POLITECNICO
MILANO 1863

DD: Design Document

Cotrone Mariarosaria, De Ciechi Samuele, Deidier Simone

Professor:

Elisabetta DI NITTO

Academic Year: 2023-2024

Contents

1 | INTRODUCTION

1.1. Purpose

This document will provide an overview of the system's architecture starting from the RASD document already compiled. All design choices will be documented and explained, providing adequate justifications for each decision taken. The document will focus on the design of the system's architecture and user interface design while also providing a general implementation, integration and testing plan.

1.2. Scope

As stated in the RASD document, the CodeKataBattle software aims to create a stimulating and collaborative environment to enhance students' software development skills, providing them with a practical, competitive, and educational experience. To achieve this goal, the software is conceptualized as a web application with a user-friendly interface, allowing all users to effortlessly access information on active tournaments, scoreboards, and earned badges.

One main actor will be Students: they will interact with the systems through the web page and will be able to register for tournaments, to participate individually or in teams in programming challenges to increase their skills and earn badges.

The other actor will be Educators: they will be able to create new tournament, battles and badges.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

Definition	Description
Student	A user registered in the system, who can participate in battles and tournaments.
Educator	A user registered in the system, who can create new battles and tournaments.
Code kata battle	Platform where you can sign up for tournaments and battles and do programming exercises.

	Depending on your skills you can also get various rewards.
Notification	An alert that a certain event occurred.
Badge	Extraordinary reward awarded for certain merits in battles.
Commit	Statements that make the changes are effective and visible to other users.
Fork	Indicates the development of a new software project that starts from the source code of an existing one, by a programmer.
Tournament	A tournament is made up of one or more battles. It is created by an educator.
Battle	A battle is a programming exercise with its own ranking. It is created by the educator who created the tournament or by a collaborator.
Code kata	Programming exercise.

1.3.2. Acronyms

Acronyms	Description
UML	Unified Modeling Language
WP	World Phenomena
SP	Shared Phenomena
G	Goal
D	Domain Assumption
R	Requirement
UI	User Interface
API	Application Programming Interface
DB	Data Base
CKB	Code Kata Battle
NFR	Non-Functional Requirements
DBMS	DataBase Management System

1.3.3. Abbreviations

Abbreviations	Description
Alt	Alternative
Opt	Optional
Id	Identifier

1.4. Revision history

- January 7, 2024: version 1.0 (first release)

1.5. Reference Documents

- Specification document: "R&DD Assignment A.Y. 2023-2024"
- Visual Paradigm Online: https://online.visualparadigm.com
- Sequence Diagram: https://sequencediagram.org
- Mock-ups: https://moqups.com/it

1.6. Document Structure

The DD is structured in the following seven chapters:

1. **INTRODUCTION:** It gives a brief description of the document that will be presented, it gives general information about what this document is going to explain.
2. **ARCHITECTURAL DESIGN**: This chapter describes the architectural choices adopted to build CKB, with an overview of the main components and their interactions, in the final section it also describes some deployment choices.
3. **USER INTERFACE DESIGN**: Mock-ups of Web UIs of Students and Educators.
4. **REQUIREMENT TRACEABILITY**: Chapter that explains how the requirements listed in the RASD maps design choices were made in this document.
5. **IMPLEMENTATION, INTEGRATION AND TEST PLAN**: This chapter describes some guidelines that will be used to implement CKB, in which order components and sub-components are integrated and how they are tested.
6. **EFFORT SPENT**: Effort spent by the team members; a table that contains the hours required to realize this document.
7. **REFERENCES**: All relevant references to external sources used in this document.

2 | ARCHITECTURAL DESIGN

2.1. Overview

In this section is given an overview of main architectural styles: the application is a distributed one, with a Client-Server paradigm. A three-tier architecture will be adopted to increase system's flexibility, scalability and maintainability. The three layers consist of:

- **Presentation Layer**: it manages the presentation logic, i.e., how the system interacts with the user: it contains how information is interfaced (graphically) and rendered. This layer is accessible by the user through a graphical user interface (GUI).
- **Application Layer**: it will handle the business logic by providing all functions made available to the Users. It will be also responsible for data access.
- **Data Layer**: it is responsible of storing and retrieving asked data, it does not implement any logic and it is only used for data storing. It will be composed of a DBMS and a database. Regarding the DBMS, it should be clustered to have scalability, fault tolerance, load balancing, incremental scalability and easier maintenance.

This separation is not only about how things are organized logically but also where they physically exist. Each layer is like a different level in a building, and each level corresponds to a machine or a cluster of machines to do its job well.

To ensure the properties mentioned above, it is useful to duplicate the application layer. Since the platform is exclusively a web application, the distributed architecture exploits a load balancer to distribute the load of requests among the web servers.

To guarantee the security of sensible data, firewalls have been designed.

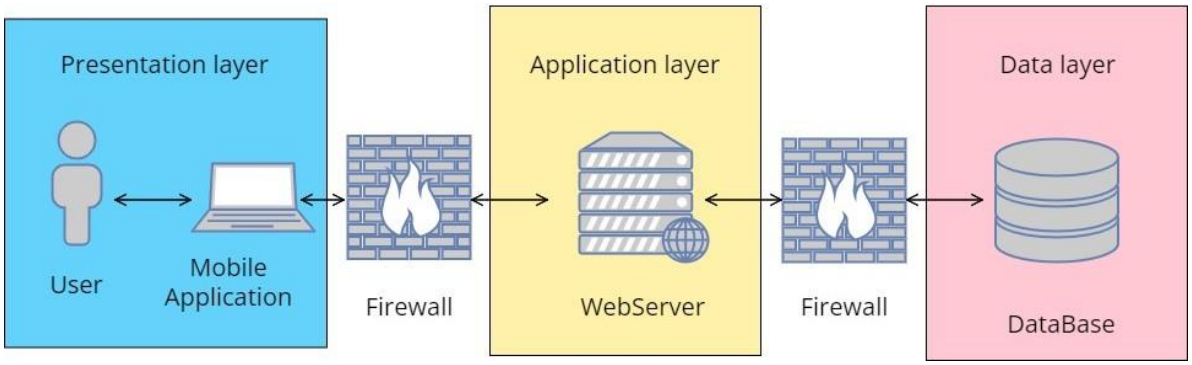

Figure 1: Three tier architecture.

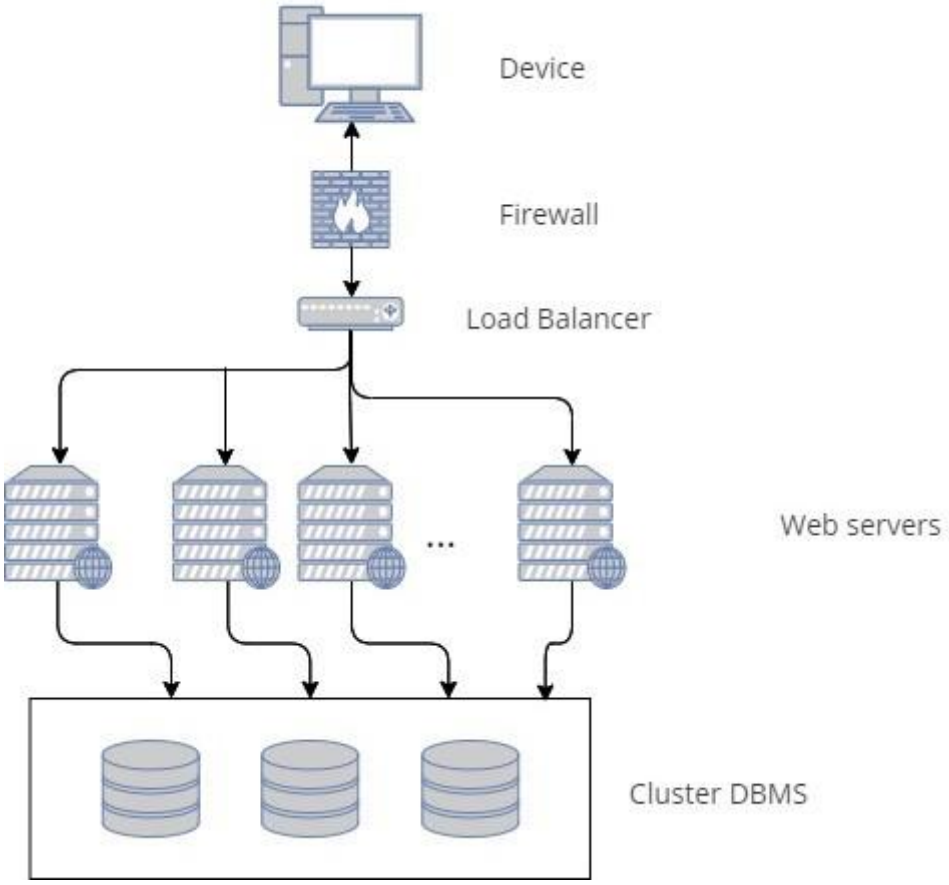

Figure 2: System topology.

2.1.1. Class Diagram

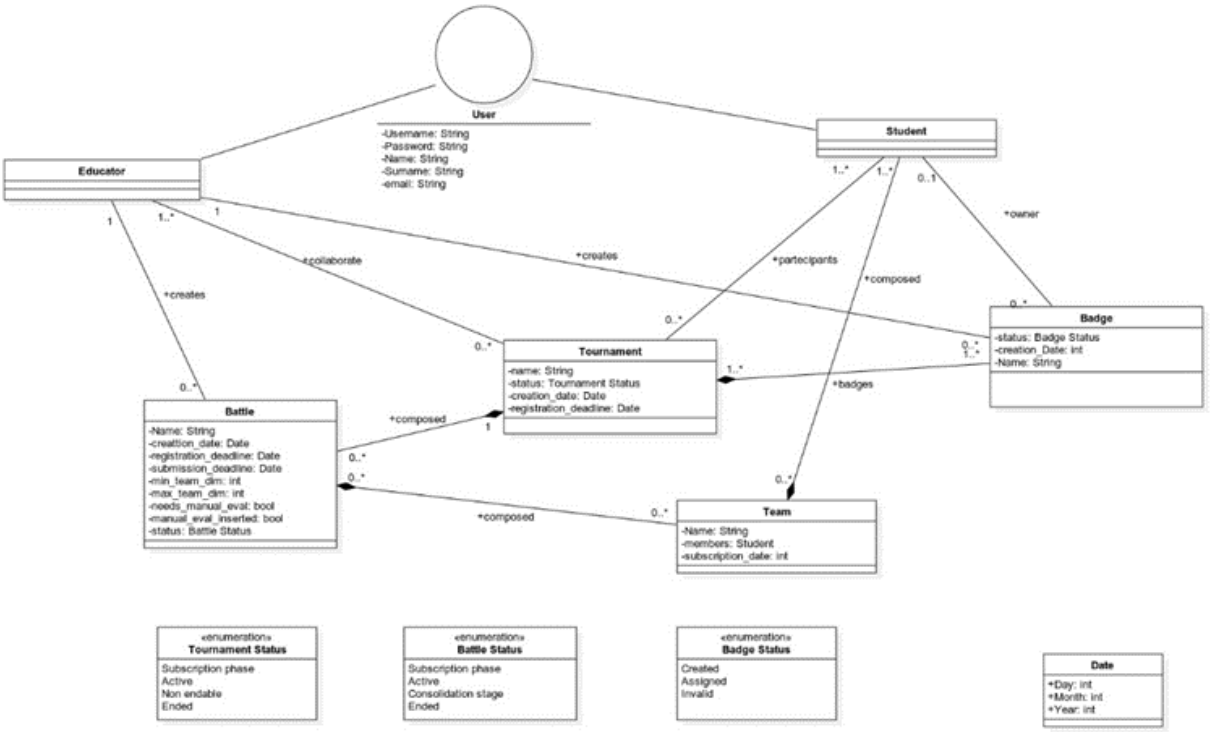

Figure 3: Class Diagram.

2.2. Component view

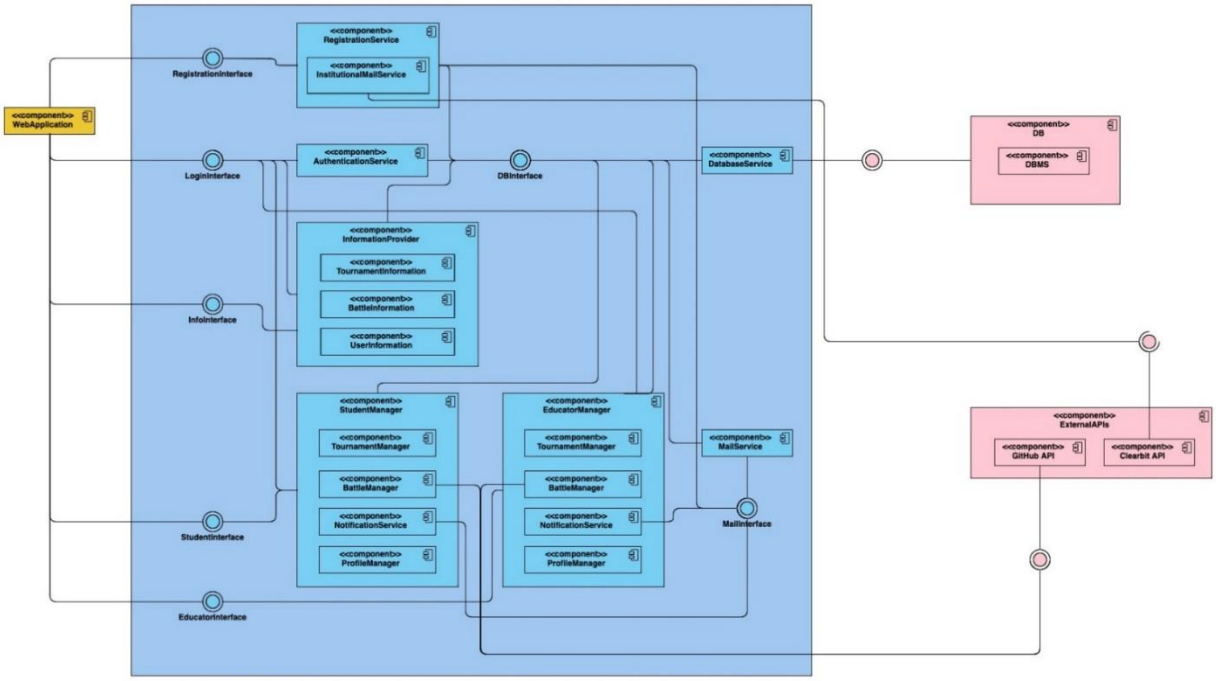

Figure 4: Component Diagram.

Here is a brief description for every component shown in the figure:

- **Web Application:** interface used by students and educators to access the CKB platform. It will display information and enable operators to engage with the system by adjusting specific parameters.
- **Mail Service:** it allows to send e-mail to all the users.
- **Registration Service:** it plays the role of managing the user registration process storing all data in the DB by interacting with the Database service. It interacts with the e-Mail Service to send a confirmation link.
    - **Institutional Mail Service:** this component is used to verify that an educator registers with an institutional email and interacts with an external API: Clearbit API.
- **Authentication Service:** allows users to login by checking their credentials. Must interact with the Database Service to access personal data and validate the login attempt.
- **Student Manager:** this component manages all the functionality related to students. All Components will save in the DB all operations performed to keep an updated record of tournament and battles in which they are entered.
    - **Tournament Manager:** it deals with the management of tournament registration by a student. To do so it will have to communicate with the DB.

    - **Battle Manager:** it will handle the life cycle of a battle registration: creation of the team and the final confirmation. To do so it will have to communicate with the DB. It interacts also with GitHub API to allow students to fork the Repository
    - **Notification Service:** it handles all notifications generated by Students' actions. The message must be created, stored and showed in the corresponding page so also this component will interact with the DB. It interacts also with e-Mail manager to send notifications to all students also via e-mail, for example when they are invited to participate in a team.
    - **Profile Manager:** it allows students to view and manage their profiles within the system such as badges won in tournaments. Users can edit personal information in their profile, such as first name, last name, nickname, and other relevant information. It provides a secure interface for managing passwords, allowing users to change them. To save all changes, this component interacts with DB.
- **Educator Manager:** this component manages all the functionality related to Educators. All Components will save in the DB all operations performed to keep an updated record of the tournament, battles and badges that they have created, tournaments in which they participate as collaborators.
    - **Tournament Manager:** it will handle the whole life cycle of a creation of a tournament that includes the creation of badges and the insertion of subscription deadlines. It will store all the information in the database.
    - **Battle Manager:** it will handle the whole life cycle of a creation of a battle that includes the insertion of subscription deadlines, code kata, minimum and maximum members in a team. It will store all the information in the database. It interacts also with GitHub API to create a repository when the battle starts.
    - **Notification Service:** it handles all notifications generated by Educator's action. The messages must be created, stored and showed in the corresponding page so also this component will interact with the DB. It interacts also with e-Mail manager to send notifications to all educators also via e-mail, for example when they are invited to participate in a tournament as collaborators.
    - **Profile Manager:** it allows educators to view and manage their profiles within the system. Educators can edit personal information in their profile, such as first name, last name, nickname, and other relevant information. It provides a secure interface for managing passwords, allowing users to change them. To save all changes, this component interacts with DB.

