POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

SOFTWARE ENGINEERING II
COMPUTER SCIENCE AND ENGINEERING

Design Document
Code Kata Battle

Authors:

[redacted]

Academic Year:
2023-24

Deliverable: RASD
Title: Requirements Analysis Specification Document
Authors:
[redacted]
Version: 1.0
Date: 21-December-2023
Download page: [redacted]

Contents

1 | Introduction

1.1. Purpose

As the software industry continues to expand, it is becoming increasingly important to train developers more effectively. Traditionally, developers are taught theory and then immediately put to work on a complex project without any prior experience. Code katas are short exercises that can be completed in a few minutes and are designed to address this issue. These exercises can involve programming or simply thinking about the issues behind programming. Code katas do not have a single correct answer, but the main point is to learn something from the experience. CodeKataBattle (CKB) is a platform where educators can create tournaments with multiple battles. Each battle is a code kata that is designed to develop various skills. The idea is that students compete in these tournaments in teams to gain practical experience and improve their software development abilities.

1.2. Scope

As mentioned in the RASD, there are two main users that interact with the system: Educators and Students. Educators are responsible for creating tournaments, which consist of multiple battles grouped by context. For each battle, they must provide a code kata, the number of participants allowed, and a deadline for both registration and submission. Additionally, Educators can allow other colleagues to add new battles to the tournament. To make the tournament a competition, they must also establish the scoring criteria. Lastly, Educators will create badges which will be assigned to each student who meets the criteria indicated in the badge. Students can join one of these tournaments in groups. The platform will then notify them when the competition begins and provide them with a repository containing all the code necessary for the kata. Students must fork this repository and then start working on the solution. They must also set up a GitHub action so that every push they make to the repository during the competition is counted towards their score. Finally, after the deadline, the system will generate a ranking based on their best score.score and the manually evaluation performed by the educator.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

- **Static analysis tool** :Method of debugging that is done by automatically examining the source code without having to execute the program ensuring that is compliant, safe, and secure.
- **Dynamic analysis tool** :Process of testing and evaluating a program thanks to running a test on the code.
- **OAuth Access Token** :An OAuth Access Token is a string that the OAuth client uses to make requests to the resource server.
- **WebHook** :A webhook is an HTTP-based callback function that allows lightweight, event-driven communication between 2 application programming interfaces (APIs). In this case it lets communication between Code Kata Battle platform and GitHub.
- **Consolidation** : Is a phase that occurs at the end of a tournament where the educator can decide to add a manual evaluation.

1.3.2. Acronyms

- CKB: Code kata battle
- UI: User Interface
- UML: Unified Modelling Language
- DB: Database
- API: Application Programming Interface
- DBMS: Database Management System
- RDBMS: Relational Database Management System

1.3.3. Abbreviations

- G*: goal
- WP*: world phenomena
- SP*: shared phenomena
- R*: functional requirement
- UC*: use case
- UI: user interface

1.4. Revision history

The following are the revision steps made by the team during the RASD development:

- Version 1.0 (9 December 2023);

1.5. Reference Documents

The document is based on the following materials:

- The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2022/2023;
- Slides of Software Engineering 2 course on WeBeep;

1.6. Document Structure

1. **Introduction:** it aims to give a brief description of the project. In particular it's focused on the reasons and the goals that are going to be achieved with its development, as previously explained in the RASD;
2. **Architectural Design:** : This part of the document describes at different levels of granularity how the architecture is designed;
3. **User Interface Design:** This section shows the wire frame that constitutes the platform UI;
4. **Requirements Traceability:** in this section it is explained how the goals defined in the previous document are satisfied by the interaction between the requirements and the component described in this document.
5. **Implementation, Integration and Test Plan:** this section provides a description of the implementation integration and testing details, useful for the developers and producer of the platform;
6. **Effort Spent:** shows the effort spent to write this document
7. **Bibliography:** it contains the references to any documents and software used to write this paper.

2 | Architectural Design

2.1. Overview

For CKB platform we will use a three tier architecture. A three-tier architecture is a popular software application structure that divides applications into three distinct computing levels: the presentation layer, which is the user interface; the application layer, which is where data is processed; and the data layer, which is where the data related to the application is stored and managed. The main advantage of this architecture is that each tier can be developed and updated independently by a different development team, and can be scaled up or down without affecting the other tiers.[2]

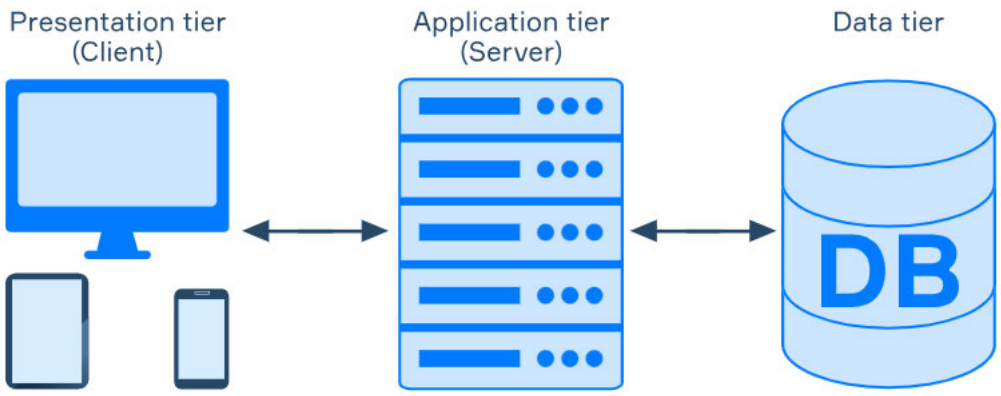

Figure 2.1: Three tier architecture
[1]

2.1.1. Distributed view

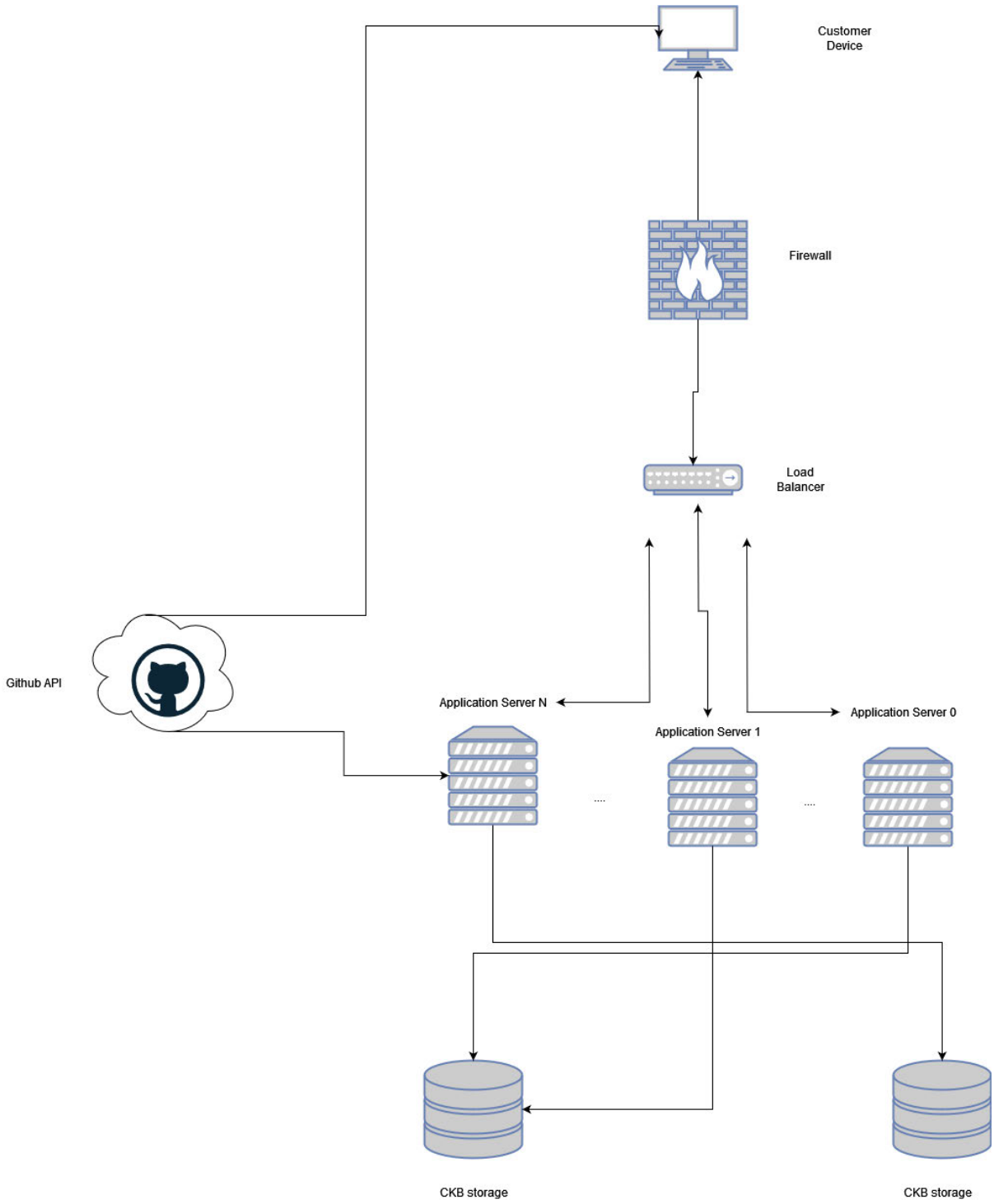

Figure 2.2: Distributed view

2.2. Component view

As mentioned for the platform we will use a three tier architecture. In figure 2.3 it is possible to find the component view diagram of the platform. The back-end will implement an API which will be used by the web app (presentation layer). As you can see from the diagram, the data layer will also interact with the back-end, as will GitHub and the static analysis tool. The main components of the back-end are:

- AuthenticationManager: provides the functionalities related to the authentication of the users to GitHub
- AccountManager: provides the functionalities related to the users general information and profile
- TeamManager: provides the functionalities related to the management of teams, necessary so students can participate in the battles.
- CompetitionStructureManager: This will implement the organizational part of the competition
- CompetitionEvaluationManager: Implements the evaluation of the competition. On contrast with the previous one, it is more dynamic because is triggered on every commit and it is also more computational intensive as it performs all the calculations needed for computing the score, the ranking and in order to assign the badges.
- DynamicAnalyser: implements the dynamic analysis tool that will run the test on the code submitted by the students. Because malicious users might try to exploit some vulnerability by providing malicious tests or code, we keep it separate from the previous.
- NotificationManager: Implements whats necessary so the platform can send notifications when needed
- Database: stores all the data of the app (Data Layer)
- API: The main purpose is to expose an API for the webApp, routing the incoming request to the appropriate internal component.

