A.Y. 2023/2024

POLITECNICO
MILANO 1863

DD: Design Document

Davide Grazzani Alessandro Masini

Professor
Matteo Giovanni Rossi

Version 1.0
January 7, 2024

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive technical overview of the system outlined in the RASD document. In this context, we will evaluate the hardware and software architectures, emphasizing the interactions among the system's constituent components. Furthermore, we will explore the details of the implementation, testing, and integration procedures. While the document primarily employs technical language tailored for programmers, it extends an invitation to stakeholders for perusal, as it can offer valuable insights into the developmental aspects.

1.2 Scope

This design document outlines the system's behavior, encompassing both general and critical scenarios. It delineates the system architecture by examining the logical assignment of components and their interactions.

1.3 Definitions, Acronyms, Abbreviations

Acronyms

- **RASD**: Requirements Analysis and Specification Document
- **DD**: Design Document
- **API**: Application Programming Interface
- **DBMS**: Database Management System
- **UML**: Unified Modeling Language
- **UI**: User Interface
- **GUI**: Graphical User Interface
- **HTTPS**: Hypertext Transfer Protocol Secure
- **HTML**: Hypertext Markup Language
- **CSS**: Cascading Style Sheets

- **JS**: JavaScript
- **MVC**: Model-View-Controller

1.4 Revision History

- Version 0.1: Setup
  - Created first layout
- Version 1.0: First release
  - Added section 1
  - Added section 2
  - Added section 3
  - Added section 4
  - Added section 5

1.5 Reference Documents

- Specification document: "Assignment RDD AY 2023-2024"
- Requirements Analysis and Specification Document: "RASD"
- UML documentation: `https://www.uml-diagrams.org/`

1.6 Document Structure

- **Section 1** offers a concise overview of the design document, outlining its purpose and scope, encompassing all the definitions, acronyms, and abbreviations employed.
- **Section 2** extensively explores the system architecture, offering an intricate depiction of the components, interfaces, and all the technical decisions undertaken for application development. It encompasses detailed sequence, component, and ArchiMate[1] diagrams providing an in-depth understanding of the system.

[1]ArchiMate is an open and independent enterprise architecture modeling language to support the description, analysis and visualization of architecture within and across business domains in an unambiguous way[1].

- **Section 3** provides a comprehensive portrayal of the UI, including all client-side mockups alongside with graphs that explains the proper execution flow.
- **Section 4** maps the goals and requirements outlined in the RASD with the functionalities presented here[2].
- **Section 5** presents the implementation, testing, and integration phases of the system components, which will be undertaken during the technical development of the application.

[2]Here refers to the totality of this design document.

2 Architectural Design

2.1 Overview

The system's architecture is organized in *three logical layers* following the *client-server* paradigm.

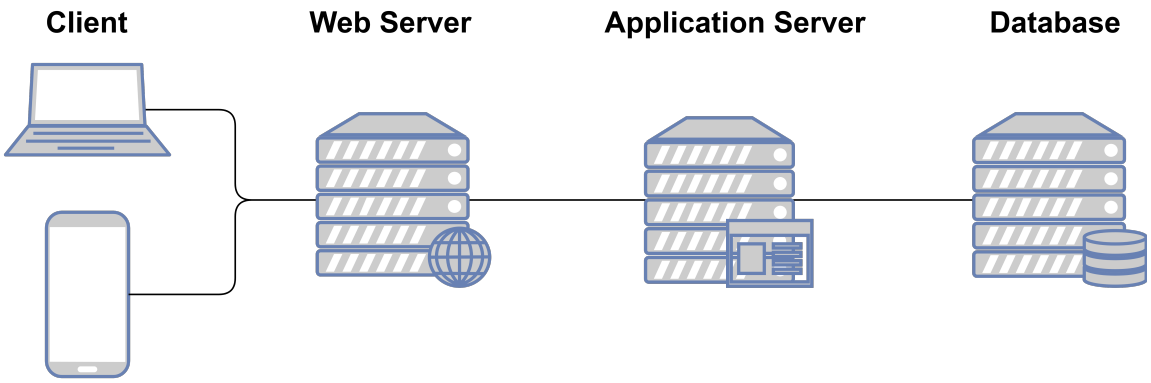


Figure 1: Three Tier Architecture (and the clients)

- **Presentation Layer (P)**: This layer is tasked with presenting the application's user interface to the client via the Web Server. It is represented by a GUI designed to facilitate efficient and comfortable user interaction with the application.
- **Application Layer (A)**: The Application Layer handles the business logic of the application. It receives requests from the Presentation Layer, processes them, and then sends back the results to be displayed(to the presentation layer again).
- **Data Layer (D)**: Responsible for accessing and managing data sources, the Data Layer provides data through some database and directs it to the other layers. Additionally, it plays a crucial role in ensuring a high level of abstraction from the database itself by providing a proper interface

2.1.1 High level architecture

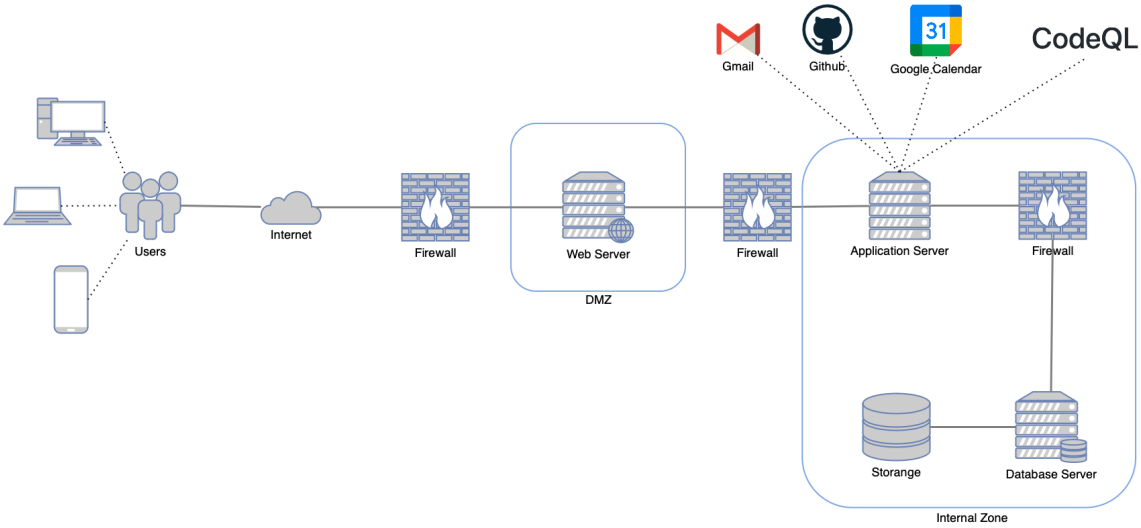


Figure 2: High Level Architecture

The figure 2 shows the high level architecture of the system. It's divided in two different zones:

- *Demilitarized Zone*: represent the area of the system where the system is partially exposed to the internet, hence the users. In this zone the *Web Server* is located. Its primary role is to handle the communication and interaction between the whole system and the final users.

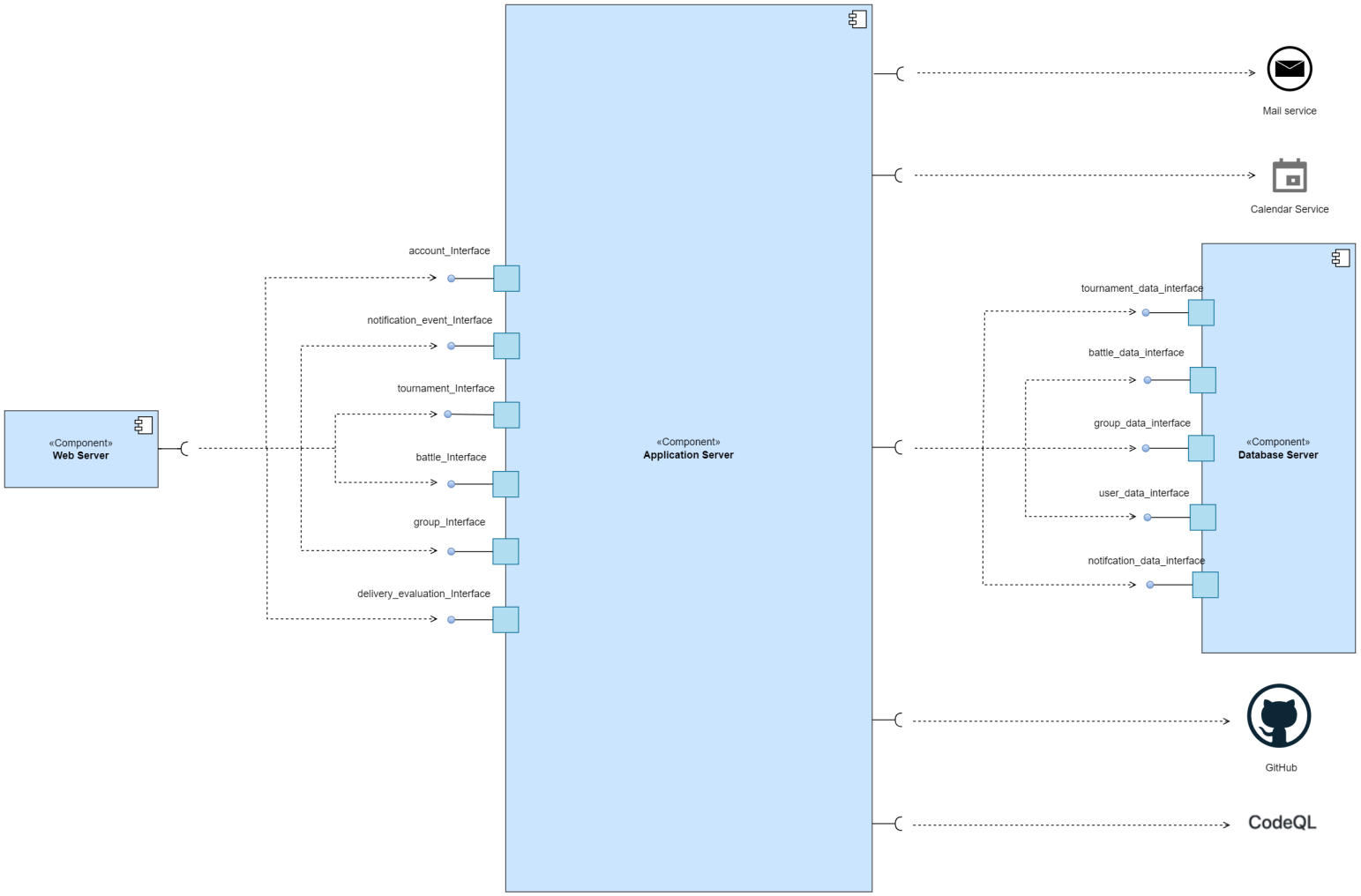
- *Internal Zone*: represent the internal area of the system where the business logic and the data are located. Here the *Application Server* and the *Database Server*[3] are present.

This division enhances the security of the system by placing the database, one of the most common target of cyberattacks, behind three different firewalls. Moreover, only the application server itself is allowed to directly query the database, further reducing the risk of unauthorized access.
It's necessary to clarify that the application server, even if not directly exposed to the internet, has to be connected to it in order to allow communication with external services such as the GitHub APIs.

[3]The database server is furthermore protected by another firewall

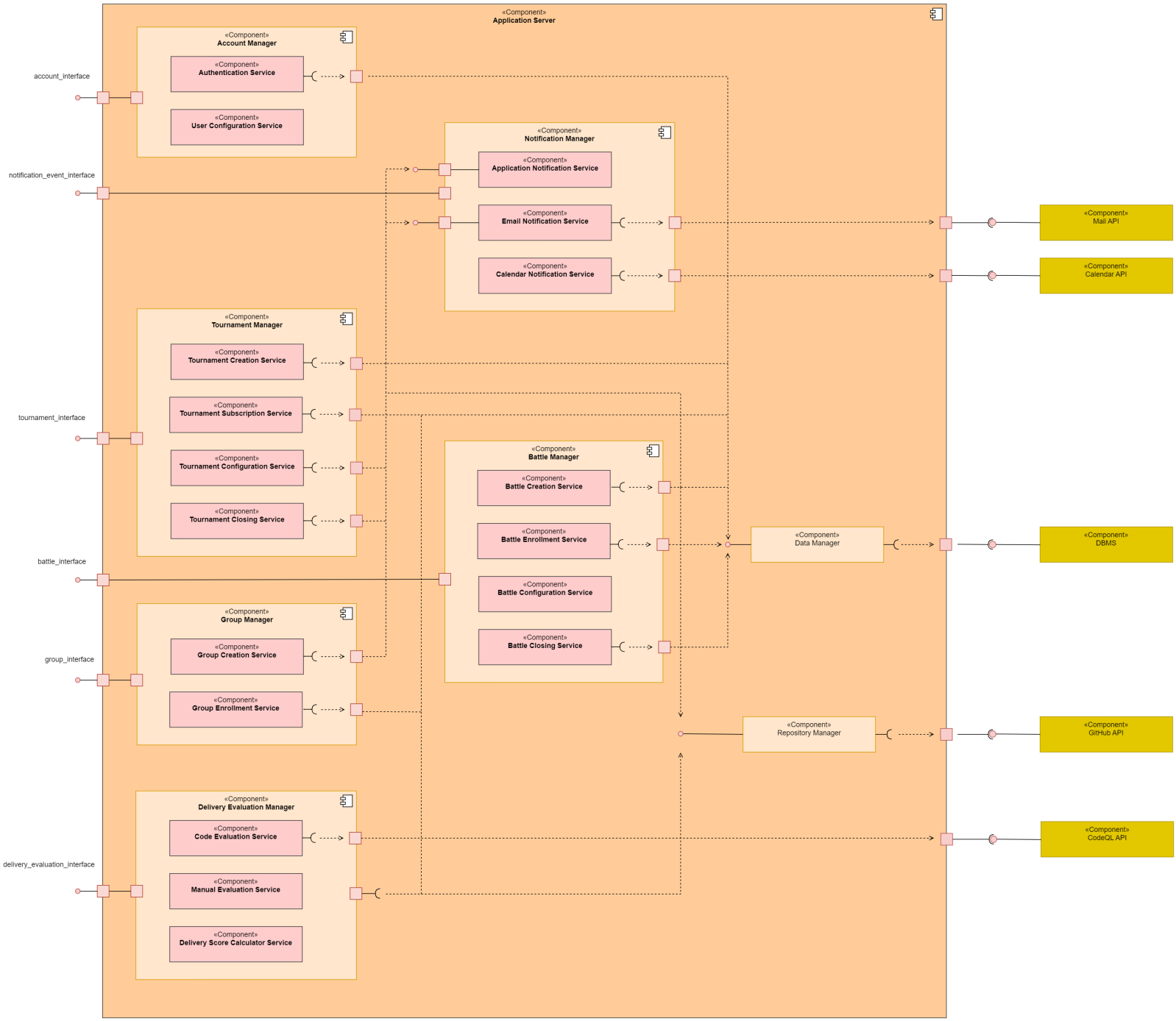
2.2 Component View

High level component view



This image gives a high level representation of the components of the system, where the application server is represented as an empty box, since its complete description will be provided in the next section. The interfaces represented on the left are the ones between the web server and the system's application server, and they represent the main functionalities requested by the clients to the web server. On the opposite side the external interfaces used by the application server are presented instead, which are for example the online repository system used by CodeKata (GitHub), but also the DBMS interfaces, which are managed by the system's DBMS service and handle all the system requests for data from its external database

