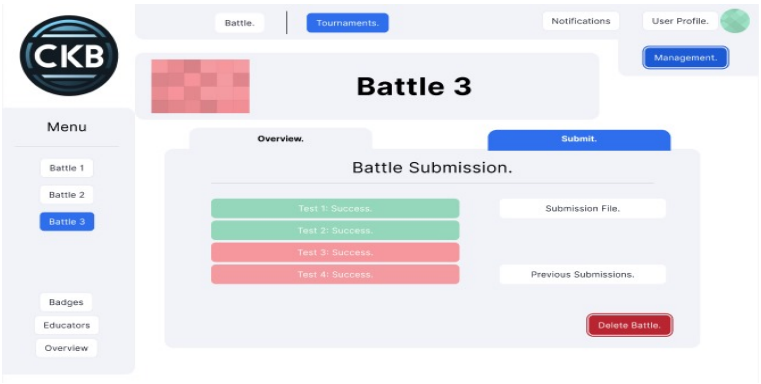


Figure 46: **Submission Interface.**

As it happened with the tournament overview screen, any battle also has its respective battle overview screen that can be accessed by any user by clicking on the respective battle button. Besides, if the user is an educator too, he/she can manually rate each group's performance in the battle if the educator created the battle.

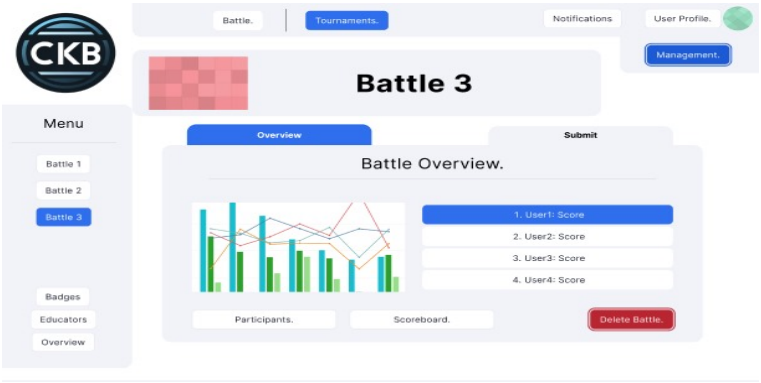


Figure 47: **Battle Overview Interface.**

The educator that created a battle can then delete it too if he/she wishes to do so. In order to do it, he/she needs to click on the corresponding battle and then click on the "Delete Battle" button. A confirmation screen will then appear and the procedure will be the same as in previous confirmation screens already explained above.

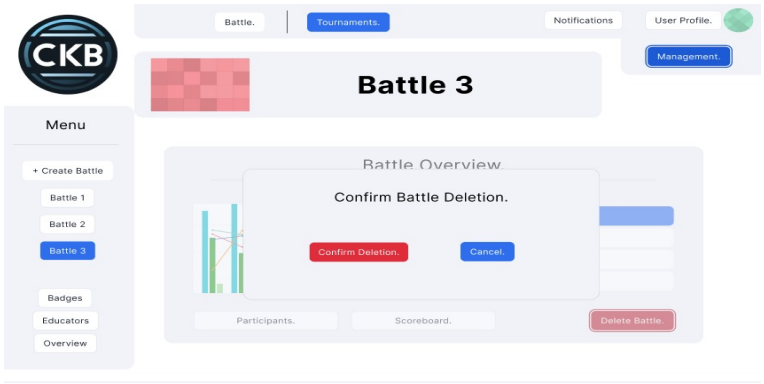


Figure 48: **Delete Battle Confirmation Interface.**

The system allows students to subscribe to challenges that will grant them badges if they fulfill all the requisites. They can also unsubscribe to those that they have already been subscribed to by clicking to the "Unsubscribe" button or to access the details of the challenge by clicking on the "Access ¿" button.

