



POLITECNICO

MILANO 1863

DD

Design Document

Authors: **Davide Currò**
Matilde Lombardo
Lorenzo Mondo

Date: 07/01/2024

Professor: Elisabetta Di Nitto

Summary

Summary	2
1. Introduction	4
1.1 Purpose.....	4
1.2 Scope.....	4
1.3 Definitions, Acronymous and Abbreviations	4
1.3.1 Definitions.....	4
1.3.2 Abbreviations	5
1.4 Revision History	5
1.5 Reference Document.....	5
1.6 Document Structure	5
2. Architectural Design	6
2.1 Overview	6
2.2 Component View.....	7
2.3 Deployment View	9
2.4 Runtime View	10
2.4.1 Student/Educator Registration	10
2.4.2 Student/Educator Login	11
2.4.3 Educator creates a new Tournament	12
2.4.4 Student participates to a new Tournament	12
2.4.5 Educator creates a new Battle within a Tournament	13
2.4.6 Student participates to a new Battle	14
2.4.7 Student submits a new solution.....	16
2.4.8 A Battle ends and the platform automatically updates the personal score of the students	16
2.4.9 Educator manually evaluates the students	17
2.4.10 Educator closes a Tournament and the platform automatically assigns the badges	18
2.5 Component interfaces	19
2.6 Selected Architectural Style and Patterns	20
2.7 Other Decision Designs	21
3. User Interface Design.....	22
4. Requirements traceability	27
4.1 Components	27
4.2 Requirements	28
4.3 Requirements Traceability Matrix	29
5. Implementation, integration, and test plan.....	32
5.1 Implementation	32

5.2 Integration & Test Plan	32
5.3 Static Analysis Tool Implementation	37

1. Introduction

1.1 Purpose

The platform CodeKataBattle (CKB) has been designed in order to help students improve their software development skills by training with peers on code Kata [1]. A code kata battle is a programming exercise in a programming language of choice. The principal objective of the platform is to provide a dynamic and engaging environment where students can develop new knowledge, improve their skills and learn how to work in a team. The scope of the educator is to challenge and motivate their students. Furthermore, the introduction of gamification elements encourages students to do their best work to strive for excellence. To summarize, the purpose of CodeKataBattle is to create an educational ecosystem that encourages collaboration, competition, and continuous improvement in the field of software development.

This document contains the overall description of the architectural design of the system CodeKataBattle, his component, and how they interact with each other. Furthermore, the document contains some mock-ups for the user interfaces and a plan for the implementation and testing of the system. Thus, this document will guide the development of the software.

1.2 Scope

The scope of the project aims to satisfy both students and educators needs. The students will use this platform to improve their skills and their knowledge, on the other hand the educator will use it to test the students' learning level. Therefore, this document is focused on their needs since they are the main consumers of the platform, in describing what CodeKataBattle should provide and how to arrange the platform's functions.

The document describes an educational platform dedicated to hands-on training and assessment of students' software development skills through "code kata" challenges, tournaments, and automated and manual assessments.

The platform also integrates gamification elements to motivate students. In fact, since the whole learning process is managed as a tournament where you must win a battle to gain more points or new badges, the students are more motivated to fulfil the challenge assigned. At the same time, they will learn in an unconventional way and perhaps this will help them to learn more easily and quickly.

1.3 Definitions, Acronymous and Abbreviations

1.3.1 Definitions

Definition	Description
User	A person who is either subscribed to or interested in subscribing to the platform, and can be either a student or an educator.
Student	A user aspiring to enhance their development skills by utilizing the platform.
Educator	A user utilizing the platform to assess students participating in their own tournaments.

1.3.2 Abbreviations

Abbreviation	Meaning
CKB	CodeKataBattle
RASD	Requirements Analysis and Specification Document
WPX	World Phenomena number X
SPX	Shared Phenomena number X
GX	Goal number X
DX	Domain assumption number X
RX	Requirement number X

1.4 Revision History

1.5 Reference Document

This document refers to the specification document: “Assignment RDD AY 2023-2024.pdf”

1.6 Document Structure

The document is structured in different sections.

The first section is the introduction of the document, it describes the topic of the system and its purpose, as well as useful information to understand better the document.

The second section describes the architectural design of the system: the architecture chosen for the system; all the components present as well as the component and deployment views and all the sequence diagrams to describe how the components communicate with each other to fulfill the system's requests. Therefore it starts with an high-level overview and then with a detailed description of all the components.

The third section shows the mock-up designs of the clients' interfaces.

The fourth section contains the requirements traceability matrix that maps all the components required to fulfill each requirement. The requirements are the same as the ones described in the RASD document.

The fifth section describes a suggested implementation and test plan for the system.

The last section contains the effort spent on this document by the authors and the references used.

2. Architectural Design

2.1 Overview

The architectural design of CodeKataBattle follows the three-tier architecture, composed of a presentation layer, business logic and a data layer. We have decided to use this type of architecture because of the division of the data layer from the business logic; we think that access to the DBMS is an additional layer of security because the data are separated and to retrieve them, it's necessary to pass through the middle layer.

- **Presentation:** It's the layer that interacts with the client, educator, or student. Sends every request that he receives from the client to the middle layer. This is represented in a different way for the student and the educator since they have different requests to make.
- **Business Logic:** It's the most important part of the architecture and system. It receives all the requests of the client and updates the DBMS accordingly. It's the layer responsible for the communication with the DBMS. It manages all the functions of the system we are describing, from the creation of the tournament to the evaluation and final score ranks.
- **Data:** It's the layer where all the persistent data are stored, such as the information about the student and the tournament with his battles. The data are retrieved when needed, depending on the status of the requests. The data stored are related for example to the credentials of the users, or the data of the battles, the team playing and their scores.

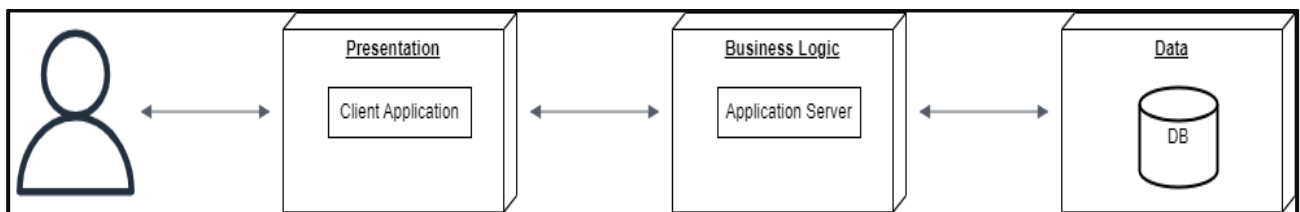


Figure 1 - Overview of the architectural design with three layers.

2.2 Component View

This section is composed of this diagram which shows an overview of the system, his components and how they interact and cooperate in order to provide all the system's functionalities. Furthermore, there is a brief description of all the components in this diagram.

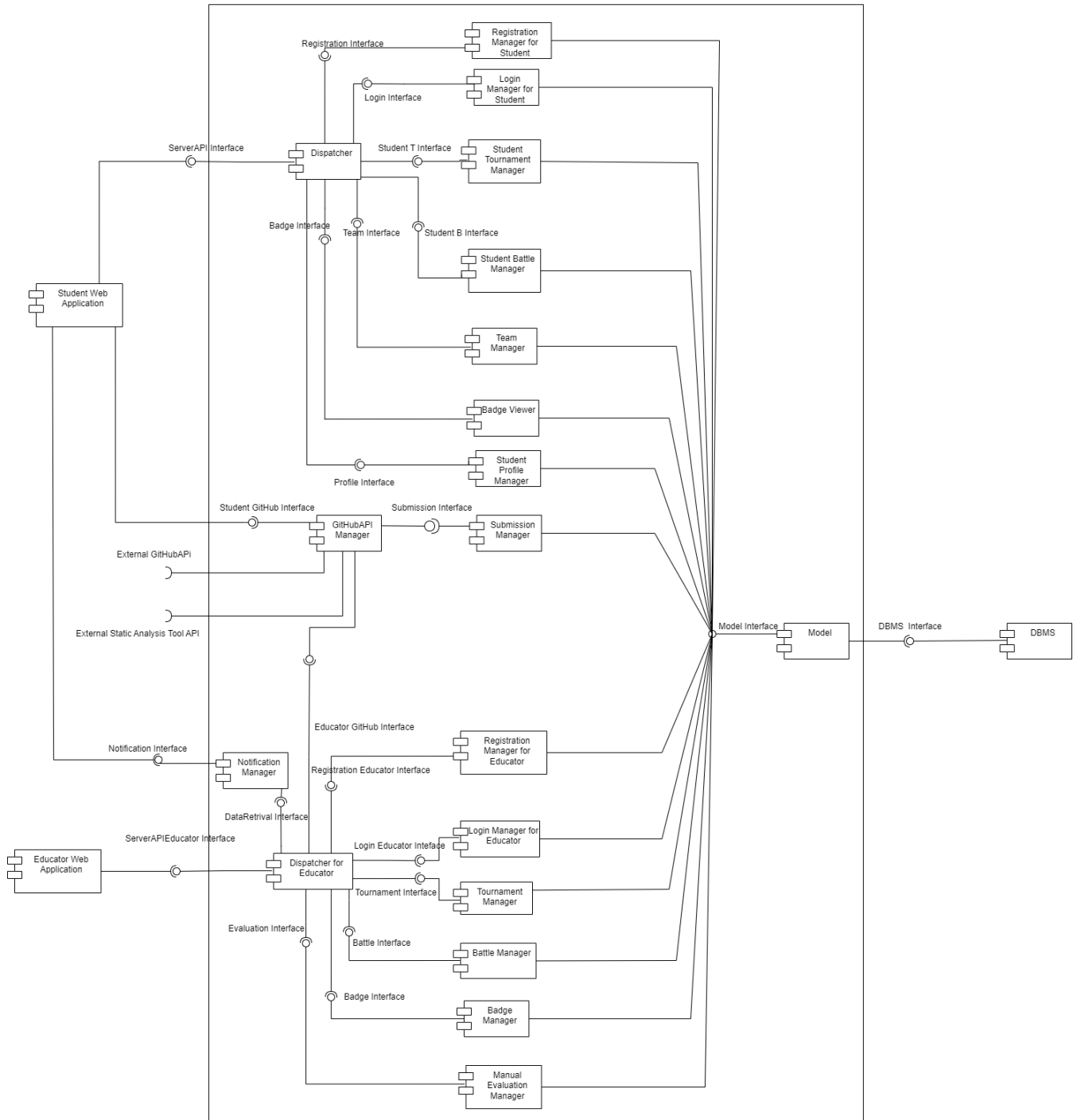


Figure 2 - Component diagram of the system eMall.

This is the list of the components with a description:

- **Student Web Application:** The web application client side for the student's perspective.
- **Dispatcher for Student:** It's the entry point for all the requests from the student. It handles all the requests by rerouting them to the corresponding component.
- **Login Manager for Student:** This component handles the sign in of the student on the platform, by verifying the student's credentials and allowing the login.
- **Registration Manager for Student:** This component has the role to manage the request about the sign up of the student on the platform.
- **Student Tournament Manager:** This component handles the registration process in a new tournament, which has to happen between selected dates, and manage, it manages also general requests that the students could ask to obtain more informations about a Tournament (i.e. Tournament Ranking View).
- **Student Battle Manager:** This component has the role to manage the registration of the student in a battle, it manages that the registration happens by the deadlines defined, it manages also general requests that the students could ask to obtain more informations about a battle (i.e. Battle Ranking View).
- **Team Manager:** This component manages the creation and the operations of the team.
- **Badge Viewer Manager:** This component is responsible for handling the requests of the students who want to see their badges, or other student's badges.
- **Submission Manager:** This component is responsible for the submission and automatic evaluation of the solutions, using also the GitHubAPI for pulling from GitHub.
- **Student Profile Manager:** This component has the role to manage the student profile.
- **GitHubAPI Manager:** Is responsible for receiving and pushing the commits from the teams' repository and handling them over to the submission manager to evaluate them in the model.
- **Notification Manager:** This component has the role to manage all the notifications from the system to the student.
- **Model:** Handles the data layer with the logic one and contains an object-relation mapping used by the other components when they want to retrieve data from the DB. It's the only component communicating with the DB.
- **DBMS:** This component is responsible for storing persistent data.
- **Educator Web Application:** The web application client side for the educator's perspective.
- **Dispatcher for Educator:** It's the entry point for all the requests from the educator. It handles all the requests by rerouting them to the corresponding component.
- **Badge Manager:** It's the component with the role of handling all the operations on the badge, such creation and visualization of it.
- **Battle Manager:** This component has the role of managing the operations about the battle and allows the educator to create a new battle.
- **Tournament Manager:** This component gives the possibility to create a new Tournament to the educator and to manage it.
- **Login Manager for Educator:** This component handles the sign in of the educator on the platform, by verifying the educator's credentials and allowing the login.
- **Registration Manager for Educator:** This component has the role to manage the request about the sign up of the educator on the platform.
- **Manual Evaluation Manager:** This component has the role to allow the educator to manually evaluate a solution in one battle.
- **Educator Profile Manager:** This component has the role to manage the educator profile.