Application server's component view



The previous component diagram provides a detailed view of the Application Server, showing its internal structure and the interactions between its components. Since such diagram is focused on the internal structure of the server, external elements are represented in a simplified way.
Description of each component:

- **Account Manager:** manages all the account related functionalities, like the authentication process and the setting of user's information.
  - **Authentication Service:** handles all the functionalities related to the authentication of the user (log in, log out, sign up), after the user has authenticated through this service, he/she can use the application
  - **User Configuration Service:** it handles the operations related to getting and setting user-configurable properties like its nickname and phone number
- **Tournament Manager:** it manages all the functions related to the set-up, management and closure of a tournament
  - **Tournament Creation Service:** it handles the creation of a new tournament by Educator, also checking in the database if another tournament with the same name does already exist
  - **Tournament Subscription Service:** handles the operations required when a user requests to subscribe to a certain tournament, like adding his/her name to the list of subscribed users in the database
  - **Tournament Configuration Service:** allows the modification of some Tournament properties, like its name, description or main picture
  - **Tournament Closing Service:** allows to perform the operations required whenever an educator requires to close a tournament, including the check whether there is still an ongoing battle
- **Battle Manager:** is responsible for all the functionalities related to code battles in the application
  - **Battle Creation Service:** it allows to perform the operations required whenever a new battle is created by an educator, like uploading the corresponding CodeKata and setting the constraints required for each group to join

  - **Battle Enrollment Service:** it handles the required operations and checks that the system has to make whenever a user wants to enroll into a battle
  - **Battle Configuration Service:** allows getting and modifying battle-related properties, like its title, or description
  - **Battle Closing Service:** it handles all the operations required when a battle is closed ahead of time, like the set-up of a message explaining the motivation of such closure

- **Group Manager:** manages all the functionalities related to groups, like the authentication process and the setting of user's information.
  - **Group Creation Service:** handles the creation of a new group by a student, including the required checks on whether he/she is enrolled into the battle he/she wants to create a group for and if that group name is already in use
  - **Group enrollment Service:** allows the enrollment of a student into an existing group, also checking if such student is enrolled in the same battle the group has been created for

- **Repository Manager:** manages the communications from and to the online repository system (GitHub), creating the repository as described by the educator, uploading the corresponding CodeKata. When the battle is started it also automatically pulls every user's push of a new solution and sends it to the Delivery Evaluation Manager

- **Delivery Evaluation Manager:** manages the whole process of evaluation of a student's delivery, starting from the automated ones (according to what has been set up at creation of the battle), to the potential manual evaluation by the educator. Lastly if such manual evaluation is performed, it computes the final delivery score (battle score) according to the composition of both these evaluations
  - **Code Evaluation Service:** performs all the automated evaluations that have been set up at battle-creation time, including the static analysis of the quality level of the sources through external APIs (CodeQL)
  - **Manual Evaluation Service:** if a manual evaluation has been set up to be performed after the automated ones at battle-creation time, this

component allows the educator to manually assign an evaluation to each student delivery

  - **Delivery Score Calculator Service:** this component allows for the composition of both the scores given by the automated evaluation and the one given manually by the educator, by adding or subtracting the score manually assigned by the educator to the one that has been automatically given, this service has also been introduced in order to allow in the future for the possible inclusion of new ways to perform this composition (like a weighted average or by allowing the educator to increase/decrease the automated score assigned for a specific aspect of the code)

- **Notification Event Manager:** controls the different kind of way in which a notification can be sent to the user

  - **Application Notification Service:** handles the in-application notifications, so the ones that will remain only in the application itself, and shown through a specific icon on the user's dashboard
  - **Email Notification Service:** handles those notifications that will also be sent as an email to the user, to highlight their greater importance
  - **Calendar Notification Service:** handles a special type of notification occurring when a battle is created in a tournament to which the student is subscribed that will also create (if such permission has been given by the user) an event in the user's Google Calendar application for the corresponding date and time

- **Data Manager:** this component manages the access of the application to the database representing its data layer, performing eventual pre-processing on the data if required

2.3 Deployment View

The deployment diagram depicted here below illustrates the distribution of different components of the web application across the network. This includes the deployment of the web server, application server, and database server.

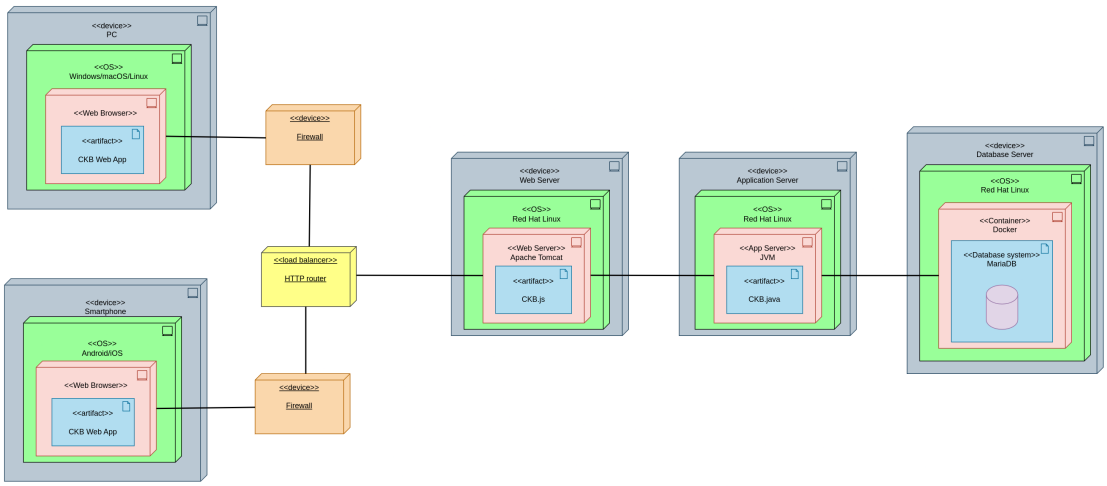


Figure 3: Deployment Diagram

The diagram visually represents the relationships and the flow of data among these components.

- **Smartphone/PC**: the user's device is used for displaying the web application. It's utilized to access and interact with the web application through the web browser and the HTTPS protocol.
- **Firewall**: fundamental security device, safeguards the network against unauthorized access by filtering both incoming and outgoing traffic. A proper configuration of the firewall is essential to ensure the security of the system.
- **Load Balancer**: it serves to evenly distribute incoming web traffic among multiple servers, ensuring the continuous availability of the web application and enhancing the system's scalability.
- **Web Server**: it's the server in charge of handling the HTTPS requests coming from the clients. It handles the communication between the clients and the platform itself.

- **Application Server**: responsible for the business logic of the system, it manages the requests coming from the web server. It's also in charge of the communication with the database server.

- **Database server**: it's the server responsible for storing all the information within the web application. A dockerized[4] version of MariaDB will be used as the database management system in order to further enhance flexibility and scalability. We opted for a relational DBMS, particularly MariaDB, due to its robust data integrity features, seamless scalability, and ACID compliance. Furthermore, MariaDB's open-source nature, active community support, and compatibility with MySQL make it a cost-effective and reliable choice for our database needs.

[4]An application is dockerized if it runs inside a Docker container

2.4 Component Interfaces

The following diagram provides a comprehensive depiction of the interfaces and their associated methods offered by each component within the CKB platform. It is essential to acknowledge that the delineated methods do not precisely mirror the final version intended for implementation; rather, they serve as a logical representation.

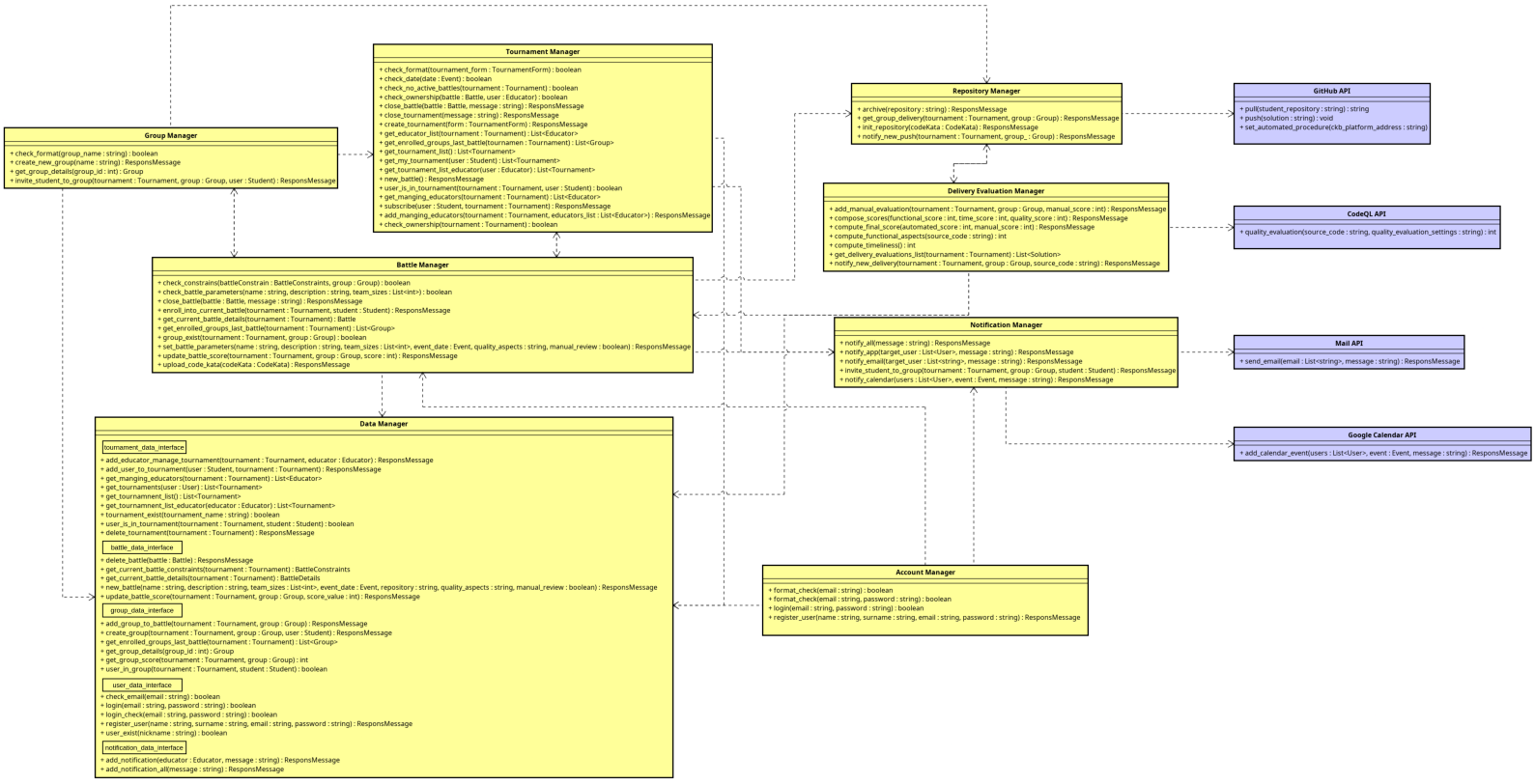
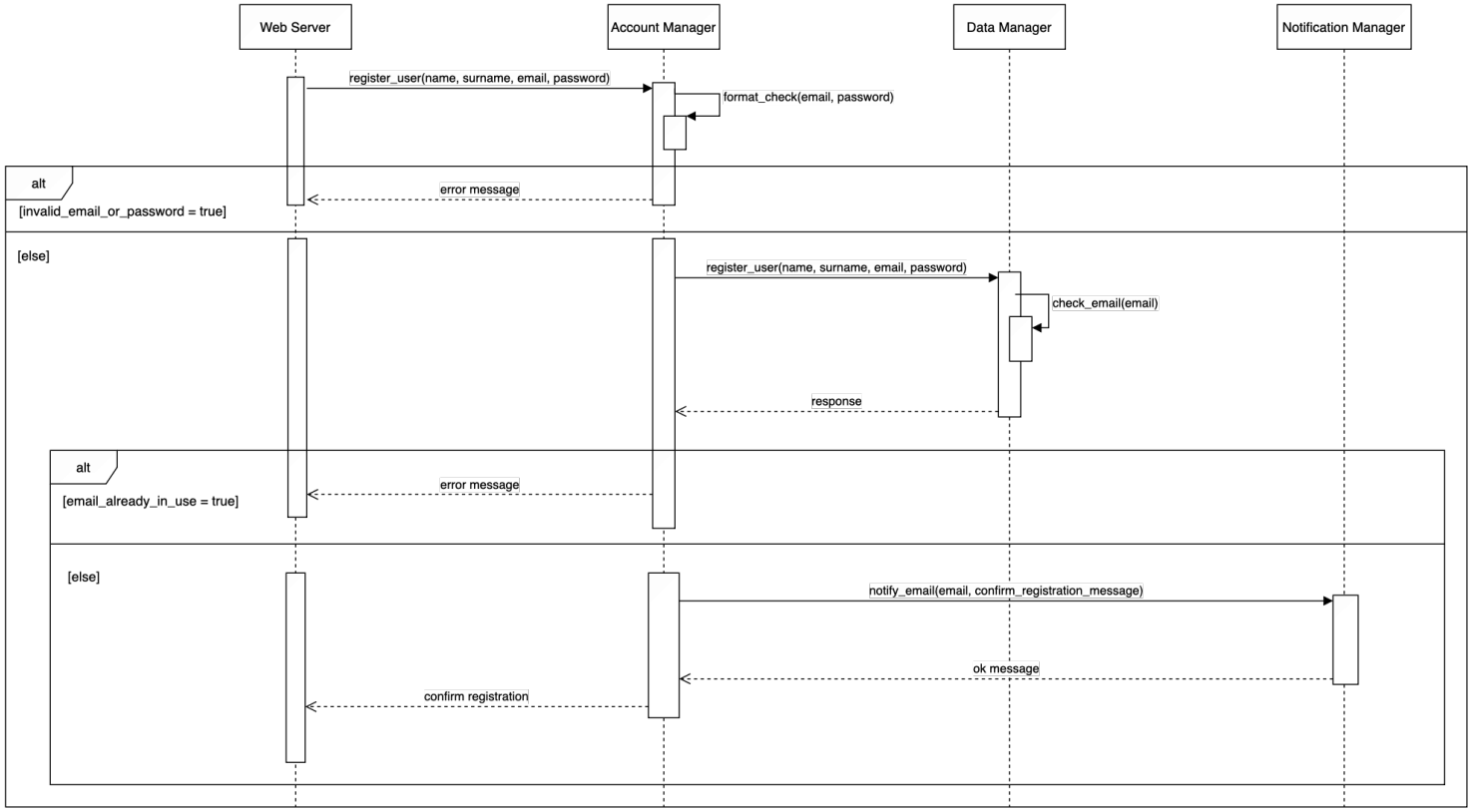