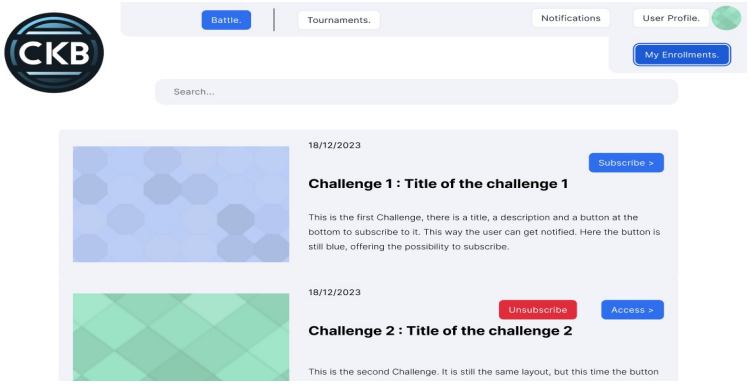


Figure 49: **Challenges Interface.**

An educator can check the badges of a tournament and decide to delete them if he/she is the author of them by clicking on the "Delete Badge" button which will erase them from the system. An educator can also decide to create a new badge by clicking on the "Add Badge" button.

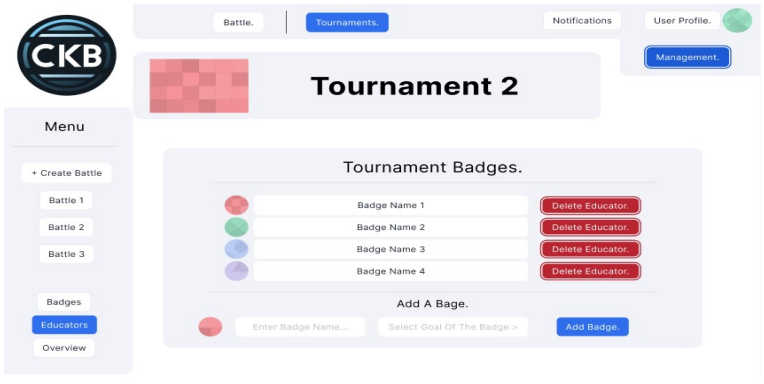


Figure 50: **Tournament Badges Interface.**

Finally, any user can check his/her notifications by clicking on the "Notifications" button that will display all the notifications that have been sent to that user in particular or in general to all users.

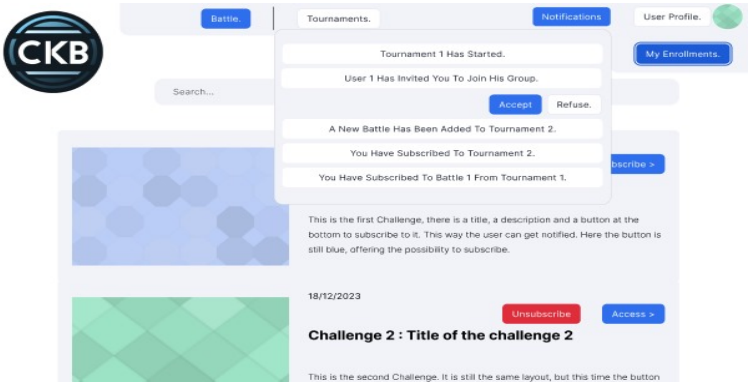


Figure 51: **Notifications Interface.**

4 Requirement traceability

Requirements:	[**R1**]: CKB allows an unregistered user that is a I.I.S.S. Bertrand Russel educator to create an educator user account [**R2**]: CKB allows an unregistered user that is a I.I.S.S. Bertrand Russel student to create a student user account
Components:	• WebApp • Web Server – Dispatcher – Profile Manager – Model • ckbDBMS

Requirements:	[**R3**]: CKB allows all users that already have an account to log in
Components:	• WebApp • Web Server – Dispatcher – Profile Manager – Model • ckbDBMS

Requirements:	[**R4**]: CKB allows all users to log out
Components:	• WebApp • Web Server – Dispatcher – Profile Manager – Model

Requirements:	[**R5**]: CKB allows educators to create new badges
Components:	• WebApp • Web Server – Dispatcher – Badge Manager – Model • ckbDBMS

Requirements:	[**R6**]: The platform allows educators to create new requisites
Components:	• WebApp • Web Server – Dispatcher – Requisite Manager – Model • ckbDBMS

Requirements:	[**R9**]: CKB will allow users to update their own profiles which means adding, deleting and/or changing information that was already available

Components:	• WebApp
	• Web Server
	  – Dispatcher
	  – Profile Manager
	  – Model
	• ckbDBMS

Requirements:	[**R10**]: CKB will allow users to delete their account under certain conditions specified in the Class Diagram if the user is a student and without restrictions if it is an educator

Components:	• WebApp
	• Web Server
	  – Dispatcher
	  – Profile Manager
	  – Model
	• ckbDBMS