2.3 Deployment View

In this section is presented a deployment diagram of the system, in which are specified the tool used to run the system and the protocol for communication among the nodes.

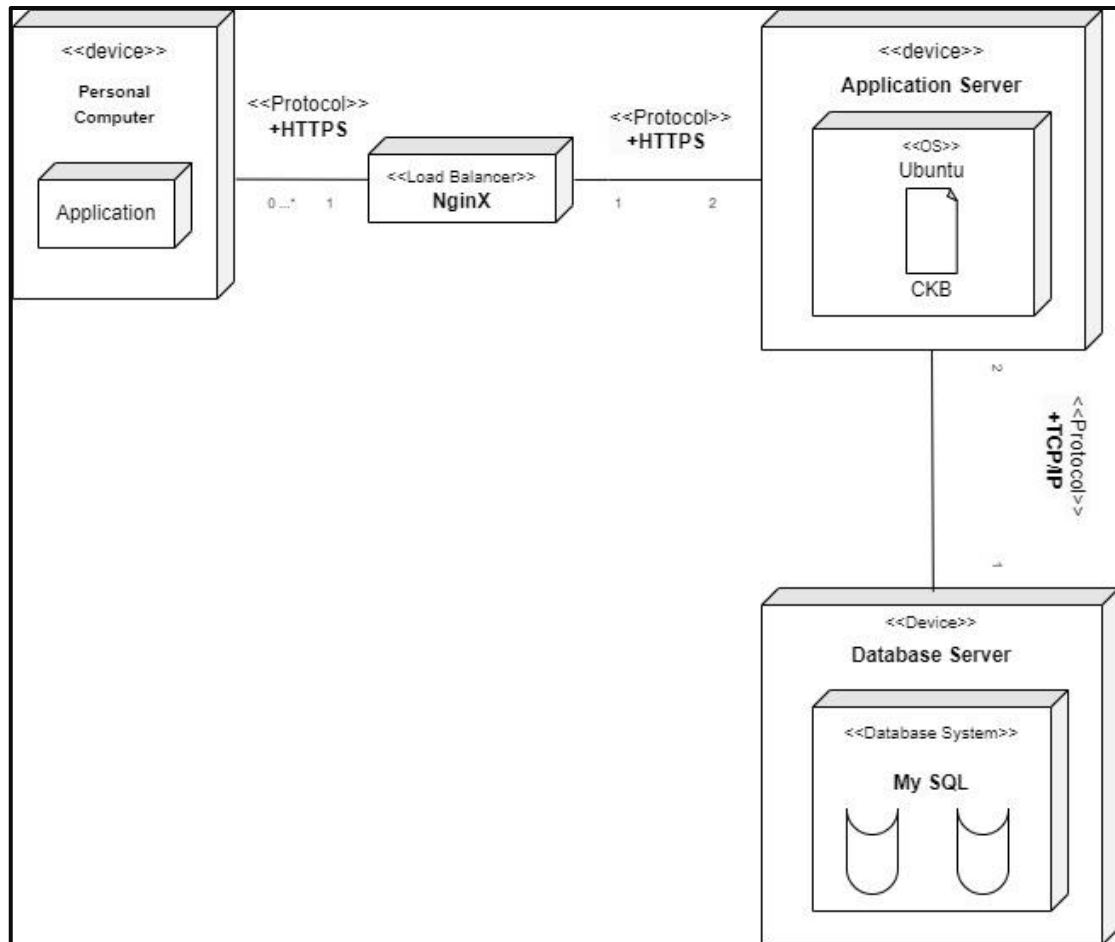


Figure 3 - This diagram shows the deployment of the system.

- **Personal Computer:** It is the device of the user that has to be connected to internet in order to communicate with the server.
- **Load balancer:** It is a web server that catches the requests from the user and balances the workload among the servers.
- **Application server:** hosts the application logic of the system. Interacts with the database server and with the client application.
- **Database server:** Host that contains the persistent data about all the system. In order to avoid loss of information the idea is to have two of them one used from the system and a replica to store the data in case of failure.

2.4 Runtime View

In this section, we present sequence diagrams that outline the key features of the system. These diagrams illustrate the flow of requests from system clients. For all sequence diagrams, with the exception of the Login process for both students and educators, it is assumed that they are already authenticated.

2.4.1 Student/Educator Registration

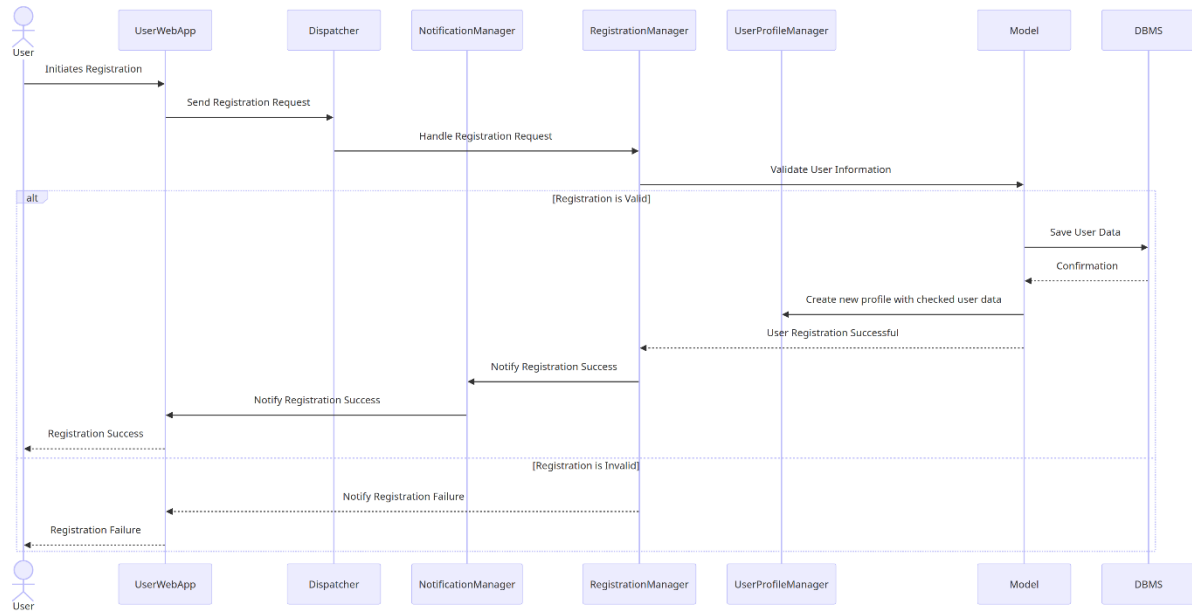


Figure 4 - This diagram represents how system behave when a new user (that can be both a Student or an Educator) sends a registration request.

2.4.2 Student/Educator Login

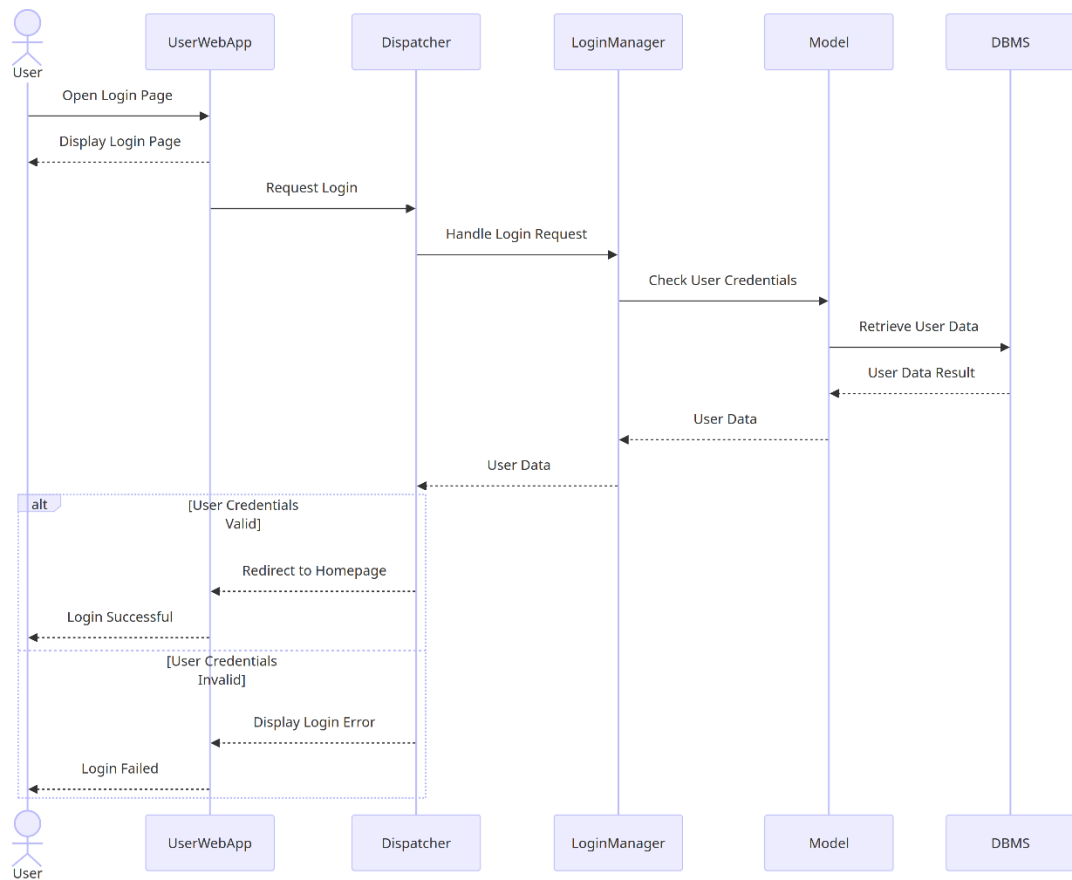


Figure 5 - This diagram represents the behavior of the system on a request of login from a user (that can be both a student or an educator).