Figure 4: Component Interface diagram

2.5 Runtime View

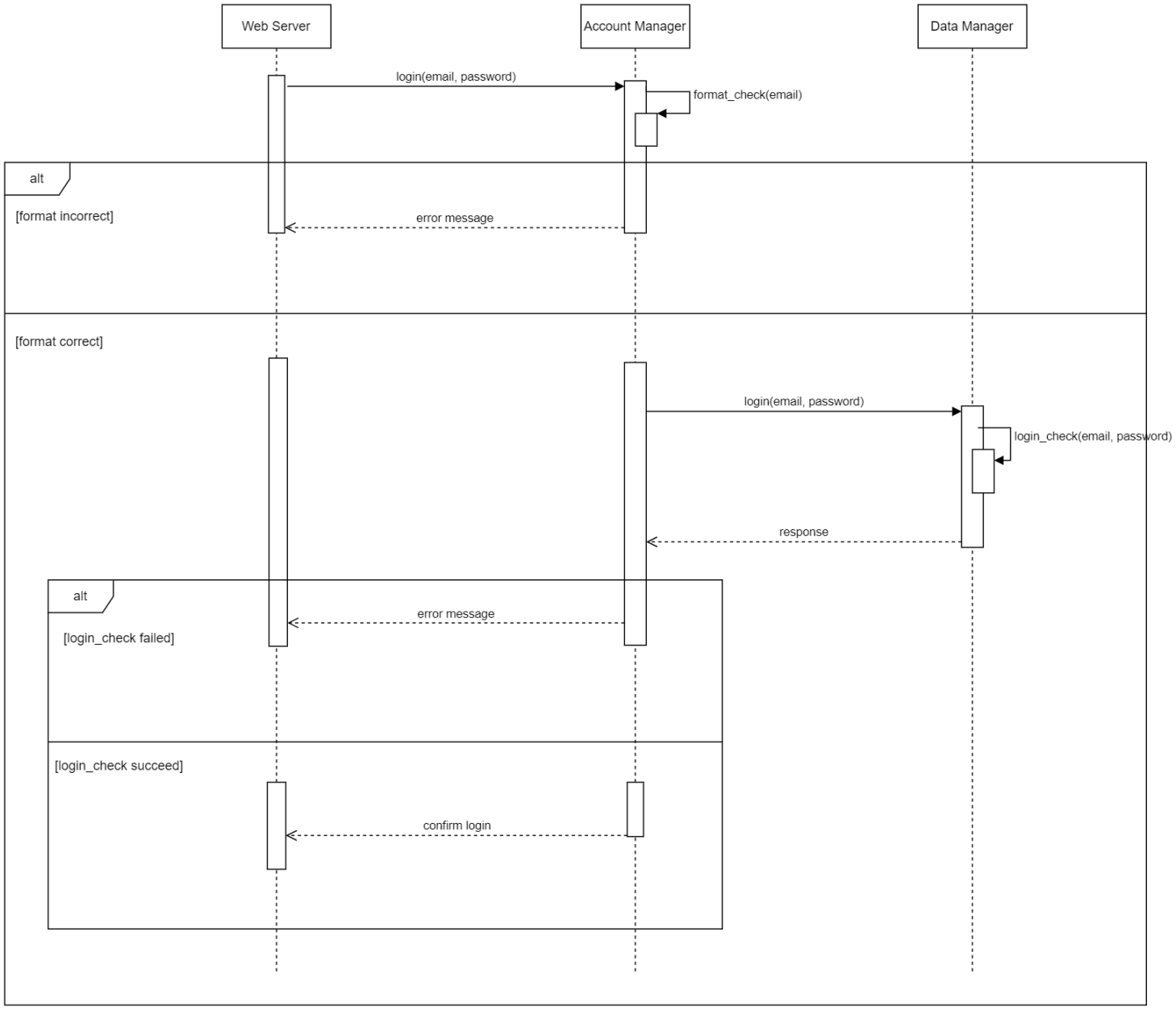
In this section various practical application of the system are presented through sequence diagrams.

Sign up



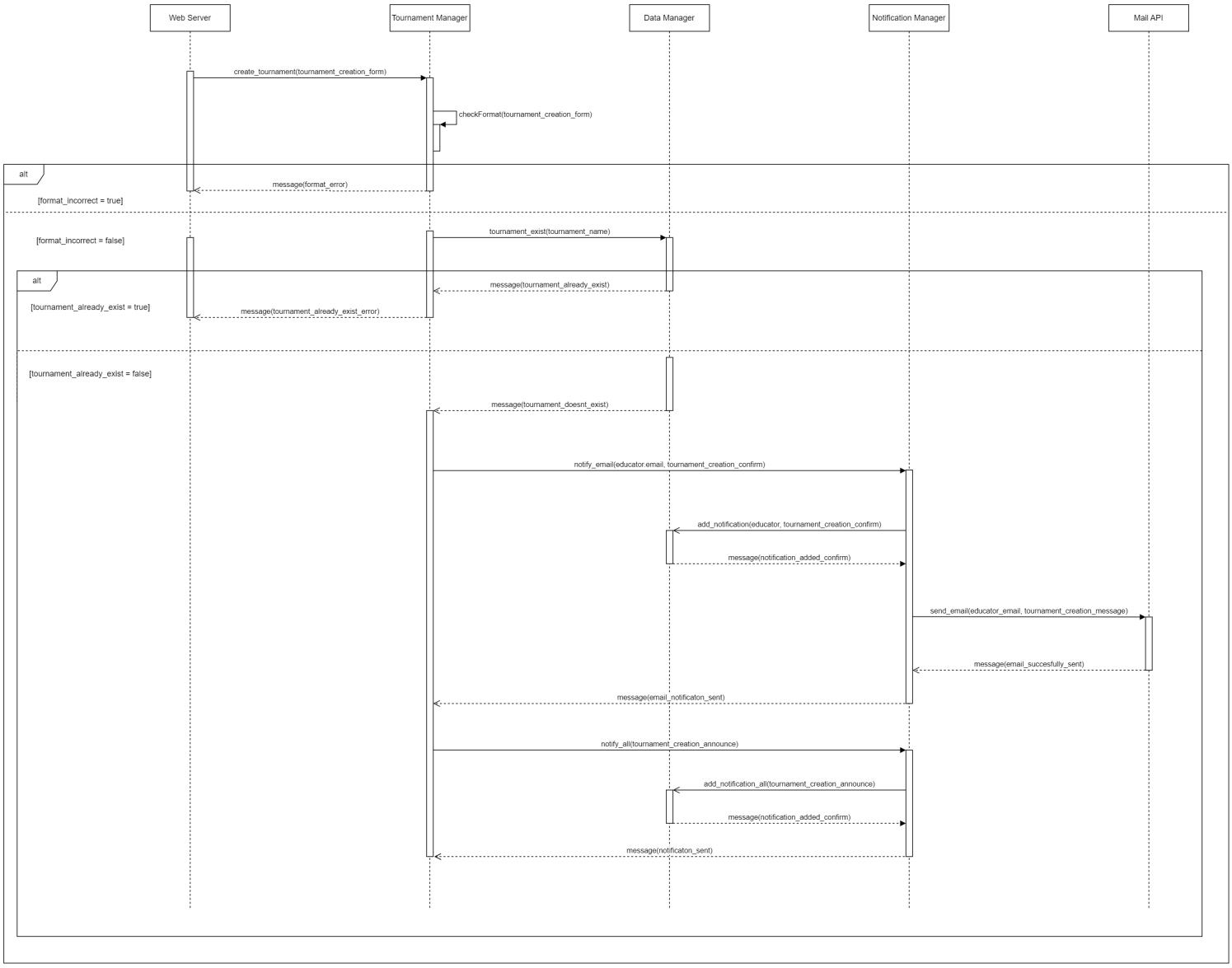
The *Sign-Up* phase involves users entering their personal information into the designated fields. Subsequently, the system verifies whether the email, serving as a unique identifier in the database, is already registered. If the email is not found, the user is eligible to proceed with the sign-up process.

Log in


Web Server
Account Manager
Data Manager
login(email, password)
format_check(email)
alt
[format incorrect]
error message
[format correct]
login(email, password)
login_check(email, password)
response
alt
error message
[login_check failed]
[login_check succeed]
confirm login

The *Log-In* phase involves users entering their credentials into the designated fields. Subsequently, the system verifies whether the email and password combination is valid. If the credentials are valid, the user is eligible to proceed with the log-in process, hence to use the CKB platform.

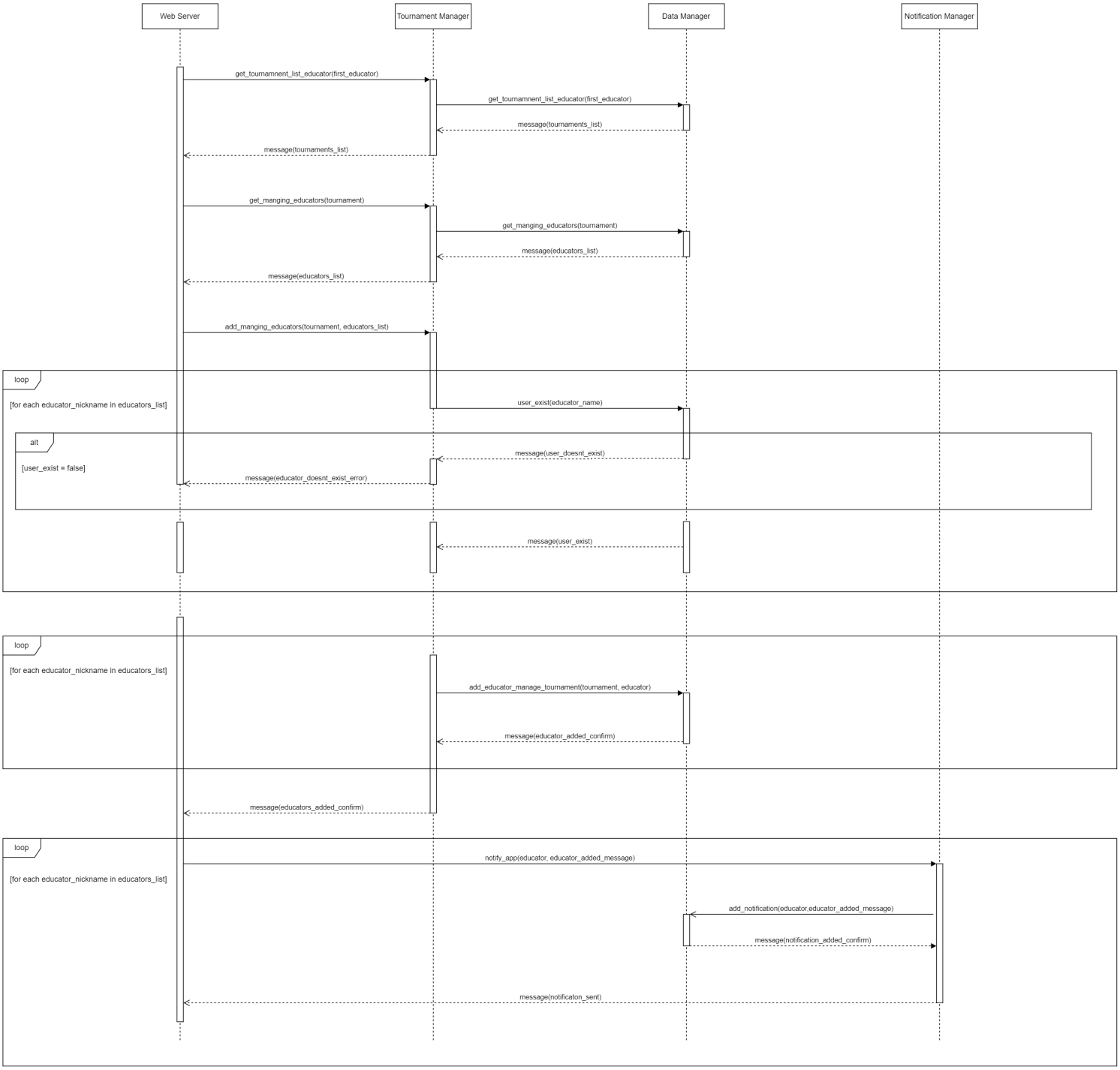
Create a tournament



The *Create a tournament* phase involves an educator entering the tournament's information into the designated fields. In particular, after the educator has entered the tournament's name, the system verifies whether there exists another tournament