Requirements:	[**R11**]: The platform automatically scores a group's submission in a battle if a manual scoring is not specified
Components:	• WebApp • Web Server – Dispatcher – Submission Manager – Model • ckbDBMS

Requirements:	[**R12**]: The system lets educators manually score a group's performance in a submission if they wish to
Components:	• WebApp • Web Server – Dispatcher – Submission Manager – Model • ckbDBMS

Requirements:	[**R13**]: CKB allows educators to create tournaments in which several code kata battles can take place
Components:	• WebApp • Web Server – Dispatcher – Tournament Manager – Model • ckbDBMS

Requirements:	[**R14**]: An educator can create one or more battles within a tournament
Components:	• WebApp • Web Server – Dispatcher – Battle Manager – Model • ckbDBMS

Requirements:	[**R15**]: An educator can delete a tournament that has not finished yet
Components:	• WebApp • Web Server – Dispatcher – Tournament Manager – Model • ckbDBMS

Requirements:	[**R16**]: An educator can delete a battle that has not finished yet
Components:	• WebApp • Web Server – Dispatcher – Battle Manager – Model • ckbDBMS

Requirements:	[**R17**]: An educator can allow other educators to create battles in a specified tournament
Components:	• WebApp • Web Server – Dispatcher – Battle Manager – Model • ckbDBMS

Requirements:	[**R18**]: CKB allows any student to create a group that can join tournaments and consequentially, battles in the tournament
Components:	• WebApp • Web Server – Dispatcher – Group Manager – Model • ckbDBMS

Requirements:	[**R19**]: CKB allows any student to join a vacant group
Components:	• WebApp • Web Server – Dispatcher – Group Manager – Model • ckbDBMS

Requirements:	[**R20**]: CKB allows a student that belongs to a group to abandon such a group
Components:	• WebApp • Web Server – Dispatcher – Group Manager – Model • ckbDBMS

Requirements:	[**R21**]: The platform lets an educator expel a student from a group
Components:	• WebApp • Web Server – Dispatcher – Group Manager – Model • ckbDBMS

Requirements:	[**R23**]: The system allows student groups to hand in several submissions for each battle, although only the last one to be submitted will be scored
Components:	• WebApp • Web Server – Dispatcher – Submission Manager – Model • ckbDBMS

Requirements:	[**R24**]: The system allows student groups to delete submissions if they wish to, although it is strongly discouraged
Components:	• WebApp • Web Server – Dispatcher – Submission Manager – Model • ckbDBMS

Requirements:	[**R25**]: CKB allows educators to update the information (description, winner_group...) of any tournament
Components:	• WebApp • Web Server – Dispatcher – Tournament Manager – Model • ckbDBMS

Requirements:	[**R26**]: CKB allows educators to update the information (description, github_link...) of any battle
Components:	• WebApp • Web Server – Dispatcher – Battle Manager – Model • ckbDBMS

Requirements:	[**R27**]: The system will automatically grant a badge to a student that has fulfilled the needed requisites
Components:	• WebApp • Web Server – Dispatcher – Badge Manager – Model • ckbDBMS

Requirements:	[**R32**]: Users can check FAQs to understand how the platform works in case they are not sure how they can do something
Components:	• WebApp • Web Server

Requirements:	[**R33**]: The platform will allow users to send direct messages amongst them
Components:	• WebApp • Web Server – Dispatcher – Notification Manager

Requirements:	[**R39**]: The system allows the educator that created a badge to erase it
Components:	• WebApp • Web Server – Dispatcher – Badge Manager – Model

Requirements:	[**R40**]: The system allows the educator that created a requisite to erase it
Components:	• WebApp • Web Server – Dispatcher – Requisite Manager – Model

5 Implementation, Integration and Test Plan

5.1 Overview

The implementation plan for the CodeKataBattle Project is aimed at creating a comprehensive, engaging, and educational platform for software development learning. This plan will be executed in phases, each focusing on a specific aspect of the system. Starting with the foundational infrastructure, the plan will gradually incorporate more complex features such as coding challenges, tournaments, and community engagement tools. Emphasis will be on user experience, scalability, and continuous improvement based on feedback. The goal is to build a platform that not only facilitates learning through challenges but also fosters a supportive community of learners and educators.

Now, let's move on to the detailed implementation plan.