2.4.3 Educator creates a new Tournament

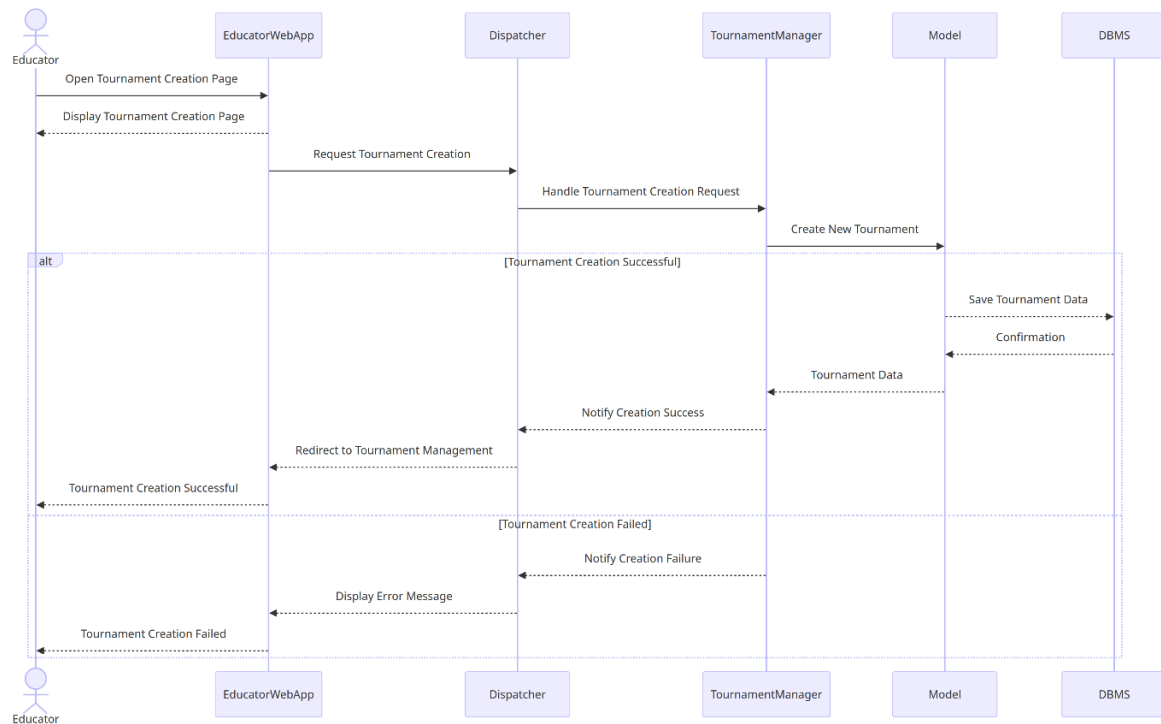


Figure 6 - This diagram represent the behaviour of the system when an educator creates a new Tournament

2.4.4 Student participates to a new Tournament

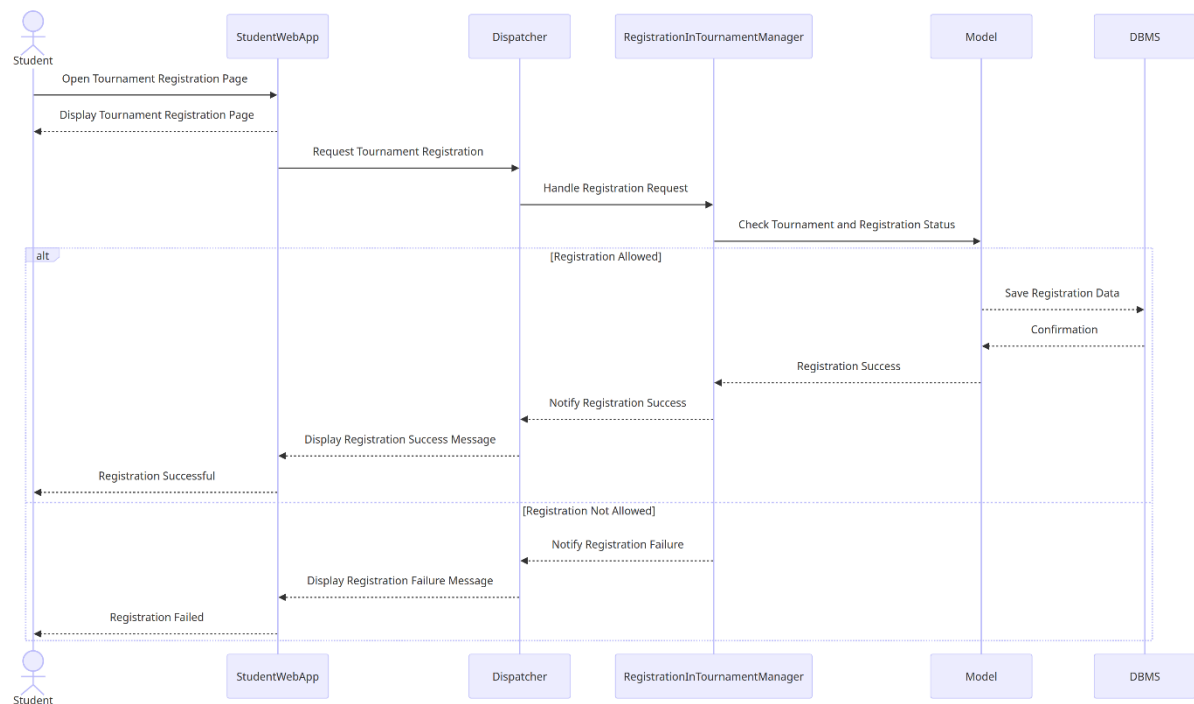


Figure 7 – This diagram represents the behaviour how the system when a student send a request to participate to a new tournament

2.4.5 Educator creates a new Battle within a Tournament

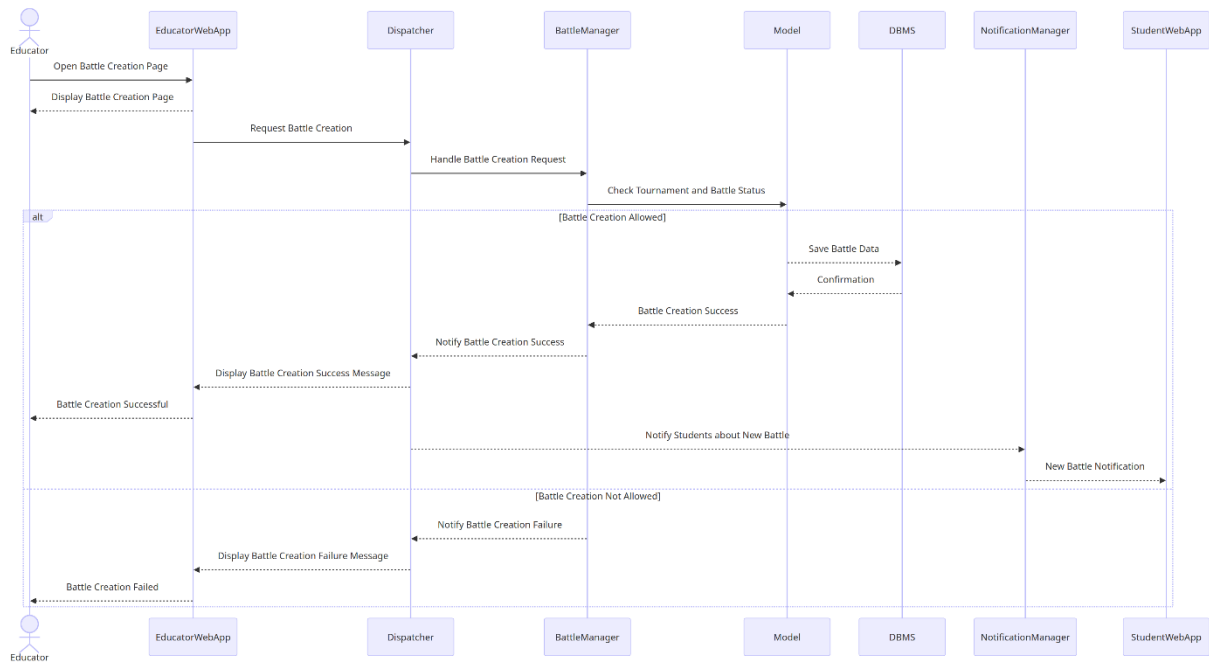


Figure 8 - This diagram represents the behaviour of the system when an educator sends a request to create a new Battle within a specific Tournament

2.4.6 Student participates to a new Battle

Student Participates To the Battle Alone

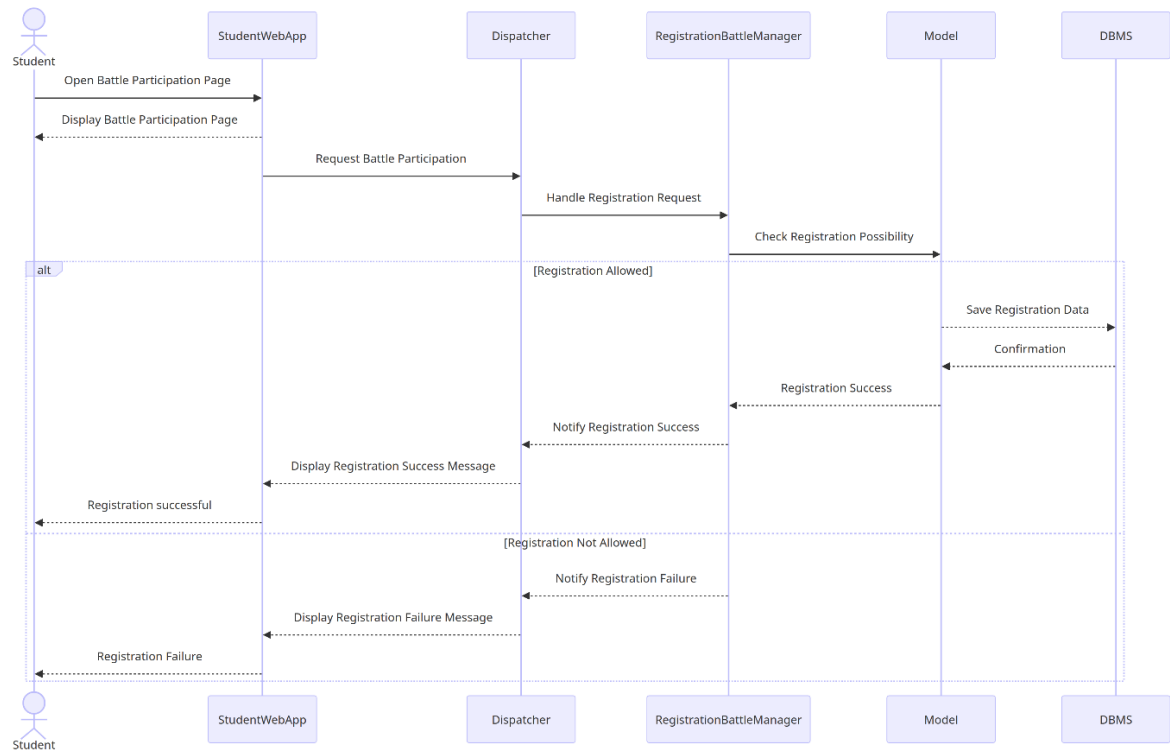


Figure 9 - This diagram describes how the system behave when a student sends a request to participate to a Battle alone.