  For the front-end, the main component is:

  - WebApp: The client app with which users will interact directly. Apart from the user interface it will contain some logic necessary to interact efficiently with the back-end REST API

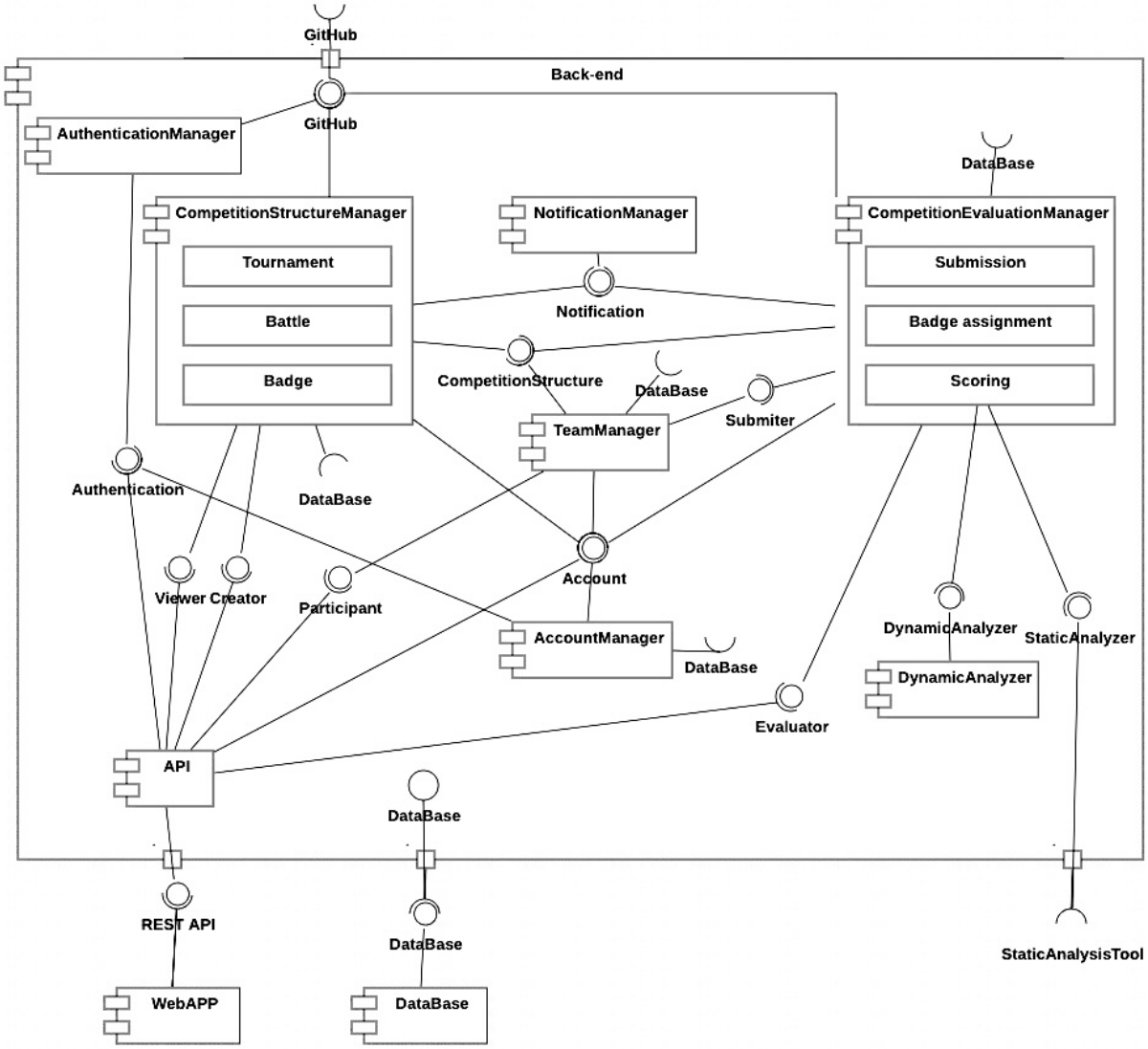

Figure 2.3: Component view

2.3. Deployment view

Here is showed the deployment view which is important because it describes the execution environment of the system but it also shows the topological distribution of the CKB application

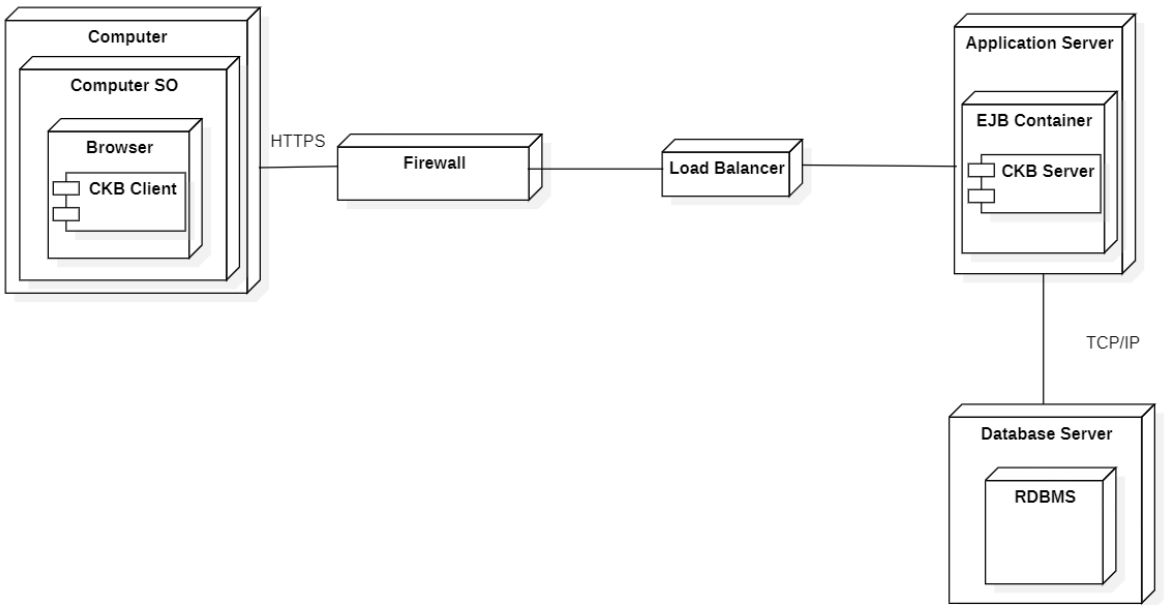

Figure 2.4: Deployment view diagram

The device SO is the one in which the browser is running from the browser it is possible to start the client-side app then every client request is processed by a firewall which is in charge of checking if some malicious request are sent to the server. A load balancer is a device useful for avoiding brute force attack to the server and, in general to not overload the server cpu capacities with a lot of request to process.The application server is able to manage the software architecture on the server-side then every change in the database is updated by the application server via TCP/IP.

2.4. Runtime view

[UC1] User Registration

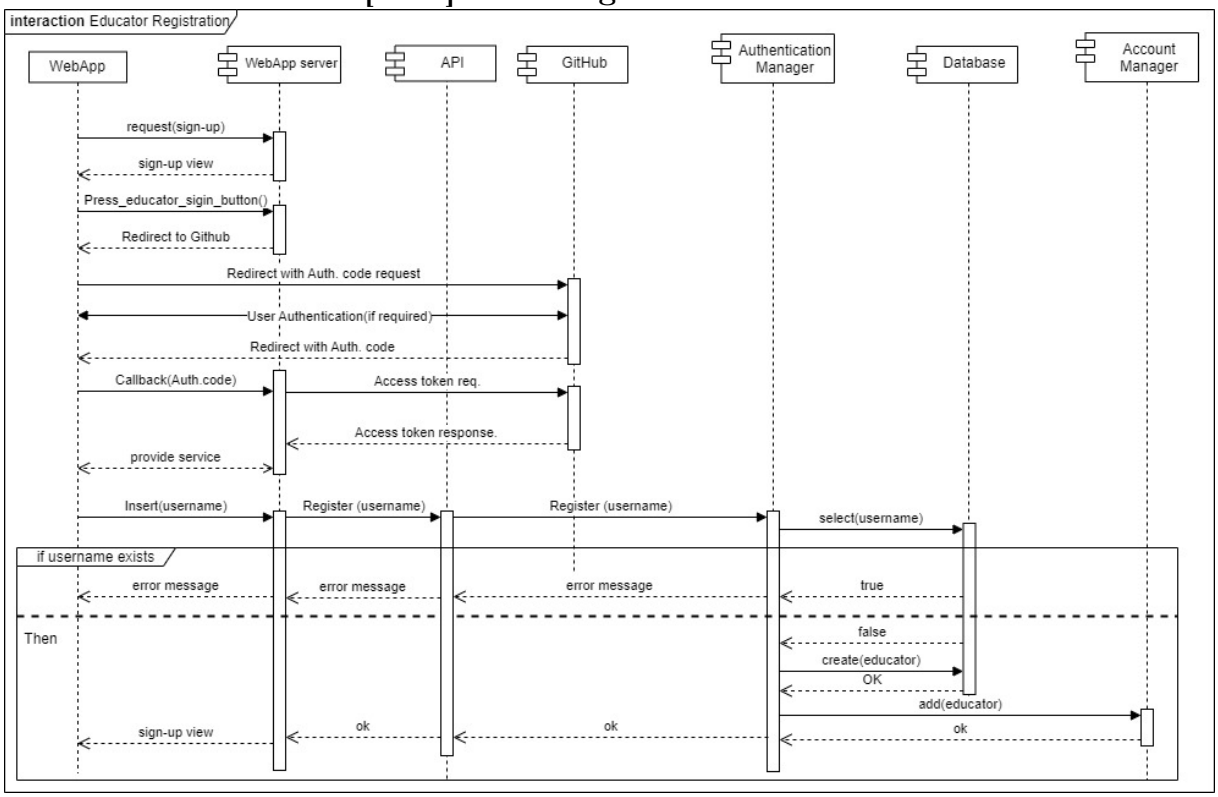

Figure 2.5: User Registration

The figure shows the process of the sign-in of the educator. Once it is sent the request to sign-up with GitHub, it will start the process of authentication, managed by the authentication manager. GitHub handles authenticated requests after the application has obtained an access token. When the process of authentication is completed, the educator fills the educator profile form with the information needed to create the profile of the user who is signing up. As soon as the form is submitted and sent to the authentication manager, it is asked to the database to check whether the username already exist in the platform. Then if the chosen username is new the database stores the information of the educator and the account manager adds the user to the table of educators.