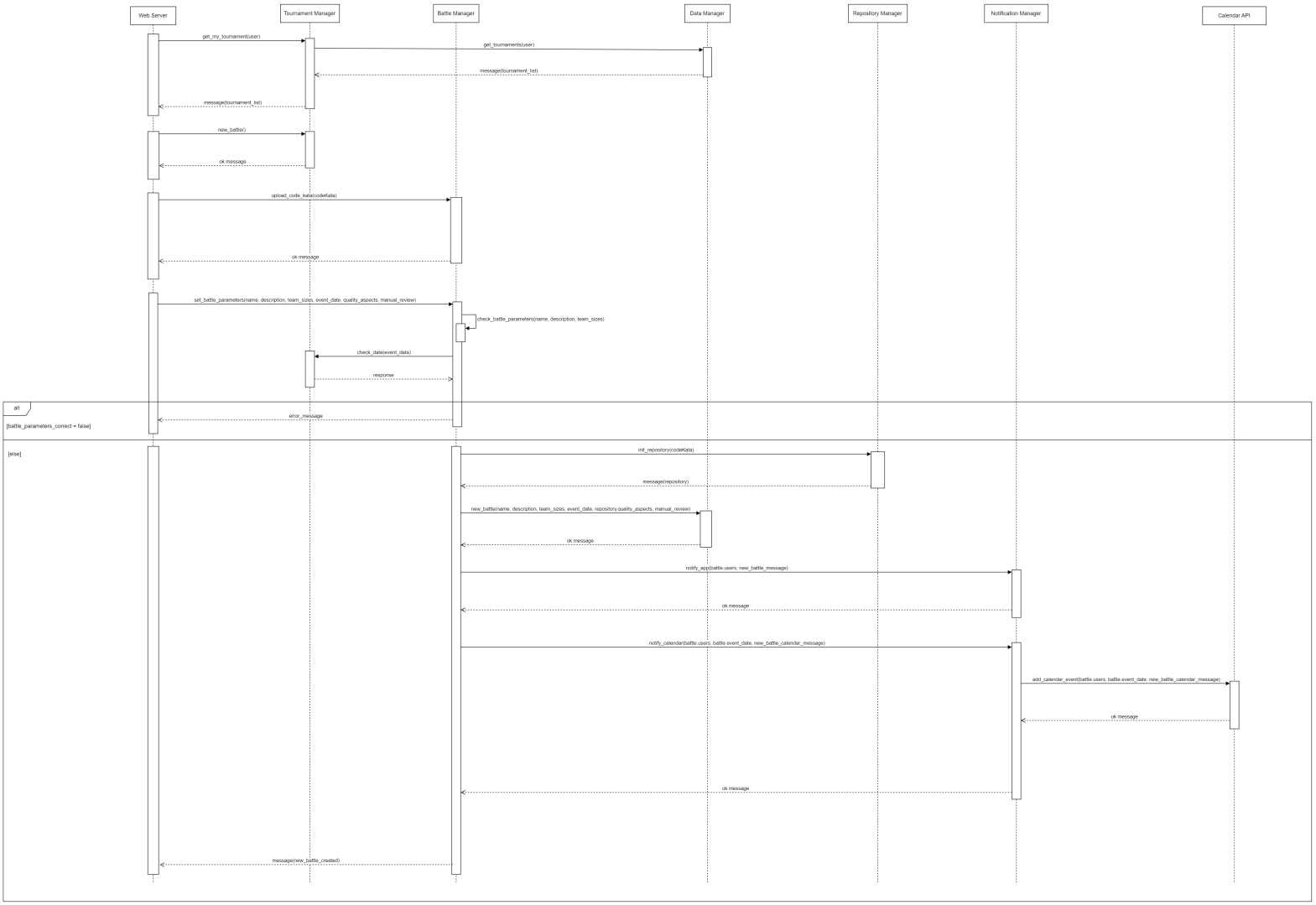
with the same name. Upon successful verification a new tournament is created. A notification to all the users of the CKB platform is sent to inform them about the creation of the new tournament.

Invite other educators to manage a tournament



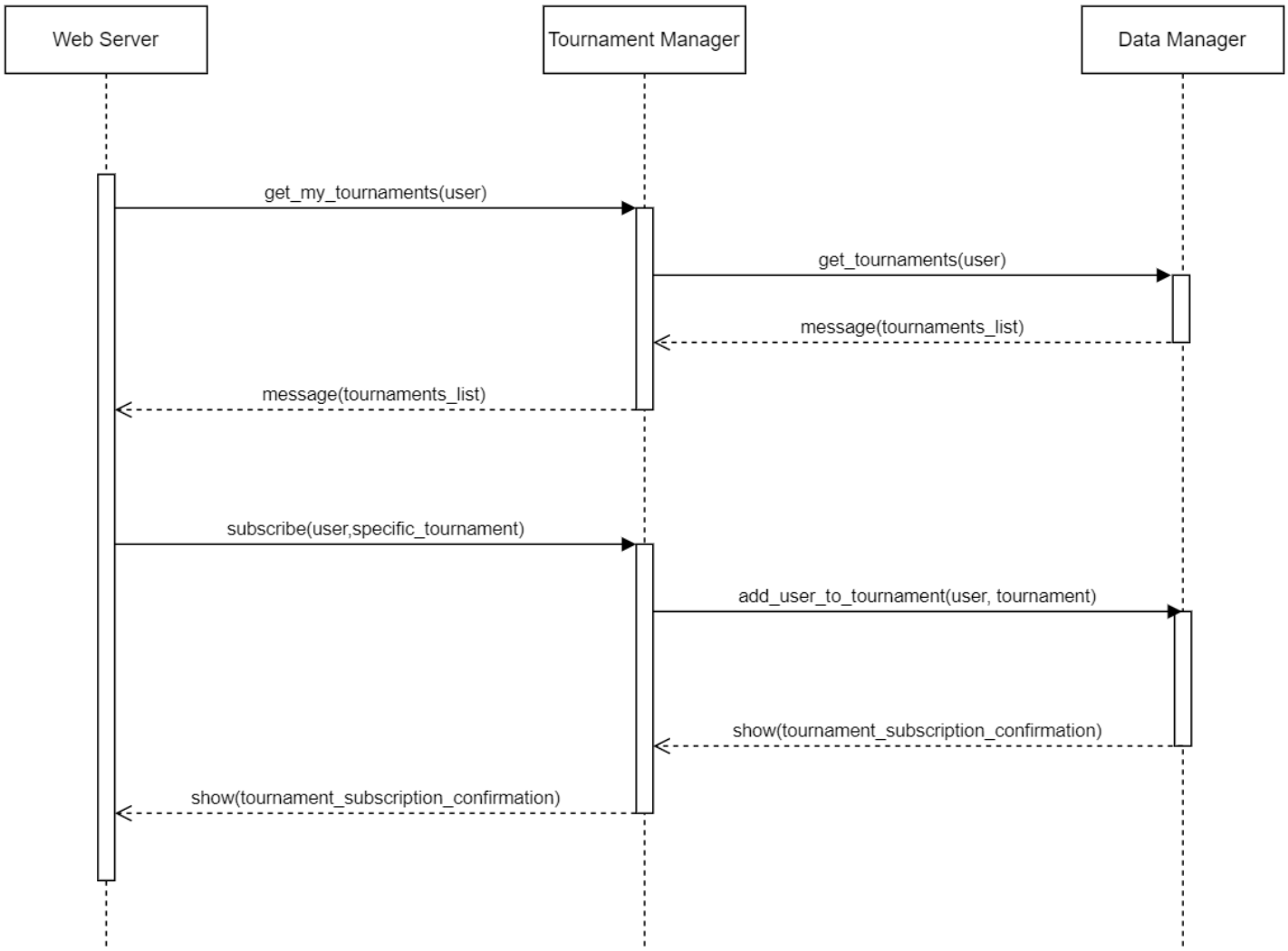
The *Invite other educators to manage a tournament* phase involves an educator entering the email of the educator he/she wants to invite into the designated field. Specifically, the system checks that the users exists and then sends an invitation to the educator. It's important to note that the educator can invite multiple educators at the same time following this procedure.

Create a battle



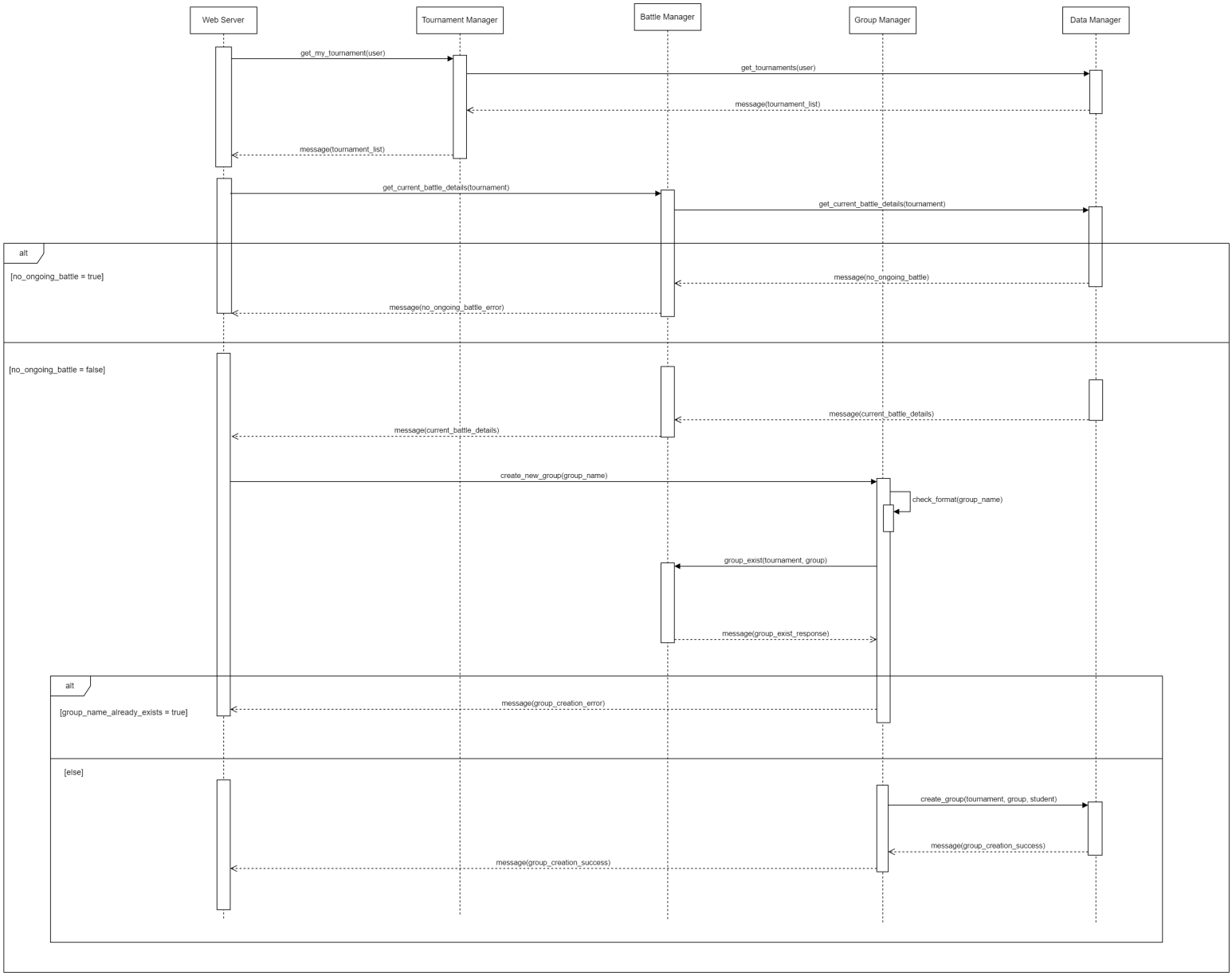
The *Create a battle* phase involves an educator entering the battle's information into the designated fields. In this phase the educators sets all the constraints for the battle, including the CodeKata, the deadlines, the maximum number of participants, and the automated evaluations settings. Before creating the battle, the system checks that the all the provided information is valid and consistent(e.g. the deadline is after the start date). Upon successful verification a new battle is created and a notification is sent to all the students subscribed to the tournament where the battle belongs.

Subscribe to a tournament



The *Subscribe to a tournament* phase involves a student selecting a tournament from the list of available tournaments in which he/she wants to participate. Upon a successful subscription, the user, now student, is eligible to enroll in the battles of the tournament.