Student Participate To the Battle Creating a new Team

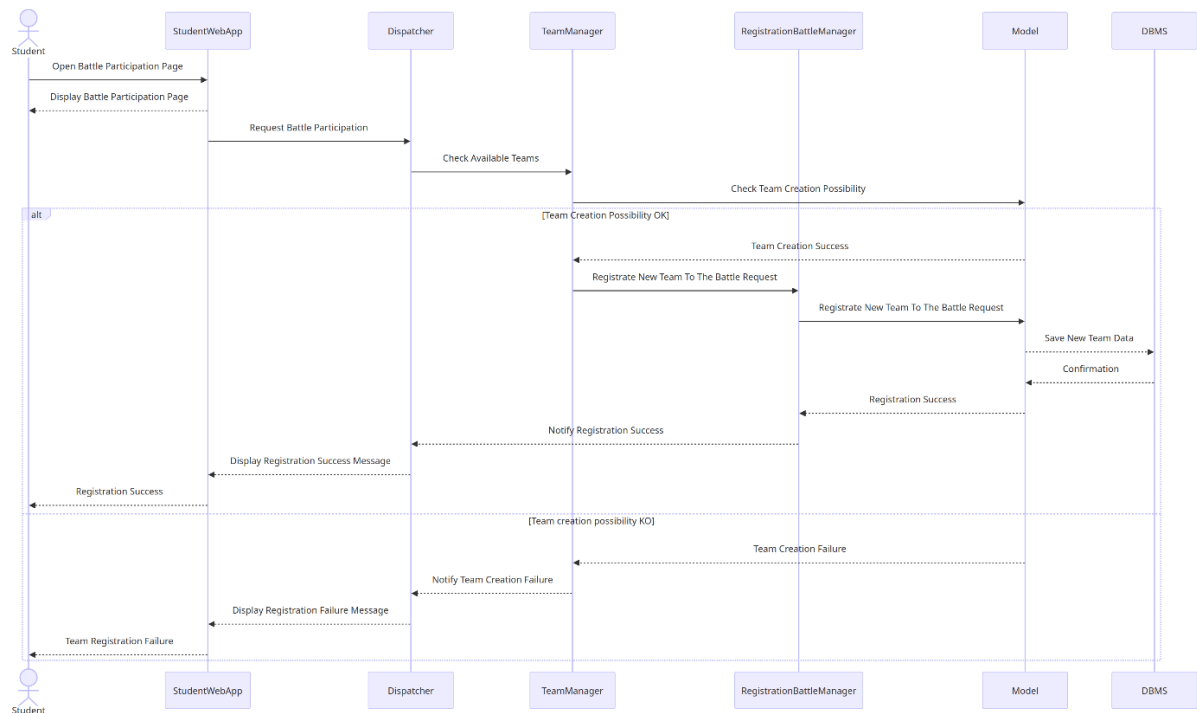


Figure 10 - This diagram describes the behaviour of the system when a student participate to a battle by creating a Team

Student Participates to the Battle joining an existing Team

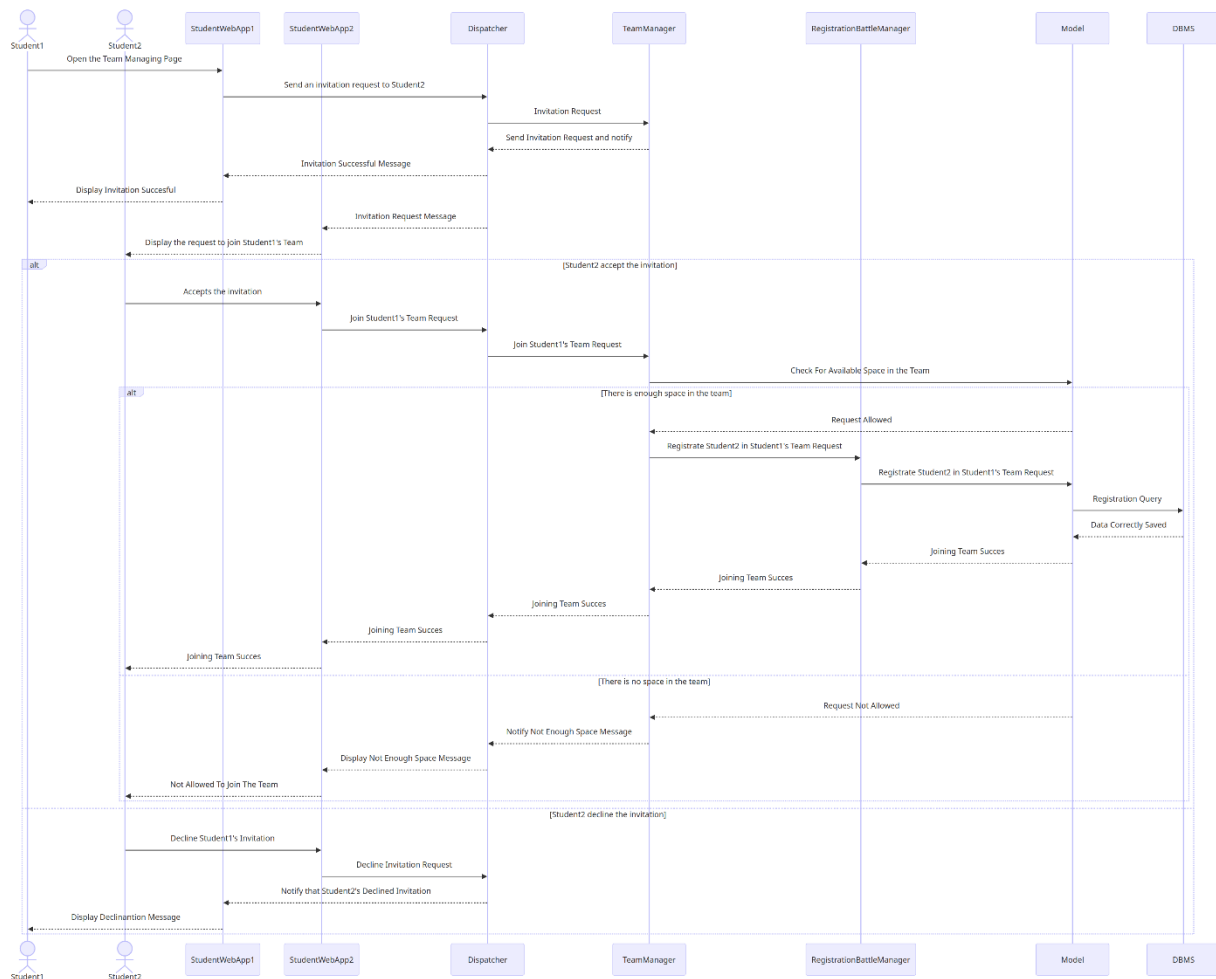


Figure 11- This diagram describes the behaviour of the system when a student participates to a battle by joining an existing Team

2.4.7 Student submits a new solution

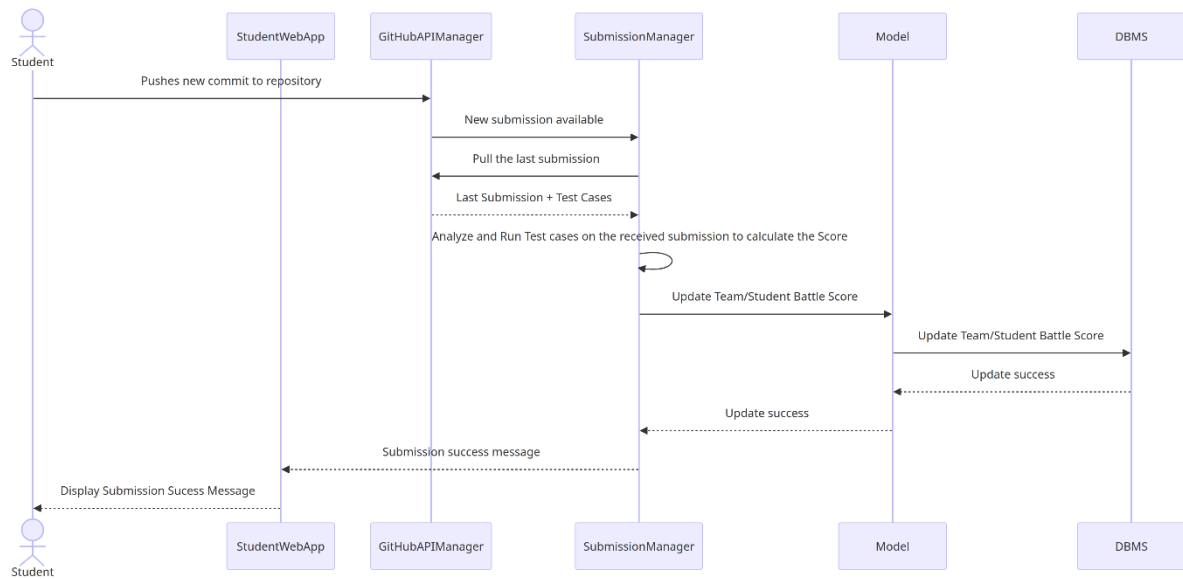


Figure 12- This diagram describes the behaviour of the system when a student submits a new solution by pushing it in the main branch of the GitHub repository.

2.4.8 A Battle ends and the platform automatically updates the personal score of the students

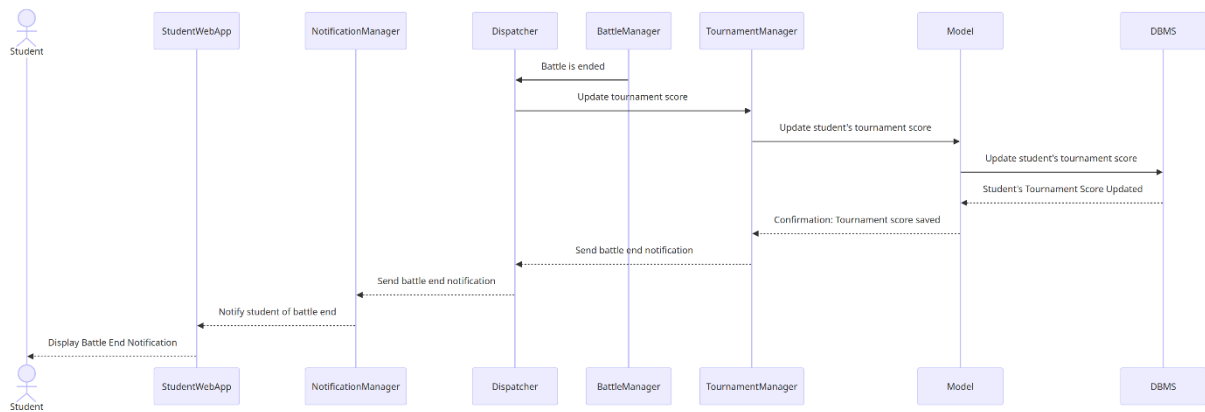


Figure 13 – This diagram describes the behaviour of the system when a battle ends and the platform automatically updates the personal score of the students

2.4.9 Educator manually evaluates the students

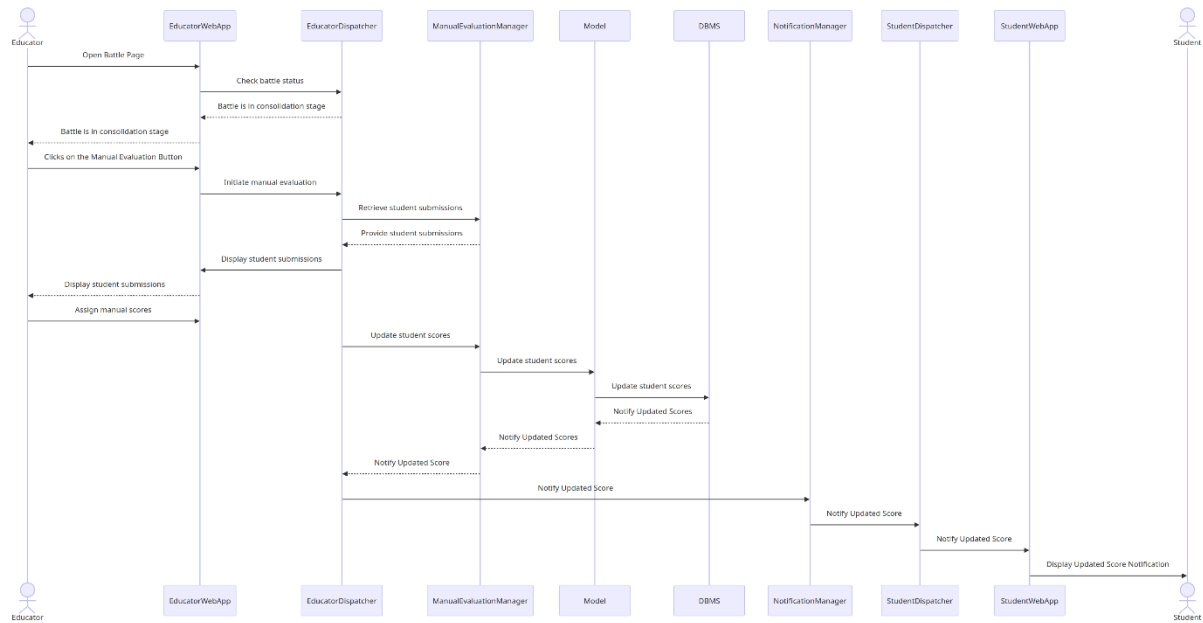


Figure 14 – This diagram describes the behaviour of the system when an educator manually evaluates the students