5.2 Implementation plan

The implementation plan that was the best suited for this system seemed to be the top-down approach. After having tested each of the components, we would start by implementing the core elements and then, adding more and more elements to the subsystem. This would be a structural implementation approach, and it would provide continuity in testing, paired with easier bug tracking.
However, we thought about combining it with a thread-based approach, to provide visibility to the stakeholders in real-time and be able to make modification decisions as soon as possible if they are needed.
Adding this second approach would allow us to get feedback and see our evolution throughout the implementation, but it also requires us to select the elements from

the subsystem that we want to implement first.
We therefore decided to first identify the different features, and then based on the results, we would decide on the threads we want to implement, each of them in a top-down approach.

5.2.1 Features identification

Here are the main features that we identified in the CodeKataBattle system. These are deduced from the RASD document of CodeKataBattle.

[F1]: Sign Up and Log In: These two features are the usual account creation and then logging in on the platform. The difference between the Educator and the Student Version will be that the Students can freely sign up, whereas the System Manager will need to grant access to registration for the Educators, changing a value to their user line in the database. This feature will be using an external Authentication Service; it is a bottom feature, easily testable.

[F2]: Update and Delete a Profile Account: This feature is related to the previous one, it is about changing the elements in the profile line of the database, or deleting it completely, causing a cascade for the other related elements (such as submissions, scores...). This again is a bottom feature.

[F3]: Create and Manage a Tournament: This is one of the core features of the system. It is only available for the Educators. This feature is required for the following ones, for the Student to join tournaments.

[F4]: Create and Manage a Battle: This is also one of the core features of the system, only available for the Educators. This feature is required for the Student to be able to join Battles and submit.

[F5]: Create and Grant Badge: This feature of the system is important for the gamification aspect of CodeKataBattle which rewards the students who complete the specific criteria in tournaments or battles. This feature creates a badge with a name and description and links it to the student's profile once all the criteria are met.

[F6]: Join or Withdraw from a Tournament or a Battle: This feature enables students to join or withdraw from tournaments and battles. This feature includes the implementation of the GitHub Service which should be able to create

and delete a GitHub repository. This feature is required for the features concerning the Grading and the Granting of Badges.

[F7]: Join a Group: This feature is designed to promote teamwork within CodeKataBattle by enabling students to become members of a group or to start their own. It offers the tools for students to locate groups that are open for new members or to establish a new group to collaborate with peers. Upon joining, students are associated with the group for collective participation in battles and tournaments. The process is managed by the Group Manager component, which maintains the link between students and their respective groups in the system's records.

[F8]: Grading Submissions: This feature lets the teacher score the work that students turn in for battles and tournaments. It is a big part of the CodeKataBattle because it measures how well students are performing. It comprises automatic grading based on predefined criteria and manual grading for more subjective assessments.
The Submission Manager component is responsible for retrieving submissions from the linked GitHub repositories, applying the grading norms, and updating the students' scores. This component also interfaces with the Notification Manager to inform students of their grades. The feature is crucial to ensure fair and transparent grading.

[F9]: Receiving and Sending Notifications: This feature is vital as it keeps the users up-to-date on all the CodeKataBattle events. It sends alerts for new battles, tournament updates, or badge awards. The system's Notification Manager partners with an Email Provider to ensure timely email communication. It's essential for user engagement on the platform.

5.2.2 Implementation Flows

To apply the thread approach, we are going to define some implementation flows that will be used to showcase our progress to the stakeholders. These threads will initially present distinct functionalities of the platform, but will then be connected as in the top-down approach.

Flow 1 - Creating an account, updating information, and deleting it : This flow will target features [F1] and [F2], showcasing the user profile system, for both the Educators and the Students. This will include showing the screens of the web page corresponding to the profile creation, account settings, and delete

account, and will involve the authentication service and the Model and CKBDBMs components for the Accounts.

Flow 2 - Creating a Tournament and a Battle, updating them, and deleting them: This flow targets the features [F3] and [F4] and will showcase the respective functionalities of the system. This will involve screens from the WebApp such as Tournament Creation, Battle Creation, Tournament Deletion, Battle Deletion, and Battle Update. It will also require the implementation of the corresponding parts of the TournamentManager, BattleManager, Model, and CKBDBMs components, for storing the Tournaments and the Battles.