The process is the same for the student sign-up, however it is sent a different request from the browser to the webapp server. Another difference lies in storing student information that is separate from that of the educator.

[UC2] User Login

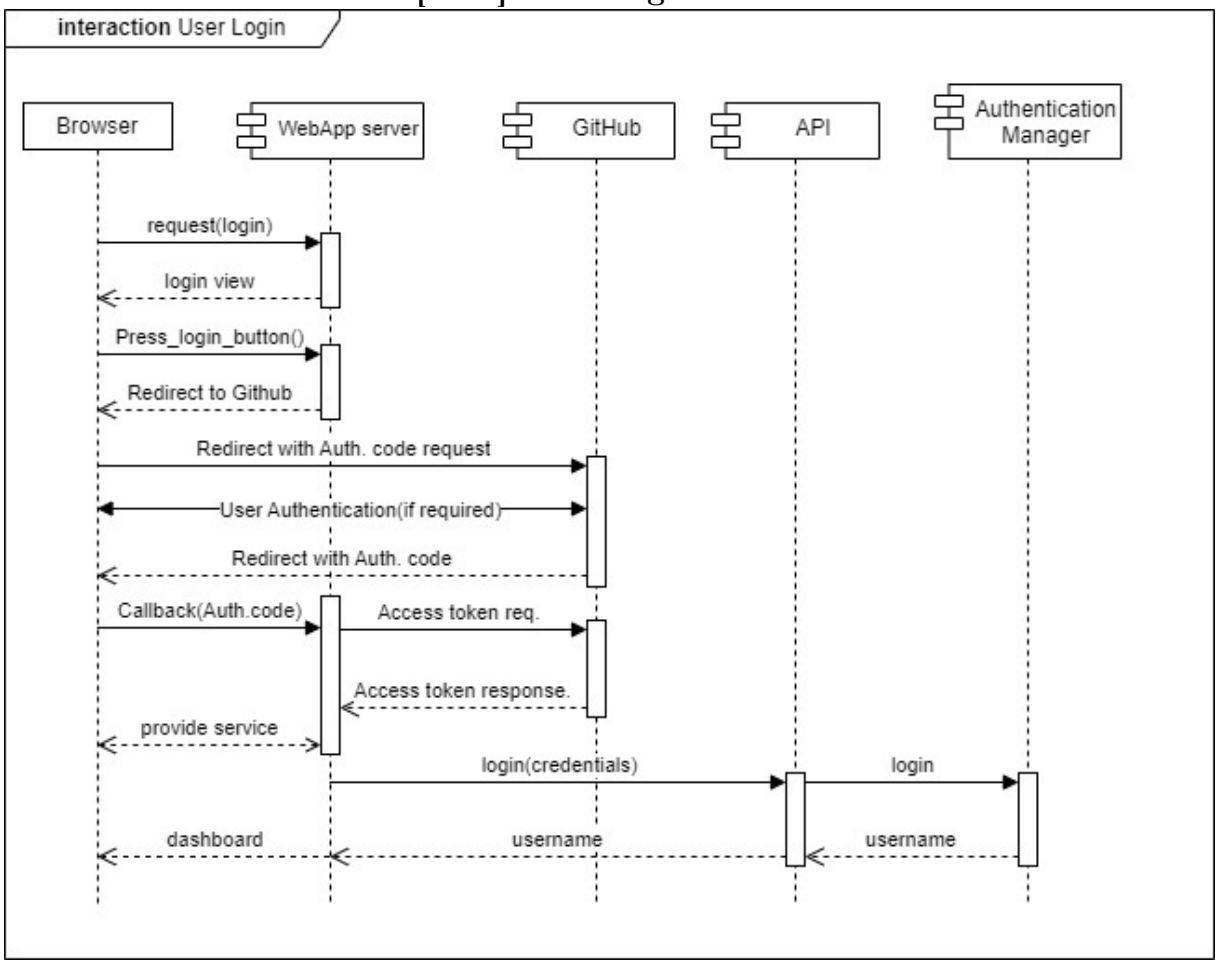

Figure 2.6: User Login

The figure shows the process of login of a user. Since the information of both students and educators have been separately stored in the database, the process of login is the same for both of the two figures. The login is managed by GitHub. Once the webapp has obtained the access token the webapp can check the credentials thanks to the authentication manager and let the user log to the platform.

[UC3] Tournament Creation

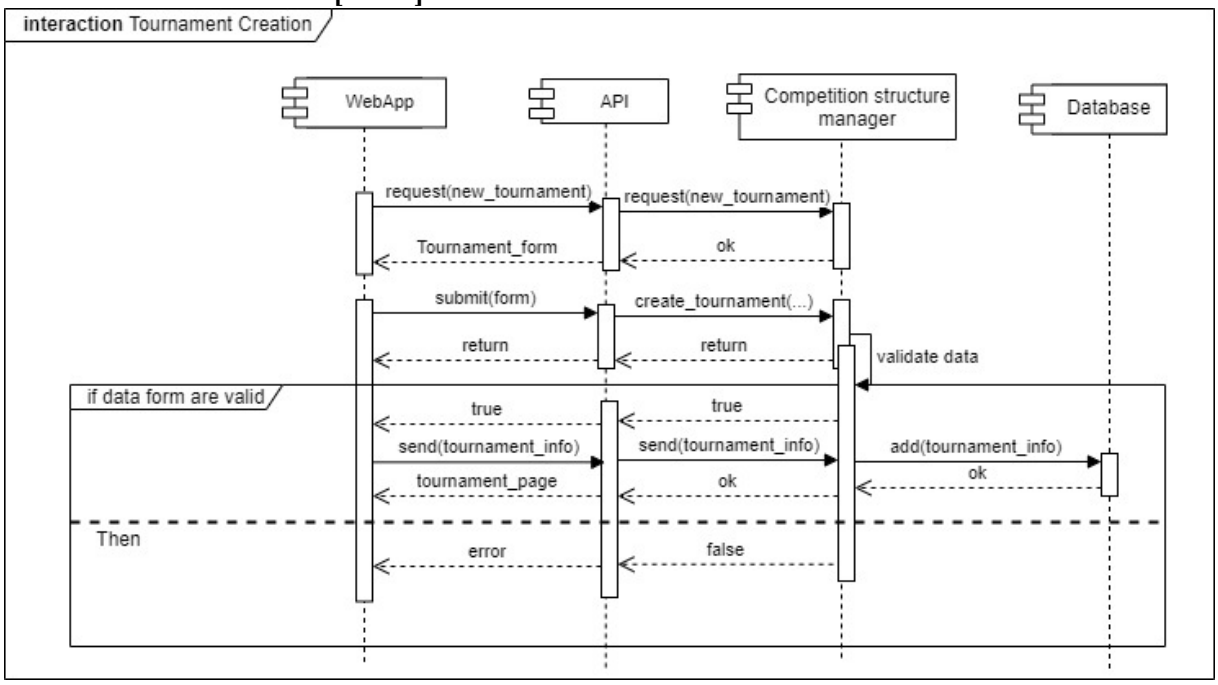

Figure 2.7: Tournament Creation

This figure shows the process of the creation of the tournament. the user send to the WebApp a request of creation of a new tournament. As a consequence the form is shown and it is filled with the tournament information. Then the competition structure manager is in charge of the validation of the information sent, if the information are valid they are inserted into the database.

[UC4] Battle Creation

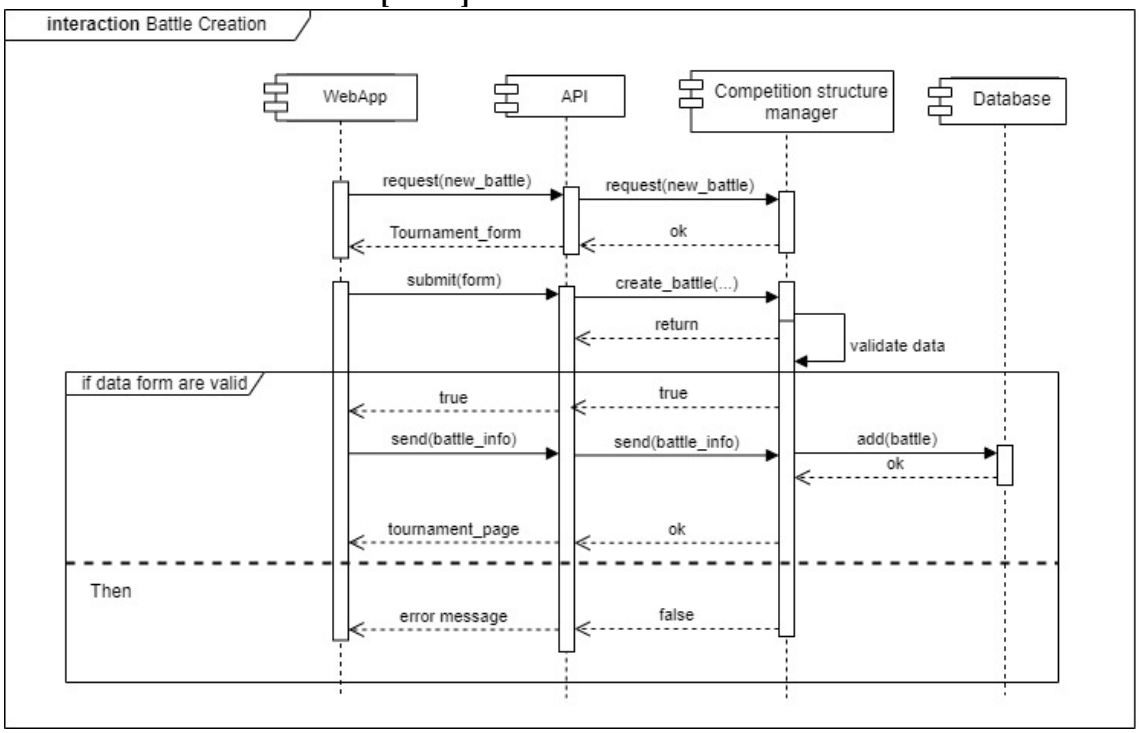

Figure 2.8: Battle Creation

The figure shows the process of the creation of a battle. As for the previous case, the user send to the WebApp a request of creation of a new battle. So the form is shown and it is filled with the battle information. Then the competition structure manager is in charge of the validation of the information sent, if the information are valid they are insert into the database, in order to create the battle.