2.4.10 Educator closes a Tournament and the platform automatically assigns the badges

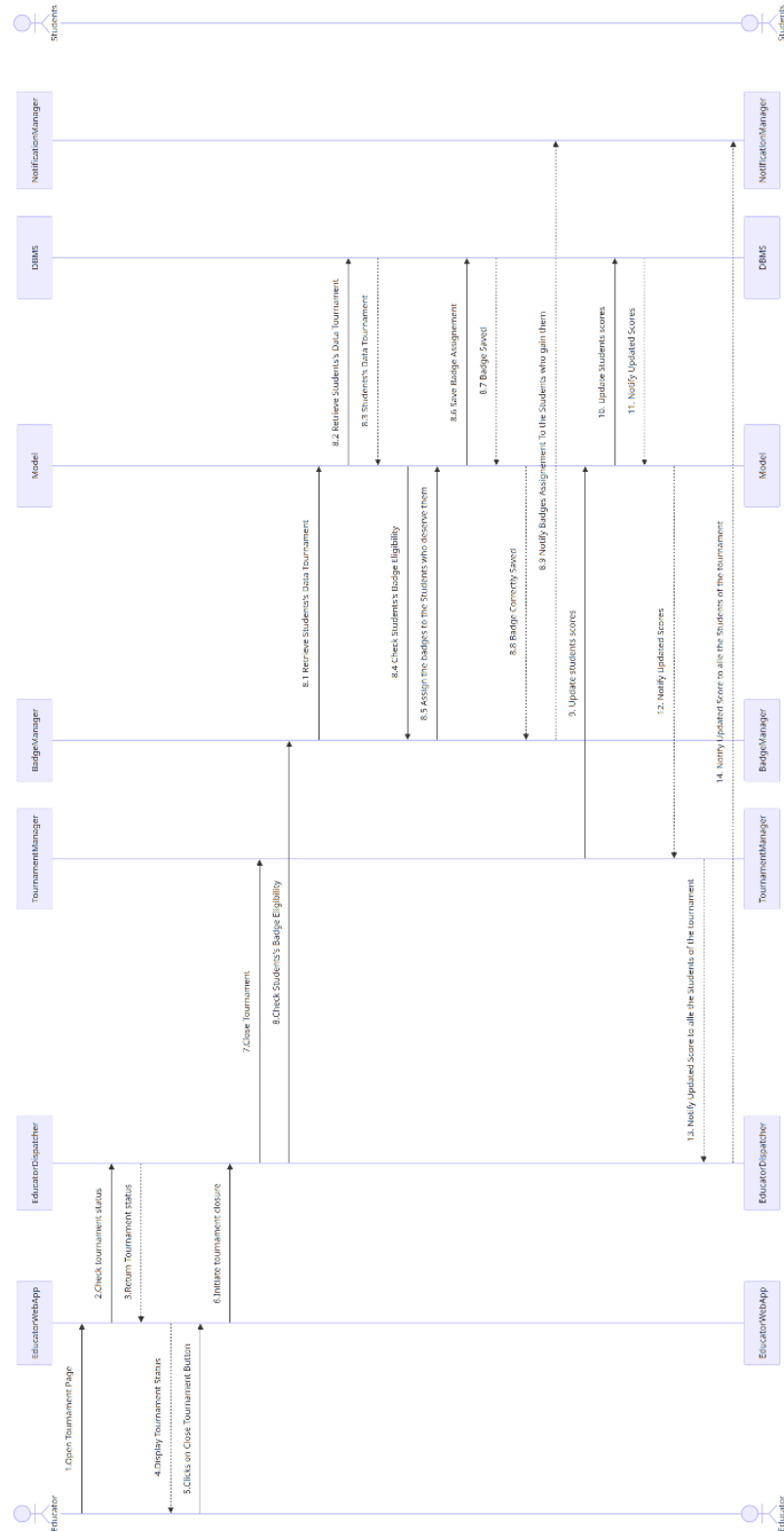


Figure 15- Educator closes a Tournament and the platform automatically assigns the badges

2.5 Component interfaces

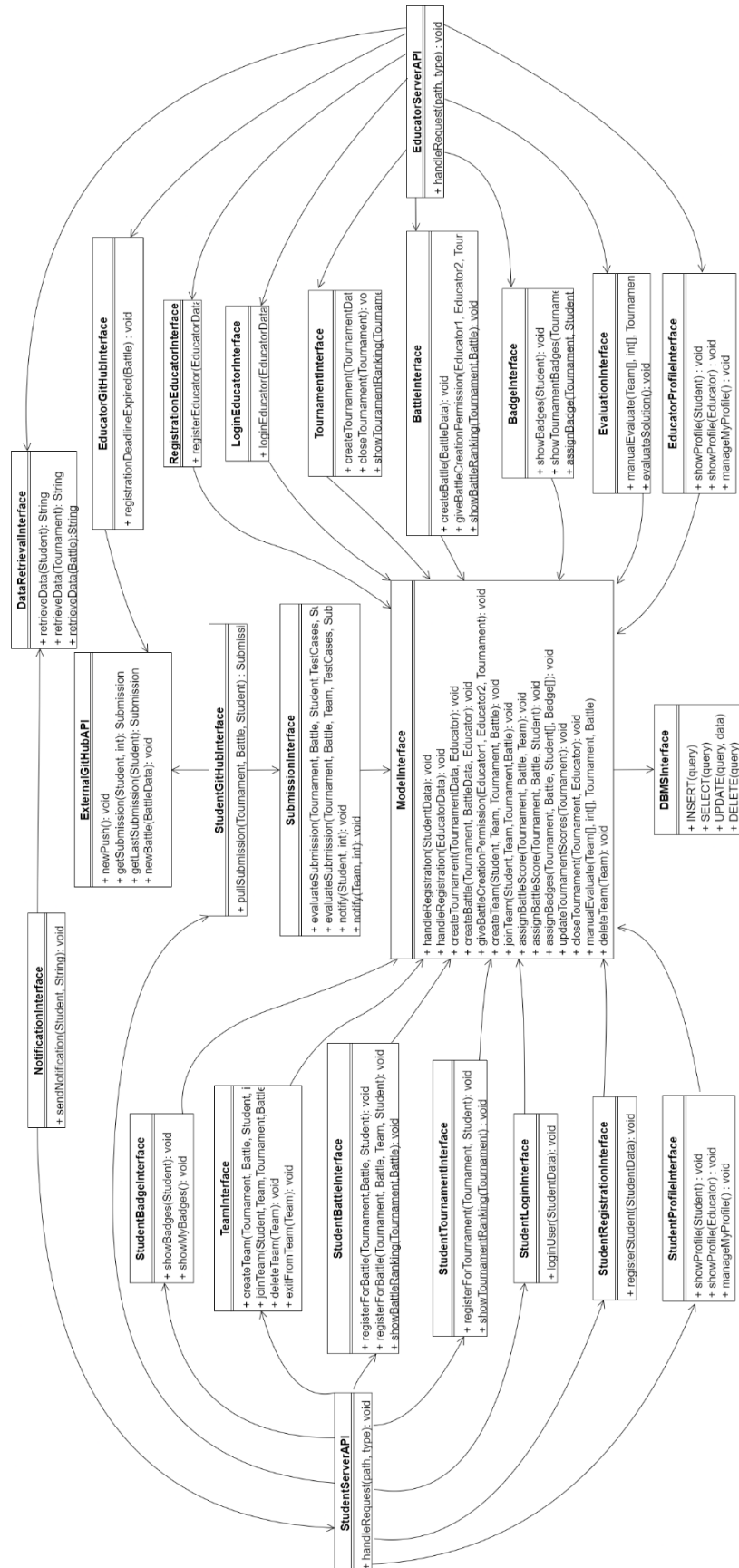


Figure 16 – Component Interfaces

2.6 Selected Architectural Style and Patterns

Model View Controller (MVC):

The Model View Controller (MVC) pattern is embraced for its ability to foster a high level of code decoupling. By distinctly separating the presentation logic and data handling, the system becomes more manageable, adaptable, and conducive to effective testing. This pattern is implemented through three fundamental components:

- **Model:** Empowers access to and management of data through a defined set of methods, primarily serving the controller.
- **View:** Presents data in a user-friendly and comprehensible format.
- **Controller:** Captures input from the view and translates it into executable commands.

Three-Tier Client-Server:

The three-tier client-server architecture represents a multi-layered structural approach where presentation, application, and data access functions are physically isolated. Our choice of a multitier architecture is driven by its capacity to enhance system flexibility and reusability, effectively decoupling the inherent complexity of the system. This architectural model enables the thorough separation of data access from the layer housing presentation and customer interaction logic, a crucial feature when dealing with sensitive information.

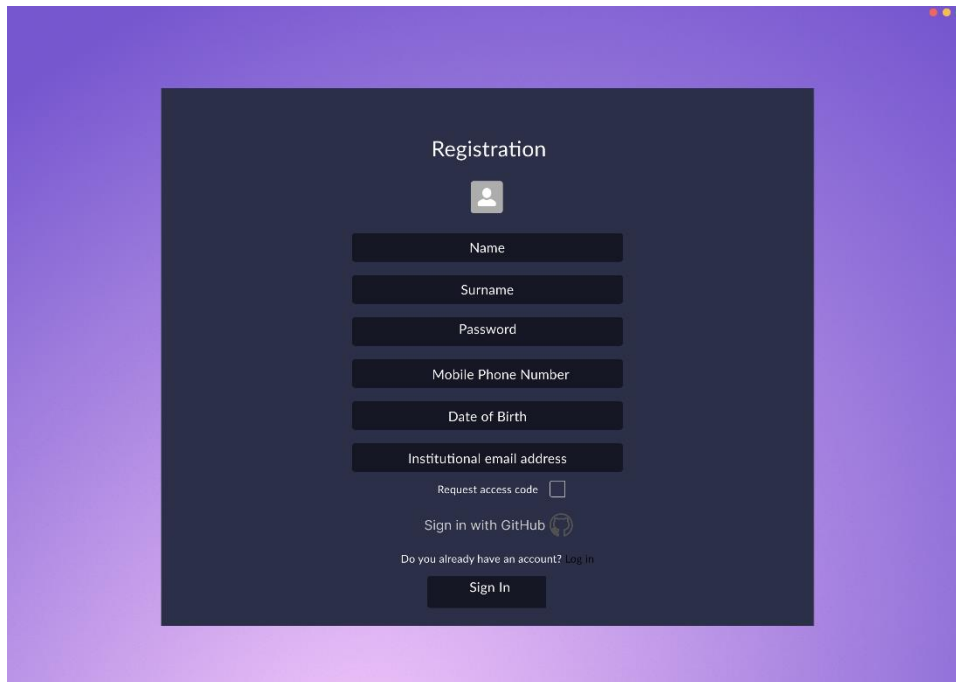
In more detailed terms, our adopted three-tier architecture consists of a Presentation Tier, an Application Tier, and a Data Access Tier. This configuration allows for the clear segregation of clients from data, facilitated by the inclusion of the application tier.

2.7 Other Decision Designs

In this section, we outline additional design decisions that influence the architecture of the Code Kata Battle (CKB) platform.

One key design choice is to treat Students and Educators as distinctly separate entities within the system. While both can be viewed as logical users of the platform, we've opted to consider them as entirely different entities at the implementation level. This demarcation is driven by the significant differences in roles and permissions between the two. Despite their shared identity as platform users, the implementation ensures that Students and Educators do not inherit from a common class. This decision is made to uphold a clear separation in their roles, permissions, and overall system behavior.