Flow 3 - Joining a Tournament and a Battle as a Student, and Joining a Group: This flow targets the features [F6], [F7], and [F9]. It will showcase the Student side of the Battle and Tournament functions. It will involve parts of the Web Page Component, such as the Battle and Tournament Browsing pages, and the Notification Page. This flow will involve parts of the TournamentManager, BattleManager, GroupManager, Model and CKBDBMs (concerning the subscriptions), and NotificationManager, and therefore the Email Provider Service.

Flow 4 - Submission and Grading: This flow targets the features [F8] and part of [F6], this flow outlines the process of student submission for battles and the subsequent grading by educators. It showcases the SubmissionManager's role in retrieving the submissions from GitHub repositories and applying the grading rules. The flow also includes interaction with the NotificationManager, which communicates the grading outcomes to students. This crucial step highlights the system's educational feedback loop, integrating grading with CodeKataBattle's commitment to learning and progress tracking.

Flow 5 - Badge Creating and Awarding and Criteria Tracking: This flow details the gamification aspect of CodeKataBattle centering on feature [F5], highlighting how students earn badges upon meeting specific criteria in tournaments or battles. It involves the BadgeManager component which is responsible for creating badges, linking them to student profiles, and tracking criteria completion. This flow is essential for motivating students and adding a layer of achievement and recognition to the educational process.

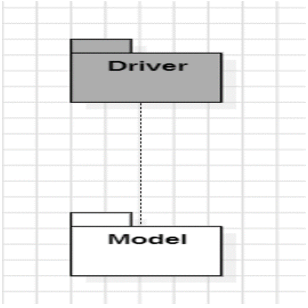
Flow 6 - Notification and Communication: Incorporating feature [F9], this flow focuses on the system's capability to keep users informed and engaged through notifications. It showcases NotificationManager's integration with the Email Provider Service, ensuring that users receive timely updates about new battles, tournament

progress, badge awards, and any direct communications within the platform. This flow is crucial for maintaining an active and informed user base and fostering a connected community within CodeKataBattle.

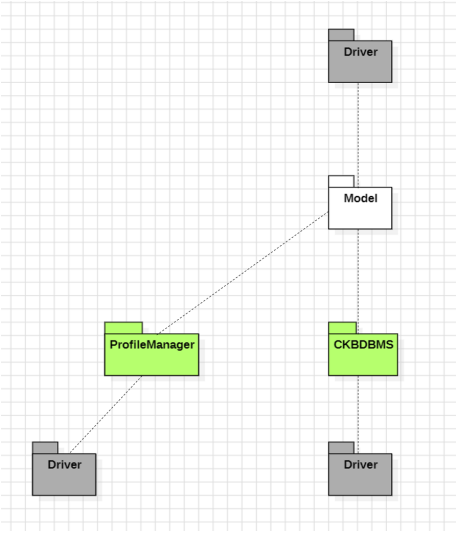
5.3 Component Integration and Testing

The component integration and testing phase for CodeKataBattle is designed to ensure that all parts of the system work together seamlessly to provide a robust educational platform. This section outlines the stages in which components are integrated and tested, with each feature corresponding to a set of components.

One of the first components to implement is the Model since it is used for all the flows. We use drivers for components still under development.

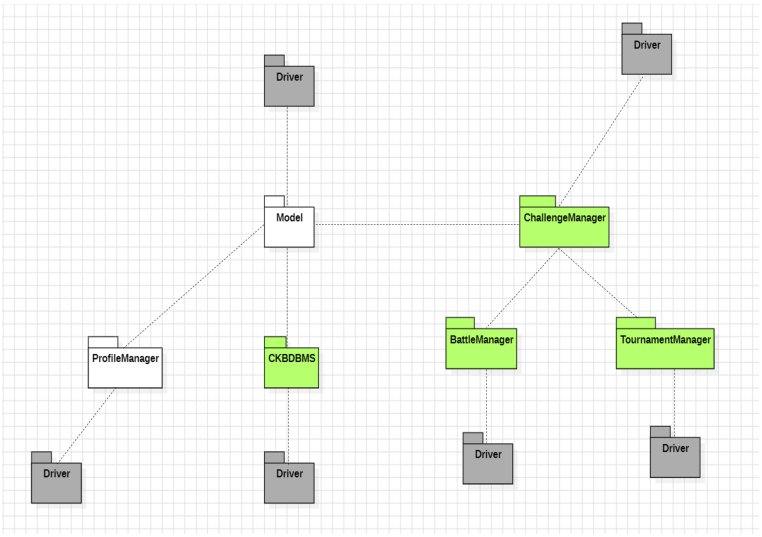


Flow 1 Implementation: Creating an account, updating information, and deleting it. For the implementation of this flow, as said earlier we will need the authentication service and the Model and CKBDBMs components for the Accounts.