[UC5] Add collaborator

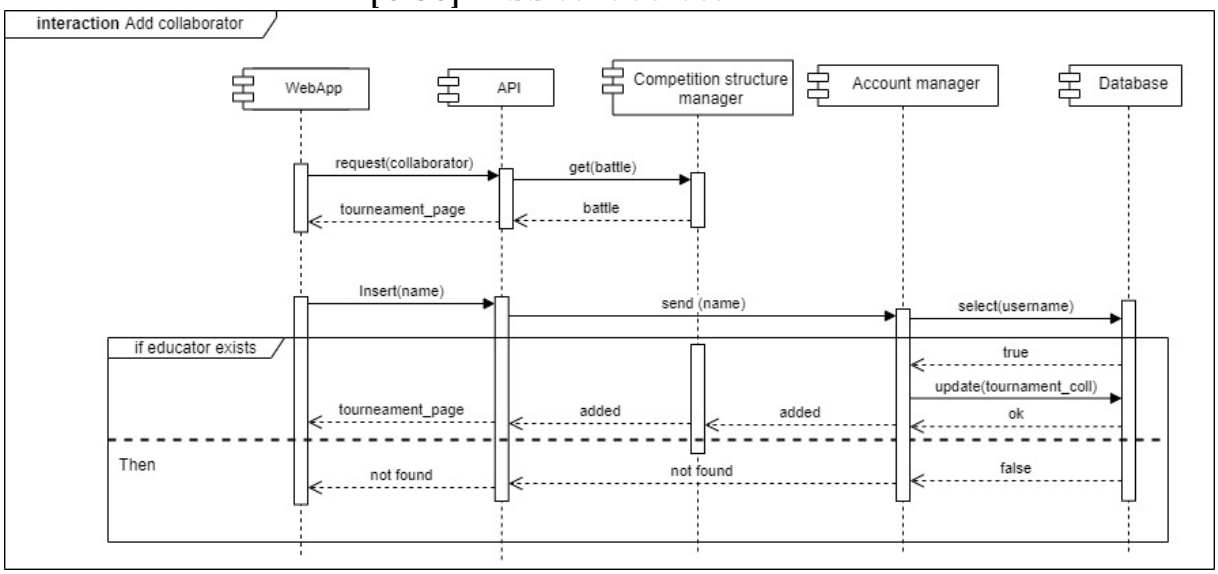

Figure 2.9: Add collaborator

The figure shows the process of adding another educator to a tournament. The request is sent form the user to the competition structure manager which is in charge of return the tournament where the educator will be added. Then the user types the name of the user to add, then the account manager will request to the database to check whether the name inserted by the user is an educator. If yes the collaborator is added to the tournament collaborators.

[UC6] Team Creation

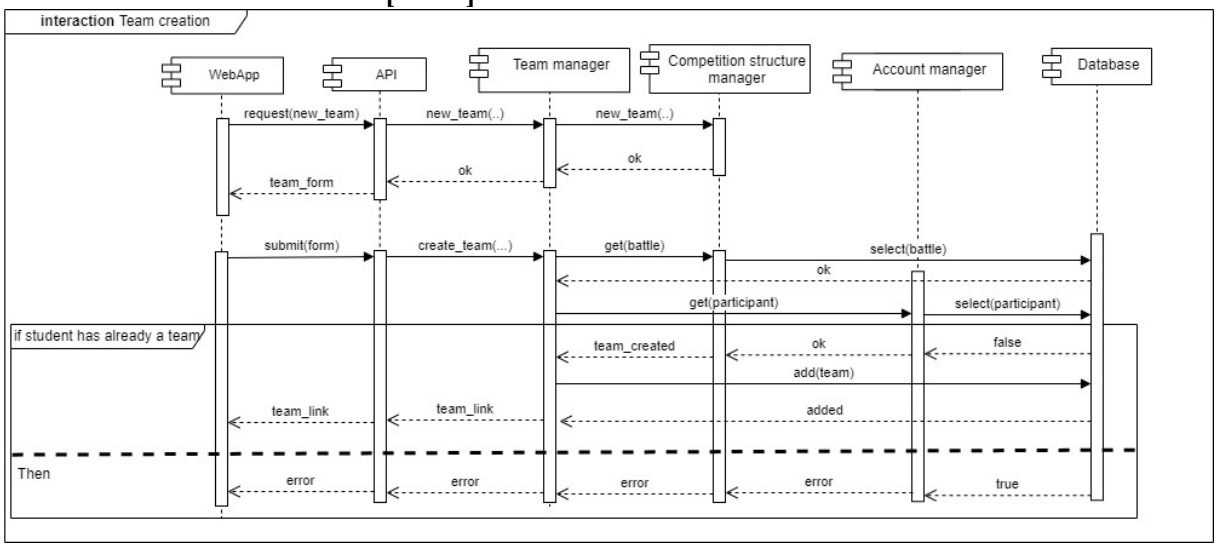

Figure 2.10: Team Creation

The figure shows how a team is created. The student request the creation of a new team managed by the team manager. When the form is submitted, the team manager request to the competition structure manager the battle in order to associate the battle with it. In the end the competition structure manager has to check whether the user who request a team is a student already enrolled in a team. Once when it has been verified then it inserts the team into the database.

[UC7] Join Team

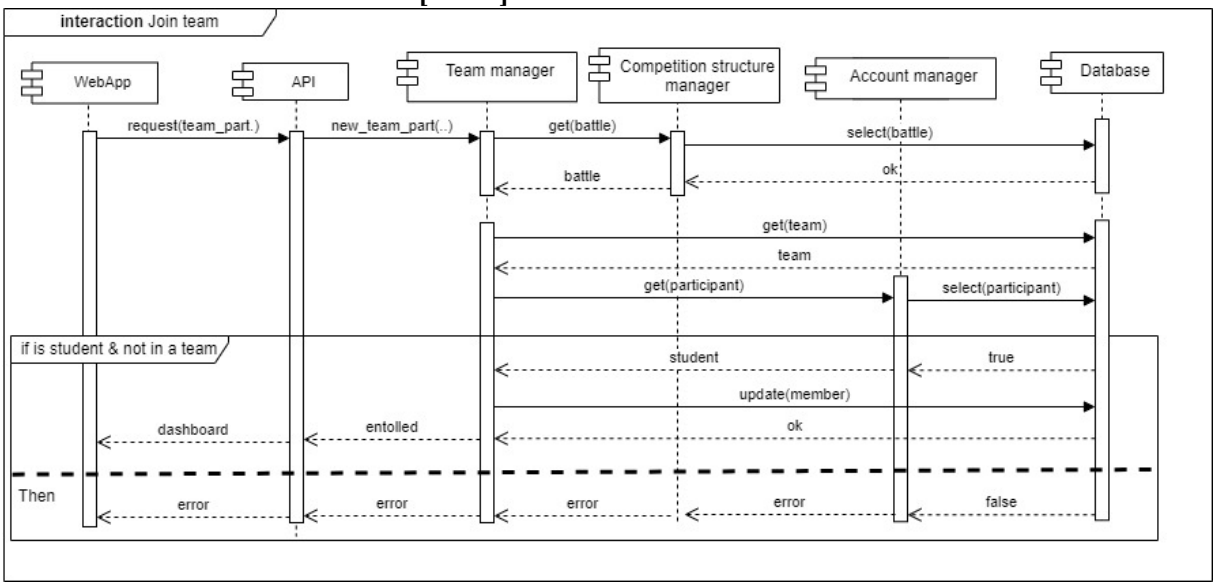

Figure 2.11: Join Team

The figure shows how a student that has already received a link can join a team. Once the link has been opened, the webapp send a request to the team manager to join the team. The manager asks the competition structure manager to get the battle information, then recover the information of the team associated to such battle. Then the manager check if the user is registered to the platform and and is not enrolled in other teams.

[UC8] Battle Begins

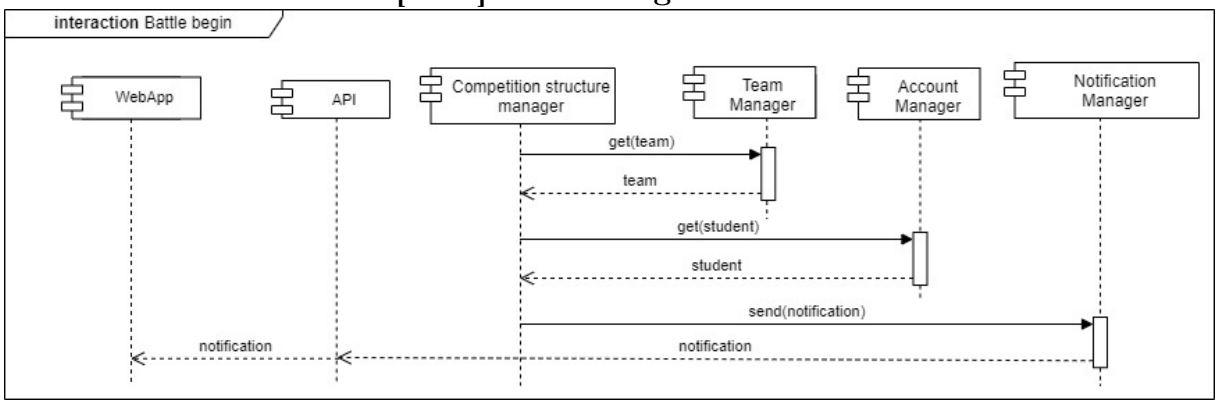

Figure 2.12: Battle Begins

The figure shows of the battle starts. As said in the RASD the battle starts as soon as the the enrollment deadline of the battle ended. Then the competition structure manager selects the students or the teams enrolled in such battle so that it can request to the notification manager to notify them.