- **DB:** it includes all components tasked with storing, maintaining and managing data
    - **Database Service:** it allows all the components that interact with it to communicate with the database.
- **Information Provider:** it includes all components that will handle all requests for information, so they interact with the database.
    - **Tournament information:** it manages all relevant tournament information in a timely and accurate manner when requested by users. It provides important deadlines associated with the tournament, such as those for registration, presents a scoreboard (leaderboard) that is always up to date. It also provides all current battles and badges that can be won in the tournament.
    - **Battle information:** it manages all relevant battles information in a timely and accurate manner when requested by users. It provides important deadlines associated with the battle, such as those for registration or the minimum and maximum number of members in a team. It presents a scoreboard (leaderboard) that is always up to date.
    - **User information:** it is designed to enable the display of information associated with another user within a system such as badges won in a tournament.
- **External API's:**
    - **GitHub API:** This component provides an interface and feature set for interacting with the GitHub API, enabling the application to communicate with repositories. It is used for the creation of a repository and for the communication between GitHub and CKB when a commit is made.
    - **Clearbit API:** This component provides an interface and feature set for interacting with the Clearbit API, enabling the application to verify an institutional mail.

2.3. Deployment View

This section will outline the deployment strategy for the upcoming system and provide insights into the involved components, both in terms of hardware and software.

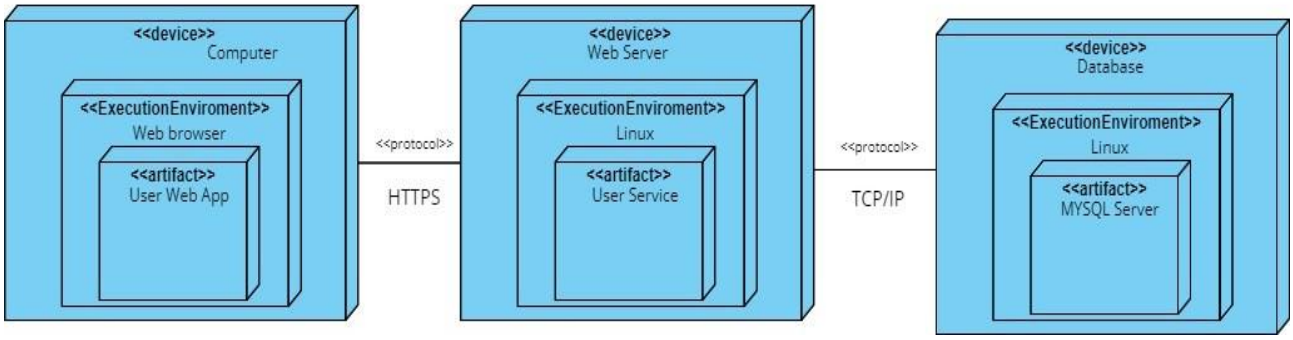

Figure 5: Deployment View.

In distributed systems, software logic layers are installed on hardware layers (tiers), where a tier represent a machine (or set of machines), each with its own processing capability. We use a three-tiered architecture where the three layers are divided among as many dedicated machines: a user workstation (computer), a middle server (web server) and a data management server. An application with three tiers has more scalability and flexibility.

- **Presentation tier:** users can interact with the system through his own computer using a web server to access CKB services.
- **Middle tier:** it is composed by the application server that receives all the requests sent from users accessing the CKB web application via a web browser. It communicates directly with the Web Application through the HTTPS protocol to handle all requests and can require some interrogations to the database.

  The application server will also act as Web Server, there is no need to decouple the Web Server from the Application Server.
- **Data tier:** students and educators will use a Database to store all persistent data needed to operate the system. It will interact with the Application Tier using the TCP/IP Protocol.
- **Load balancer:** the use of replication nodes requires the introduction of a load balancing system that will distribute the processing load among these nodes. It is included between the presentation tier and the middle tier. This will reduce the time needed to process all Request resulting in faster response times and improved reliability.
- **Firewall:** it is a set of hardware and/or software components that connect a trusted network to insecure networks. It controls by implementing appropriate security policies, the traffic entering and leaving the secure network, with the purpose of preventing, detecting and aborting any attacks and unauthorized requests. They are placed between presentation tier and middle tier, and between the middle tier and data tier.

In the figure load balancer and firewall are omitted.

2.4. Runtime View

In the following sequence diagrams, we are going to represent the behaviour of the most important components of the CKB platform. The diagrams show just a high-level representation of the interaction between the various components.

2.4.1 Student’s sequence diagrams

2.4.1.1 Sign up

This sequence diagram illustrates the student's registration process using the Web Application. After transitioning from the login page to the registration page, the student completes the registration form. Subsequently, the Web Application validates all entered credentials.

Upon validating the credentials, the Web Application sends the registration request to the RegistrationService component. The RegistrationService is responsible for verifying the accuracy of the credentials and, if valid, storing them in the database via a call to the DatabaseService. In the event of an error, the RegistrationService notifies the Web Application, which displays an alert to the student, informing him of the encountered problem. This allows the student to attempt registration again.

If the credentials are correct, the RegistrationService requests the MailService to send a verification message to the student’s e-mail. Upon receiving the e-mail, the student can click on the provided link within the specified timeframe to complete the registration.

Finally, the Web Application displays the login page once the entire registration process is completed.

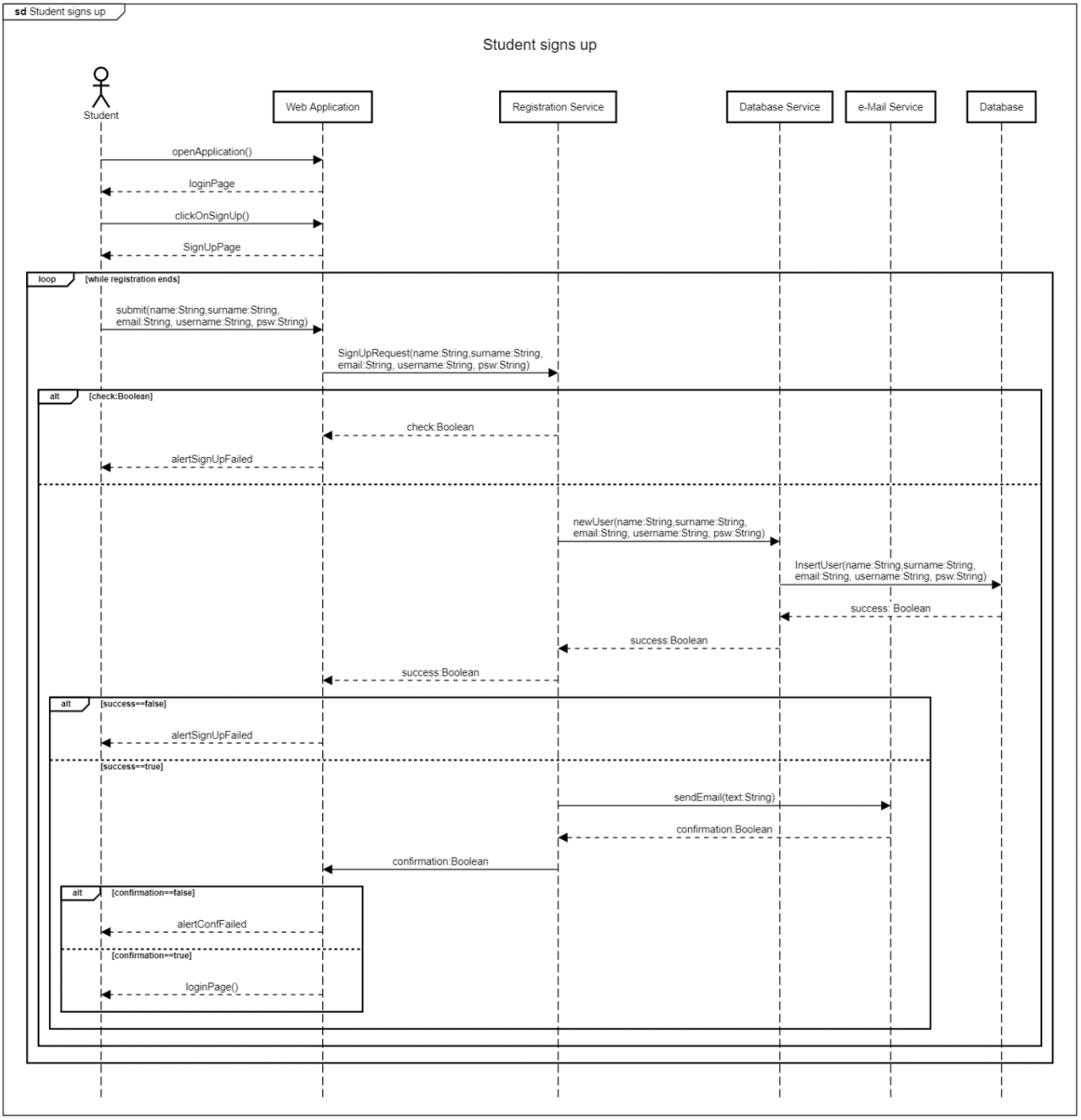

Figure 6: Student signs up.

2.4.1.2 Log in

A student, upon opening the website, is shown the login page. Submitting the login form, the data is sent to the AuthenticationService component, responsible for accessing the database, through the DatabaseService component for checking if the data inserted by the user is correct. The student must provide his username or his email address and his password. The Database responds with an integer, with a value of zero if the login data was incorrect, one if the user logging in is a student or two if the user logging in is an educator, and the userID (minus one in case the boolean is zero), an integer value

relative uniquely to the user. The WebApplication receives this data, showing an error and redirecting the user to the login page in case of a value zero or creating the session for the user, retrieving all tournaments from the database using the TournamentInformation component, accessed through the InformationProvider. The application splits the tournaments into two categories, “My tournaments” and “All the tournaments”, using the userID stored in the session and comparing it to the admins in a tournament, and redirects the user to his personal page.

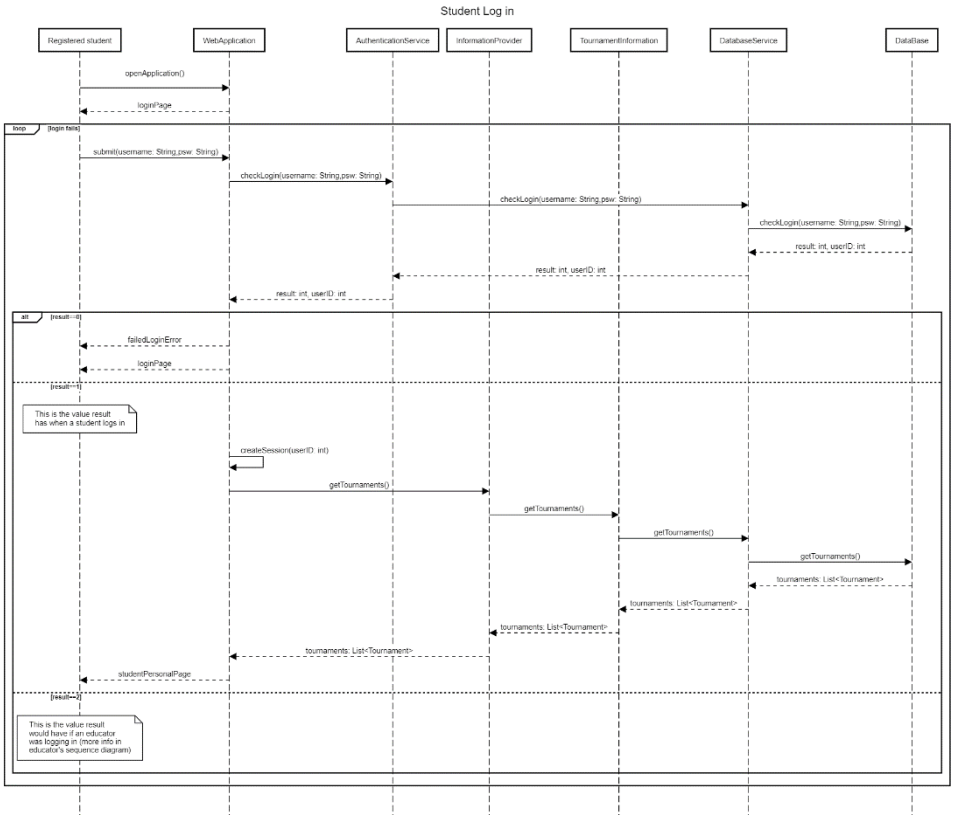

Figure 7: Student log-in.

2.4.1.3 Log out

A student can log out of his or her profile from any page on the site since the button with the corresponding function is always located in the side banner that is present on every page. Once the student requests to log out, the application simply invalidates the session and performs a redirect to the log in page, thus forcing the student to log in again.

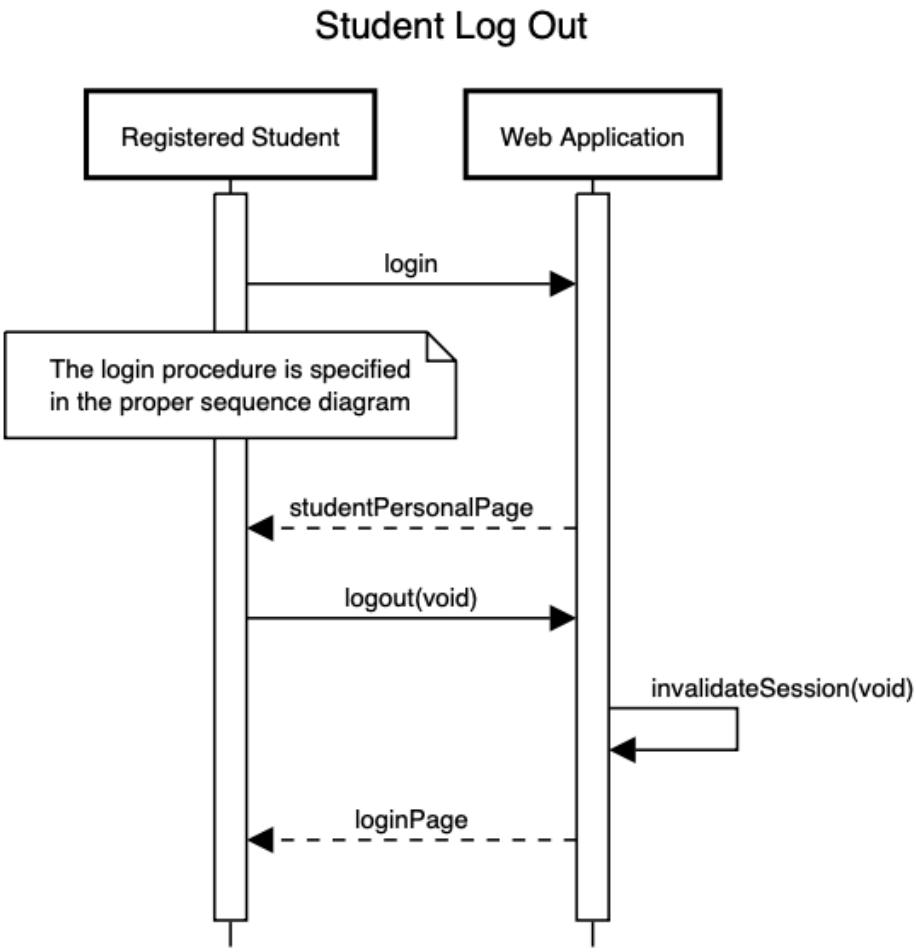

Figure 8: Student log-out.

2.4.1.4 Show tournament's information

A student able to perform the login operation is redirected to his personal page. He has access to all the tournaments, split in two categories: "My tournaments" and "All the tournaments". When he clicks on "More information on this tournament!", next to a tournament, a request containing the tournamentID is sent to the application. The request is so redirected through the InformationProvider for the TournamentInformation provider, who then works with the DatabaseService to get all the data of the tournament from the database. The Database replies with a tournament object, that is NULL if no tournament was found with that ID. The object is provided back to the WebApplication that loads the tournament page, using the data in the tournament object, or shows an error in case of a NULL object.

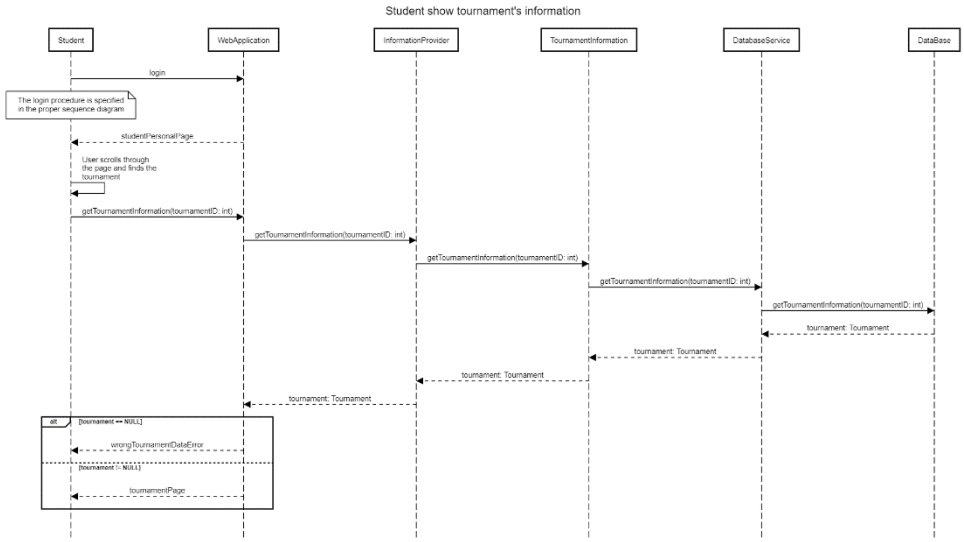

Figure 9: Student show tournament information.