3. User Interface Design



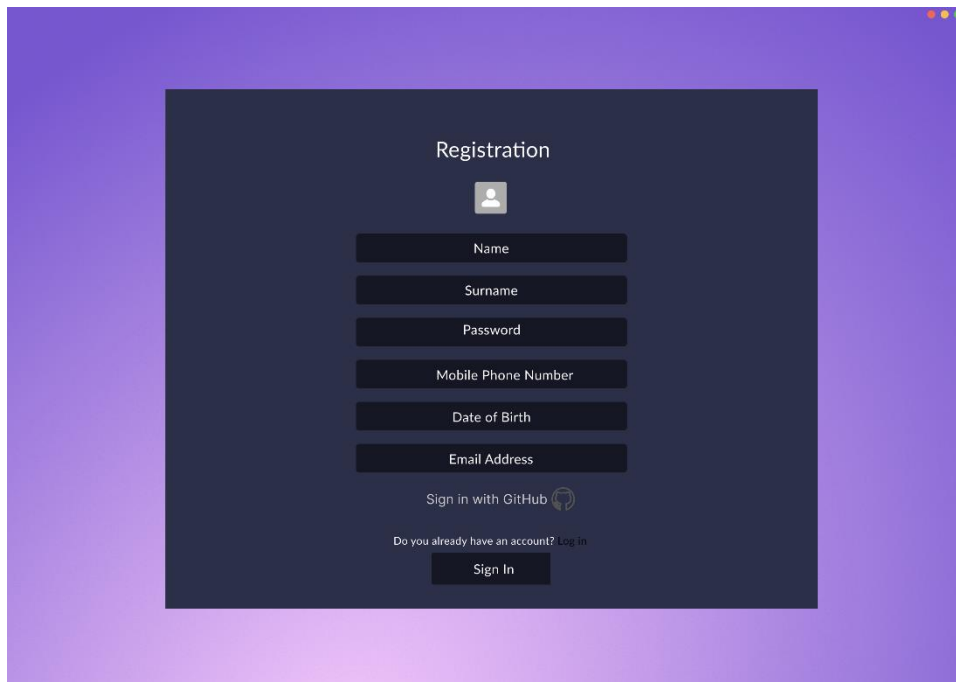
The image shows a registration form for educators. It is titled "Registration" and features a user icon. The form contains several input fields: Name, Surname, Password, Mobile Phone Number, Date of Birth, and Institutional email address. Below these fields is a checkbox for "Request access code". There is a "Sign in with GitHub" button with the GitHub logo. At the bottom, there is a link "Do you already have an account? Log In" and a "Sign In" button.

Registration

☐ Request access code

[Do you already have an account? Log In](#)

Figure 17 - this interface represents the educator registration page with the corresponding information to be entered.



The image shows a registration form for students. It is titled "Registration" and features a user icon. The form contains several input fields: Name, Surname, Password, Mobile Phone Number, Date of Birth, and Email Address. Below these fields is a "Sign in with GitHub" button with the GitHub logo. At the bottom, there is a link "Do you already have an account? Log In" and a "Sign In" button.

Registration

[Do you already have an account? Log In](#)

Figure 18 - this interface represents the student registration page with the corresponding information to be entered.

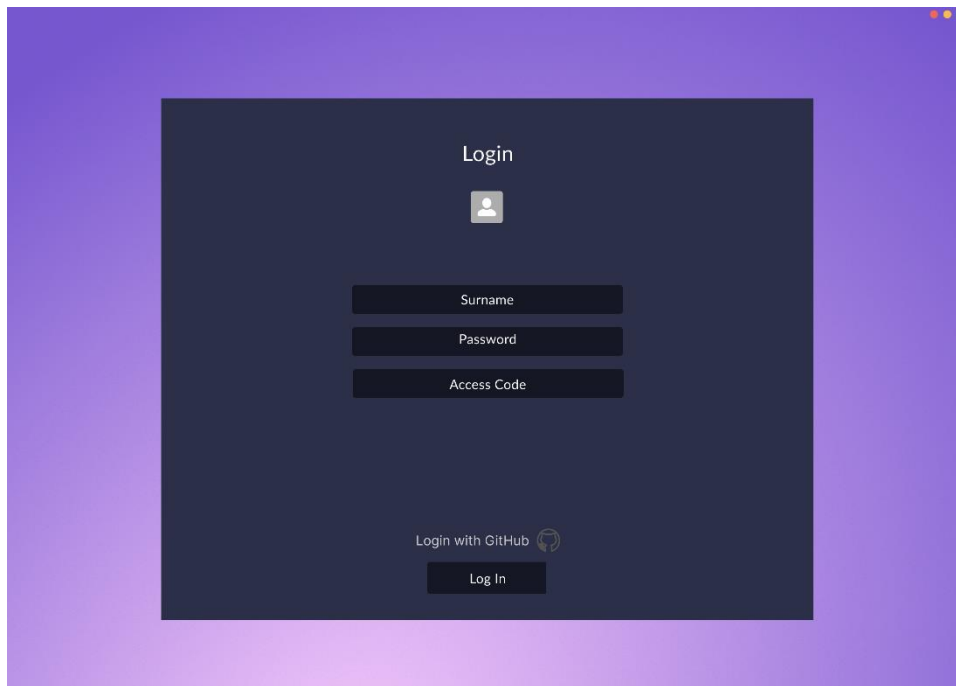


Figure 19 – this interface represents the educator login page with the corresponding information to be entered.

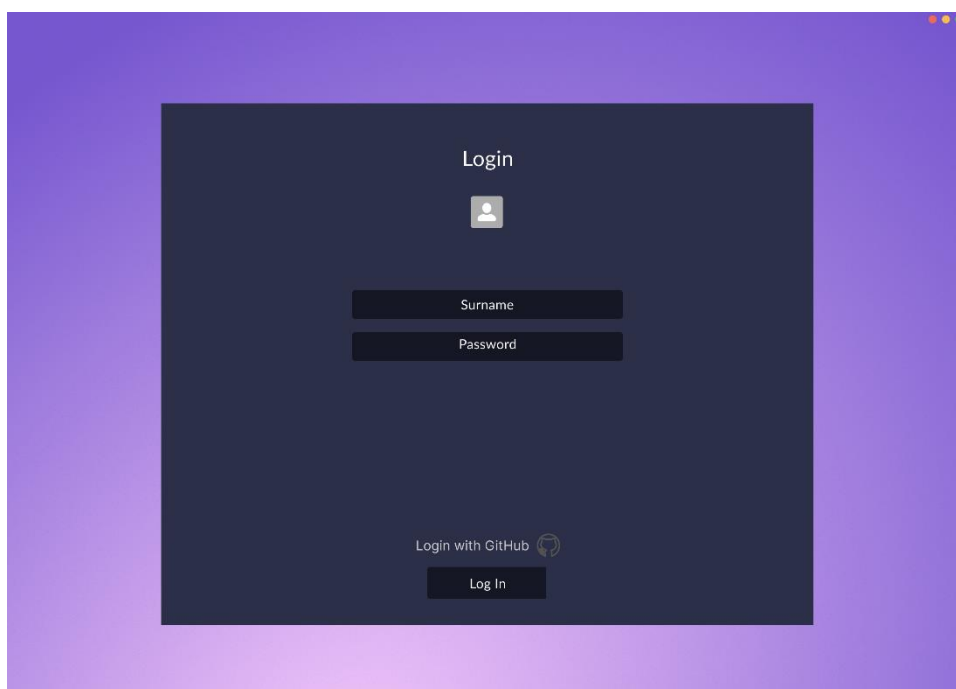


Figure 20 – this interface represents the student login page with the corresponding information to be entered.

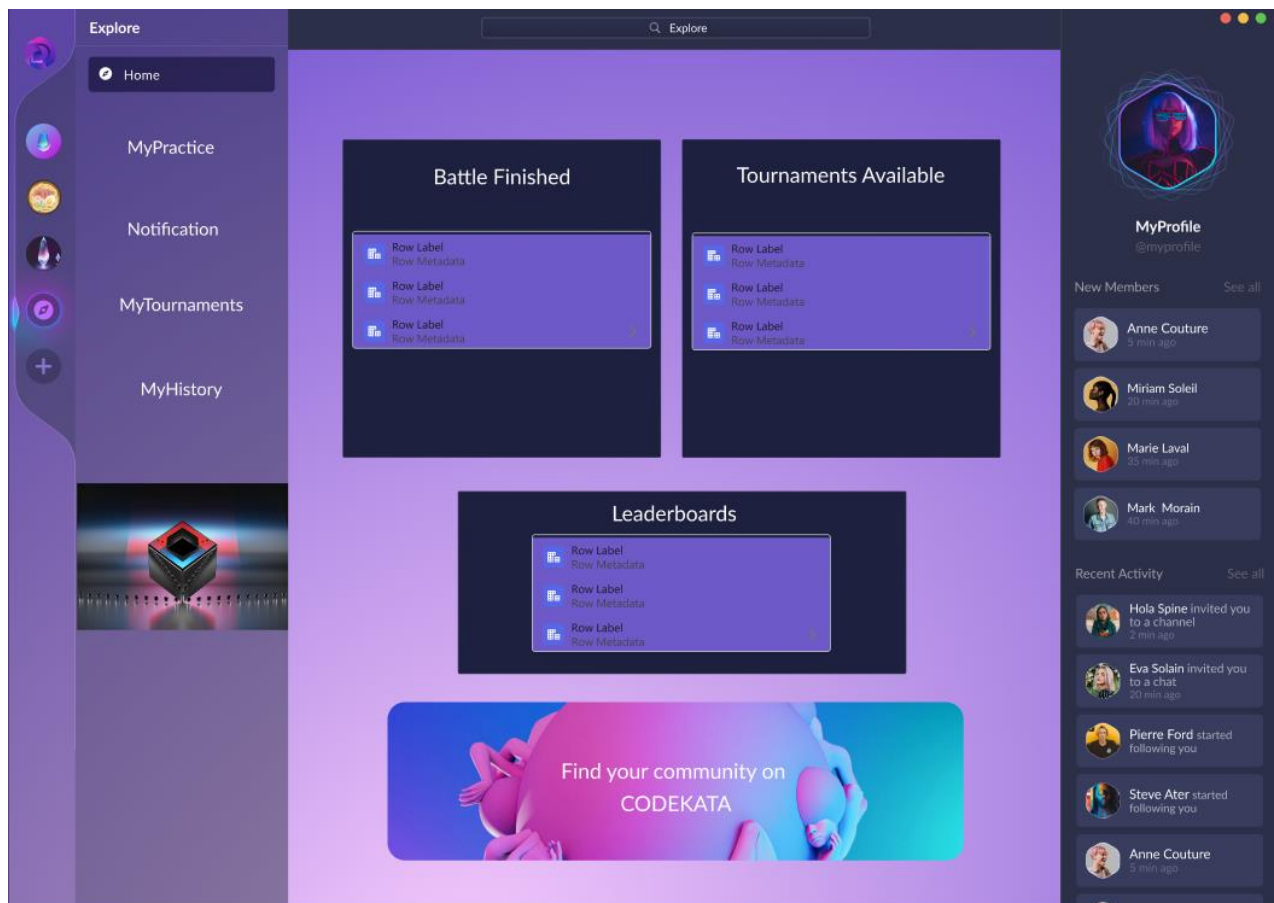


Figure 21 – This interface represents the main page of the platform. Here, you can see various features such as notifications, available tournaments, completed battles and other information

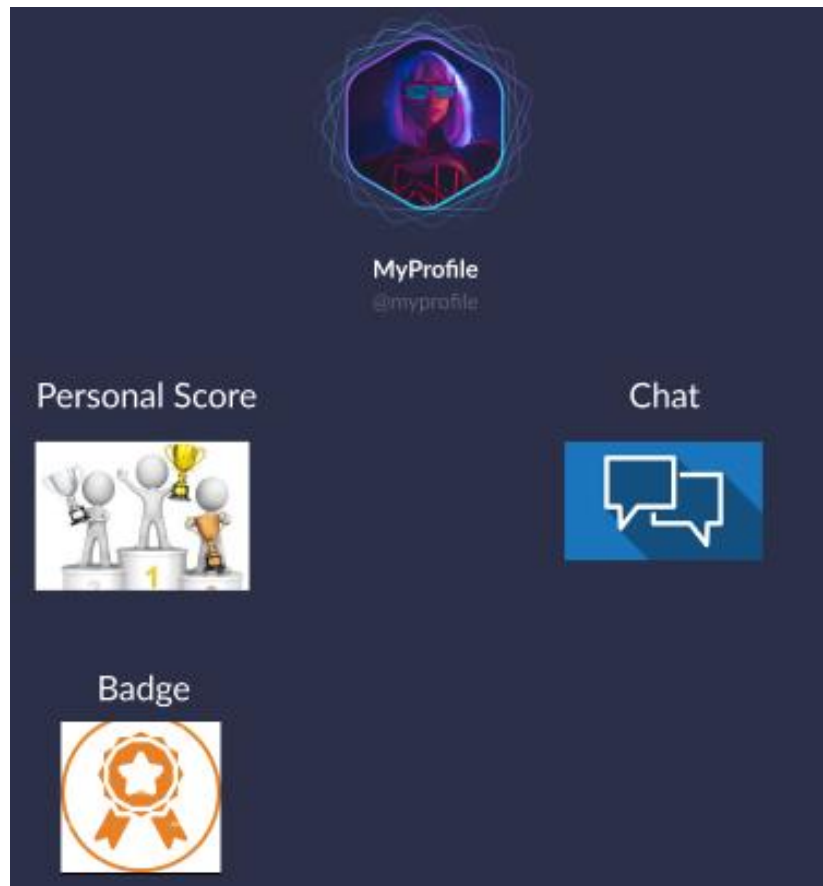


Figure 22 – this interface represents what the student can see within their profile. They can view the assigned badges, the score or the chat.