[UC9] Code submission

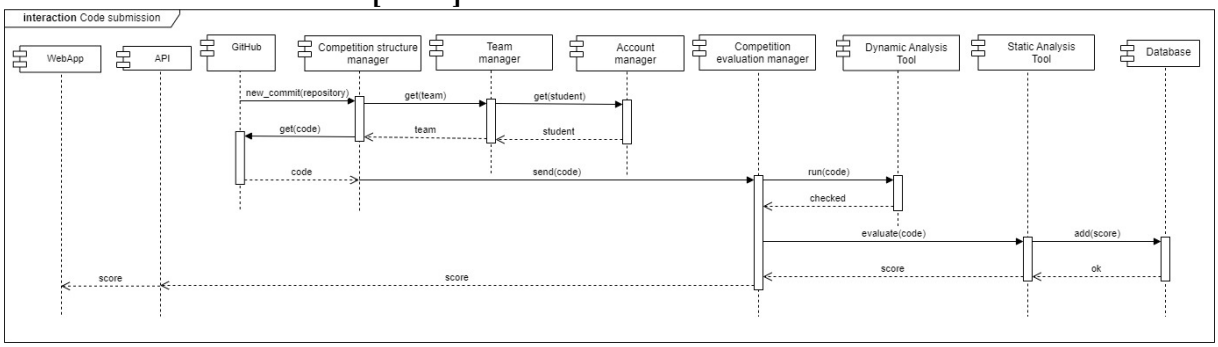

Figure 2.13: Code submission

The figure shows the process of submission of a code from a certain team. The competition structure manager is triggered by GitHub as soon as a new commit has been done, as a consequence it looks for the team and its participants, in order to associate the code to the team. Then through a callback the manager gets the code so that it can be sent to the competition evaluation manager which is in charge of request to the dynamic analysis tool to check the code. Once the code has been checked the manager request the evaluation to the static analysis tool. The resulting score will be added to the database and sent to the webapp.

[UC10] The battle ends

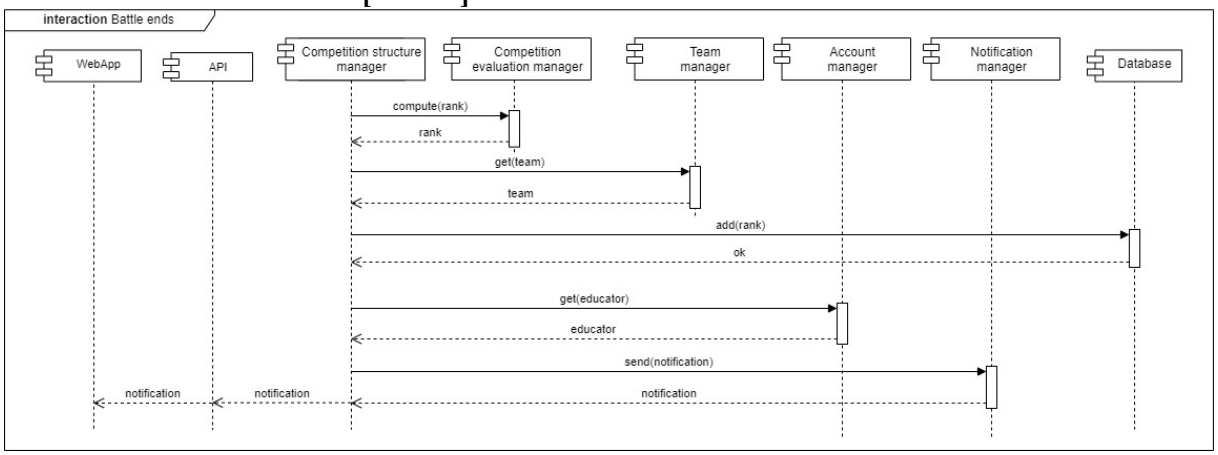

Figure 2.14: The battle ends

The figure shows how is it managed the end of a battle. The competition structure manager ask the competition evaluation manager to compute the rank of the battle, than request the team manager to get the team in order to associate the ranks to the teams and store it in the database. In the end it ask the notification manager to notifies students about the new rank.

[UC11] Code Evaluation

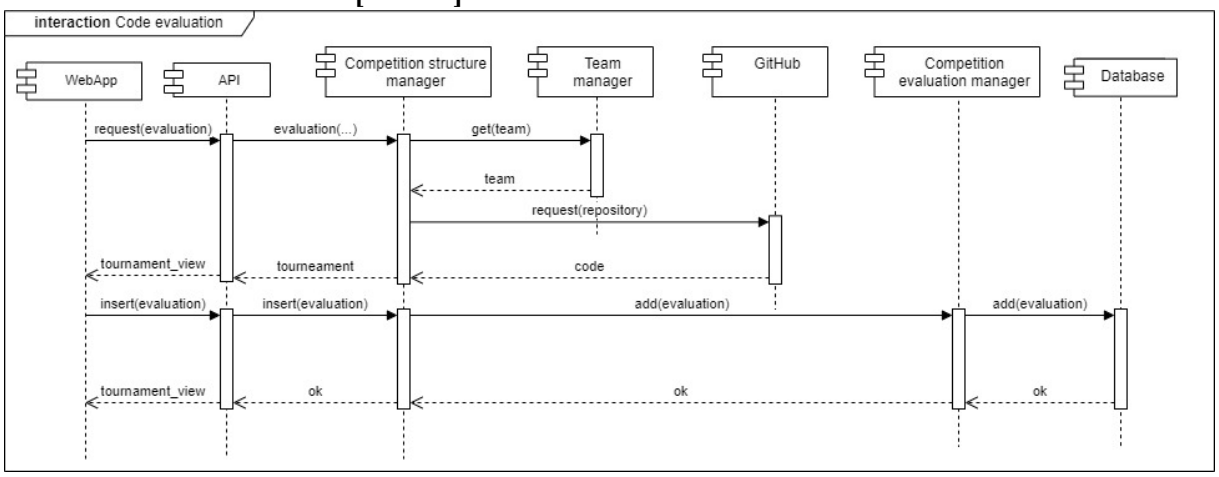

Figure 2.15: Code Evaluation

The figure shows the process of manual evaluation done by the educators at the end of the tournament,if necessary. The consolidation phase started. The webapp request the competition structure manager for a manual evaluation, as a consequence the manager request the team to the team manager and the code from GitHub which retrieves it through the repository. Then the user can inset his evaluation in the platform which will be sent to the competition structure manager and the competition evaluation manager and finally stored in the database.

[UC12] Publish Final Rank

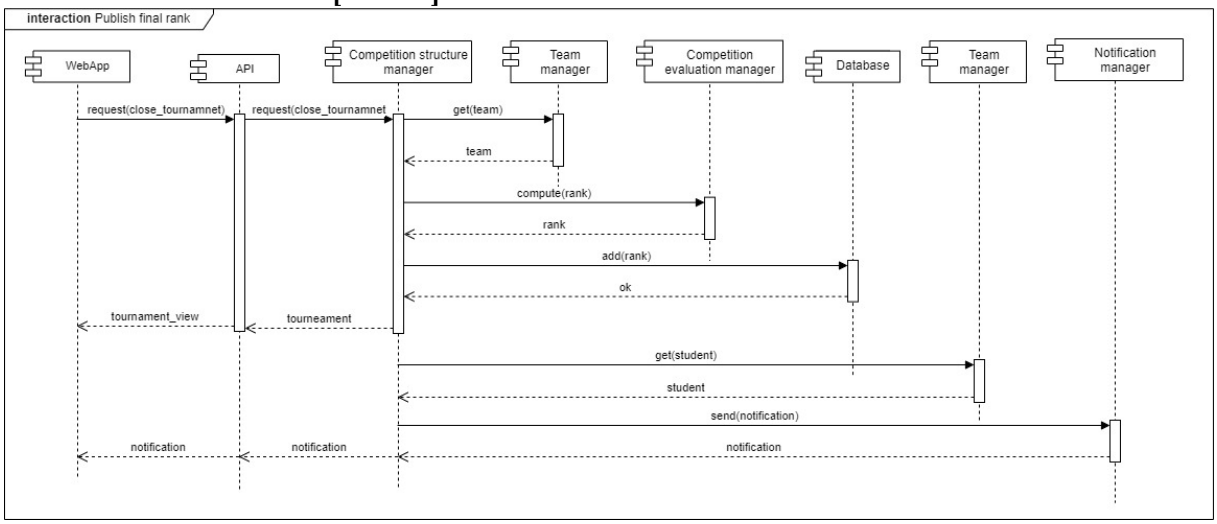

Figure 2.16: Publish Final Rank

The figure shows the process of closure of the tournament. When the educator closes the tournament, the competition structure manager takes care of recovering the name of the team and asks the competition evaluation manager to calculate the final rank and so updates the rank in the database. After that the competition structure manager is in charge of notify the student about the final rank update. Asks the account manager the name of the student who will receive the notification from the notification manager.