2.4.1.5 Register in a tournament

This sequence diagram illustrates the student's tournament registration process using the Web Application. From the tournament page for which the student previously applied, he or she clicks on the registration button. At this point the Web Application through the StudentManager and TournamentManager updates the Database. If the addition was unsuccessful an error page is shown, otherwise the student returns to the tournament page.

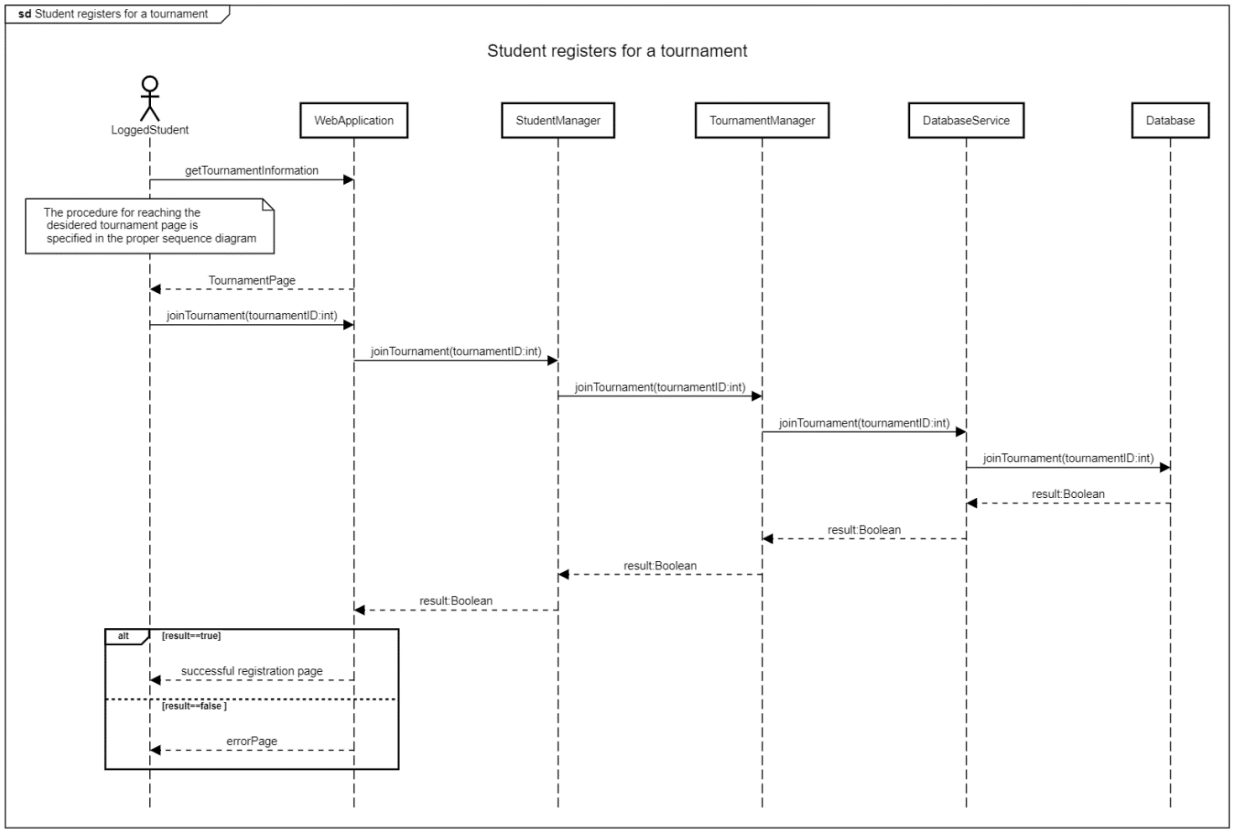

Figure 10: Student registers for a tournament.

2.4.1.6 Show battle's information

This sequence diagram illustrates when a student wants to see the battle's information using the Web Application. After transitioning from the login page to his personal page and then to the tournament page, the student clicks the button for getting battle information.

At this point the information provider asks the battle information to query the database and show it all the battle information. If the information is loaded correctly the battle page is shown, otherwise the error is presented and the student returns to his main page

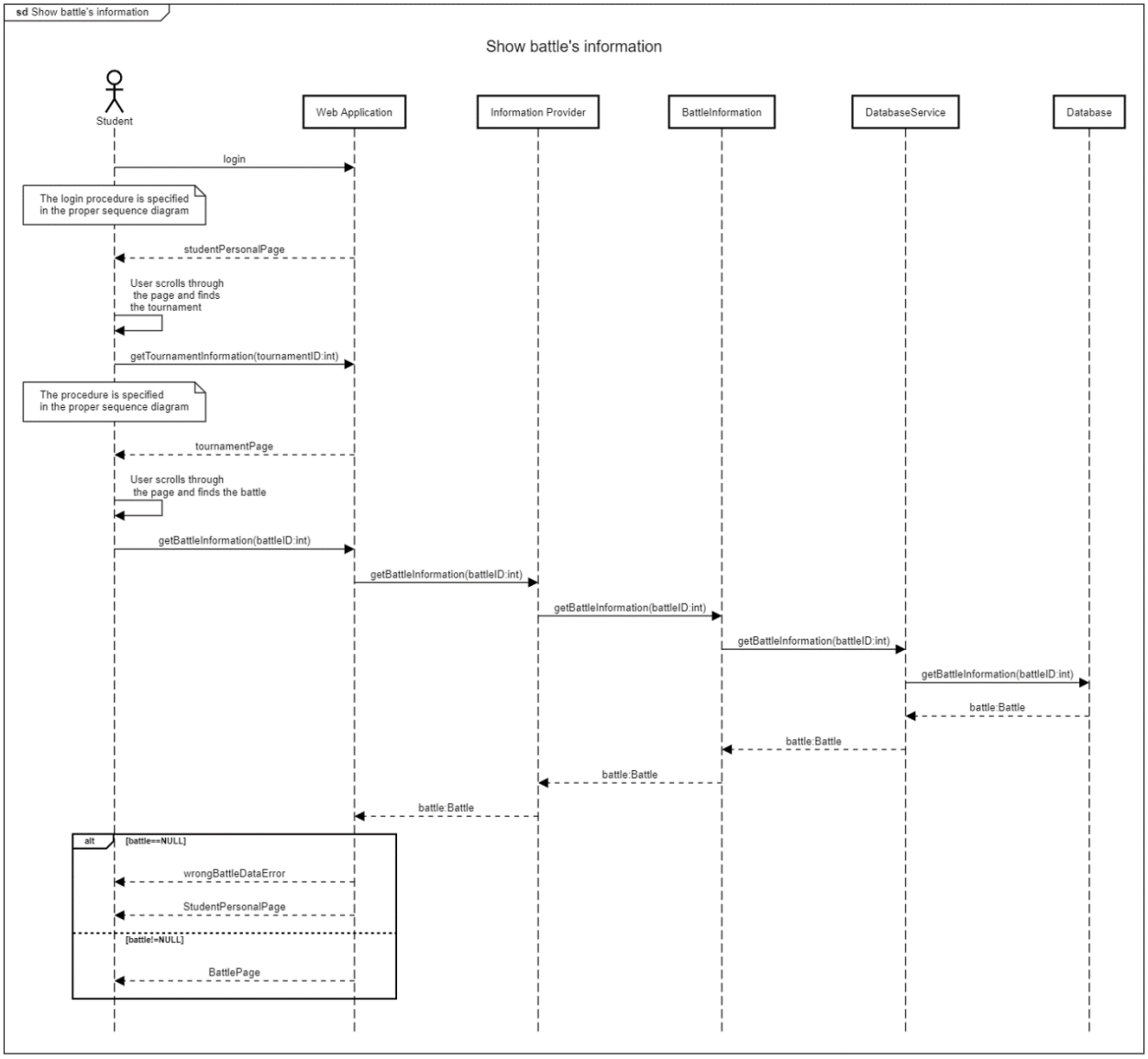

Figure 11: Student show battle's information.

2.4.1.7 Register in a battle

The procedure for a student to view a battle information page is explained the previous sequence diagrams. If the battle shown in the page still has not reached the subscription deadline, it is possible for the user to click the 'Register!' Button. A request is sent to the server with the battleID and the WebApplication processes it, adding the studentID integer value, stored in the session, to link the request to the specific student. The request goes through the StudentManager and reaches BattleManager, that requests the operation to the database. The database replies with a boolean result value, true if the operation was completed, false otherwise, a String called error, which may contain information about an error if the boolean is false or specify how the operation was completed if the boolean is true, and the battleID, the same as the request. The WebApplication receives this data and processes it. In case the result boolean is false an error page is loaded, using the information contained the String error; instead if the result value is true there are two possibilities, depending on the information contained in the error String: if the battle had as values one for both the minimum number of player and for the maximum number of players in a team, the error value is “ONE_STUDENT_BATTLE” and so the database already confirmed the subscription of the student, that is so redirected to the “successful registration page”; if the value of error equals NULL, the user is redirected to a waiting room, where he can assemble his team for the battle, by sending invitation to other students.

When the team has been assembled, the user can click on the 'Join the battle!' Button, sending a request to the server with the integer battleID. The application links to the request the userID of the student, contained in the session and through the StudentManager, the request reaches the BattleManager component. The database is accessed through the DatabaseService. The database replies with a boolean result value, that is true when the subscription to the tournament is accepted, false otherwise, and a String, containing information about the error in case the boolean is false or NULL instead. The application processes this data and shows the user an error page if result is false or the successful registration page if result is true.

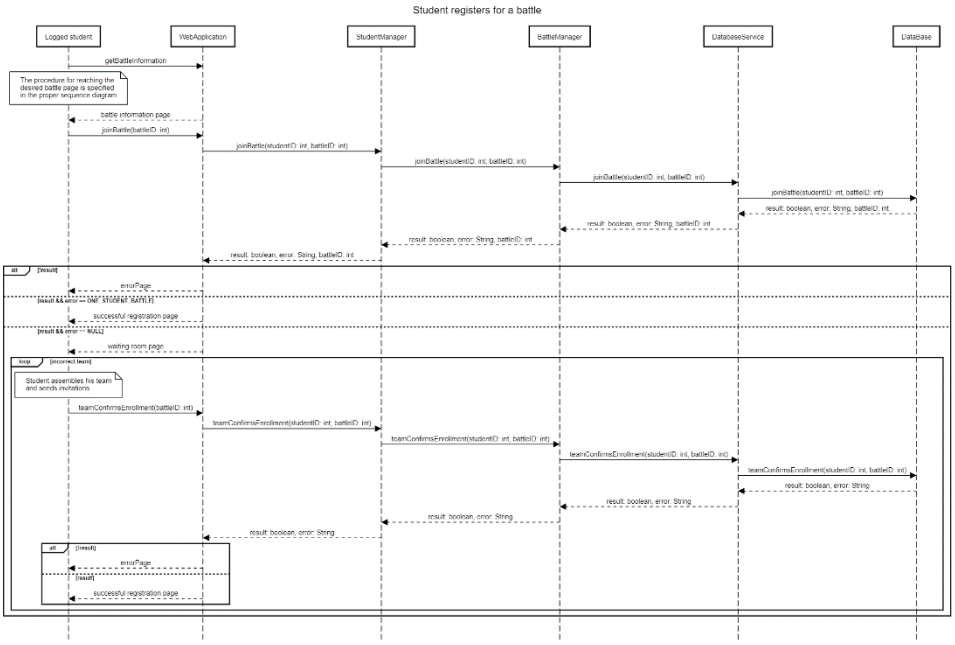

Figure 12: Student registers for a battle.

2.4.1.8 Invite a student into the team

When a student is on the waiting room page for the creation of a team for a battle, he/she can perform two actions: modify the search criteria on the page for inviting new students, in this case dynamically the application takes the data in the search form, passing through the InformationProvider and UserInformation accesses the database to search for possible matches, and the result proposes it as a possible completion of the search criteria entered in the form. After correctly completing the name of a user, the student can click the button to send the team invitation to it; the application passing through the StudentManager and the BattleManager arrives at the database where, thanks to the studentID data, it checks that the selected student is not already a member of the team (result = false in case he already is). In case he is not, the application accesses his web address and thanks to the NotificationService and the MailService sends the invitation to the desired user, sets the value of the boolean variable result to true, and notifies the student that the submission was successful.

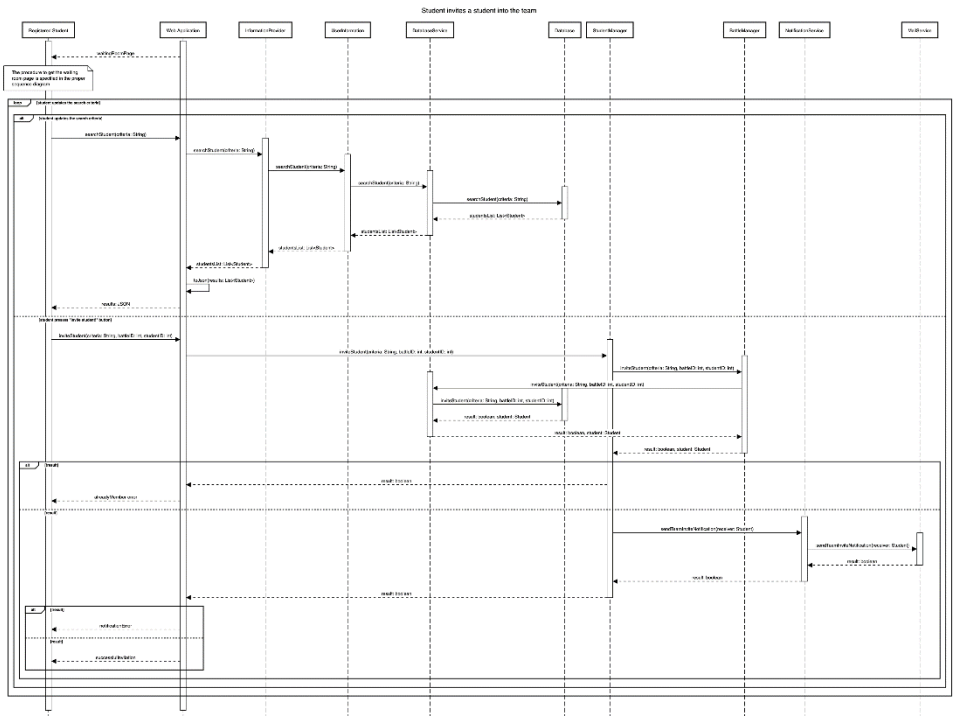

Figure 13: Student invites a student into the team.

2.4.1.9 Confirm team registration

A student after inviting other students to the team for a battle can enter the name of the team they are creating and register the team. In doing so, the application performs a check on the data entered and the data in the database, passing through the StudentManager and the BattleManager. The function will return an integer value (result) that takes value 0 if the team has been successfully created, 1 in case the student has chosen a team name that is already in use, 2 in case the team that is trying to register does not meet the minimum and maximum membership requirements for the battle, or 3 in case the student is trying to register the team for a battle whose registration date has already passed; in all cases the user is notified and only in the last one is a redirect to the page of the tournament to which the battle in which they were trying to register is part of.

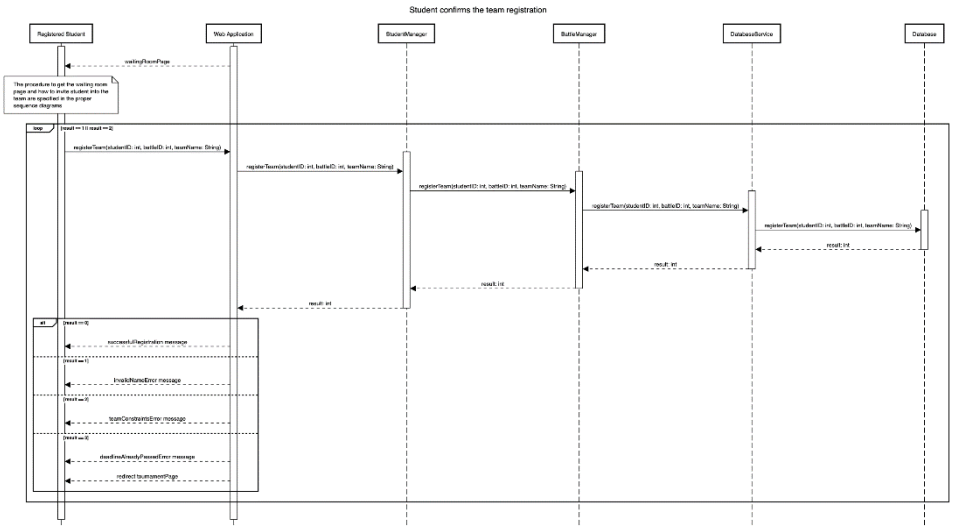

Figure 14: Student confirms the team registration.

2.4.1.10 View my profile

A student, after completing the login procedure (specified in the previous "login" sequence diagram), has access to his personal page. By clicking on "My profile" a request is sent to the server, containing the user's identification ID. The ID is redirected through the InformationProvider to the UserInformation component; the database is accessed through the DatabaseService and an object student is what the database provides, that is NULL in case of errors. The student object is then processed by the WebApplication that loads the student's profile if the object is not NULL and the student's ID is the same as the userID stored in the session, an error page otherwise.