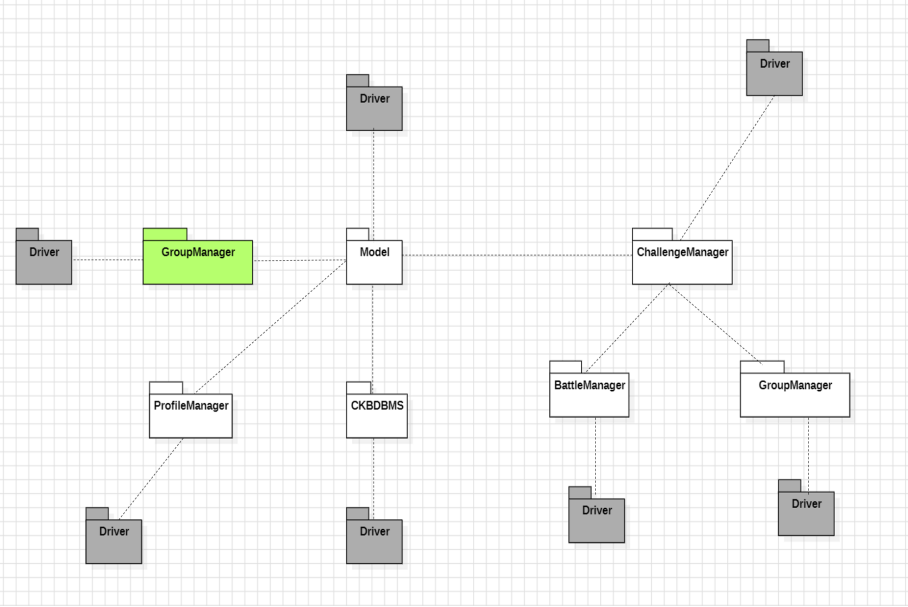


These components will be enough to showcase the Account creation, update, and deletion to the stakeholders since all the components required for this task are present. The authentication service will be linked to the ProfileManager.

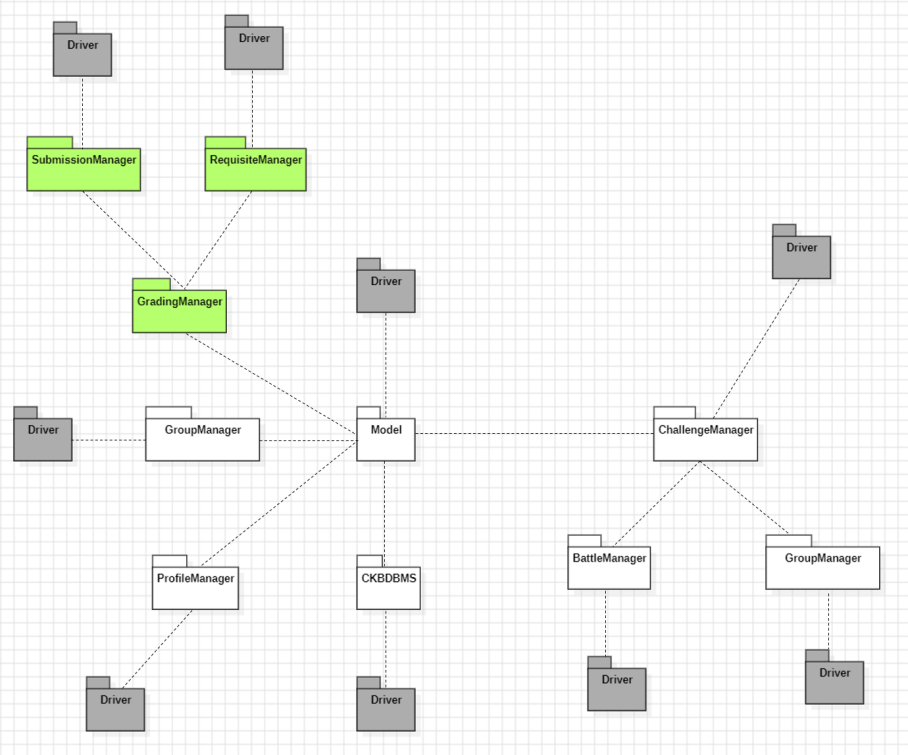
Flow 2 Implementation: Creating a Tournament and a Battle, updating them, and deleting them. This flow requires the following components to be at least partially implemented: TournamentManager, BattleManager, the Model, and the CKBDBMS.