[UC13] See Profile

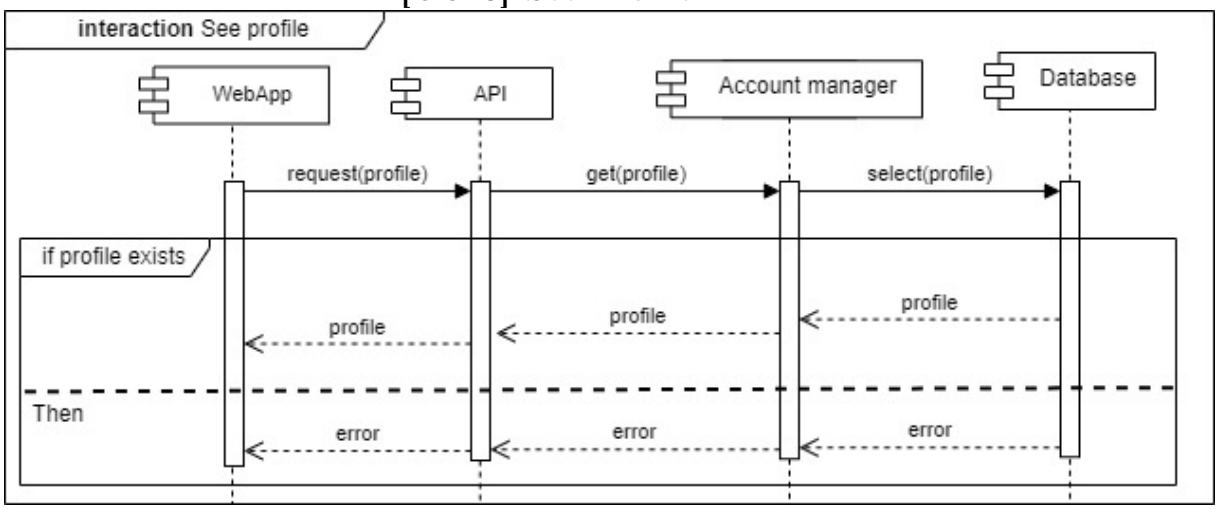

Figure 2.17: See Profile

The figure shows how the user can look for a profile of a user in the platform. As soon as the name is searched, is sent a request of getting a profile to the account manager, which ask the database to check whether the name is a user of the platform.

[UC14] Badge Creation

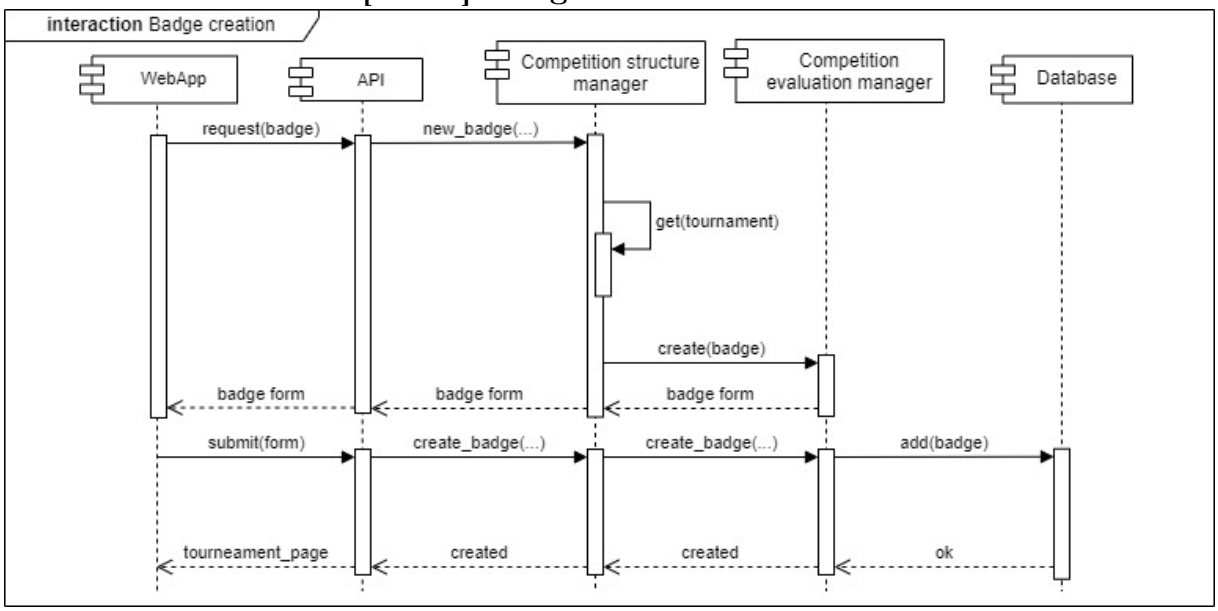

Figure 2.18: Badge Creation

The figure shows the creation of a badge. The request is sent to the competition structure manager which gets the tournament associated to the badge. When the form of the badge has been submitted, the competition structure manager forward the request of creation of the badge to the competition evaluation manager and then the badge is added to the database.

2.5. Component interfaces

WebApp server

- request(sign-up)
- press educator sigin button()
- callback(Auth.code)
- request(login)
- Press login button()

GitHub

- redirect with Auth. code request
- access token req.
- request(repository)

API

- register (username)
- login(credentials)
- login
- request(new tournament)
- submit(form)
- send(tournament info)
- request(new battle)
- send(battle info)
- request(collaborator)
- insert(name)
- request(new team)
- request(evaluation)
- insert(evaluation)
- request(close tournament)
- request(profile)
- request(badge)

AuthenticationManager

- register (username)

CompetitionStructureManager

- request(new tournament)
- send(tournament info)
- validate data
- request(new battle)
- get(battle)
- new team(...)
- request(team part.)
- new get(student)commit(repository)
- evaluation(...)
- insert(evaluation)
- request(close tournament)
- new badge(...)
- get(tournament)
- create badge(...)

CompetitionEvaluationManager

- send(code)
- compute(rank)
- add(evaluation)
- create(badge)
- create badge(...)

NotificationManager

- send(notification)

TeamManager

- new team(...)
- create team(...)
- new team part(...)

- get(team)

AccountManager

- select(username)
- add(educator)
- add(student)
- send (name)
- get(participant)
- get(student)
- get(educator)
- get(profile)

DynamicAnalyzer

- run(code)

StaticAnalysisTool

- evaluate(code)

Database

- create(educator)
- create(student)
- add(tournament info)
- add(battle)
- select(username)
- update(tournament coll)
- select(battle)
- select(participant)
- add(team)
- get(team)
- update(member)
- add(score)
- add(rank)
- add(evaluation)

- select(profile)
- add(badge)

2.6. Selected architectural styles and patterns

2.6.1. 3-tier Architecture

The architectural style that is used is the 3-tier architecture style because it provides many benefits the main one is the modularization in three independent layers:

- The **web server**:is the presentation tier and provides the user interface.This is usually a web page or a web site.The content can be static or dynamic, and is usually developed using HTML, CSS and Javascript.
- The **application server**:it is the middle tier, implementing the business logic.
- The **database server**:it is the backend tier of a web application. It runs on DBMS, such as MySQL.

2.6.2. Model View Controller Pattern

The CKB is basically a web application that allows users to have access to the CKB features.So The MVC pattern is the the best choice to implement the application:

- **Model**:it manages the data, logic and rules of the application.
- **View**:representation of information such as a chart. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- **Controller**:it accepts input and converts it into commands for Model and view.

The tasks division between these three elements provide a strong decoupling which gives some benefits such as reliability, re usability .

2.6.3. Facade Pattern

The facade pattern is a software design pattern that is useful for implementing the requests dispatching.In fact, the facade is an object that hides the more complex underlying. it improves the readability and usability of a software library by masking interaction with more complex components behind a single API. it provides a context-specific interface to more generic functionality and it serves as a launching point for a broader refactor of monolithic or tightly-coupled systems in favor of more loosely-coupled code.

2.7. Other design decisions

In this section are explained some design decisions that have been taken in order to make the system work

2.7.1. Availability

The concept of Load Balancing has been introduced to design a system which is highly available and to be able to manage an high amount of request at the same time.it's also important to have some sort of replication to avoid a single point of failure.

2.7.2. Data Storage

The data storage is managed by using a database to store the personal data of educator and students. Moreover, are used some additional data structure to have better query performances

2.7.3. Security

The security is managed by means of a firewall that is necessary to filter some cyber attacks.