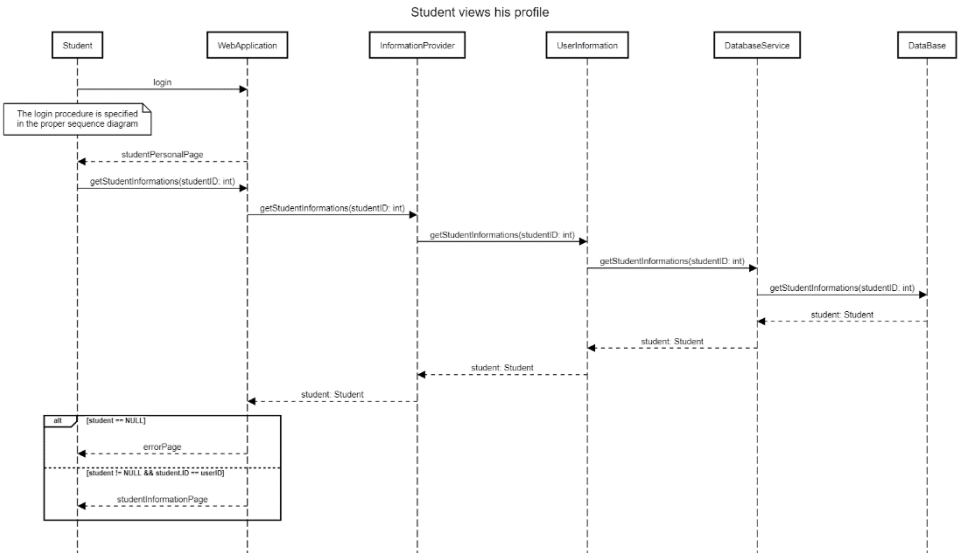

Figure 15: Student views his profile.

2.4.1.11 Search a student's profile

A student, after entering the search criteria in the appropriate bar always present in the side banner of the Web site, pressing enter from the keyboard makes the call to the function that through the InformationProvider and UserInformation accesses the database to do a search for all students who have a match with the criteria entered. All students who present a match are entered into a list, and if the list is empty, an error message is presented to the user, otherwise a page containing the list of all students in the above list is loaded. The user, by clicking on one of the profiles on the page, accesses a new page where all data related to the chosen student (identified by his ID) are presented.

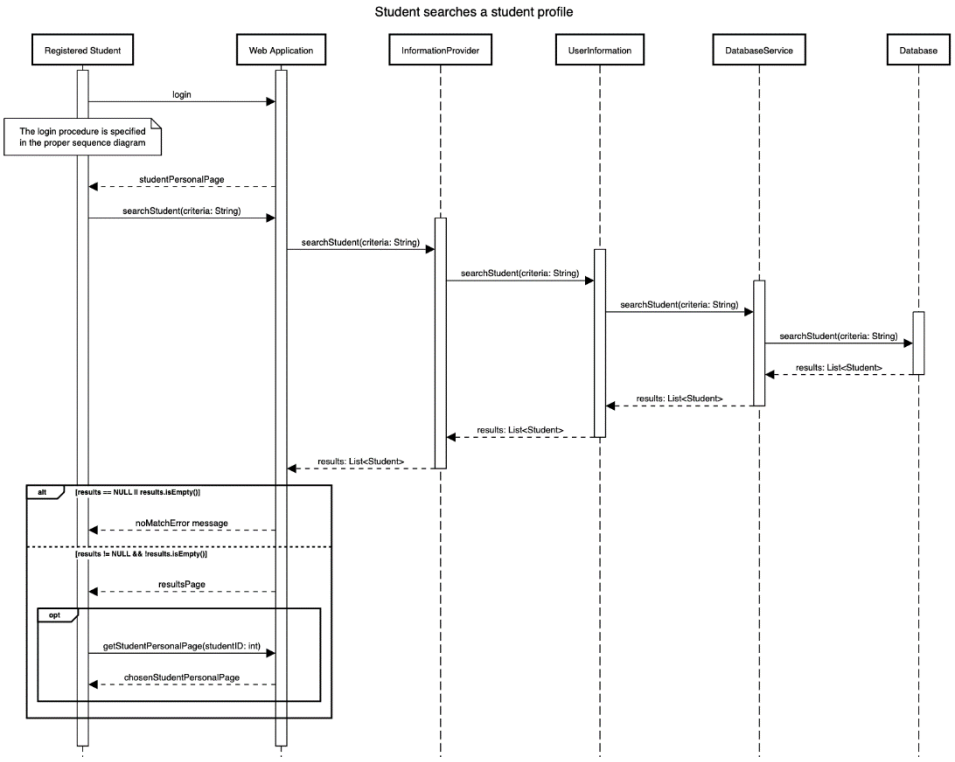

Figure 16: Student searches a student profile.

2.4.1.12 View my notifications

This sequence diagram illustrates when a student wants to see his notifications using the Web Application. After transitioning from the login page to his personal page, the student clicks the “View my Notification” button.

At this point the Student Manager asks the Notification Service to query the database and show it all the information about its notifications. If the information is loaded correctly the notifications page is shown, otherwise the error is presented and the student returns to his main page.

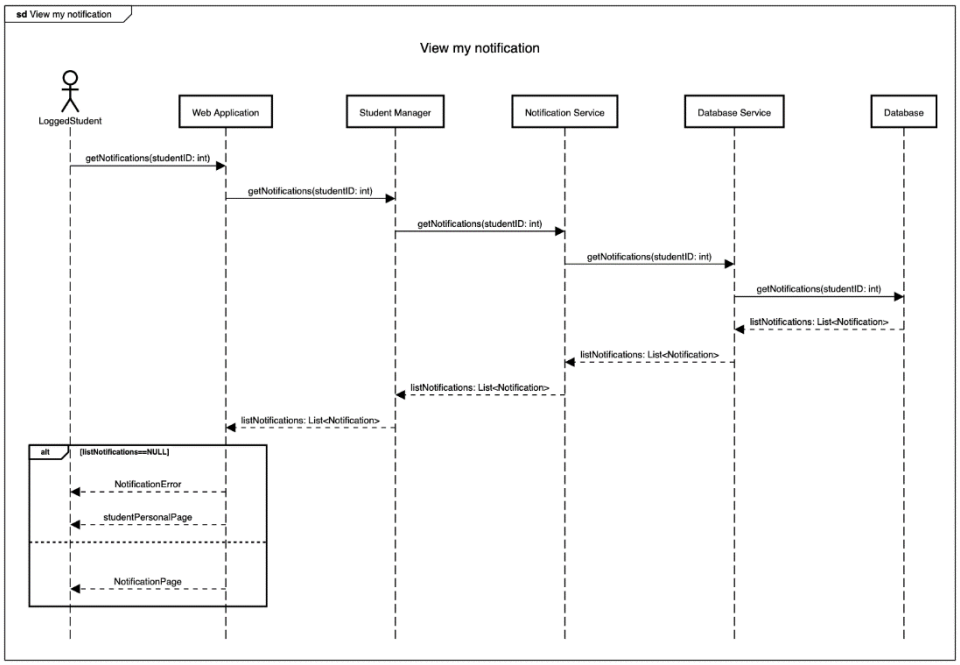

Figure 17: Student views my notification.

2.4.1.13 Reply to a team's invitation

This sequence diagram illustrates when a student wants to respond to a team invitation using the Web Application.

From its main page the student clicks the button to see its notifications. At this point he decides to respond to an invitation. The Student Manager, through the Battle Manager, makes the update of the invitation status to the database. If the update is not successful, an error page is shown. On the other hand, if the update was successful, the Battle Manager takes care of adding the student to the team through a database modification. If this change is unsuccessful an error page is shown, otherwise the updated notification page is shown.

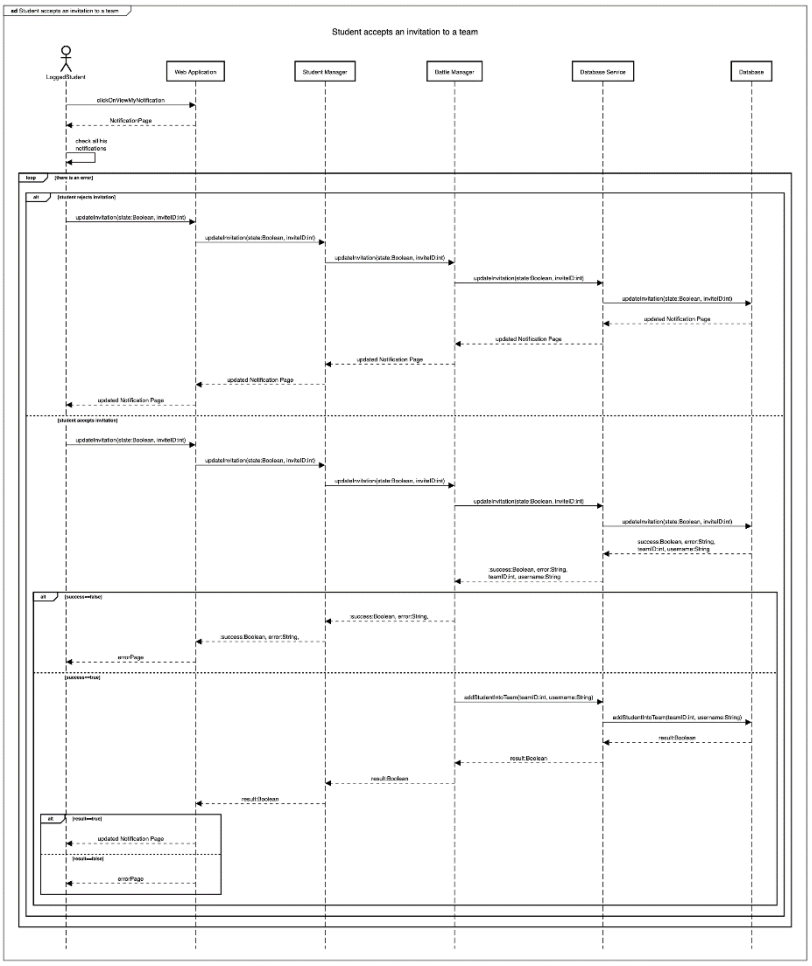

Figure 18: Student accepts an invitation to a team.

2.4.2 Educator’s sequence diagrams

2.4.2.1 Sign up

This sequence diagram illustrates the educator's registration process using the Web Application. After transitioning from the login page to the registration page, the educator completes the registration form. Subsequently, the Web Application validates all entered credentials.

Upon validating the credentials, the Web Application sends the registration request to the RegistrationService component. The RegistrationService is responsible for verifying the accuracy of the credentials. After that RegistrationService call ClearbitAPI to check if the e-Mail is institutional or not and, if valid, then stores credentials in the database via a call to the DatabaseService. In the event of an error, the RegistrationService notifies the Web Application, which displays an alert to the educator, informing him of the encountered problem. This allows the educator to attempt registration again.

If the credentials are correct, the RegistrationService requests the MailService to send a verification message to the educator's e-mail. Upon receiving the e-mail, the educator can click on the provided link within the specified timeframe to complete the registration.

Finally, the Web Application displays the login page once the entire registration process is completed.

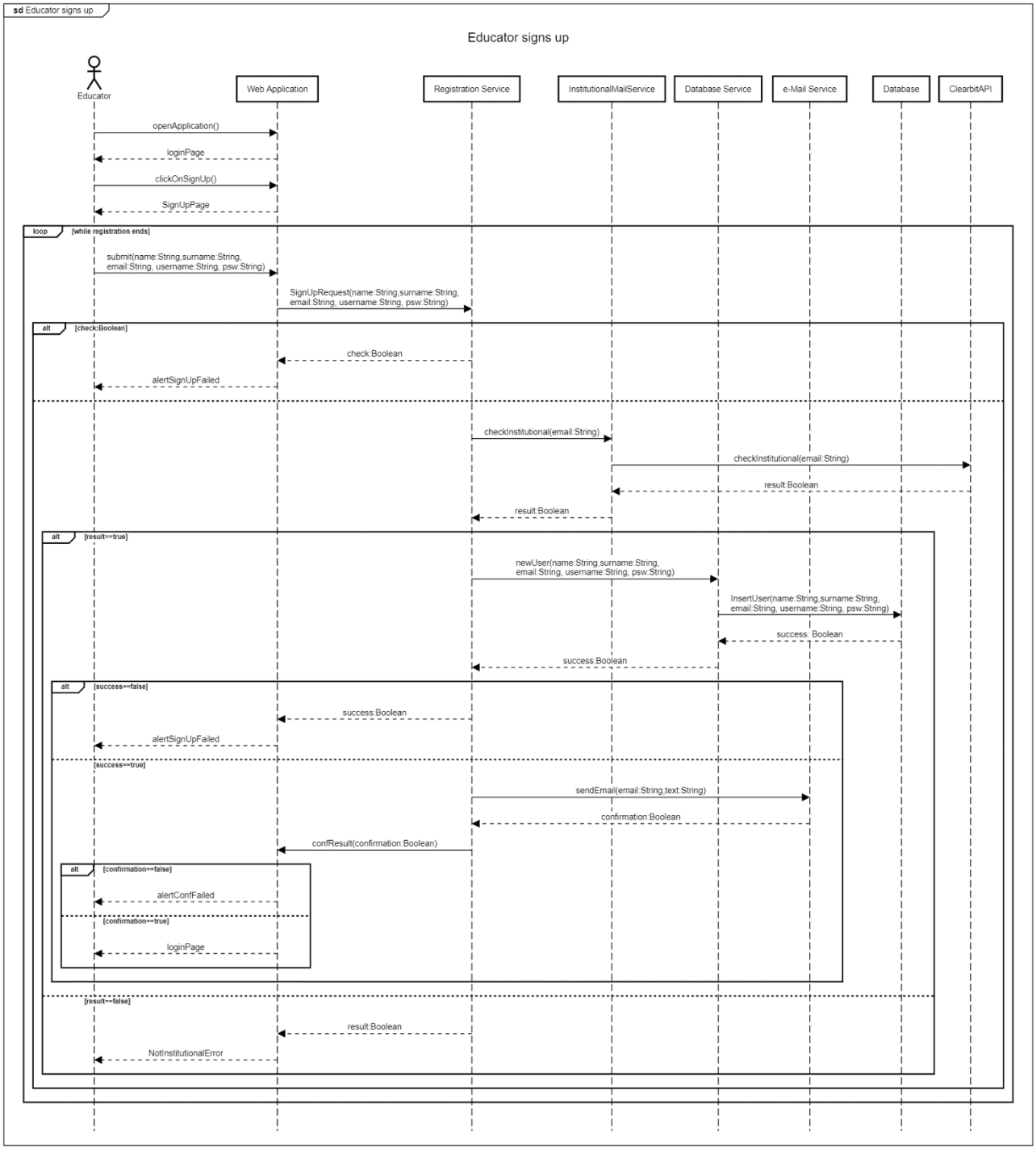

Figure 19: Educator signs up.

2.4.2.2 Log in

An educator, upon opening the website, is shown the login page. Submitting the login form, the data is sent to the AuthenticationService component, responsible for accessing the database, through the DatabaseService component for checking if the data inserted by the user is correct. The educator must provide his username or his email address and his password. The Database responds with an integer, with a value of zero if the login data was incorrect, one if the user logging in is a student or two if the user logging in is an educator, and the userID (minus one in case the boolean is zero), an integer value relative uniquely to the user. The WebApplication receives this data, showing an error and redirecting the user to the login page in case of a value zero or creating the session for the educator, retrieving all tournaments from the database using the TournamentInformation component, accessed through the InformationProvider. The application splits the tournaments into two categories, "Tournaments created" and "All the tournaments", using the userID stored in the session and comparing it to the admins in a tournament, and redirects the user to his personal page.

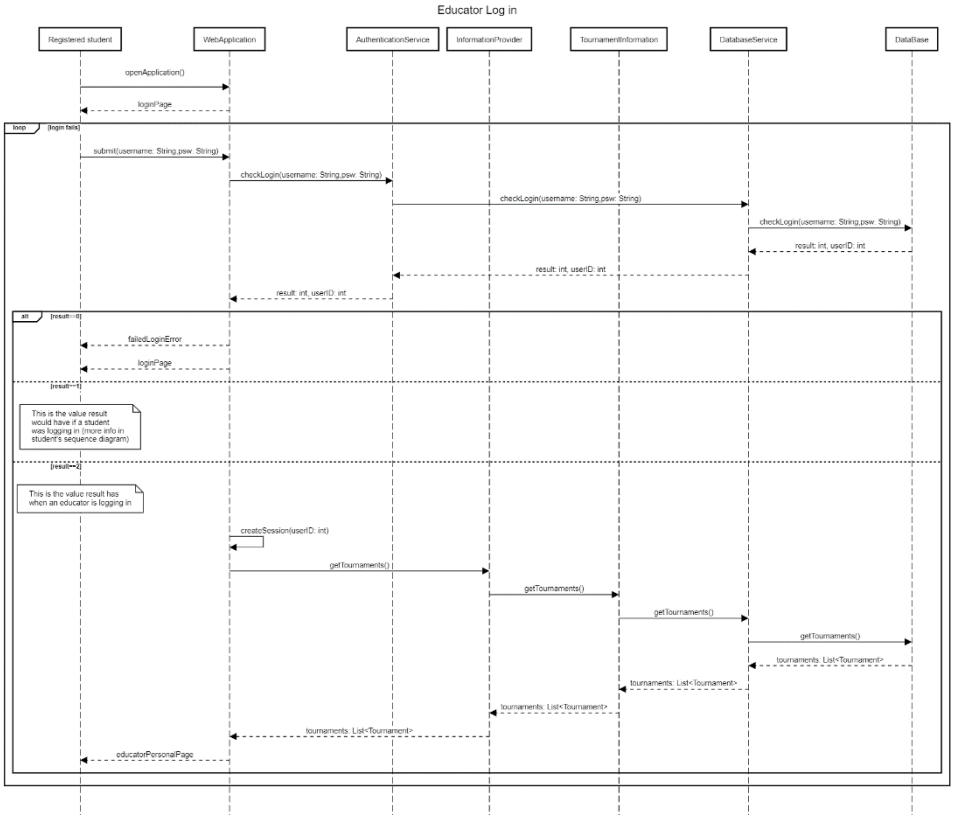

Figure 20: Educator log-in.