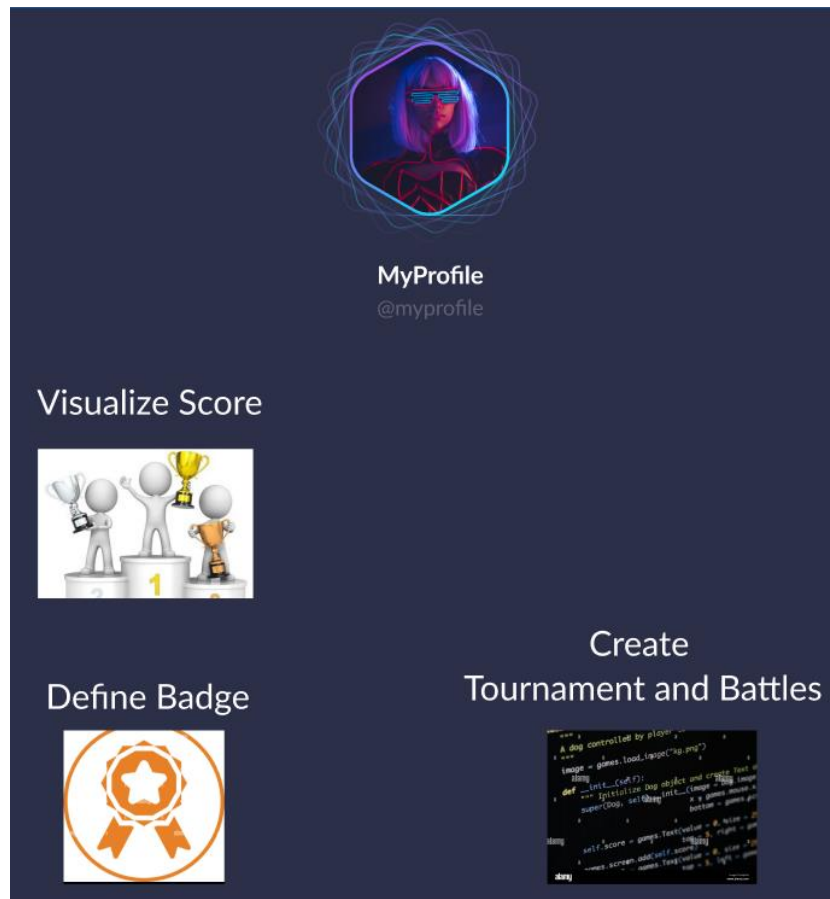


Figure 23 – this interface represents what the educato can see within their profile. They can view the assigned badges, the score or create tournament and battles.

4. Requirements traceability

This section describes which components of the system are required to satisfy the requirements specified in the RASD document. In order to explain it, a requirements traceability matrix is used to mark which components are involved or not.

4.1 Components

To understand better and clearer the matrix, a list of abbreviations of the names of the components is shown as follows:

- **Student Web Application – SWA**
- **Dispatcher for Student – DS**
- **Login Manager for Student – LMS**
- **Registration Manager for Student – RMS**
- **Student Tournament Manager – STM**
- **Student Battle Manager – SBM**
- **Team Manager – TM**
- **Badge Viewer Manager – BVM**
- **Submission Manager – SM**
- **GitHubAPI Manager – GM**
- **Notification Manager – NM**
- **Model – M**
- **DBMS – DB**
- **Educator Web Application – EWA**
- **Dispatcher for Educator – DE**
- **Badge Manager – BM**
- **Battle Manager – BAM**
- **Tournament Manager – TOM**
- **Login Manager for Educator – LME**
- **Registration Manager for Educator – RME**
- **Manual Evaluation Manager – MEM**
- **Student Profile Manager – SPM**
- **Educator Profile Manager – EPM**

4.2 Requirements

For a clearer description of the traceability matrix, here are provided all the requirements written in the RASD document:

Requirements	Description
R1	The system shall allow user to register a personal account.
R2	The system shall allow users to login.
R3	The system shall allow users to modify personal information into account.
R4	The system must check the validity of the information inserted by the users.
R5	The system shall allow the Educator to create tournaments.
R6	The system shall allow the Educator to setup deadlines for the tournament.
R7	The system shall allow the Educator to create battles.
R8	The system shall allow the Educator to grant to other colleagues the permission to create battle.
R9	The system shall allow the students to view the list of available tournaments.
R10	The system shall allow the students to subscribe for a tournament.
R11	The system shall allow the students to view the list of battles.
R12	The system shall allow the students to create/join a team.
R13	The system shall allow the students to select a battle.
R14	The system shall allow the team to participate in the battle.
R15	The system must create a GitHub repository containing the code kata.
R16	The system should send the link of the GitHub repository to all student who are members of subscribed team.
R17	The system must pull and analyse the latest source from GitHub repo.
R18	The system must run the tests on the corresponding executables.
R19	The system must be able to automatically evaluate the job done by the students.
R20	If manual evaluation is required (by the Educator), the system shall allow the Educator to check and evaluate the work done by students.
R21	The system must update the battle score of a team as soon as new push action on GitHub are performed.
R22	The system shall allow the users to see the current rank evolving during the battle.
R23	If manual evaluation is required (by the Educator), the system shall allow the Educator to through the sources produced by each team.
R24	At the end of each battle, the system must update the personal tournament score of each student.
R25	The system shall allow the Educator to create the badge and rules to achieve a badge in the Tournament.
R26	The system must automatically assign the badges at the end of each tournament.
R27	The system shall allow the users to visualize the badges in their profile.
R28	The system must notify students registered on the platform.
R29	The system must notify students every time a battle is created within the tournament.
R30	The system must notify students when the battle registration time is about to expire.
R31	The system must notify students when the final submission is about to expire.
R32	The system must notify users when the final battle rank becomes available.
R33	The system must notify students involved in a tournament when the educator closes that specific tournament.

4.3 Requirements Traceability Matrix

For the sake of clarity and ease of reading, we have divided the table into three separate tables, each containing a partial number of requirements. In each row of the tables, the required components for each requirement are indicated.

[illegible]

[illegible]

C	R29	R30	R31	R32	R33
SWA	X	X	X	X	X
DS	X	X	X	X	X
LMS	X	X	X	X	X
RMS					
STM					X
SBM			X	X	
TM					
BVM					
SM					
GM					
NM	X	X	X	X	X
M	X	X	X	X	X
DB	X	X	X	X	X
EWA					X
DE					X
BM					
BAM	X	X	X	X	
TOM	X	X	X		X
LME					X
RME					
MEM					
SMP					
EMP					

5. Implementation, integration, and test plan

This section will describe the methods and plans for the implementation and testing of the CodeKataBattle system. The testing is one of the most important parts of developing a system, because it helps finding all the bugs and problems that create a unsatisfiable software for the final user.

5.1 Implementation

The implementation of the system will follow a bottom-up approach as well as the testing. This has been selected as the better solution because it allows the implementation to continue at the same time as the testing, without waiting for the end of the development of each component unit in the system. In addition, testing each component as soon as it's completed ensures that the system is working perfectly, and each component is satisfying its purposes.

The bottom-up approach has been chosen because it allows the individual testing of each component as well as the edit of a single component without editing the other ones and their behaviors.

5.2 Integration & Test Plan

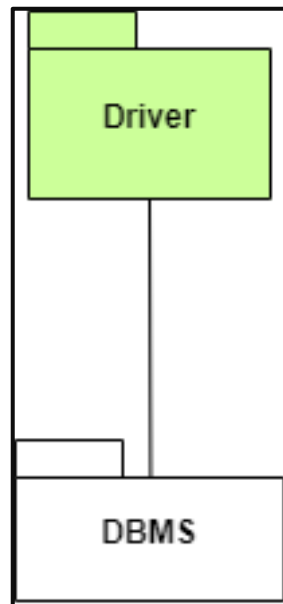
The approach used in the integration and testing of the components of the system will be the bottom-up approach. This approach has been chosen because of its benefits, among all there is the incremental development process which allows to have a more manageable development of the system step by step, as well as testing each component individually before including it in the system. That allows also to find problems and bugs easily and without affecting the whole system with their behavior.

The first components that will be tested are the ones in the lowest layers of the hierarchy and from there the testing will continue with the components higher that recall the ones that have already been tested, until all the system has been developed and tested.

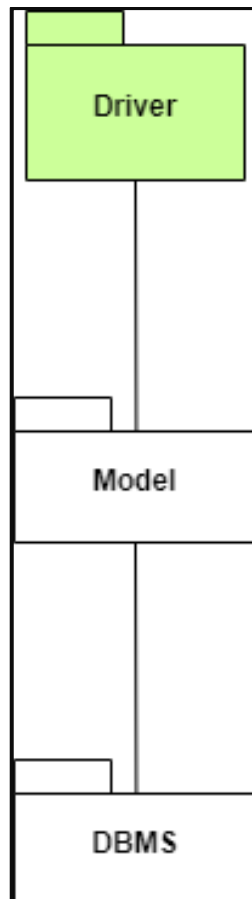
The graphics below will show the flow of integration of all the components of the system. Not every component has been included in the diagrams, but only examples of calls between some important components in the system.

The components considered are the model, the DBMS, the controllers, and the Clients web applications.