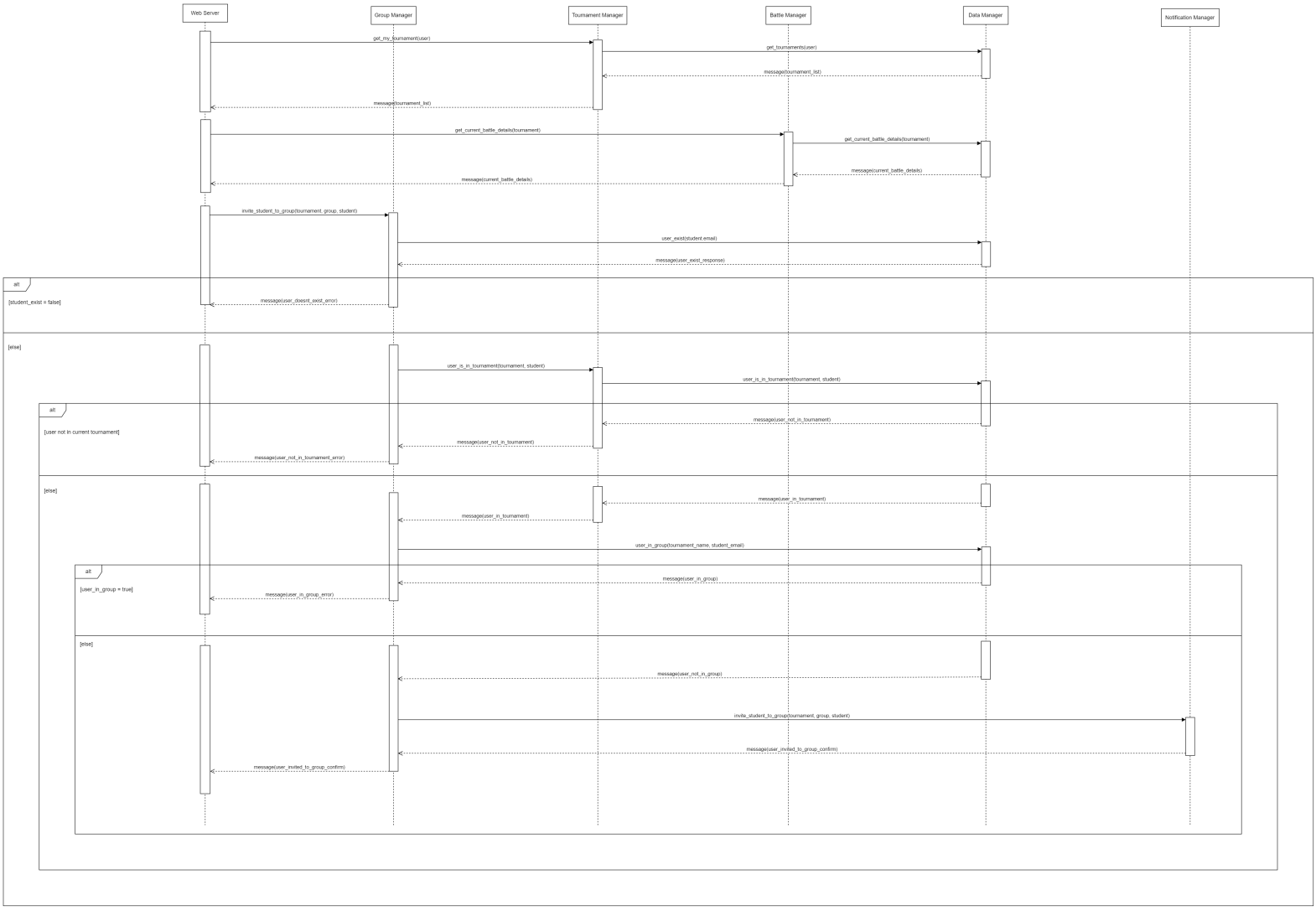
Create a group



The *Create a group* phase involves a student creating a new group in order to compete with other colleagues to battles. To do so the student has to provide the

name of the group using the provided field. The system checks that no other group has the same name in the context of the battle in which the student is going to create such a group.

Invite a student to an existing group

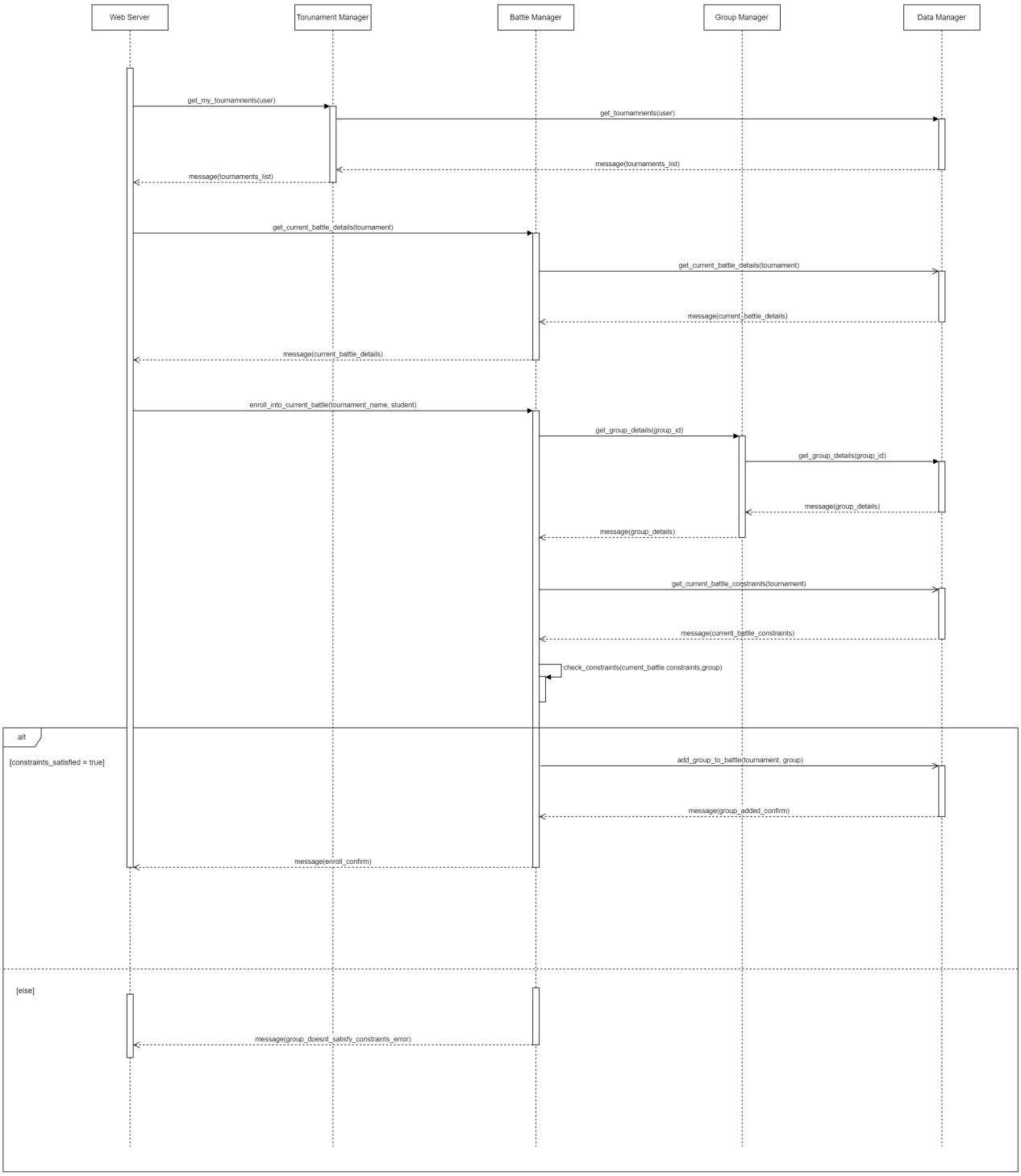


The *Invite a student to an existing group* phase involves a student inviting another student to join an existing group. In particular, the system checks:

- if the invited student exists;
- if the invited student is enrolled in the same tournament and battle as the inviting student;
- if the invited student is already enrolled in another group in the same battle.

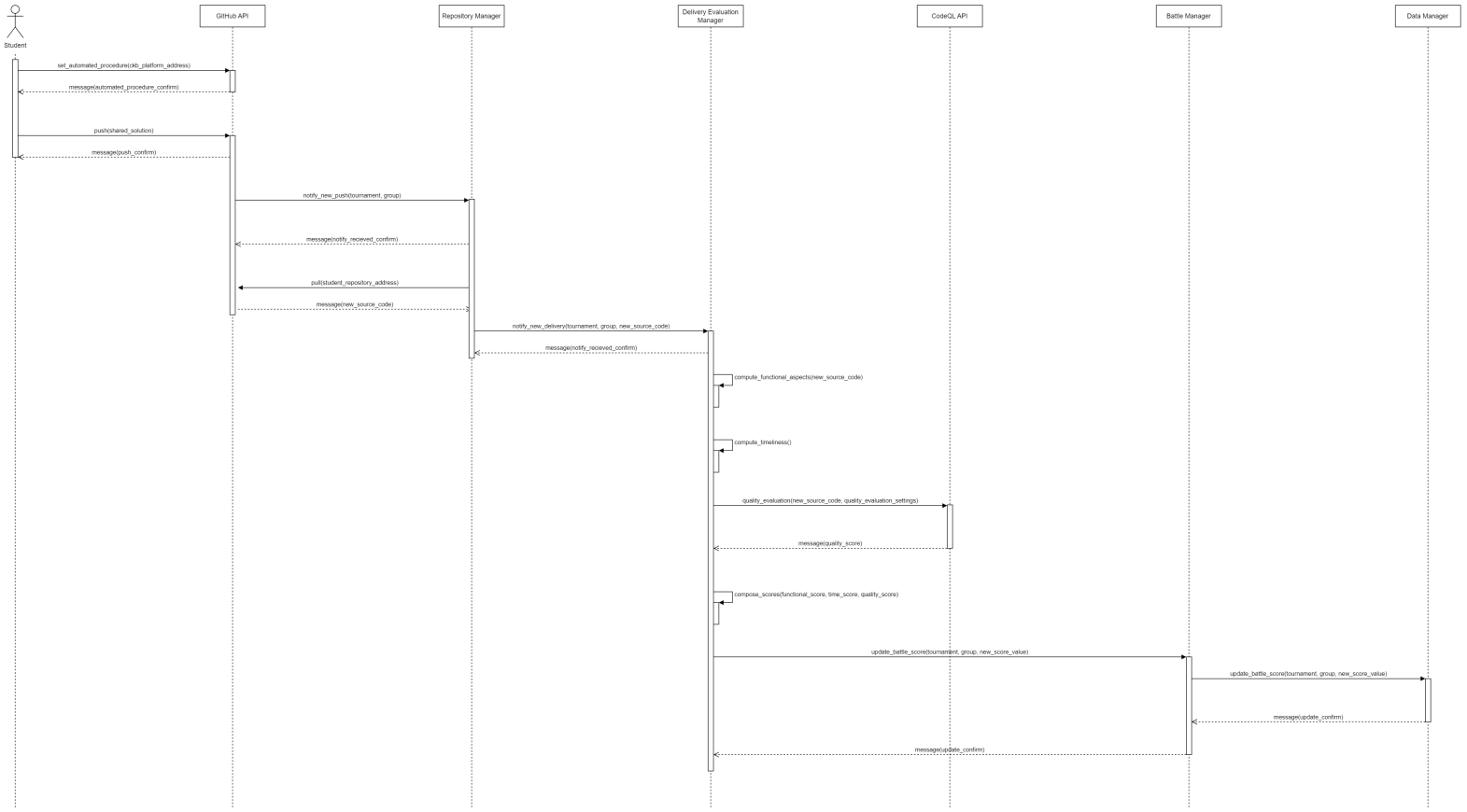
Upon successful verification of the aforementioned conditions a notification is sent to the invited student.

Enroll in a battle



The *Enroll in a battle* phase involves a student enrolling in a battle. In this stage the system ensures that the group enrolling in the battle has the correct size. Upon successful check the student is enrolled in the battle and can compete.

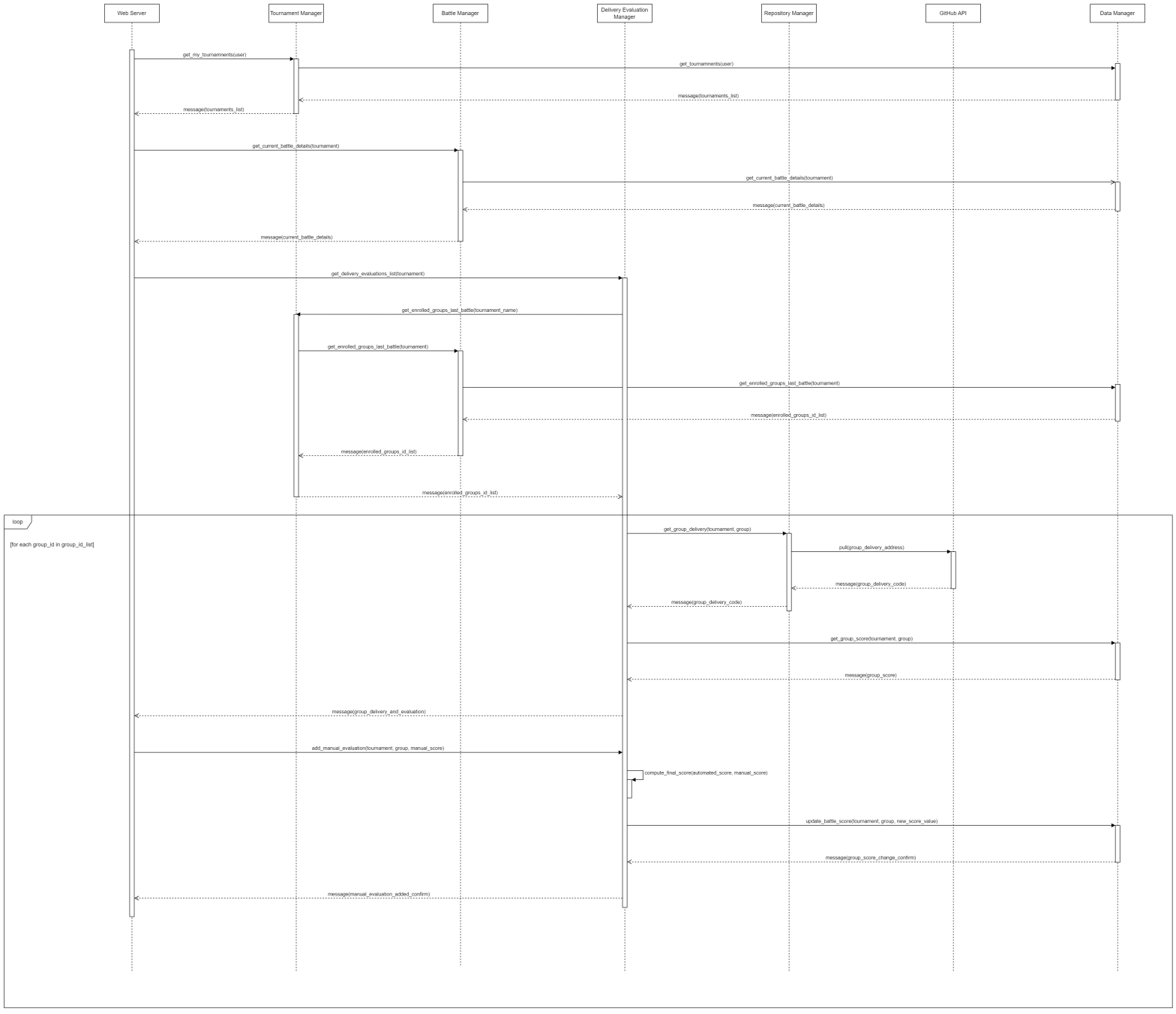
Student delivers a solution



The *Student delivers a solution* phase involves a student delivering a solution to a battle. In this phase, after the student has pushed the solution to the online repository, the system automatically evaluates the solution according to the criteria set by the educator in the battle configuration. The solution is first evaluated in terms of timeliness and number of passed test. A second code quality evaluation is performed by *CodeQL*[5] based on the battle settings. In order for this procedure to be successful the user has to set up an automated workflow using *GitHub Actions*.