2.4.2.3 Log out

An educator can log out of his or her profile from any page on the site since the button with the corresponding function is always located in the side banner that is present on every page. Once the

educator requests to log out, the application simply invalidates the session and performs a redirect to the log in page, thus forcing the educator to log in again.

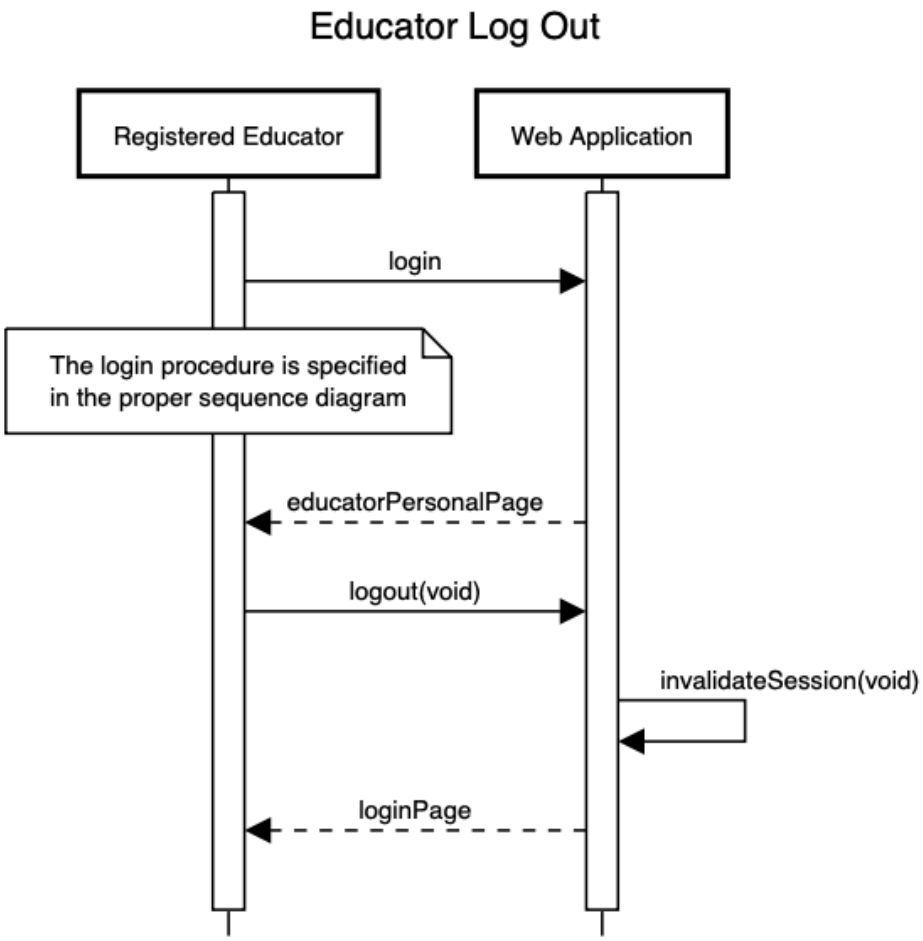

Figure 21: Educator log-out.

2.4.2.4 Show tournament's information

An educator able to perform the login operation is redirected to his personal page. He has access to all the tournaments, split in two categories: "Tournaments created" and "All the tournaments". When he clicks on "More information on this tournament!", next to a tournament, a request containing the tournamentID is sent to the application. The request is so redirected through the InformationProvider for the TournamentInformation provider, who then works with the DatabaseService to get all the data of the tournament from the database. The Database replies with a tournament object, that is NULL if no tournament was found with that ID. The object is provided back to the WebApplication that loads the tournament page, using the data in the tournament object, or shows an error in case of a NULL object.

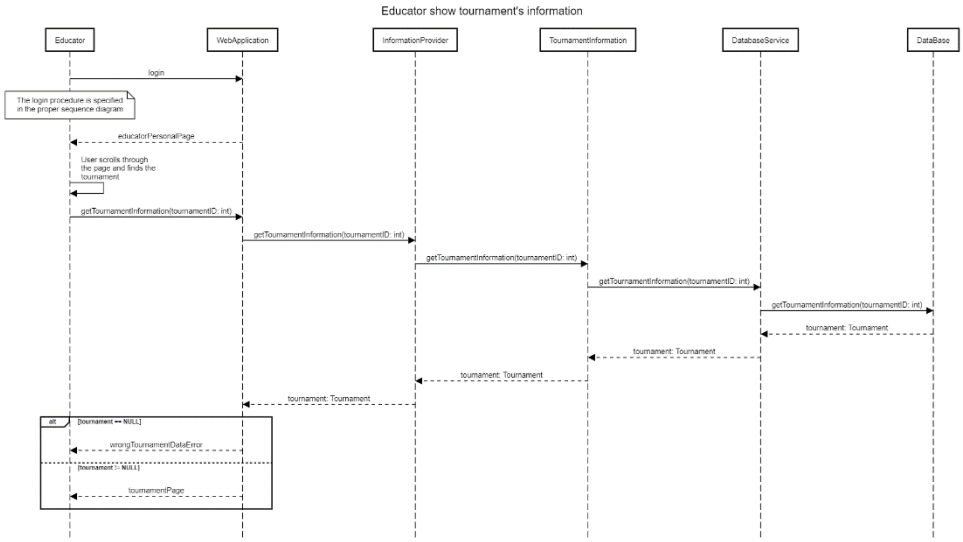

Figure 22: Educator shows tournament's information.

2.4.2.5 Create a new tournament

An educator, after completing the login procedure (specified in the previous "login" sequence diagram, is redirected to his personal page. By clicking on "Create tournament", a request is sent to the server, that redirects him to the tournament creation page. The educator completes the form and submits it. The data is managed by the WebApplication, where also the userID of the educator is added to the request, to identify from who it was generated. The request goes through the EducatorManager component to the TournamentManager, where the request for the tournament creation is sent to the database, through the DatabaseService. The Database replies with a boolean result, true if the tournament was correctly created, false otherwise, and an integer tournamentID that is a unique identifier for the tournament if the result is true, minus one instead. The data reaches the WebApplication that processes it by creating an error page if result equals to false, or by showing the tournament creation successful page instead.

The application in case the tournament was successfully created needs to send a notification to all students. The EducatorManager sends a request to the NotificationService component for handling the notification process. This component contacts the DatabaseService for obtaining a list of all the students from the database and then goes through the list and notifies each student: the first part consists of adding the notification to the user's notification list in the database. In case the boolean database response is true, then the action was completed with no problem and the component instructs the MailService to notify the user of the new tournament via email; in case the boolean is false the system retries three times to apply the modification to the database; upon failing it goes on to the next user, meaning the database may have processing issues.

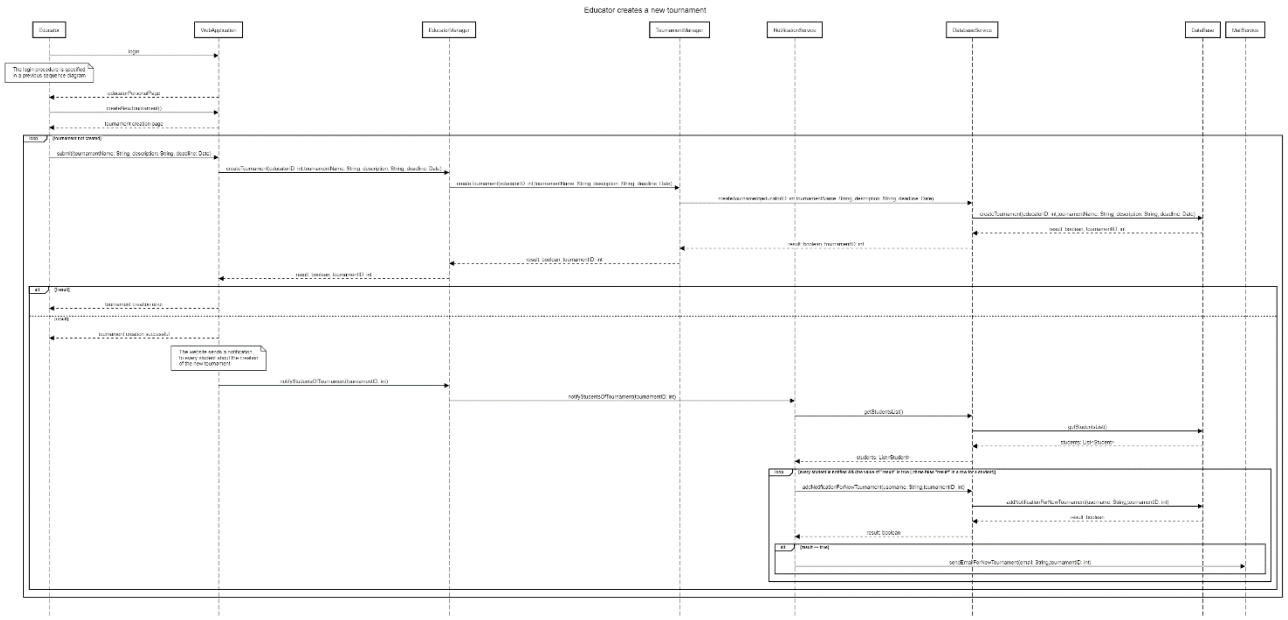

Figure 23: Educator creates a new tournament.

2.4.2.6 Create a new battle

The educator when on a tournament page can, using the appropriate button, create a new battle for that tournament. In doing so, the application passing through the EducatorManager and the TournamentManager accesses the database to check that the educator (identified by educatorID) is the creator or contributor of the chosen tournament (tournamentID); if not, a message for lack of permissions is shown to the educator, otherwise the page for creating the battle is presented. Here the educator enters all the necessary data and, using the appropriate button, submits the request for the creation of the battle; the application checks on the "code kata," i.e., the set of files that is uploaded such as description, test cases, source code, etc., and checks on the other data such as constraints for team creation and dates entered for registration and final submission. If no error is present, the application through the NotificationService and MailService sends a notification to all students registered in the tournament (tournamentID) about the creation of the new battle, and finally notifies the educator of the correct creation. If the value of errCount, on the other hand takes a value other than 0, it goes to indicate that an error is present between "code kata" (errCount = 1), between constraints for teams (errCount = 2) or between dates (errCount = 3), and in all cases the user is notified.

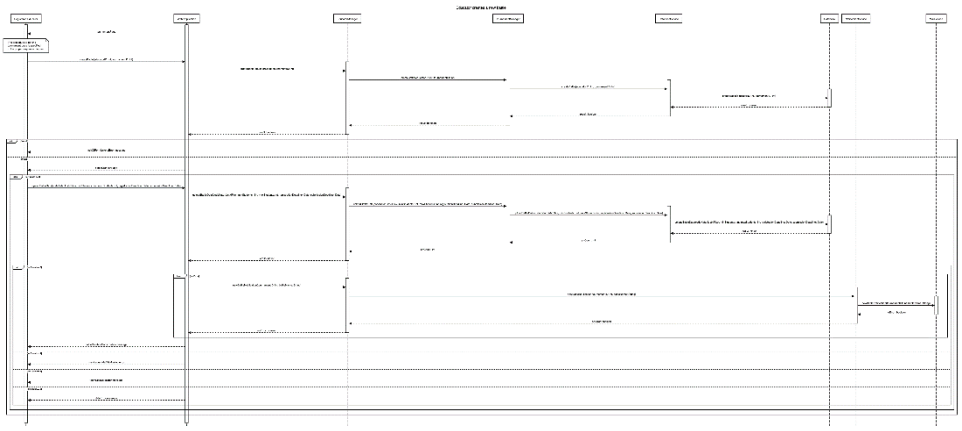

Figure 24: Educator creates a battle.

2.4.2.7 Show battle's information

This sequence diagram illustrates when an educator wants to see the battle's information using the Web Application. After transitioning from the login page to his personal page and then to the tournament page, the educator clicks the button for getting battle information.

At this point the information provider asks the battle information to query the database and show it all the battle information. If the information is loaded correctly the battle page is shown, otherwise the error is presented, and the educator returns to his main page.

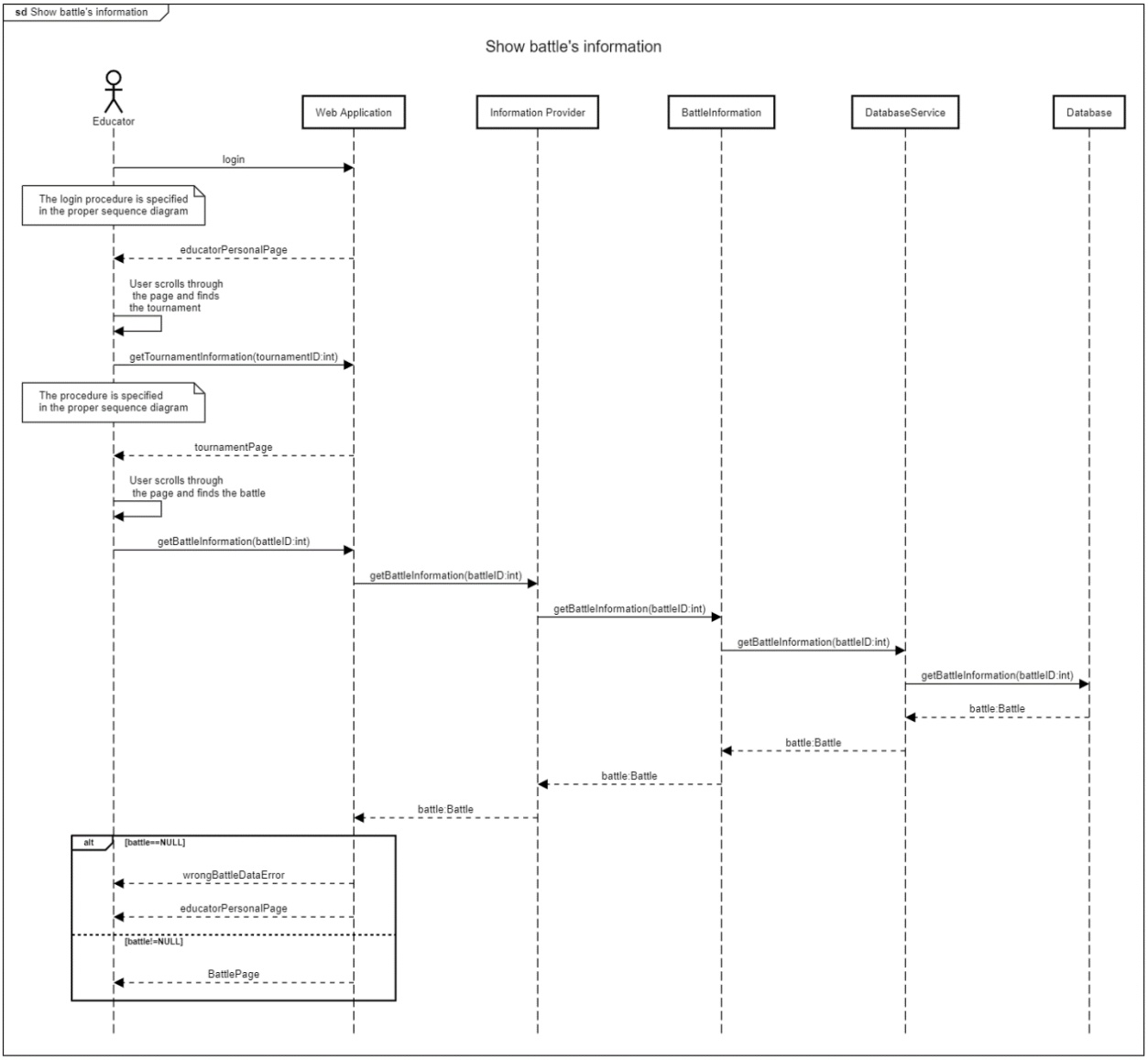

Figure 25: Educator shows battle's information.

2.4.2.8 Enter the manual evaluation

The educator, when on the page of a battle in consolidated stage can, using the appropriate button, enter the final manual evaluation to then be able to close the battle. In doing so, the application passing through the EducatorManager and BattleManager accesses the database to check that the educator (identified by educatorID) is the creator of the chosen battle (battleID); if not, a message for lack of permissions is shown to the educator, otherwise the page for entering the evaluation is presented. Here the educator can through the form enter the various evaluations; he/she can decide to save the current evaluations entered, by doing so all the form data are checked by the application, passing through the EducatorManager and BattleManager, arriving to be saved in the database. The application checks for non-numeric data entered (in which case an exception is raised and the user is

notified) and checks how many assessments still need to be entered; if there are no more assessments to be entered the user is notified and can properly close the battle, while in the case of missing assessments, the user's page is dynamically refreshed showing the assessments he has just saved.