Flow 3 Implementation: Joining a Tournament and a Battle as a Student, and Joining a Group For this flow we will require the TournamentManager, the BattleManager, the GroupManager, the Model, and CKBDBMS. Since some of these core features are already implemented thanks to the previous flows and the top-down approach, we only need to implement the GroupMagager for this flow.

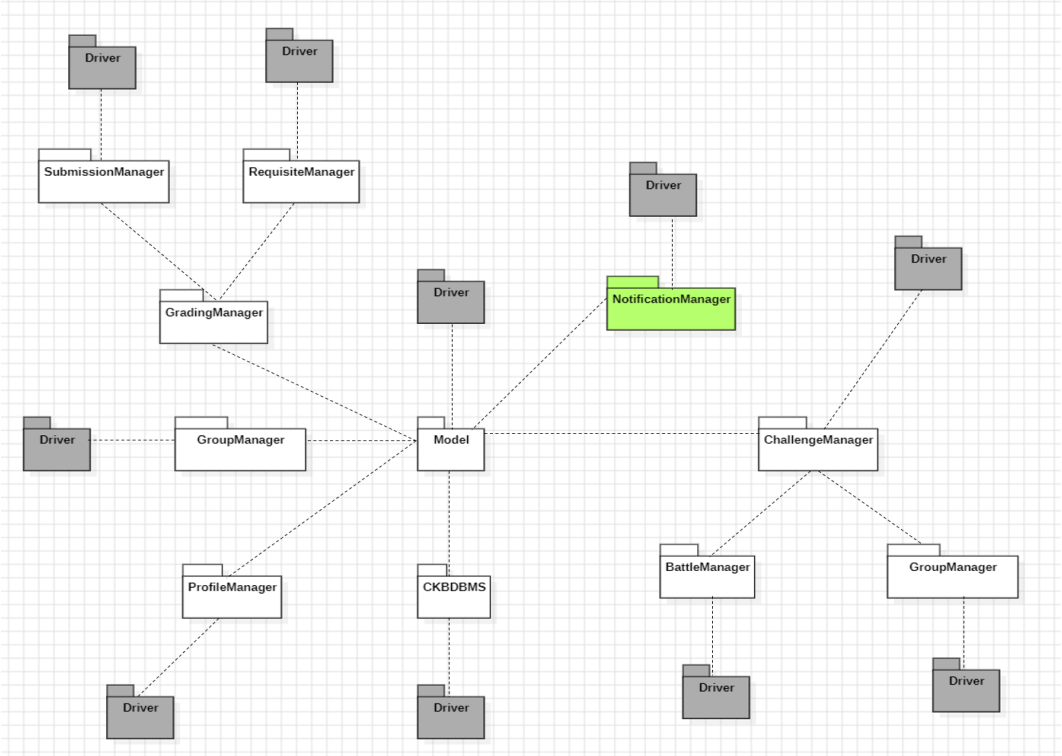


Flow 4 Implementation: This flow will require the GradigManager, which is made of both the RequisiteManager and the SubmissionManager, and is linked to the GitHub Service.



Flow 5 Implementation: Badge Creating and Awarding and Criteria Tracking. This Flow uses the RequisiteManager and the SubmissionManager from the GradingManager, which are both already implemented thanks to the other Flows. The demonstration to the stakeholders can therefore be done.

Flow 6 Implementation: Notification and Communication - This Flow uses the NotificationManager which is linked to the Email Providing system.



5.4 System testing

This phase regroups a series of rigorous tests designed to ensure the system operates flawlessly under various scenarios, particularly those relevant to its deployment in educational environments. These tests are essential to guarantee that the system adheres to the specified requirements and provides a seamless, secure, and engaging experience for both students and educators engaged in coding battles and tournaments. The testing phase is characterized by its focus on real-world usability, performance under stress, and overall system integrity, reflecting the diverse needs and activities of its users in an educational context.