3 User Interface Design

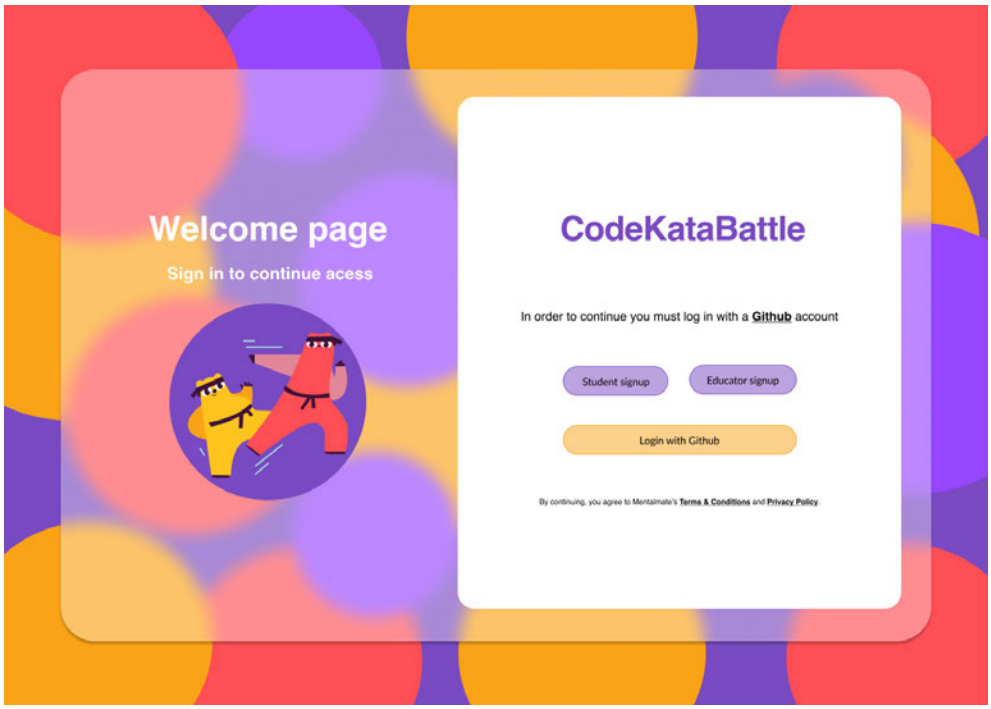

Figure 3.1: User login

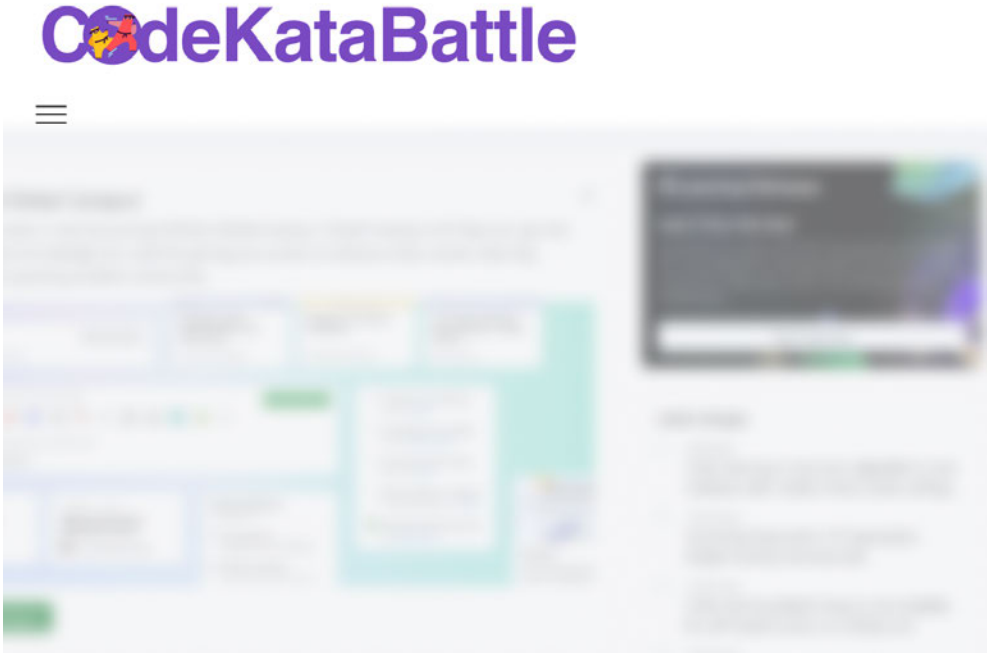

Figure 3.2: selection bar

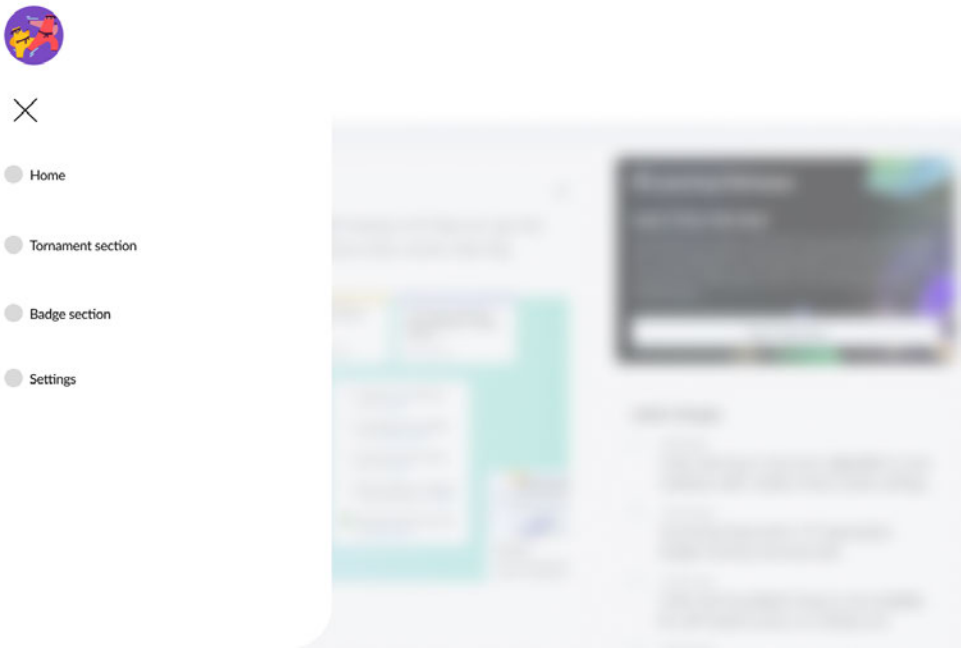

Figure 3.3: menu

4 | Requirements Traceability

In this section it is explained how the requirements defined in the RASD, map the design components explained in this document.

Requirments	[R1] The System allows users[1] to register by providing their personal information (Full Name, etc.), a valid email address and a password. [R2] The System allows registered user to log in
Components	WebApp API AuthenticationManager AccountManager Database GitHub

Requirments	**Tournament and battles management** [R3] The System allows Educators to create/modify a battle upload the code kata (description and software project, including test cases and build automation scripts) [R4] The System allows to create/modify/terminate a tournament by selecting the existing battles, setting the minimum and maximum number of students per group, the registration and final submission deadline. [R5] The System allows an educator to give or deny permission to his colleagues to modify a tournament. **Scoring and ranking** [R10] The system allows educators to define the scoring criteria for a specific battle which they have permissions to edit **Badges** [R12] Educators can create a badge and a set of rules associated with that badge
Components	WebApp API CompetitionStructureManager AuthenticationManager AccountManager Database GitHub

Requirments	**Tournament and battles management** [R6] The System must notify subscribed user about upcoming battles and deadlines.
Components	CompetitionStructureManager AccountManager AuthenticationManager Database GitHub NotificationManager

Requirments	**Student's teams** [R7] The System allows students to create a team [R8] The System allows students to invite other students into one of their teams [R9] The System allows students to join a new team which they were invited
Components	WebApp API TeamManager CompetitionStructureManager AccountManager AuthenticationManager Database GitHub

Requirments	**Scoring and ranking** [R11] The system maintains and computes the scores of each battle **Tournament and battles consolidation** [R21] The system updates the personal tournament score for each student enrolled in the tournament right after the battle ends
Components	CompetitionEvaluationManager CompetitionStructureManager TeamManager AccountManager Database

Requirments	**Badges** [R13] The system assigns the badges that are created by educators as a reward for the rules they fulfill
Components	CompetitionEvaluationManager CompetitionStructureManager AccountManager Database

Requirments	**Badges** [R14] The system shows the badges that are assigned to students
Components	WebApp API AccountManager AuthenticationManager Database GitHub

Requirments	**Tournament and battles participation** [R15] The System creates a repository on GitHub containing the code kata right after the registration deadline [R16] The system sends the link to all the enrolled students after creating the repository with the code kata
Components	CompetitionStructureManager AccountManager TeamManager Database GitHub NotificationManager[2]

Requirments	**Tournament and battles participation** [R17] The System receives notifications from GitHub regarding the students registered repositories commits [R18] The System pulls the repository after receiving a notification for that repository before the deadline of that battle
Components	CompetitionEvaluationManager CompetitionStructureManager AccountManager TeamManager Database GitHub

Requirments	**Tournament and battles participation** [R19] The System runs the appropriate test on the new code after every pull of the repository [R20] The System calculate and update the team's score for that battle after rerunning the tests
Components	CompetitionEvaluationManager DynamicAnalyzer CompetitionStructureManager TeamManager Database StaticAnalysisTool

Requirments	**Tournament and battles consolidation** [R22] The system allows educators to manually evaluate the code after the deadline [R23] The system allows educators to finish the consolidation stage after completely performing the manual evaluation
Components	WebApp API AccountManager AuthenticationManager CompetitionEvaluationManager CompetitionStructureManager TeamManager Database GitHub

Requirments	**Tournament and battles consolidation** [R24] The system computes the final ranking of the tournament immediately after consolidation finishes [R25] The system sends a notification about the tournament's termination to students
Components	AccountManager AuthenticationManager CompetitionEvaluationManager CompetitionStructureManager TeamManager Database NotificationManager[3]

5 | Implementation, Integration and Test Plan

5.1. Overview

In this chapter is explained how the platform that has been described will be implemented and tested. The aim of testing is to find the majority of the the bugs in the code that the working team has been generated. Moreover a detailed description of how components inside the code are integrated will be provided (chapter 5.3).While instead in chapter 5.2 there it's presented a description of which are the most important implementation strategy used for making the project.

5.2. Implementation Plan

The aim of this section is to describe the implementation strategies that will be used to implement, integrate and test the different components. The intention is to combine the pros of the bottom-up and threads strategies.

Using a thread strategy is functional because you can make progresses visible for users and other stakeholders.it's possible to use less drivers than expected but the integration progress is more complex.

The Top-down methodology will be used this way a basic sketch will be designed and then more complex functionalities will be added as thread unit when they are validated.

This implementation strategy allows different teams to work in parallel by accomplishing different task on their own and then every unit that has been validated will be added to the whole software architecture.

5.2.1. Features identification

The features to implement are described starting from the requirements. Some requirements need the implementation of new components while instead others require only some small changes. Here a small recap of the most important ones.

[F1] Sign-in and sign-up

This two features are really simple to test and the test methodology is equal for both operations it's important to distinguish the two figure of educator and student because they have different privileges and different types of interaction.

[F2] Creation of a Tournament and a Battle

This feature is a core feature which has to be tested right after the sign-up and sign-in feature. it enables the educator to create a tournament and a battle therefore it enables the other characteristics that is possible to use inside the battle.

[F3] Team Creation:

this functionality gives the possibility to a student to create its own team and this gives also the possibility to a student to enroll in a tournament and on the other hand to be able to unlock some other features such as upload the team code and so on.

[F4] Battle Management

For what concerns the battle management this feature groups a lot of functionalities such as verifying if the battle is ended eventually additional evaluation are needed and a notification is sent when the battle has ended so the core functionalities need to be tested as specified in previous part

[F5] Code evaluation

The code evaluation is a feature in which the system send the code uploaded by the students to the appropriate tools and this component send the score to the platform.