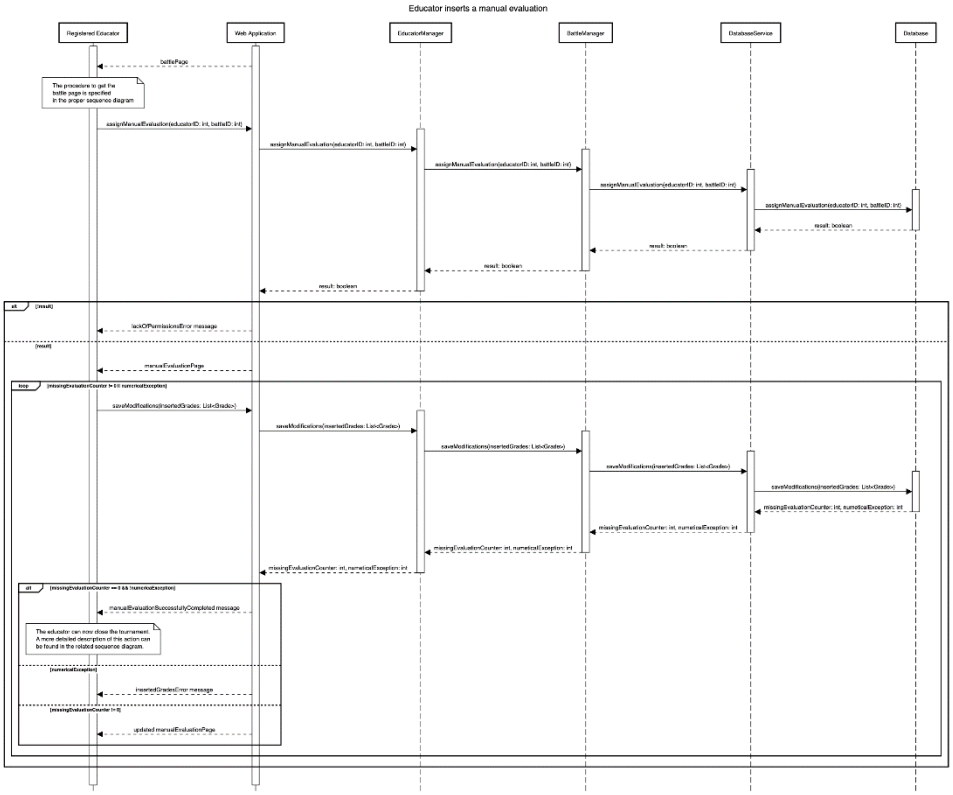

Figure 26: Educator inserts a manual evaluation.

2.4.2.9 Create a new badge

During the creation of a tournament an educator can choose to create new badges for the tournament; to do so through the appropriate button he or she accesses the new badge creation page. Once on the page the educator can decide to create new variables, or define the name, the expressions that define how the badge is assigned, and the related variables used. When the educator is satisfied, he or she can request the finalization of the creation using the appropriate button; the application performs checks on the data entered, saves it to the database via the EducatorManager and TournamentManager, and notifies the user. In case there are problems with the entered expressions, with the used variables or mandatory fields have not been filled in, the application raises exceptions, and the user is notified.

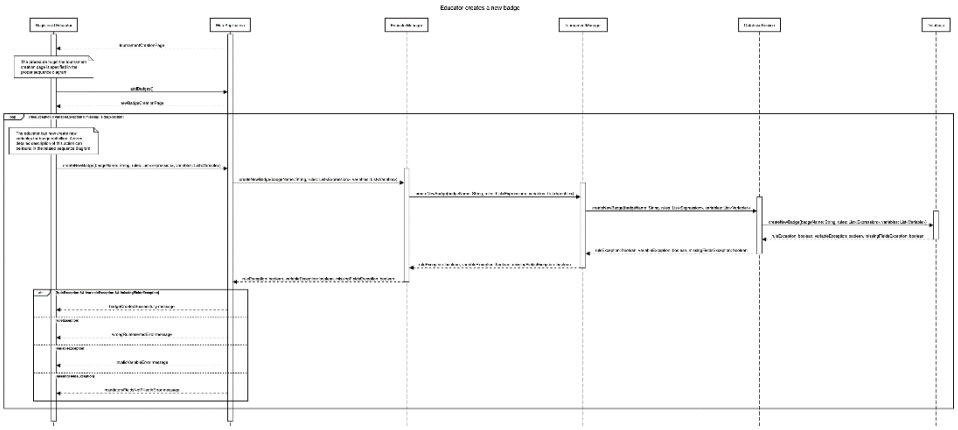

Figure 27: Educator creates a new badge.

2.4.2.10 Create a new variable

During the creation of a badge an educator may choose to define new variables; to do so through the appropriate button he/she accesses the new variables definition page. Once in the page the educator enters the name of the variable and the expression that defines it in the form and thanks to the appropriate button, requests the finalization and saving; the application performs checks on the entered data, saves them in the database passing through the EducatorManager and the TournamentManager and notifies the user. In case there are problems with the entered expression or mandatory fields have not been filled in, the application raises exceptions, and the user is notified.

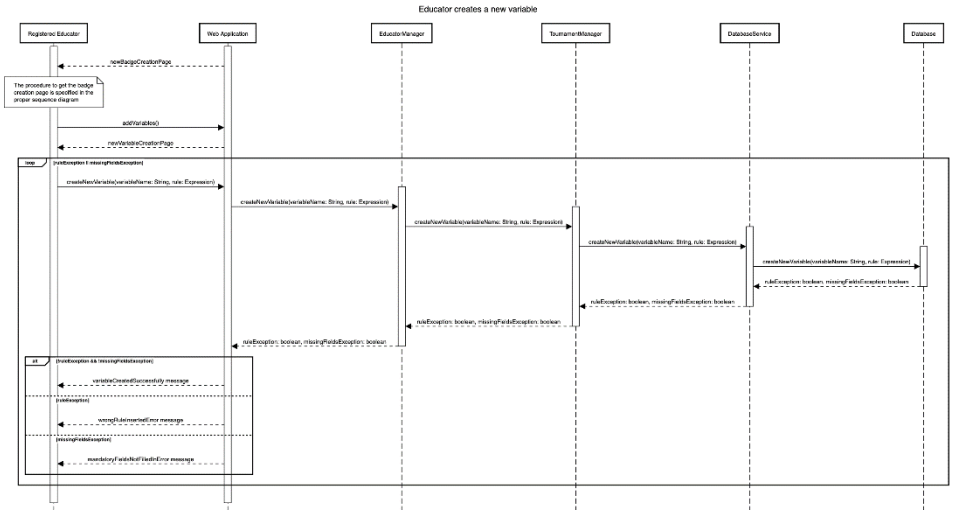

Figure 28: Educator creates a new variable.

2.4.2.11 Invite another educator as collaborator

The educator is redirected to a tournament page (for the exact procedure it is possible to look at the **“**2.4.2.4 Show tournament’s information” sequence diagram), one where he is an administrator. He clicks on “Add collaborator”, sending a request to the server, containing the unique code relative to

the tournament, tournamentID. The WebApplication, before sending the request to the EducatorManager, adds the userID of the educator. The component calls a method from the TournamentManager that contacts the database through the DatabaseService, that firstly checks for the actual presence of the educator in the admin's list of the of the tournament and then proceeds with the request. The database replies with a boolean result value and with the tournamentID of the tournament. The WebApplication processes the values and, in case of a false value for the result variable, shows the user an error page, or, otherwise, redirects the educator to the new collaborator page. Whenever the user inserts or deletes a character from the search bar an asynchronous request is sent through the InformationProvider. The component forwards the request to the UserInformation component that is responsible to contact the database through the DatabaseService. The database replies with a list of educator objects, empty if no educator was found. The WebApplication is then responsible to convert the list into JSON format and transmit the information to the client, that shows the five more accurate results to the user. When the educator finds the user, he was looking for, he clicks on "Send invitation" next to the educator's name, thus sending a request to the server containing the unique username of the invited user and the tournamentID. The application processes the request by adding the userID of the educator that is sending the invitation and forwards it to the EducatorManager, responsible to pass the request to the TournamentManager. This component interacts with the database that replies with a boolean result value, true if the educator was found correctly and the user is not yet a collaborator, an educator object of the found user, NULL if the result value is false and the tournamentID of the tournament. The EducatorManager component processes this data, prompting the WebApplication to show an error page if the result value is false or contacting the NotificationService otherwise. This component is responsible for adding the notification to the user's notification entity in the database and, for whatever reason, if the database is unable to process the request, it retries three times before prompting the WebApplication to report the error to the user. The database replies with a boolean value result, false when there have been problems with the database, true otherwise. If the value returned from the database is true, then the notificationService instructs the MailService to also notify the user via mail before returning a result value to the WebApplication that then redirects the user to an error page if the value of result is false, or showing the user that the educator invitation was successful otherwise.

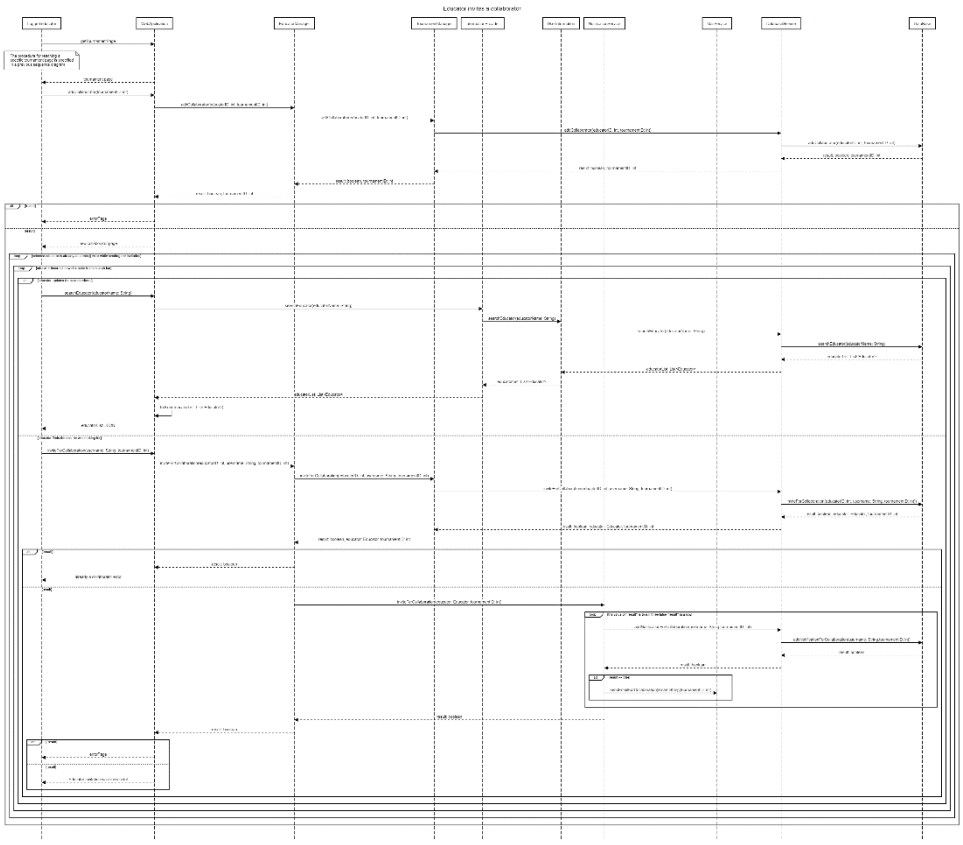

Figure 29: Educator invites a collaborator.

2.4.2.12 Close a tournament

This sequence diagram illustrates when an educator wants to close a tournament using the Web Application. After transitioning from his personal page to the tournament page he clicks on “Close tournament Battle”, sending a request to the server, containing the unique code relative to the tournament, tournamentID. The WebApplication, before sending the request to the EducatorManager, adds the userID of the educator. The component calls a method from the TournamentManager that contacts the database through the DatabaseService, that firstly checks if the educator is the creator and then proceeds with the request. The database replies with a boolean result value.

The WebApplication processes the values and, in case of a false value for the result variable, shows the user an error message and then his personal page, otherwise through a function call the system queries the database and obtains the list of enrolled students and their e-mail. This way the notification is added to the database and each student is notified of the closing of the tournament.

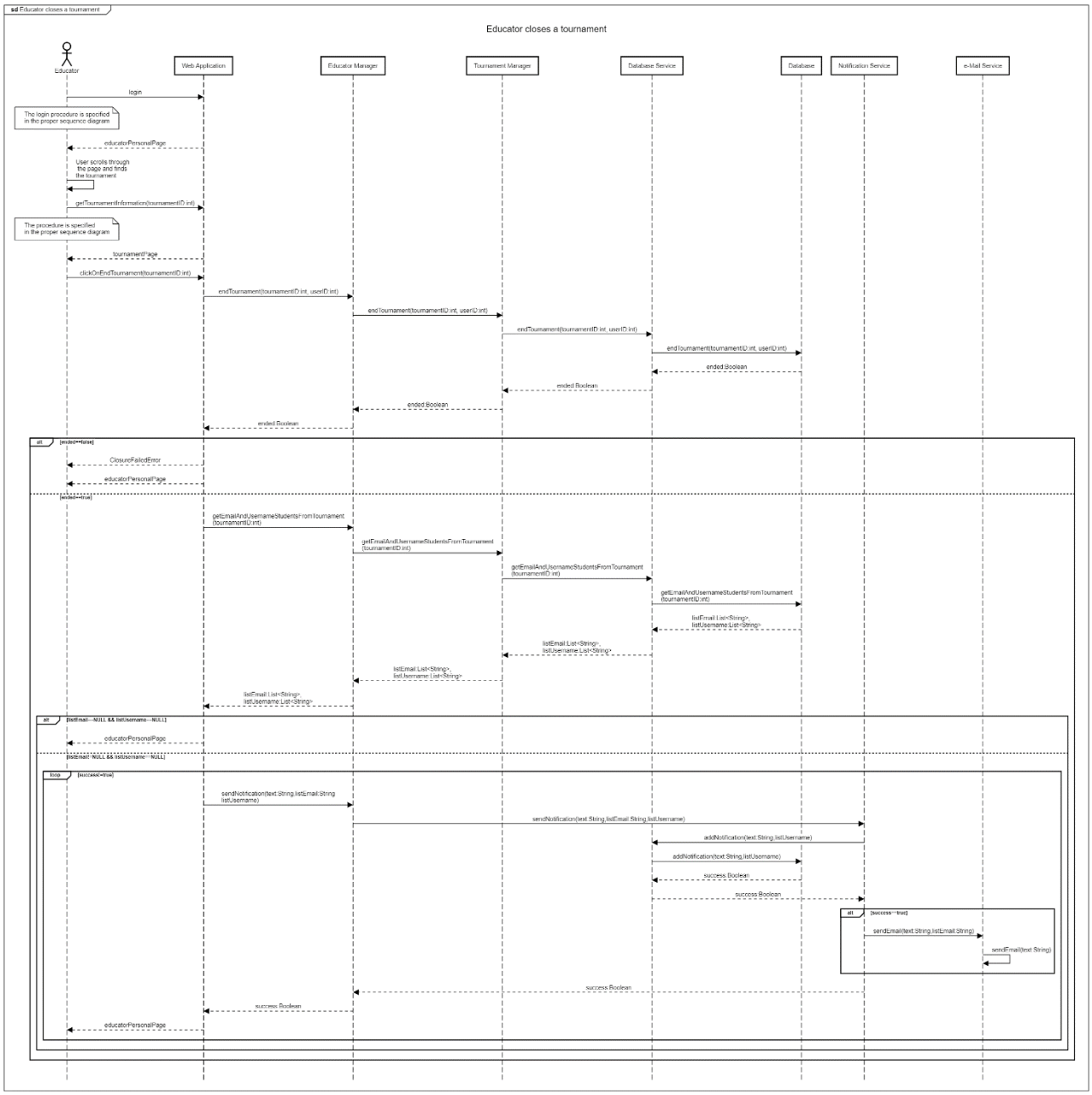

Figure 30: Educator closes a tournament.

2.4.2.13 Search a student's profile

An educator, after entering the search criteria in the appropriate bar always present in the side banner of the Web site, pressing enter from the keyboard makes the call to the function that through the InformationProvider and UserInformation accesses the database to do a search for all students who have a match with the criteria entered. All students who present a match are entered into a list, and if the list is empty, an error message is presented to the user, otherwise a page containing the list of all students in the above list is loaded. The user, by clicking on one of the profiles on the page, accesses a new page where all data related to the chosen student (identified by his ID) are presented.