Testing Strategy :

- **Functional Testing:** This testing phase consists of ensuring all system functionalities align with the requirements, including user management, tournament setup, and badge awarding.
- **Performance Testing:** This testing phase evaluates the system's efficiency under different loads, to find out the bottlenecks and efficient parts that should be fixed. This testing will be done when trying to overload the system with sending notifications, creating tournaments or battles, testing submissions, creating and granting badges and requisites... Multiple of these tests will be combined.
- **Usability Testing:** This test phase assesses the ease of use for students and educators to use the interface. This will be done by having user testing sessions for each category of user.
- **Security Testing:** This test phase verifies the protection of sensitive data and the system's defense against security threats.
- **Load Testing:** This test phase analyzes the system behavior under high user traffic and data processing demands.
- **Stress Testing:** This test phase observes the system's resilience and recovery in extreme operational conditions. This can be done by overloading some parts or cutting out resources in others.

Continuous Feedback Loop: This part is not a real testing phase but it concerns a way to always improve the system. Some test sessions with end users need to be scheduled throughout the development, letting educators or students give feedback about the current system state. This would help to fine-tune the system to meet real-world educational needs.

5.5 Additional specifications on testing

The additional specifications on testing for the CodeKataBattle system are designed to ensure the platform not only aligns with its technical and functional objectives but also excels in delivering a quality experience to its users, particularly students, and educators. This phase extends beyond conventional testing methods to encompass a holistic evaluation of the system, emphasizing user-centric design, performance efficiency, and continuous improvement. The approach integrates iterative testing, user feedback, and performance monitoring to create a dynamic and responsive development process. These additional testing procedures are crucial in refining the system's features, usability, and overall quality, ensuring that it remains adaptable to the evolving needs of an educational environment.

Testing Methodology :

- Iterative testing at each development stage for early bug detection and resolution.
- Regular integration of user feedback for system refinement.

Key Testing Aspects :

1. **Regular Bug Testing**
   - Continuously identify and fix system bugs.
   - Combine automated and manual testing methods.
   - Focus on features like tournament creation and badge awarding.
2. **Integration with User Feedback**
   - Align development with user needs.
   - Collect and apply feedback from educators and students.
   - Evaluate interface design and system navigation.
3. **Performance Monitoring**
   - Maintain consistent system performance.
   - Monitor load times and server response.
   - Address performance issues during high-usage periods.
4. **Quality Assurance Checks**
   - Uphold high-quality standards across the system.
   - Conduct code reviews and maintain documentation.
   - Review new features for quality assurance.
5. **Scalability Testing**
   - Test the system's capacity for growth.
   - Simulate increased user numbers and data volume.
6. **Security Audits**
   - Protect user data and system integrity.
   - Regularly update and audit security measures.

- Focus on sensitive data handling.

7. **Accessibility Testing**
   - Ensure the system is accessible to all users.
   - Implement and test for accessibility standards.
   - Ensure compatibility with various accessibility tools.

Continuous Development and Testing Cycle :

- Maintain a responsive environment for testing and development.
- Adapt to changing user needs and technological trends.

6 Time Spent

Name	Section	Hours Spent
Pablo García Alvarado	ARCHITECTURAL DESIGN	15h
	USER INTERFACE DESIGN INFORMATION	10h
	REQUIREMENT TRACEABILTY	6h
	REVIEWING ENTIRE DOCUMENTATION	2h
Adrian Valica	ARCHITECTURAL DESIGN	7h
	REQUIREMENT TRACEABILTY	6h
	COMPONENT INTEGRATION AND TESTING	12
	REVIEWING ENTIRE DOCUMENTATION	2h
Rishabh Tiwari	INTRODUCTION	3h
	OVERALL DESCRIPTION	2h
	IMPLEMENTATION PLAN	8h
	COMPONENT INTEGRATION AND TESTING	10h
	OVERLEAF DOCUMENTATION	10h

7 References

References

[1] Codekata main page. `http://codekata.com/`.

[2] Example of a similar site (codewars). `https://www.codewars.com/`.

[3] What is test-driven development (tdd)? `https://en.wikipedia.org/wiki/Test-driven_development`.

[4] Draw.io. https://www.draw.io, Accessed: 2024. High level architectures are made using www.draw.io.

[5] Staruml. http://staruml.io, Accessed: 2024. Testing models and component architecture made with StarUML; sequence diagram models made with StarUML.

[6] Politecnico di Milano. Software engineering 2 course material, 2022-2023. Course material for the academic year 2022-2023.

[7] Visual Paradigm. What is sequence diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/. Accessed: 2023-12-21.