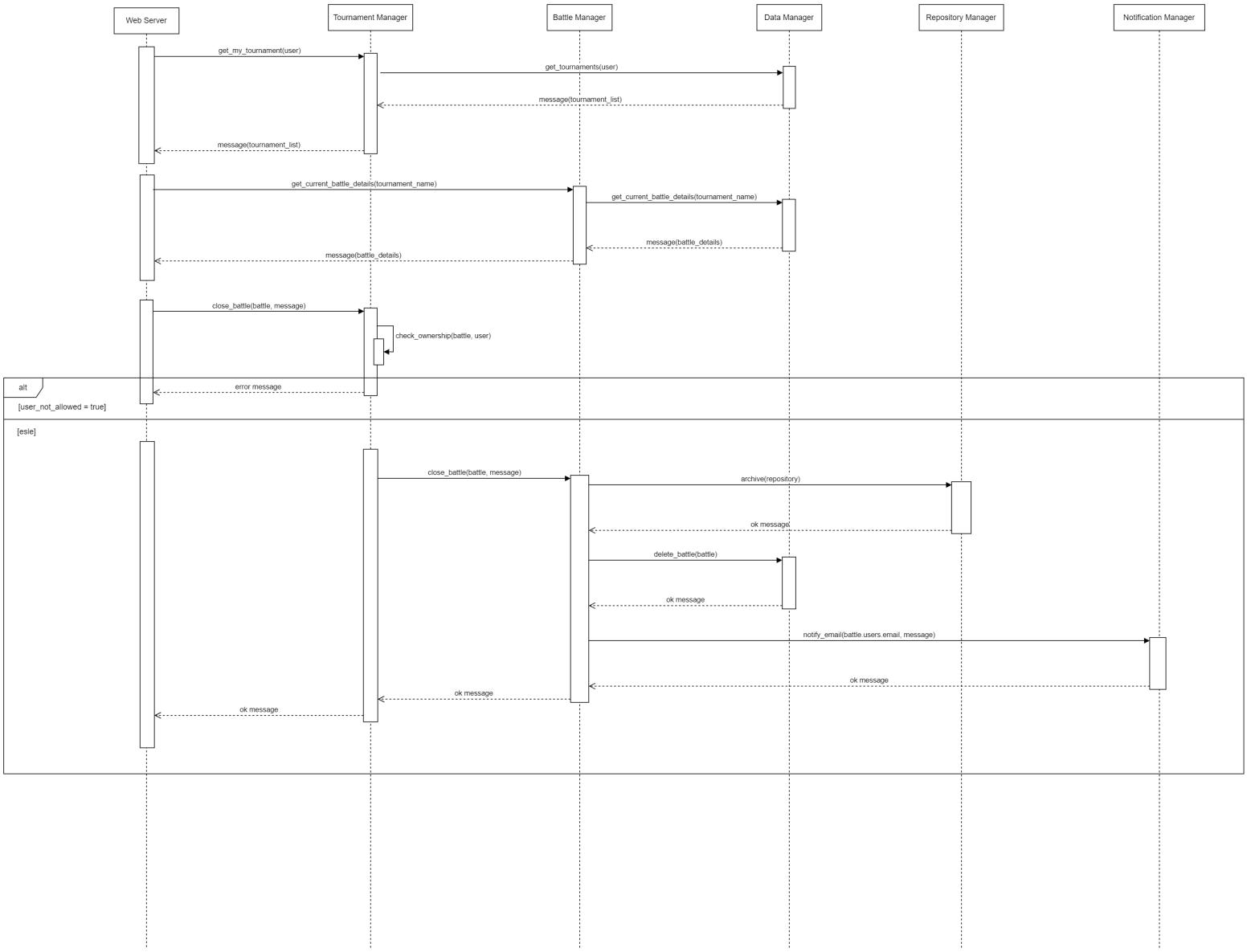
[5]CodeQL is a semantic code analysis engine that allows to write queries to find vulnerabilities in code

Educator does a manual code review



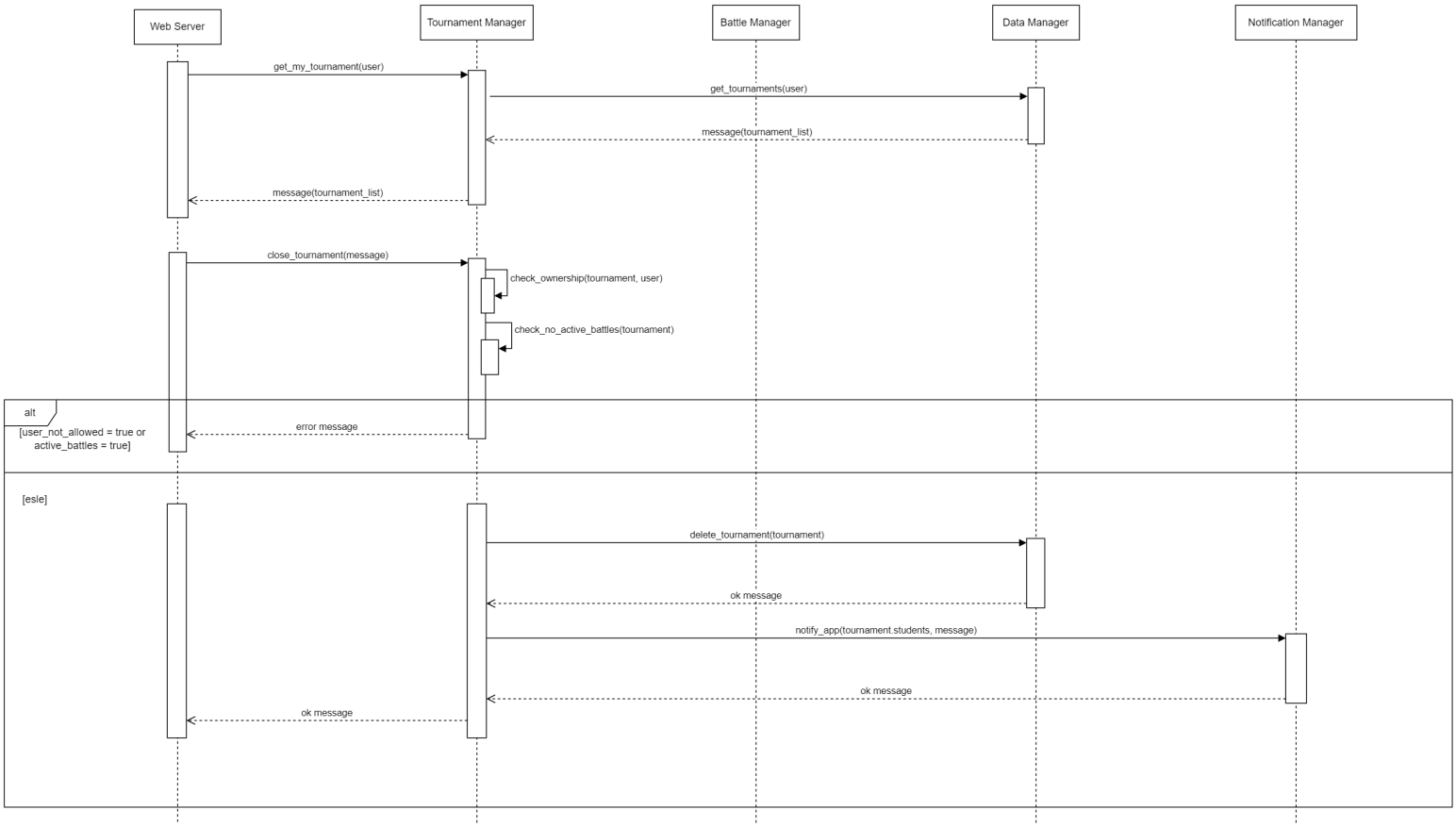
The *Educator does a manual code review* phase involves an educator reviewing a student's solution. In this procedure, after the *Student delivers a solution* phase, the educator can manually evaluate the solution. The solution score is updated starting from the automated evaluation score. It's important to note that the manual review does not substitute the automated one, but it's rather a complement to it that aims to fix, at some extent, the initial automated one.

Close a battle early



The *Close a battle early* phase involves an educator closing a battle before the deadline. In this phase the system checks that the educator has the ownership, hence the permission to close the battle. Upon successful verification the system closes the battle by deleting it from the database and archiving the corresponding CodeKata repository. The educator, using the provided field, also needs to specify a reason for the early closure of the battle, which will be included in the notification sent to all the previously subscribed students.

Close a tournament



The *Close a tournament* phase involves an educator closing a tournament. In this state, the systems checks that no battles are still ongoing in such a tournament. Upon successful check the system closes the tournament by deleting it from the database. A notification informing the students of the tournament's closure is sent to all the previously subscribed students.

2.6 Selected Architectural Styles and Patterns

Here below are discussed the utilized architectural styles and patterns:

- **Three Tier Architecture**: The chosen system architecture is a three-tier architecture comprising a presentation tier, a logic tier, and a data tier.
  - The *presentation tier* is responsible for presenting the application to the user and for handling user interaction;
  - The *logic tier* processes the input, performing necessary calculations or operations necessary to the correct functioning of the application;
  - The *data tier* handles the storage and retrieval of data from a database.
- **Thin client**: The thin client approach enhances security by avoiding local storage of sensitive data and improves scalability by facilitating the deployment of new clients. Additionally, it enables a more centralized management helpful in case of a dynamic environment where the application gets frequently updates.
- **Scalability**: The three-tier architecture enables a modular design, making it easier to scale individual components of the system. Changes to one component do not necessarily require modifications to the other components.
- **Model-View-Controller**: The MVC design pattern is a software design pattern that divides an application into three primary components: the model, the view, and the controller. The model constitutes the data and business logic, the view represents the user interface, and the controller facilitates communication between the model and the view. Specifically:
  - *Model*: it's the central component of the pattern, serving as the dynamic data structure of the application. The model is independent of the user interface, not concerned with how data is displayed nor how the user interacts with the application. It directly manages the data, logic and rules of the application.
  - *View*: responsible for representing the user interface of the application, the view is tasked with displaying information and accepting input from the user. It defines how application data should be presented and how the user can effectively interact with it.
  - *Controller*: this component is responsible for mediating between the model and view components. It receives input from the user through the view

and performs necessary actions on the model based on that input. Subsequently, it updates the view to reflect the modified model. Given the nature of this component, the controller is independent of both model and view.

3 User Interface Design

Sign-Up and Log-In:

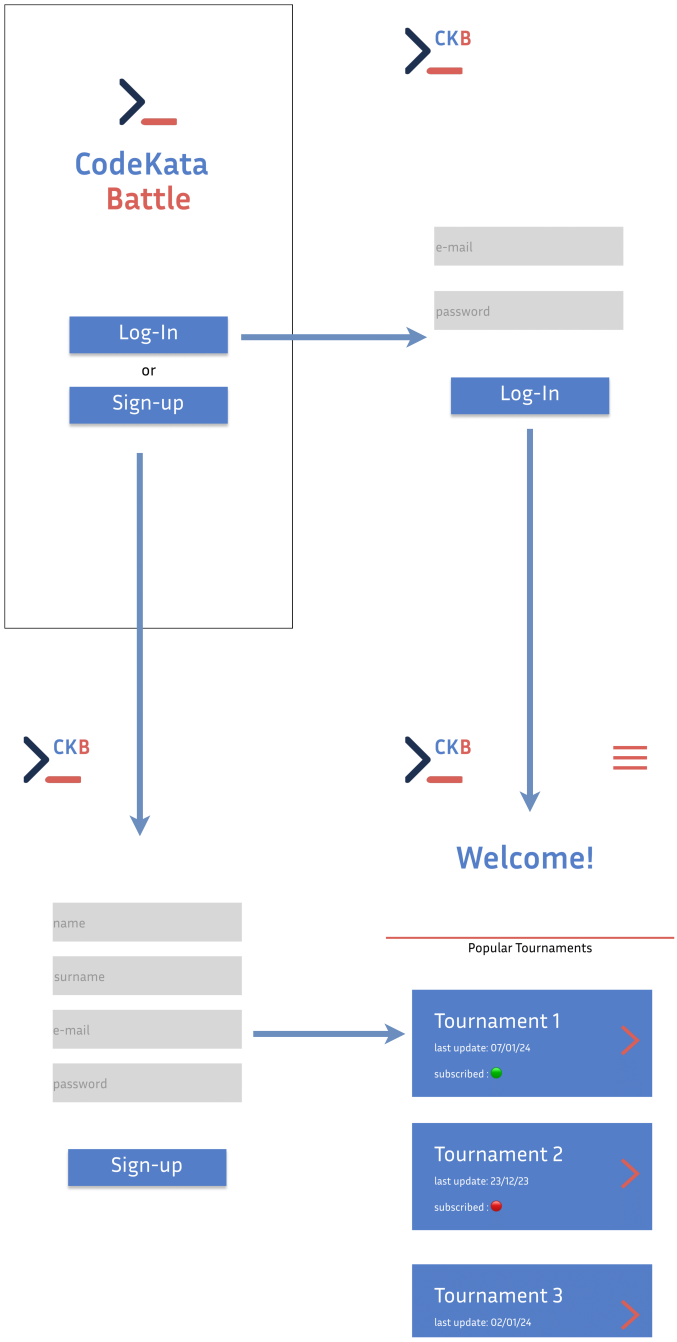


Figure 5: Sign-Up and Log-In

Create a tournament:

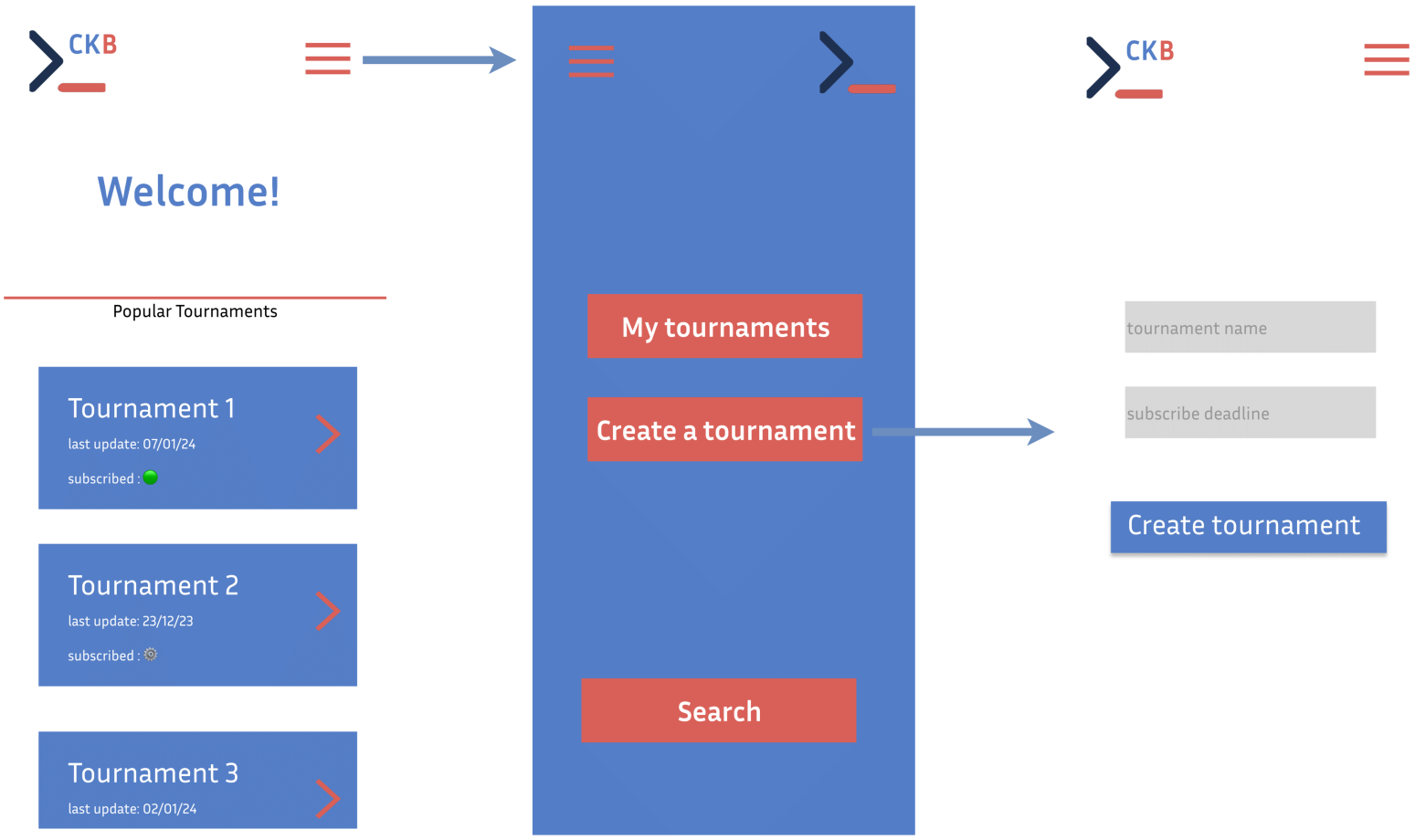


Figure 6: Create a tournament

Enroll into a battle:

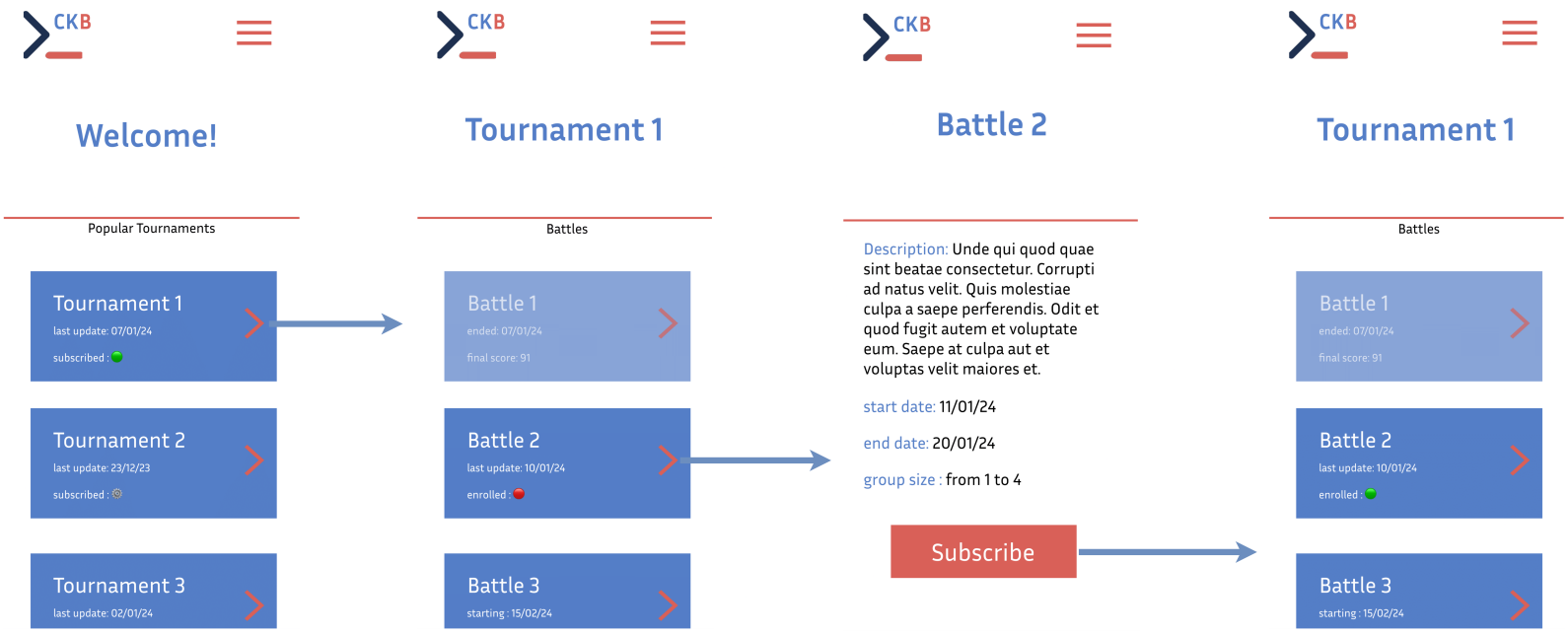


Figure 7: Enroll into a battle

4 Requirements Traceability

Requirements	Components
R1) The system must allow the registration of new users using their e-mail	• Account Manager
R2) The system must notify a user about his/her successful registration	• Notification Event Manager
R3) The system must allow registered users to log-in using their e-mail.	• Account Manager – Authentication Service • Data Manager
R4) The system must allow logged-in users to use the application.	• Account Manager • Data Manager
R5) The system must respect the GDPR	• Account Manager • Data Manager
R6) The system must allow a user to set a nickname for his/her profile	• Account Manager – User Configuration Service
R7) The system must allow a user to set a phone number for his/her profile	• Account Manager – User Configuration Service
R8) The system must allow an educator to create a tournament if another tournament with the same name doesn't already exist	• Tournament Manager – Tournament Creation Service

Requirements	Components
R9) The system must reject an educator request to create a tournament if another tournament with the same name already exist	• Tournament Manager – Tournament Creation Service
R10) The system must allow the owner of a tournament to invite other educators to manage it	• Tournament Manager – Tournament Configuration Service
R11) The system must notify an educator when he/she is invited to manage a tournament	• Tournament Manager • Notification Event Manager
R12) The system must allow an educator to accept the invitation of another educator to manage a tournament	• Tournament Manager • Notification Event Manager
R13) The system must allow an educator to set a registration deadline when creating a tournament	• Tournament Manager – Tournament Creation Service – Tournament Configuration Service
R14) The system must allow to set a minimum number of students for each group when creating a new battle	• Battle Manager – Battle Creation Service – Battle Configuration Service

Requirements	Components
R15) The system must allow to set the maximum number of students for each group when creating a new battle	• Battle Manager – Battle Creation Service – Battle Configuration Service
R16) The system must allow to set a registration deadline when creating a new battle	• Battle Manager – Battle Creation Service – Battle Configuration Service
R17) The system must allow to set a final submission deadline when creating a new battle	• Battle Manager – Battle Creation Service – Battle Configuration Service
R18) The system must allow an educator to choose while creating a new battle which aspects of the quality level of the sources will be automatically evaluated	• Battle Manager – Battle Creation Service – Battle Configuration Service
R19) The system must allow an educator to choose while creating a new battle whether a manual evaluation will be needed after the automated ones	• Battle Manager – Battle Creation Service – Battle Configuration Service
R20) The system must be able to receive a new code kata from an educator when he/she is creating a battle	• Repository Manager • Battle Manager

Requirements	Components
R21) The system must allow the educators managing a tournament to create new battles in the context of such tournament	• Tournament Manager • Battle Manager – Battle Creation Service
R22) The system must notify all users of the platform when a new tournament is created	• Tournament Manager – Tournament Creation Service • Notification Event Manager
R23) The system must notify all students subscribed to that tournament when a new battle of such tournament starts	• Tournament Manager – Tournament Subscription Service • Battle Manager – Battle Creation Service • Notification Event Manager
R24) The system must allow users to subscribe to a tournament	• Tournament Manager – Tournament Subscription Service
R25) The system must allow students to create groups in the context of a battle	• Battle Manager – Battle Enrollment Service • Group Manager – Group Creation Service

Requirements	Components
R26) The system must allow a student to invite another to his/her own group, if such student is enrolled into the corresponding battle	• Group Manager • Battle Manager
R27) The system must reject a student request to invite another student to his/her own group, if such student is not enrolled into the corresponding battle	• Group Manager • Battle Manager
R28) The system must notify a student when he/she is invited to a group	• Group Manager • Notification Event Manager
R29) The system must allow a student to join an existing group if invited	• Group Manager • Notification Event Manager
R30) The system must allow groups to join a newly created battle if they respect that battle's constraints	• Group Manager • Battle Manager
R31) The system must allow all students enrolled into a battle, to access its code kata	• Battle Manager • Repository Manager
R32) The system must allow a group to submit his/their solution	• Repository Manager
R33) The system must update a group battle score after each new solution is delivered	• Delivery Evaluation Manager • Data Manager

Requirements	Components
R34) The system must show to each group his/their rank in the current battle	• Delivery Evaluation Manager • Data Manager
R35) The system must show to each group his/their current battle score	• Delivery Evaluation Manager • Data Manager
R36) The system must notify all students subscribed to a tournament when the consolidation phase of a battle of such tournament has ended	• Notification Event Manager • Tournament Manager
R37) The system must be able to evaluate functional aspects of a group's delivery	• Delivery Evaluation Manager – Delivery score calculator Service
R38) The system must be able to evaluate the timeliness of a group's delivery	• Delivery Evaluation Manager • Repository Manager
R39) The system must be able to evaluate the quality level of the sources of a group's delivery	• Delivery Evaluation Manager – Code evaluation Service
R40) The system must show the results of its automated analysis to the educators managing the tournament	• Delivery Evaluation Manager • Data Manager
R41) The system must allow an educator to see the solution provided by each group	• Delivery Evaluation Manager • Repository Manager

Requirements	Components
R42) The system must allow an educator to manually mark a solution in the consolidation phase, if this was set up in the initial Configuration of the battle	• Delivery Evaluation Manager – Manual evaluation Service • Battle Manager
R43) The system must update the tournament score of each student participating into that tournament after the consolidation phase has ended	• Delivery Evaluation Manager • Data Manager
R44) The system must allow a student to see his/her rank in a certain tournament	• Data Manager
R45) The system must allow a student to see his/her tournament score	• Data Manager
R46) The system must allow all users to see the list of ongoing tournaments	• Data Manager
R47) The system must allow all users to see the current tournament score leaderboard for each ongoing tournament	• Data Manager
R48) The system must allow an educator to close a battle before its expected conclusion	• Battle Manager – Battle Closing Service
R49) The system must allow an educator to set up a message explaining his/her motivations when Closing a battle ahead of time	• Battle Manager – Battle Closing Service

Requirements	Components
R50) The system must notify all students subscribed to a tournament when a battle in such tournament is closed ahead of time	• Battle Manager – Battle Closing Service • Notification Event Manager • Tournament Manager
R51) The system must show the educator's motivation message when notifying students about the early closure of a battle	• Battle Manager – Battle Closing Service • Notification Event Manager
R52) The system must allow an educator to close a tournament, if there is not an ongoing battle in such tournament	• Battle Manager • Tournament Manager – Tournament Closing Service
R53) The system must reject an educator request to close a tournament, if there is an ongoing battle in such tournament	• Battle Manager • Tournament Manager – Tournament Closing Service
R54) The system must notify all students subscribed to a tournament when such tournament is closed	• Tournament Manager • Notification Event Manager

5 Implementation, Integration and Test Plan

5.1 Implementation Plan

In order to increase the pace of our development, prioritizing the parallelization of various components becomes a top priority. A recommended strategy involves adopting a bottom-up approach, therefore focusing on foundational components with higher dependencies and subjecting them to rigorous testing. This testing will be initially performed by unit testing, since incorporating this kind of testing during the development phase will ensure the construction of a robust application infrastructure, thereby mitigating the need for extensive revisions of previous work. Additionally, early detection of bugs and errors is facilitated through unit testing, preventing instances where errors may necessitate a comprehensive overhaul of the entire implementation.

5.2 Integration strategy

Considering the architecture of our system and the implementation plan in place, the most rational integration and testing strategy would be bottom-up. This approach, on the whole, facilitates thorough testing at each level, the early identification of issues before they become too widespread. Moreover, bottom-up encourages modularity and reusability. Finally, thanks to its inherent characteristics, bottom-up also facilitates the seamless addition of features to the subsystem as needed.
Components not yet implemented are simulated via test drivers that will be later replaced by the actual component.
In the following section the order of development, integration and testing of the components is defined. Components not yet implemented will be simulated with test drivers or stubs that will be later replaced by the actual component once it's actually developed.