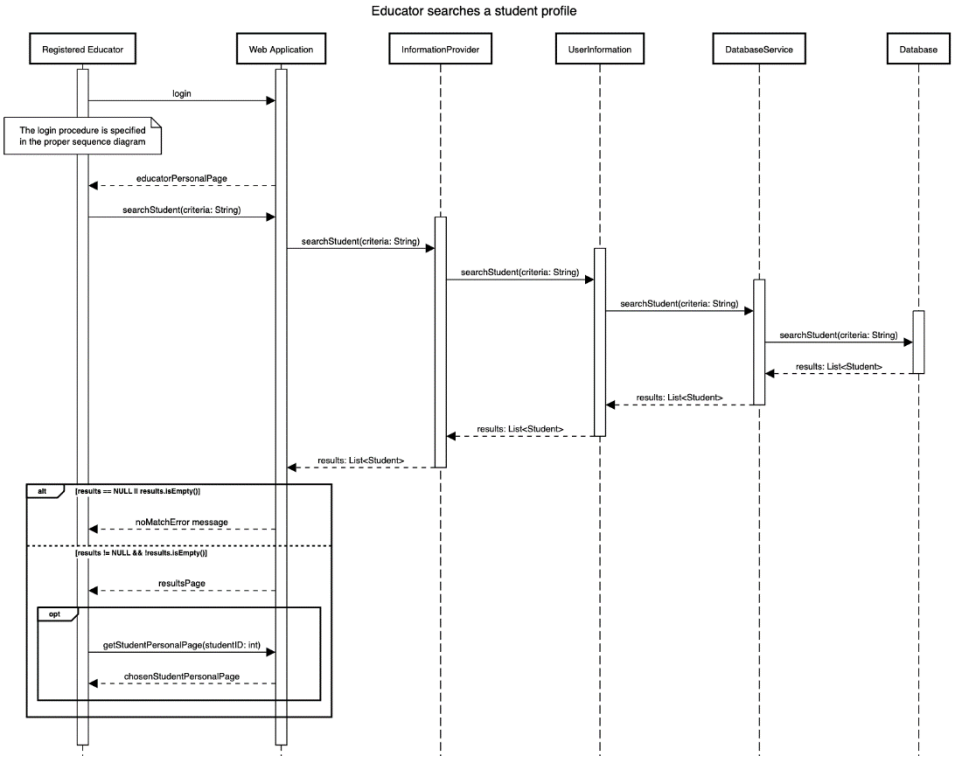

Figure 31: Educator searches a student profile.

2.4.2.14 View my notifications

This sequence diagram illustrates when an educator wants to see his notifications using the Web Application. After transitioning from the login page to his personal page, the educator clicks the "View my Notification" button.

At this point the Educator Manager asks the Notification Service to query the database and show it all the information about his notifications. If the information is loaded correctly the notifications page is shown, otherwise the error is presented, and the educator returns to his main page.

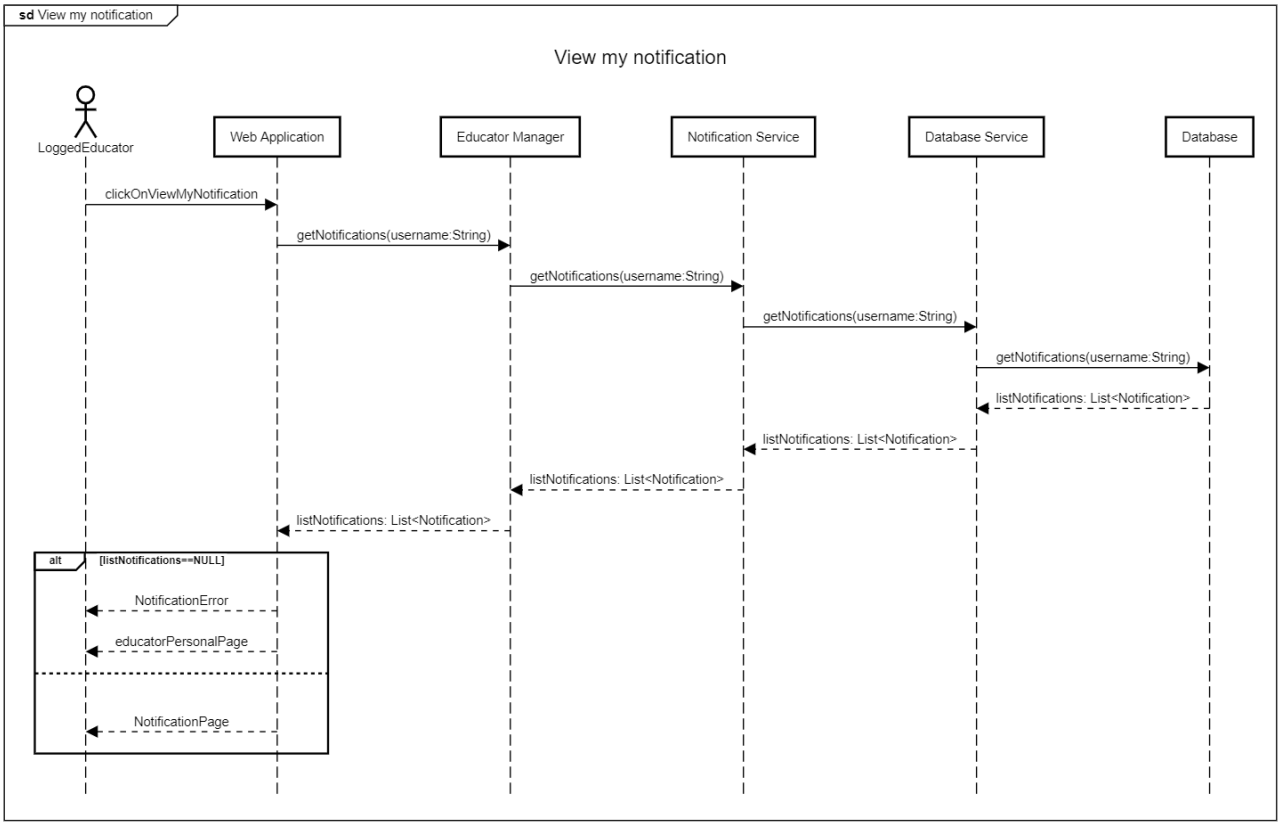

Figure 32: Educator views my notification.

2.4.2.15 Reply to a collaboration invitation

The educator is redirected to his notifications page, following the procedure illustrated in the previous sequence diagram. The educator can click either “Accept” or “Reject” when he sees he has received a notification about joining a tournament as an administrator. Both in the case the user accepts or rejects the invitation the application sends the response through the EducatorManager to the TournamentManager component, that then interacts with the database through the DatabaseService. If the user rejects the invitation, after the response is sent to the database, there is no response from the database and the user is notified of the completion of the operation (the only problem may be a database error and the worst case is that the user may have to re-click the “Reject” button a second time). If the user accepts, the database will reply with a boolean result value, true if everything went according to plan and the invitation was actually accepted, false otherwise, a String containing information about possible errors when result is false, NULL otherwise, the unique tournamentID of the tournament and a String containing the unique username of the educator that have sent the request. The data is processed by TournamentManager. When the result value is false, the component instructs the WebApplication to show an error page to the user, containing the information in the error String. When the result is true, the component instructs the database to set the educator as collaborator

through the DatabaseService component and return to the WebApplication the result of this operation. The WebApplication processes the data and shows an error page to the user if the result value is false or the updated notification page otherwise.

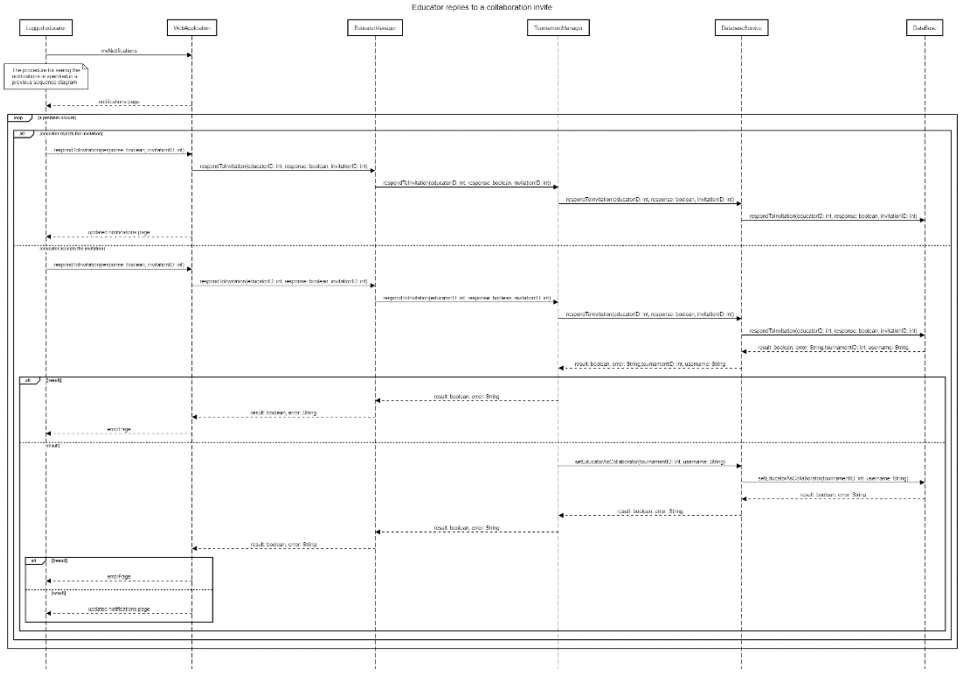

Figure 33: Educator replies to a collaboration invite.

2.5. Component Interfaces

This section provides all methods that each component interface offers to other components:

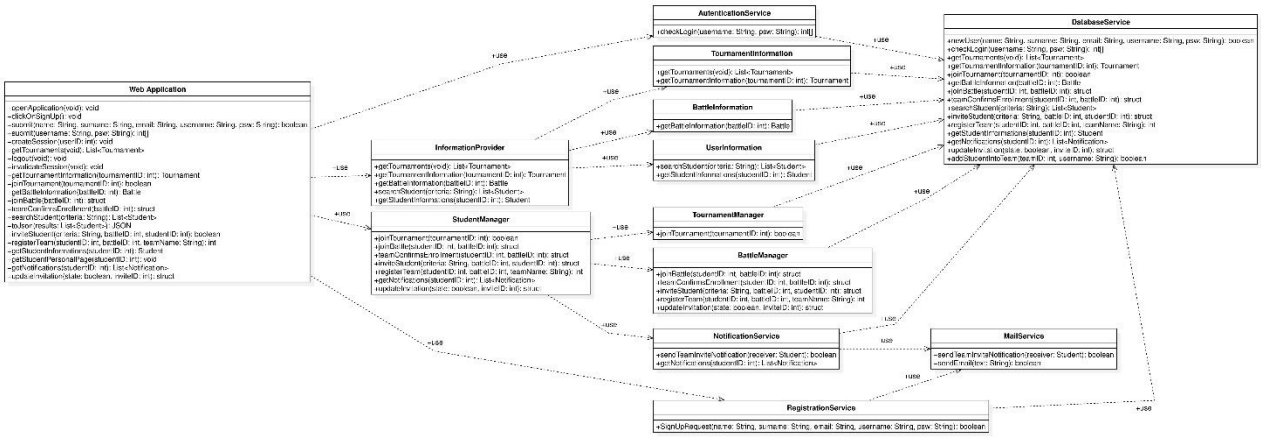

Figure 34: Student Component Interfaces.

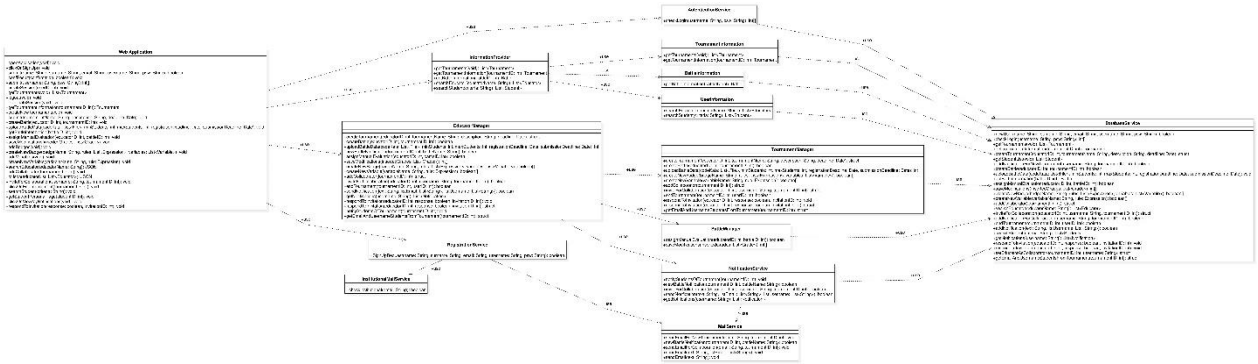

Figure 35: Educator Component Interfaces.

2.6. Selected architectural styles and patterns

- **Client-Server architecture:** it is an architectural style used in distributed applications. There are two components: clients that issue requests and servers that provides responses. Communication occurs over a network, typically using the HTTP protocol. This model facilitates scalability, maintenance, and resource management, enabling efficient deployment of applications and services over local networks or the Internet.
- **Three tier architecture:** as written in the previous sections we used an architecture with 3 tiers: presentation tier, middle tier and data tier that correspond to the 3 logical layers. It provides an organized and efficient approach to system design, enabling better resource management, simplified maintenance, and greater flexibility in meeting evolving system requirements. The development process is also made easier because each Tier can be developed simultaneously by a separate development team and can be updated or scaled as needed without impacting the others.
- **Model-View-Controller Pattern (MVC):** Architectural Design pattern that separates the application into three components:
    - **Model:** it contains the state and the application logic.
    - **View:** it contains the visualization logic of the model; it can have a reference to the model. It interacts with users.
    - **Controller:** it receives users' input from the view, then it modifies the model and the view.

  This pattern minimizes interdependencies, allows the system to increase maintainability. Each level can potentially be developed by different working groups.

2.7. Other design decisions

- **Clearbit API:** This API enables the application to verify an institutional mail.
- **GitHub API:** This API enables the application to communicate with repositories. It is used for the creation of a repository and for the communication between GitHub and CKB when a commit is made.
- **JSON:** All data being shared over communication channels will be in the JSON file format.
- **RESTful:** In our project, we are utilizing RESTful architecture along with the Spring framework. RESTful is an architectural style that uses a set of constraints to design web services. It promotes scalability, simplicity, and a stateless communication model between clients and servers.
- **Pylint, Bandit, Mypy:** We have decided to use the following tools in our software platform: Pylint, Bandit, and Mypy. These tools are essential for assessing the quality level of the source code, ensuring security, reliability, and maintainability.

3 | USER INTERFACE DESIGN

Some interfaces of the Web Application have already been presented in RASD document. Following figures display a graph, called flow graph, that represents the main screens of CKB, there is an image containing the main page of the student subsystem and another containing the most important page of the educator subsystem. In order to avoid redundancy, the login and sign-up pages leading to the educator's and student's dashboard respectively have been excluded from the diagram as they are common to both subsystems and are described in detail in the RASD document.

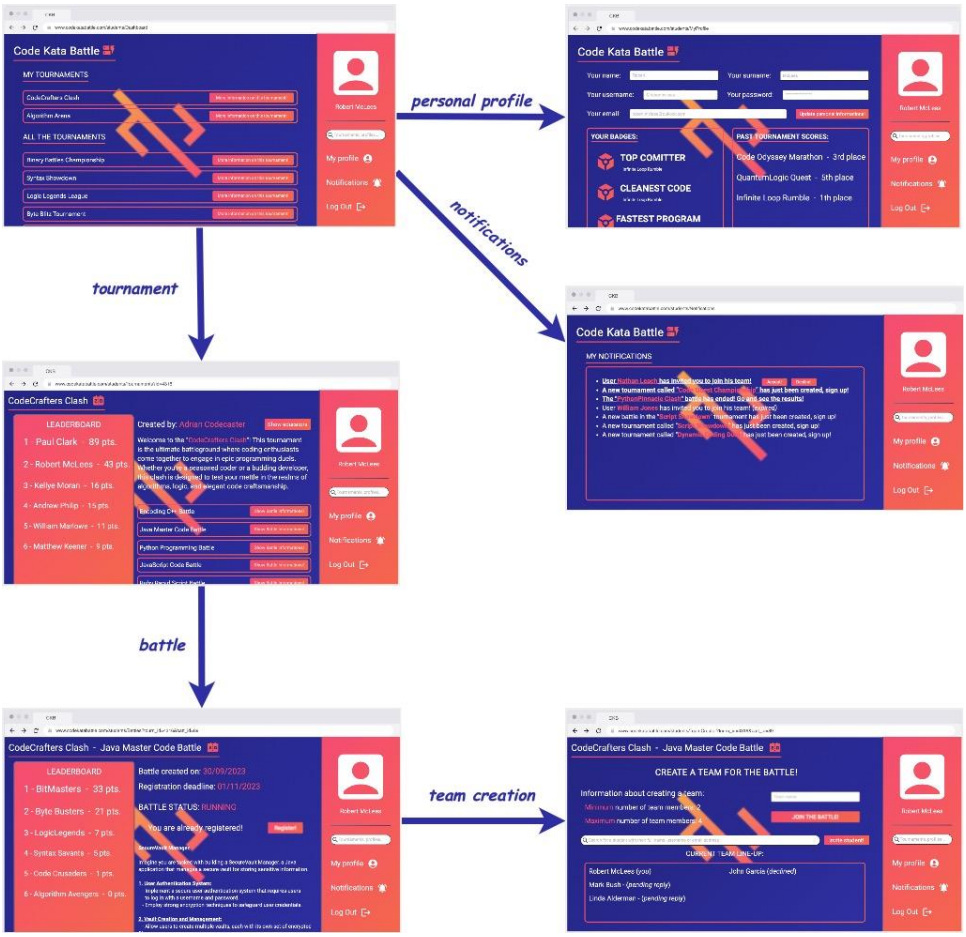

Figure 36: Student interfaces.