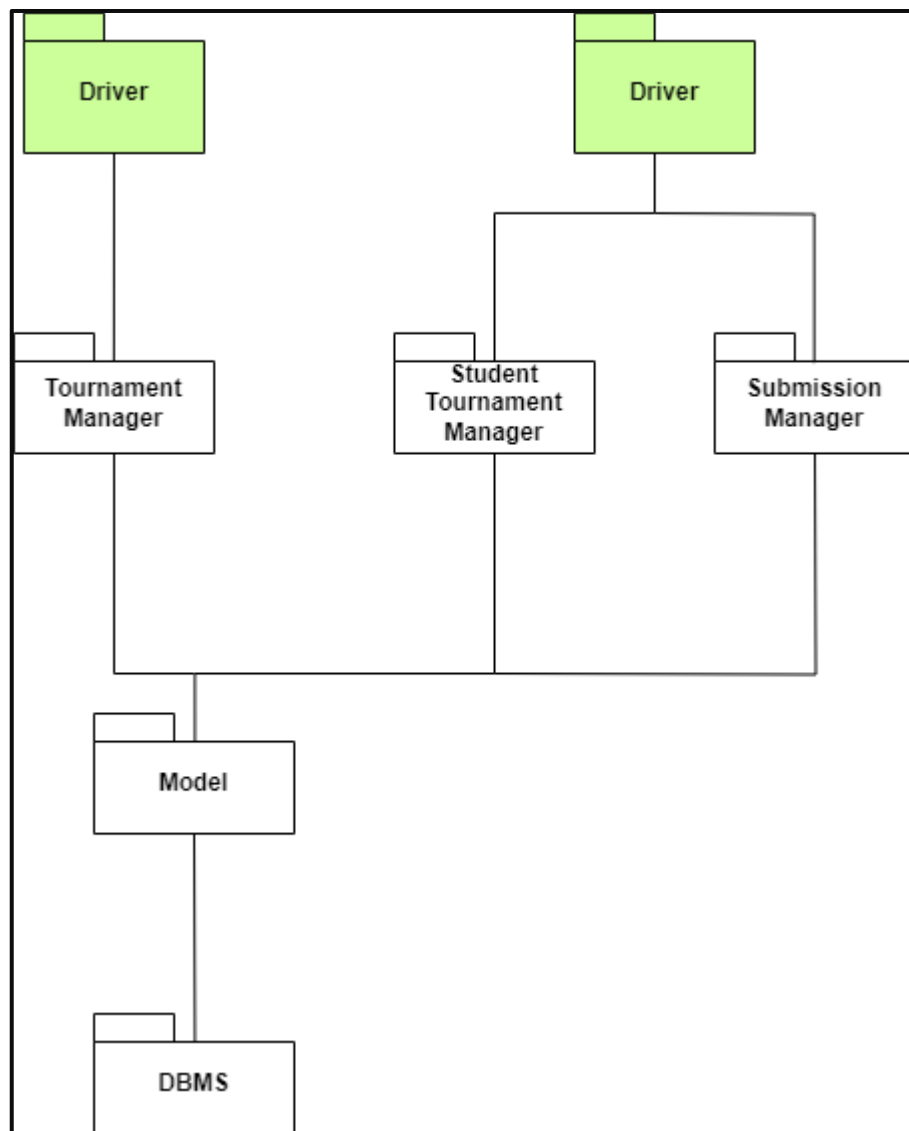
1. DBMS



2. Model

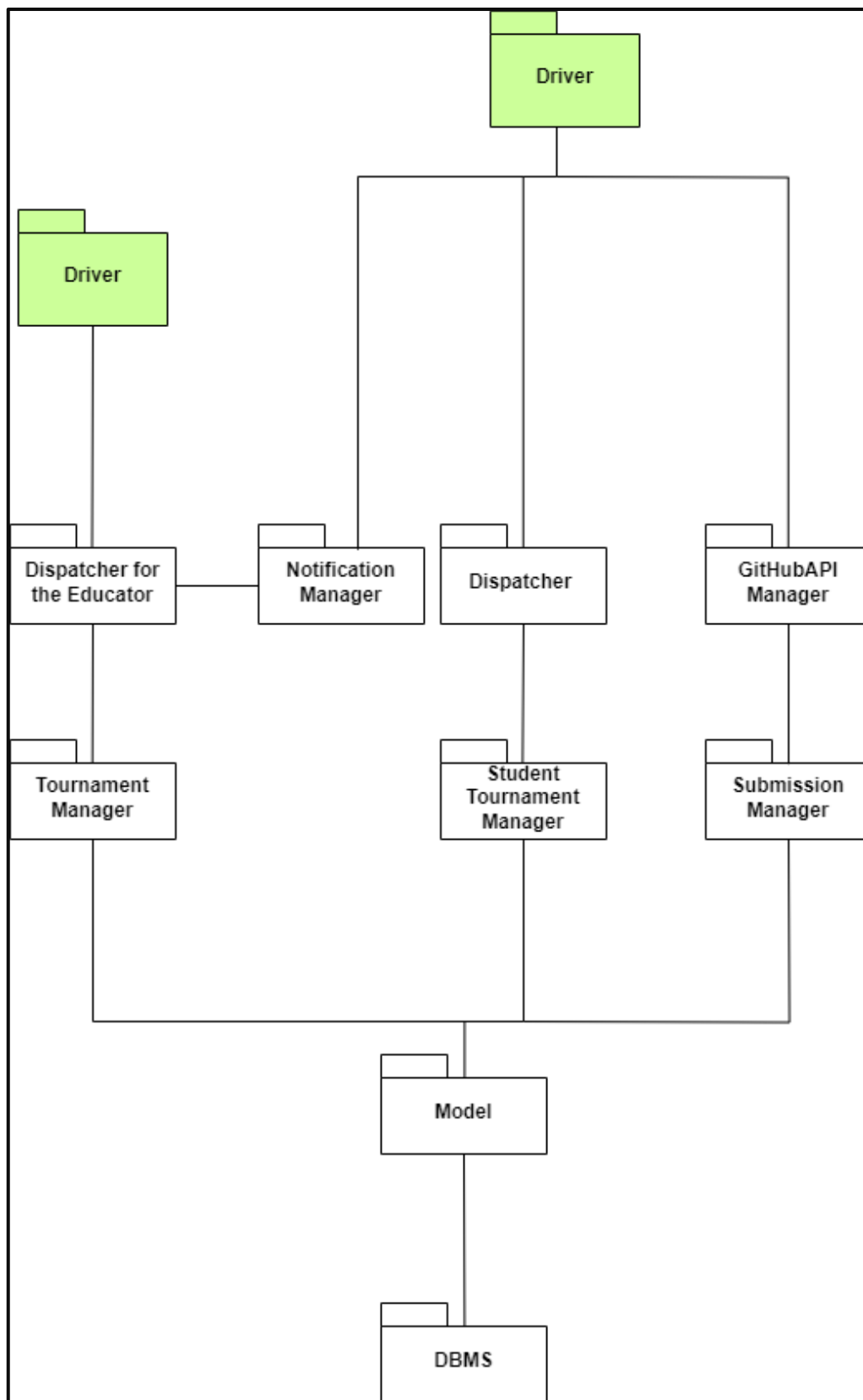


3. Controllers



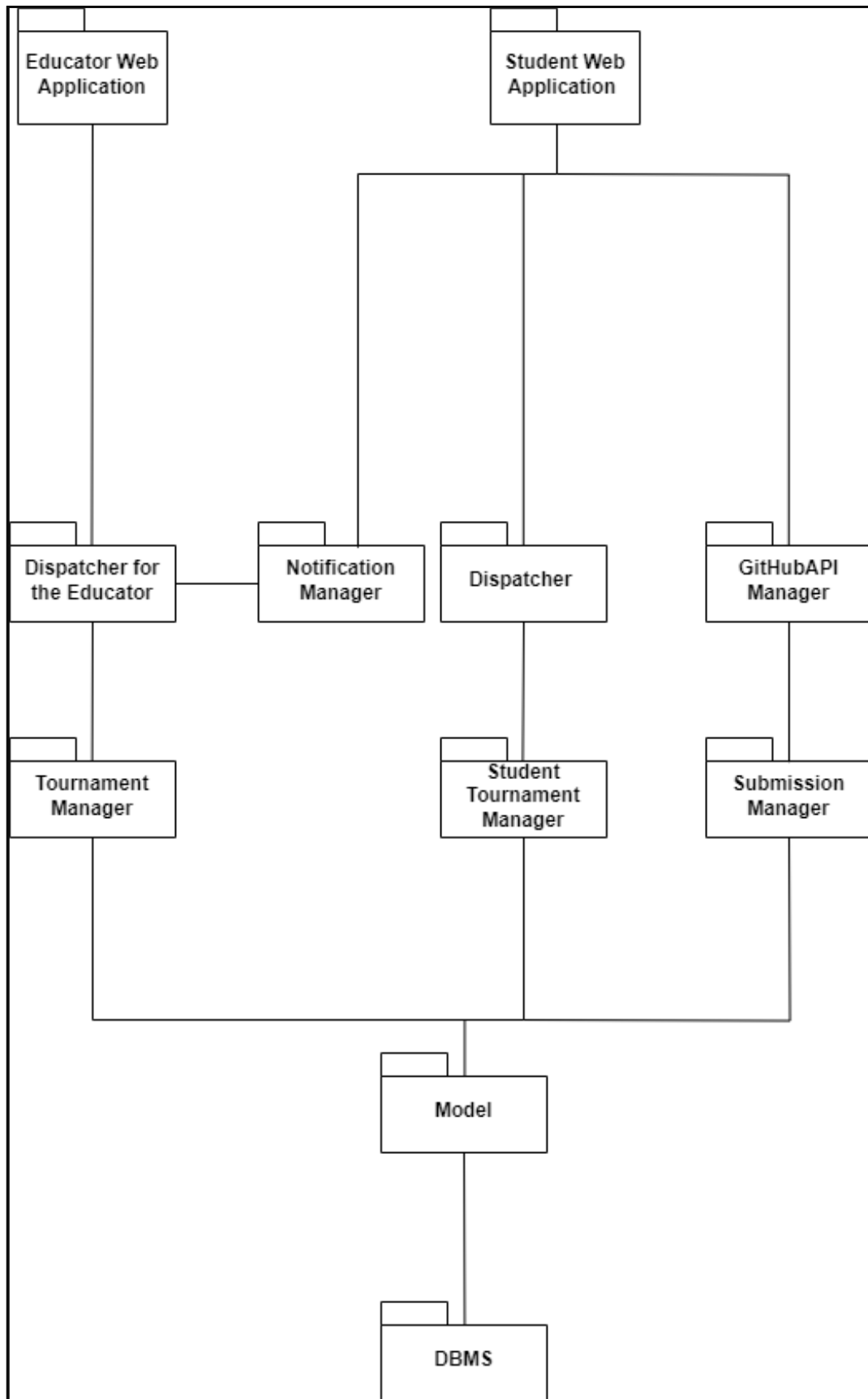
The diagrams, as said before, only show some of the controllers among all the ones present in the system, because they manage different requests but they work very similarly with each other and they rely on the same component, which is in particular the Model.

4. Dispatcher and other Controllers



In this diagram are presented the two dispatchers, one for the educators' requests and the other for students' ones. In addition, there are two controllers that don't rely on the dispatcher handling. One communicates with the dispatcher of the educator and the other one with the manager of the Git hub API which relies on an external component.

5. Clients Web Applications



When all the components of the system have been tested and integrated, the system can be run on the clients applications to check whether all the requirements defined are satisfied or not.

5.3 Static Analysis Tool Implementation

In this section, we explore the utilization of an automatic code analysis tool to evaluate the code quality produced by students in the Code Kata Battle (CKB) system. The aim is to integrate this tool seamlessly into our workflow for assessing student submissions and generating team battle scores.

This code analysis tool functions as a self-managed, automatic code review tool, designed to be integrated into existing workflows. It continuously inspects project codes, supporting over 30 different programming languages. The tool assesses code in terms of reliability, availability, security, maintainability, and other metrics, providing a numeric value as a result.

To interface with the CKB platform, it is feasible to establish a server using web APIs.

Here is a high-level workflow describing how the code analysis tool should interact with the CKB platform to automatically evaluate student submissions:

1. **Battle Creation:** The educator initiates the creation of a battle within a tournament on the CKB platform. He configures battle settings, uploads test cases, and chooses metrics (e.g., security, maintainability, reliability) for evaluating student submissions.
2. **Code Analysis File Generation:** Based on the chosen metrics, the CKB platform generates a "codeanalysisfile" (possibly a configuration file) and stores it in the GitHub repository dedicated to the battle.
3. **Repository Creation:** Once the battle is created, the CKB platform automatically generates a GitHub repository for the battle and notifies students with the repository link.
4. **Repository Forking:** Students fork the battle repository and set up GitHub Actions to interact with the CKB platform.
5. **Commit and Code Analysis:** With each student commit to the main branch, GitHub Actions trigger the CKB platform to pull the submission, test cases, and the codeanalysisfile. This information is then forwarded to the code analysis server, which evaluates the code based on specified metrics and returns the results.
6. **Battle Score Generation:** The CKB platform receives the analysis results and employs them to calculate the team's battle score, contributing to the overall assessment of student performance.

6. Effort spent

Task	Time spent
Introduction	1h
Architectural design	30h
Graphical interface design	8h
Requirements traceability	3h
Implementation, integration and test plan	5h

7. References