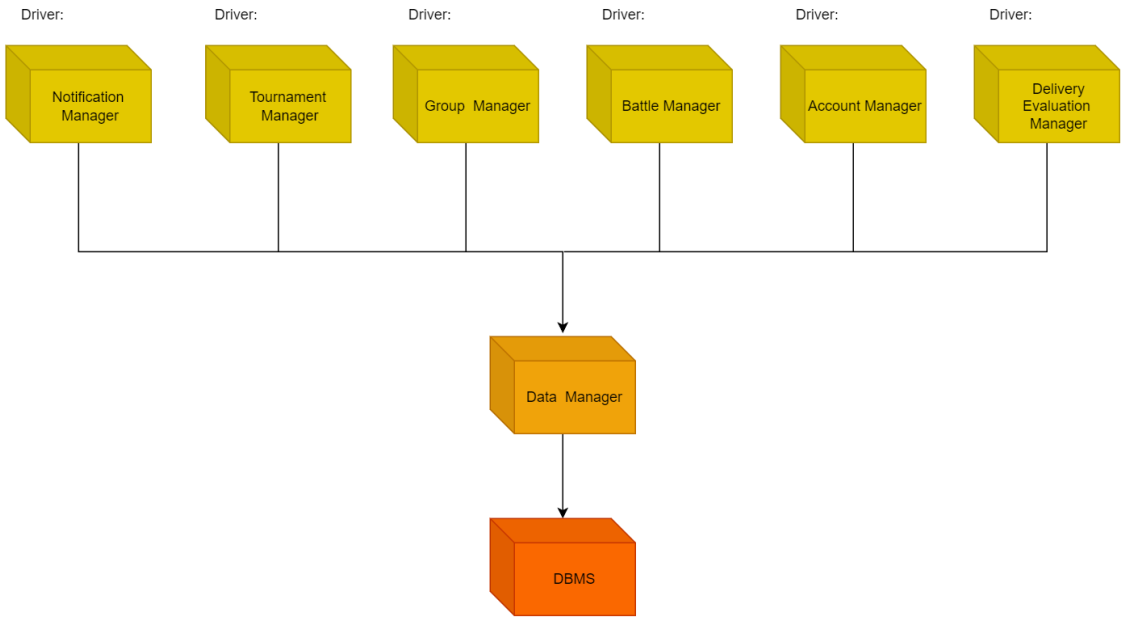


Figure 8: Integration of Data Manager

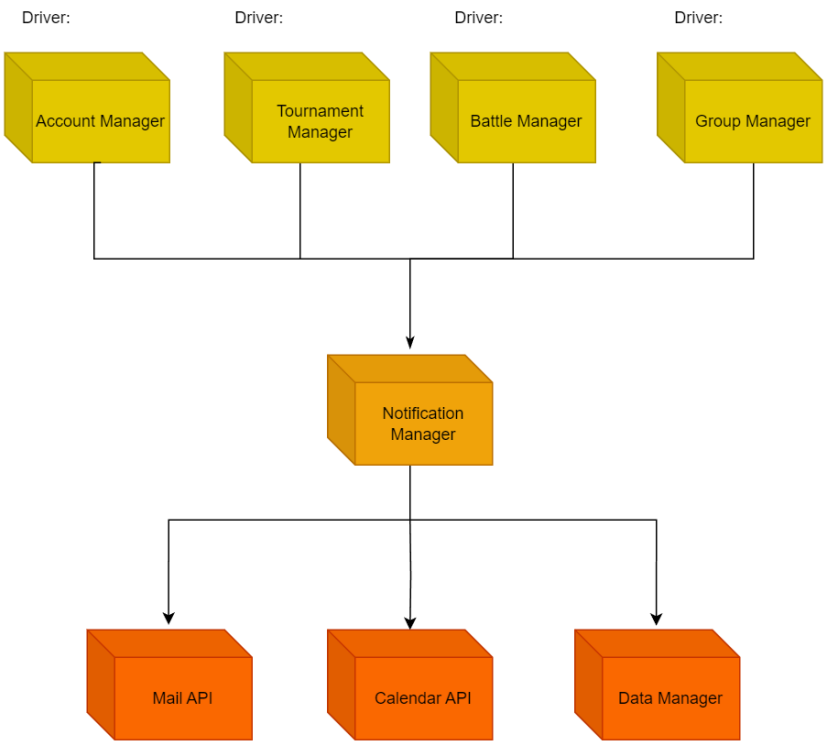


Figure 9: Integration of Notification Manager

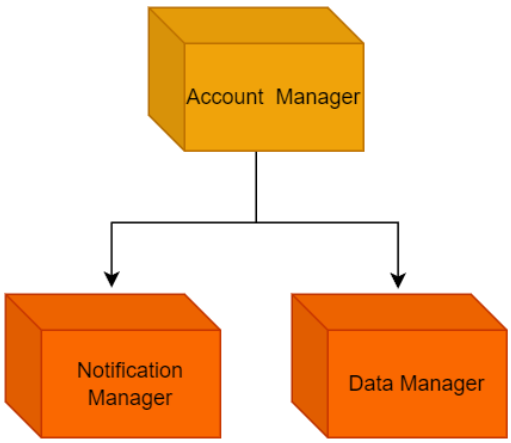


Figure 10: Integration of Account Manager

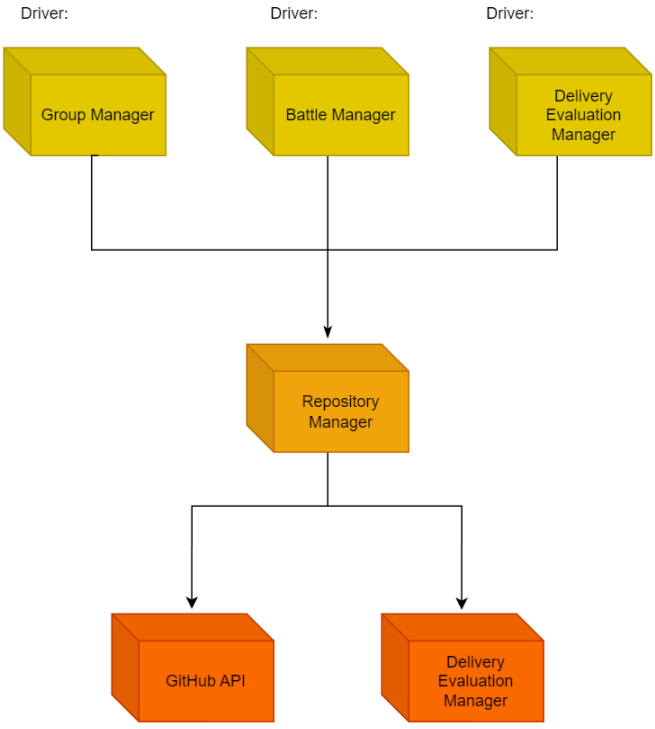


Figure 11: Integration of Repository Manager

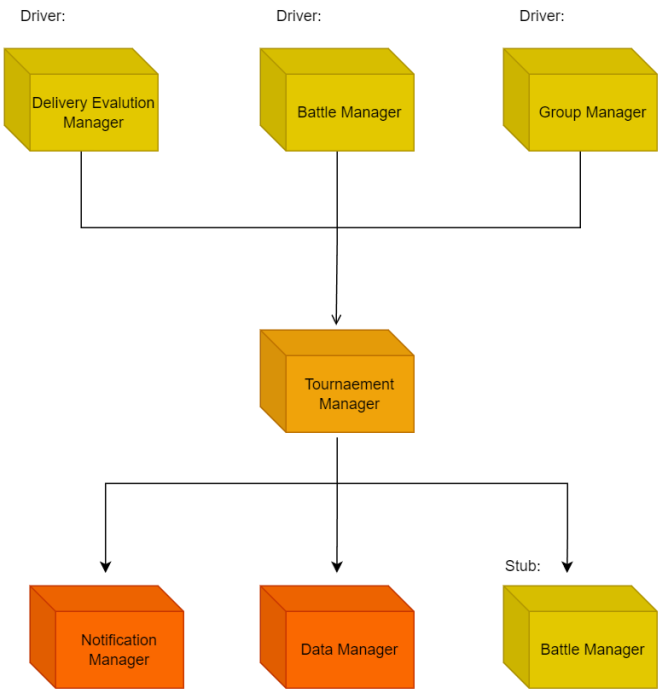


Figure 12: Integration of Tournament Manager

Figure 13: Integration of Delivery Evaluation Manager

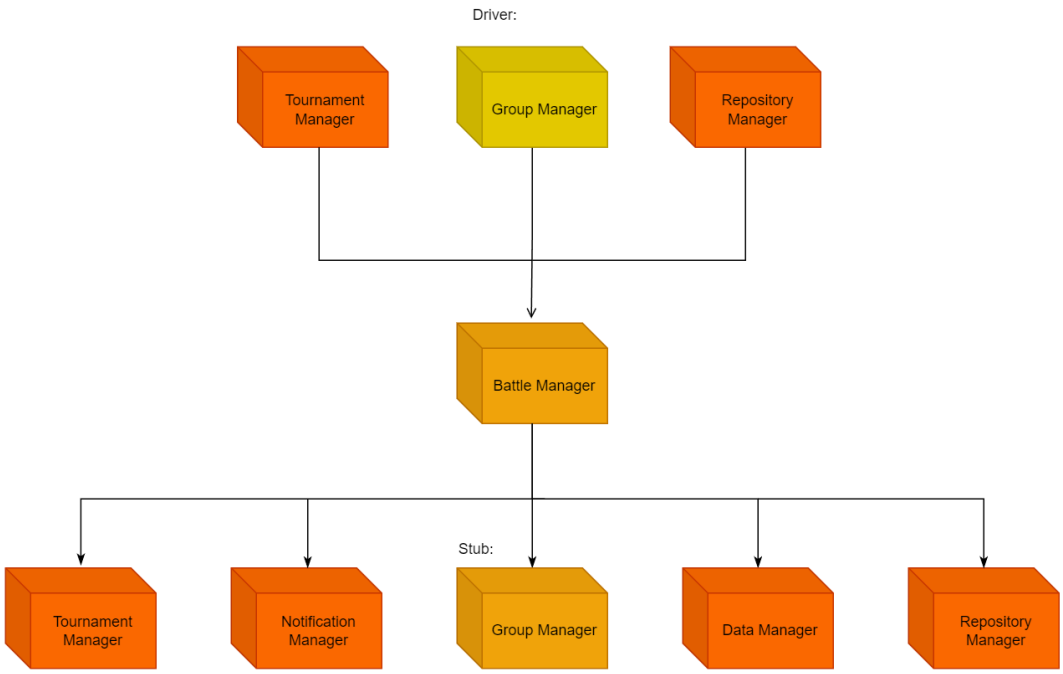


Figure 14: Integration of Battle Manager

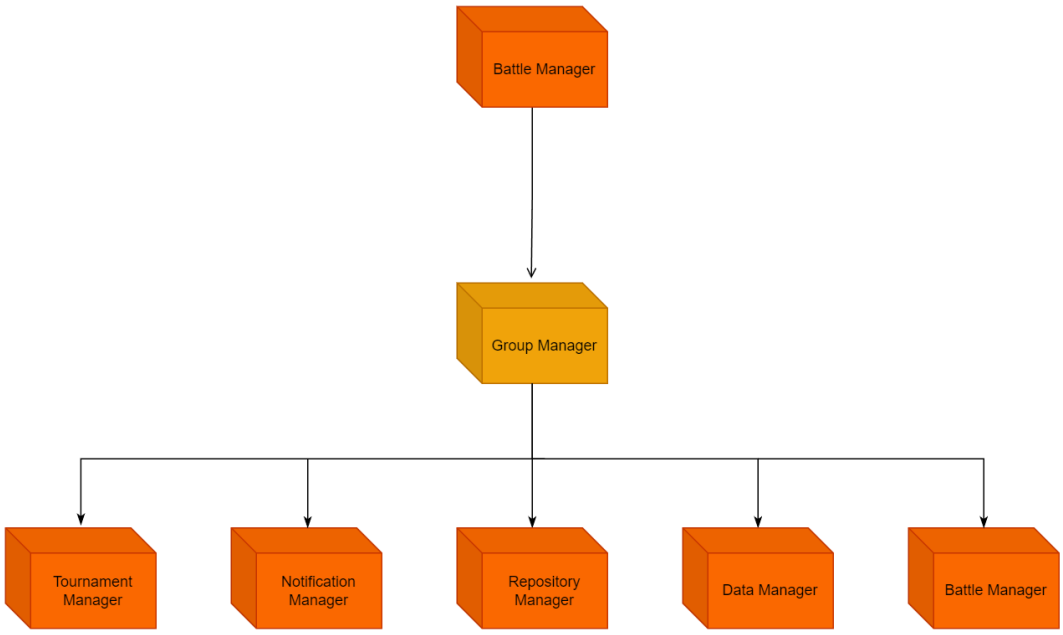


Figure 15: Integration of Group Manager

5.3 System testing

Once the individual testing for each component is completed, and it's verified as operational, the last phase involves evaluating the system as a cohesive entity. Therefore, we need to conduct comprehensive testing on the entire system before its deployment and fix any issues that might negatively affect our functionalities or our user experience. For these reasons, we'll run specific tests to assess the complete system's performance, stability, security, and usability.
A crucial aspect of this evaluation will be the performance testing of the system, particularly under high workload conditions, to determine its capability to handle multiple requests concurrently. Furthermore, given the potential for a substantial user base, usability testing will also be of paramount importance, for example, after the application has been developed, we could ask a heterogeneous group of users to complete some surveys about its intuitiveness and user-friendliness. In the event of very negative feedback, a complete UI overhaul may be necessary.

6 Effort Spent

Student	Time for S.1	S.2	S.3	S.4	S.5
Davide Grazzani	4h	45h	6h	2h	2h
Alessandro Masini	2h	60h	2h	6h	6h

7 References

References

[1] `https://en.wikipedia.org/wiki/ArchiMate`