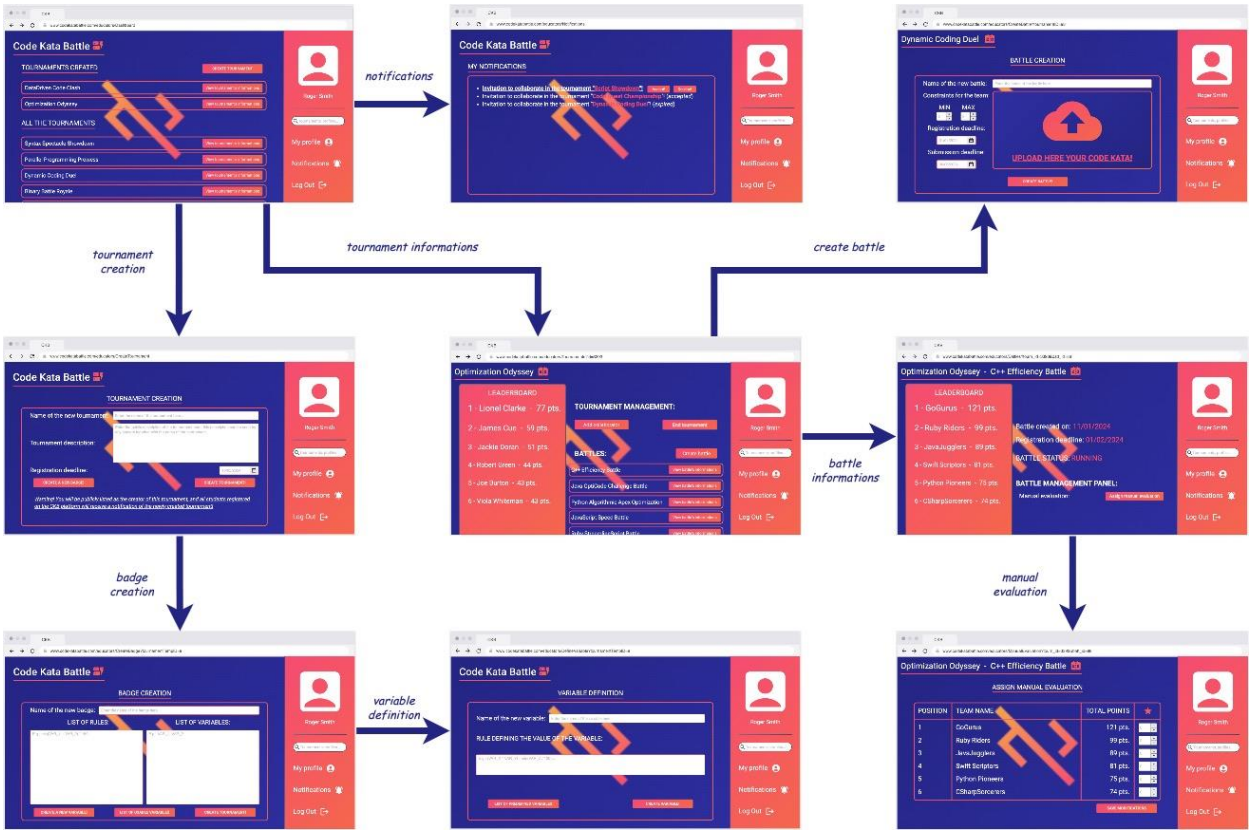

Figure 37: Educator interfaces.

4 | REQUIREMENTS TRACEABILITY

In this chapter, the mapping between the requirements and the corresponding components for fulfilling them is presented.

R1.1 The system shall allow an unregistered student to register.

- Web Application
- Registration Service
- Mail Service
- Database Service
- Database

R1.2 Allow Students to login using username/email and password

- Web Application
- Authentication Service
- Database Service
- Database

R1.3 Allow Students to log out

- Web Application

R1.4 The system allows each student to see the tournaments in which they are enrolled

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R1.5 The system allows each student to see the rankings of the tournaments in which the student is entered

- Web Application
- Database Service
- Database
- Information Provider

- Tournament Information

R1.6 The system allows each student to see if a battle is taking place within the tournament in which the student is enrolled

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information
- Battle Information

R1.7 The system allows each student to see the ranking of active battle

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information
- Battle Information

R1.8 The system allows each student to see all active tournaments

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R1.9 The system allows each student to see the rankings of active tournaments

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R1.10 The system allows each student to see if there are running battles in active tournament

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information
- Battle Information

R1.11 The system allows each student to see their badges

- Web Application
- Database Service
- Database
- Information Provider
- User Information

R1.12 The system allows each student to see the badges of the other students.

- Web Application
- Database Service
- Database
- Information Provider
- User Information

R1.13 The system allows each student to see all available badges in each tournament and the rules for winning them

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R1.14 The system allows each student to see information and deadlines of each battle

- Web Application
- Database Service
- Database
- Information Provider

- Tournament Information
- Battle Information

R1.15 The system sends the repository link to all teams registered for the battle

- Web Application
- Database Service
- Database
- Educator Manager
- Battle Manager
- Notification Manager
- Mail Service

R1.16 The system notifies when a new tournament is created

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager
- Notification Manager
- Mail Service

R1.17 The system notifies when a new battle is created

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager
- Battle Manager
- Notification Manager
- Mail Service

R1.18 The system notifies each student when they are invited to participate in a battle

- Web Application
- Database Service

- Database
- Educator Manager
- Tournament Manager
- Battle Manager
- Notification Manager
- Mail Service

R1.19 The system notifies each student when a deadline has passed

- Web Application
- Database Service
- Database
- Notification Manager
- Mail Service

R1.20 The system notifies each student when the ranking of the battle, in which the player has participated, is ready

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager
- Battle Manager
- Notification Manager
- Mail Service

R1.21 The system notifies each student when the ranking of the tournament, in which the player has participated, is ready

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager
- Notification Manager
- Mail Service

R1.22 The system allows students to register for the tournament

- Web Application
- Database Service
- Database
- Student Manager
- Tournament Manager

R1.23 The system allows students to create their own team in a battle

- Web Application
- Database Service
- Database
- Student Manager
- Tournament Manager
- Battle Manager

R1.24 The system allows students to invite other students onto their team

- Web Application
- Database Service
- Database
- Student Manager
- Tournament Manager
- Battle Manager
- Notification Manager
- Mail Service

R1.25 The system allows students to register for a battle

- Web Application
- Database Service
- Database
- Student Manager
- Tournament Manager
- Battle Manager

R2.1 The system shall allow an unregistered educator to register.

- Web Application
- Registration Service
- Institutional Mail Service
- Mail Service
- Database Service
- Database
- ClearbitAPI

R2.2 Allow Educators to login using username/email and password

- Web Application
- Authentication Service
- Database Service
- Database

R2.3 Allow Educators to log out

- Web Application

R2.4 The system allows educators to see the tournaments they have created

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R2.5 The system allows educators to see the tournaments in which they were invited to participate as collaborators

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R2.6 The system allows educators to see the active tournaments

- Web Application

- Database Service
- Database
- Information Provider
- Tournament Information

R2.7 The system allows educators to see the updated ranking of the tournaments that he has created

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R2.8 The system allows educators to see the updated ranking of tournaments in which they were invited to participate as collaborators

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R2.9 The system allows educators to see the updated ranking of the active tournaments that he has created

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R2.10 The system allows educators to see the running or finished battles of their tournament

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

- Battle Information

R2.11 The system allows educators to see the running or finished battles of the tournaments in which they were invited to participate as collaborators

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information
- Battle Information

R2.12 The system allows educators to see the running or finished battles of all active tournaments

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information
- Battle Information

R2.13 The system allows each educator to see all available badges in each tournament and the rules for winning them

- Web Application
- Database Service
- Database
- Information Provider
- Tournament Information

R2.14 The system allows each educator to see the badges of the students.

- Web Application
- Database Service
- Database
- Information Provider
- User Information

R2.15 The system notifies each educator when they are invited to participate in a tournament

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager
- Notification Manager
- Mail Service

R2.16 The system notifies each educator when a battle is created in a tournament that he has created or in a tournament in which he is invited to participate as collaborator

- Web Application
- Database Service
- Database
- Educator Manager
- Battle Manager
- Notification Manager
- Mail Service

R2.17 The system notifies each educator when a manual evaluation is required in a battle that he has created

- Web Application
- Database Service
- Database
- Educator Manager
- Battle Manager
- Notification Manager
- Mail Service

R2.18 The system allows each educator to create a new tournament

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager

- Notification Service
- Mail Service

R2.19 The system allows each educator to create a new battle (each educator can insert codekata, set deadlines, set maximum and minimum number of students per group, set additional configurations for scoring).

- Web Application
- Database Service
- Database
- Educator Manager
- Battle Manager
- Notification Service
- Mail Service

R2.20 The system allows each educator to send an invitation to another educator for becoming a collaborator

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager
- Information Provider
- User Information
- Notification Service
- Mail Service

R2.21 The system allows each educator to make a manual evaluation, if it is required

- Web Application
- Database Service
- Database
- Educator Manager
- Battle Manager

R2.22 The system allows each educator to close a tournament

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager
- Notification Service
- Mail Service

R2.23 The system allows each educator to create badges

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager

R2.24 The system allows each educator to create new variables

- Web Application
- Database Service
- Database
- Educator Manager
- Tournament Manager

5 | IMPLEMENTATION, INTEGRATION AND TEST PLAN

This section will showcase the order in which the system's subcomponents will be implemented and how they will be integrated. Additionally, details about the test plan will be provided in the section 5.3.

5.1 Implementation Plan

When choosing an implementation plan the two main choices are: Bottom Up and Top Down. The choice for the implementation of the system to be is Bottom Up: this will allow to develop, integrate and test first single Components before later checking their interaction with other modules. This choice is also perfect for future addiction of functionalities, with less effort and better merging capabilities.

The tests resulting from this approach are simpler and easily interpretable, leading to a better testing environment. For the choice of which component to implement before others it is important to check dependencies between components, for checking which subsystems are more suitable for a precedence in development.

All the external components can be considered reliable services, so the only testing included in our test plan about them will be about their integration in the system.

The database and the mail service can be regarded as external units' component, always monitored by specialized personnel.

The list underneath shows the development order of all Components:

1. Database Service
2. Mail Service
3. Registration Service
4. Authentication Service
5. Information provider
    - Tournament Information
    - Battle Information
    - User Information
6. Educator Manager

- Tournament Manager
- Battle Manager
- Notification Service
- Profile Manager

7. Student Manager

- Tournament Manager
- Battle Manager
- Notification Service
- Profile Manager

8. Link of the external components
9. Web Application

5.2 Integration and Test Plan

This section will showcase the system's integration plan and the order in which the interaction between Components will be tested. Each Component will have been tested right after development so the only focus is on communications between different Components. Since the implementation will proceed following a Bottom-Up approach the testing will reflect that process, creating the need for Drivers: these are place holders needed to act as top-level Components not yet implemented, they will be able to invoke low level Components to complete all tests needed. For example, to test the Database Service a Driver will be required to provide input and invoke the Component's methods since it will be the first implemented.

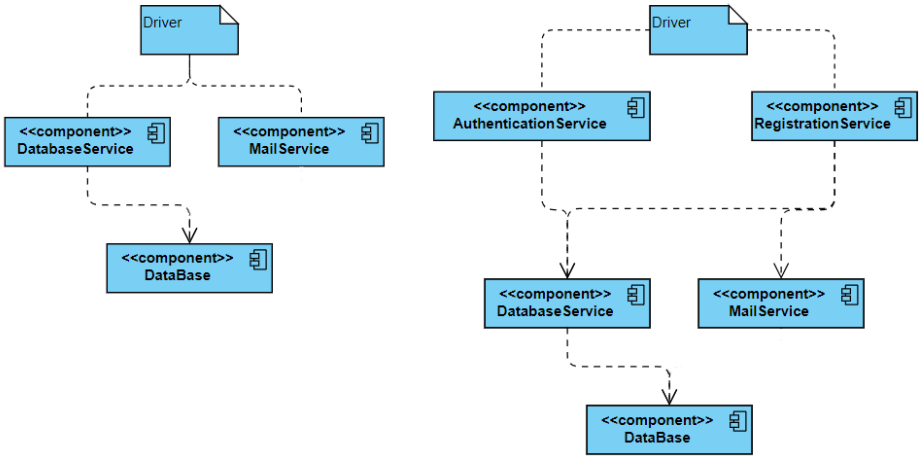

Figure 38: Integration plan.

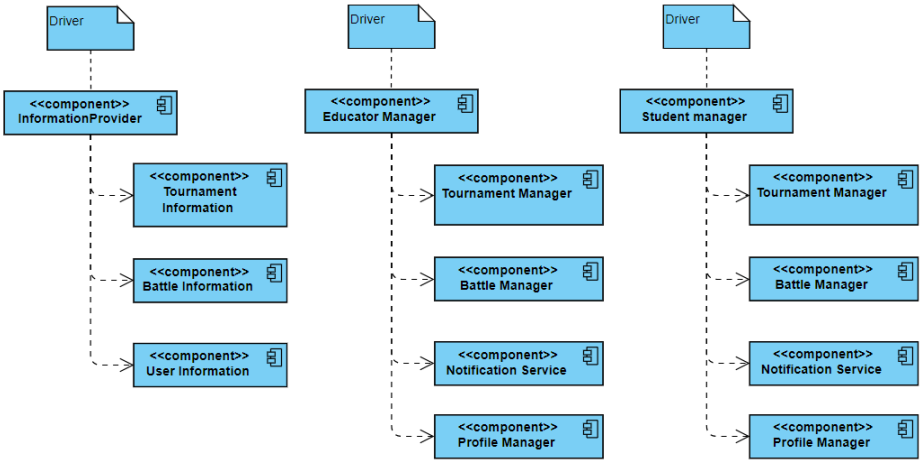

Figure 39: Integration plan.

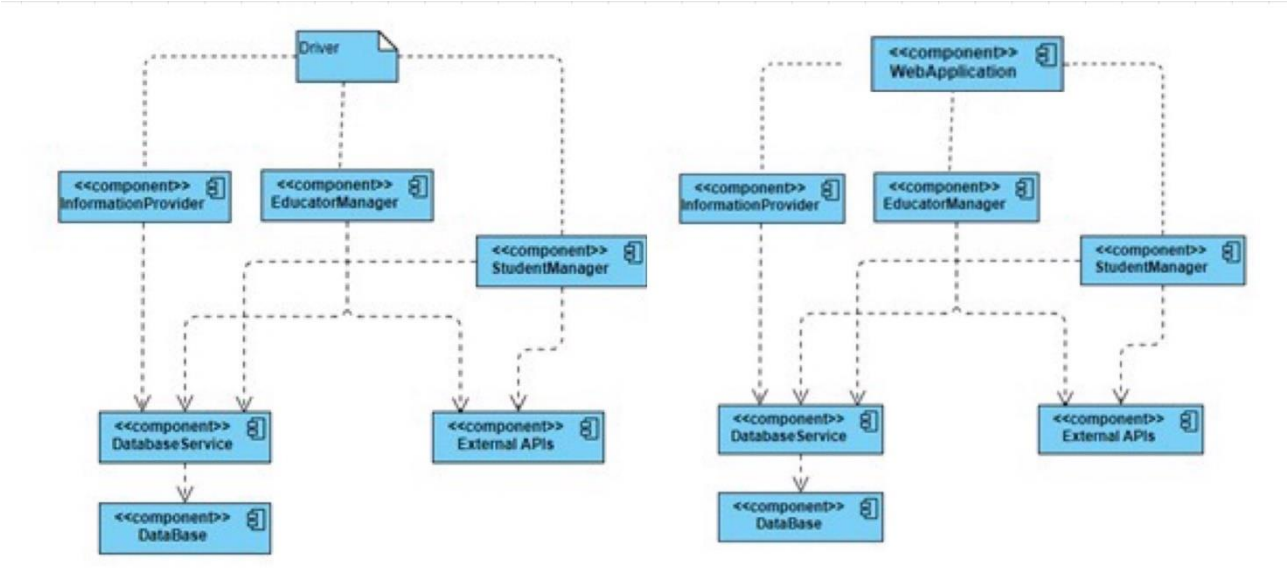

Figure 40: Integration plan.

In addition, the following system tests will be performed:

- **Load Testing**: By progressively increasing the load on the system, some critical points might be exposed such as scalability issues or memory leaks. This is also useful to identify the upper limit of components.
- **Performance Testing**: This part of the testing plan will identify any eventual bottlenecks, providing a base performance that can be used to evaluate the system.
- **Functional Testing**: All previously defined requirements and product functionalities must be fully satisfied by the system.

- **Usability Testing**: This non-functional testing aims at evaluating the User-friendliness of the system, for example the Presentation Layer should offer interfaces easily understandable and of ease of use.
- **Stress Testing**: Some of the system's resources will be put under excessive stress or removed to evaluate the system's behaviour during failures and how the system replies to it.

6 | EFFORT SPENT

6.1 Cotrone Mariarosaria

Task	Hours
Introduction	2
Component Diagram	2.30
Overview, Class diagram, Component View, Deployment View, Selected architectural styles and patterns, other design decisions	8.15
Sequence Diagram	11
Requirements traceability	2.30
Document revision	4

6.2 De Ciechi Samuele

Task	Hours
Component Diagram	2.30
Document revision	5
Sequence diagrams	12
Implementation, integration and test plan	6

6.3 Deidier Simone

Task	Hours
Component Diagram	2.30
Sequence Diagrams	7
Component Interfaces	7.30
Document revision	3.30
Mock-ups	6.45
User Interface Design	1.30