[F6] Tournament Consolidation

The tournament consolidation is a feature that enables the educator to close the tournament and in the meanwhile after the end of the tournament a notification in send to students involved in the tournament

[F7] Badge Creation and assignment

This functionality is the embodiment of the system gamification so it is a core characteristic of the system. Moreover it involves both user kind of users the educator who creates a badge and the student who has it assigned. So it is compulsory to test it

The majority the features that are listed require the interaction between application between client and server of the application. So the dispatcher has to be developed at the very beginning,

5.3. Component Integration and Testing

In this section it is detailed which component is implemented in every stage of the development process and how the different components are implemented and tested

[F1] Sign-in and sign-up

The only part that need to be tested is the interaction of the system with the interface with github because the authentication API provided is considered reliable by definition.

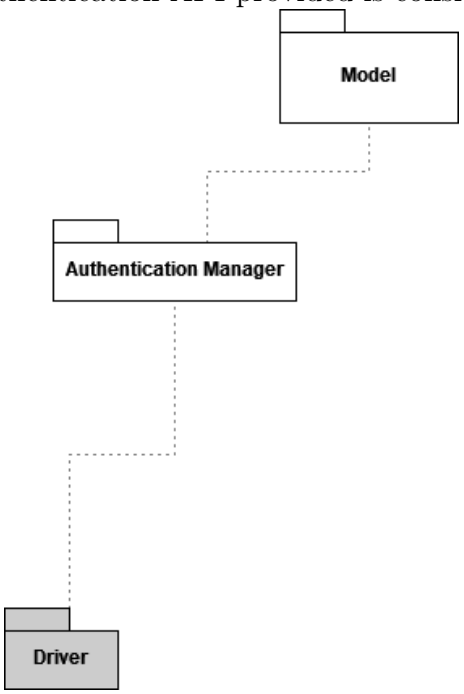

Figure 5.1: Sign in and Sign up developing

[F2] Creation of a Tournament and a Battle

Here are added new features concerning the battles and the tournaments creation so new components and mangers need to be tested.

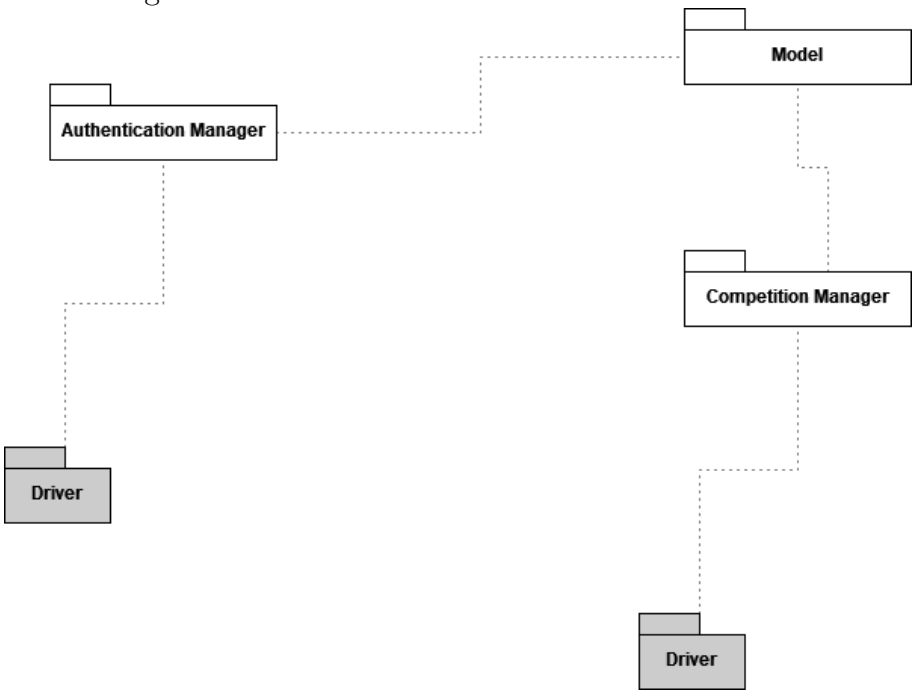

Figure 5.2: Creation of a Tournament and a Battle

[F3] Team Creation

Here it is explained how a team of student is created and for doing this some components are added to the previous diagram.

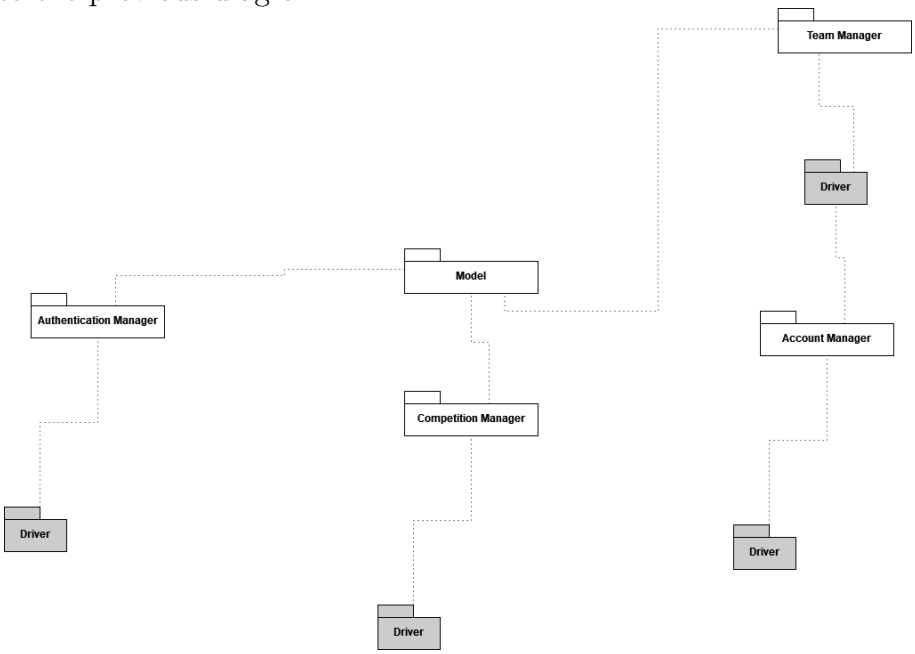

Figure 5.3: Team Creation

[**F4**] **Battle Management**

The battle management is managed by the Competition Manager that has been tested in a previous Diagram.The new component that has been tested is the Notification Manager important for sending notification to the different users.

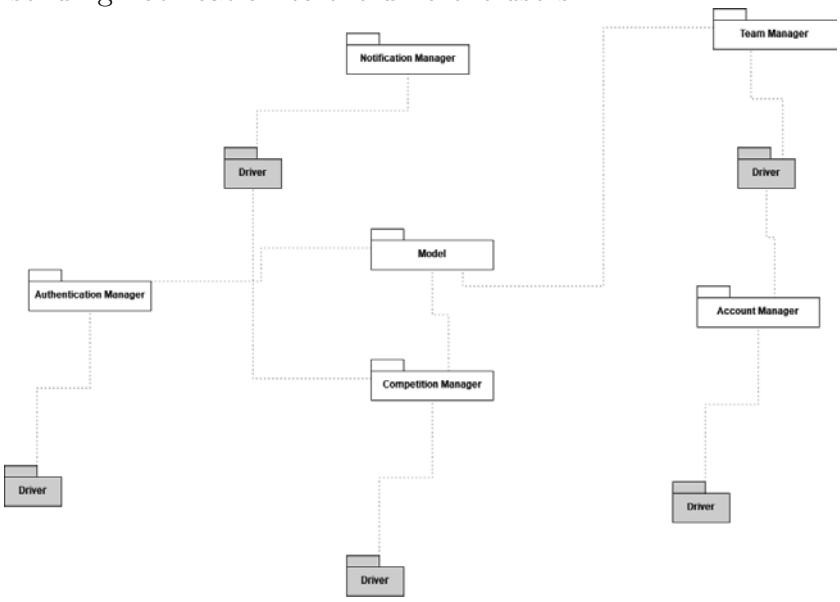

Figure 5.4: Battle Management

[**F5**] **Code Evaluation**

Since the approach that has been used is the bottom-up with the combination of thread strategy the components has been added incrementally. The last component that has to be tested is the Competition Evaluation Manager which is necessary for the whole process of code evaluation and scoring but also for the submission of the codes and badge assignment.

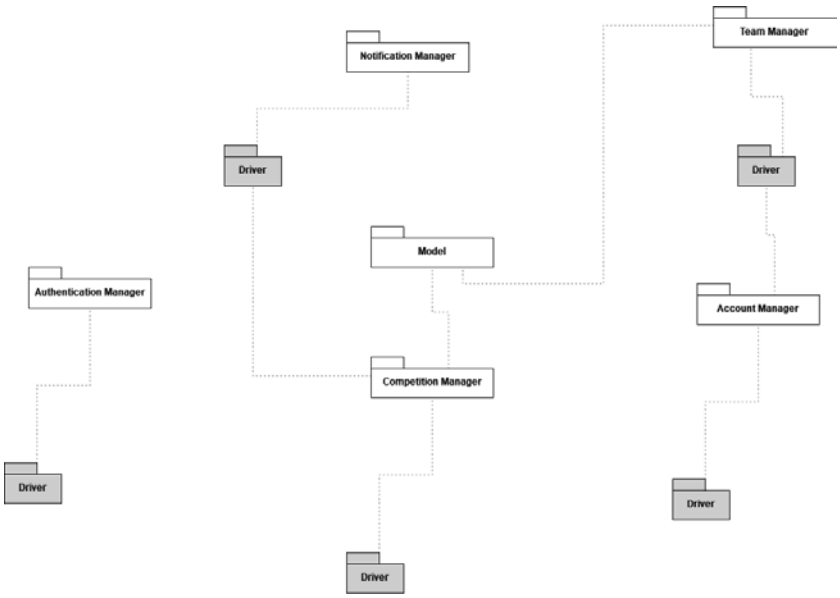

Figure 5.5: Code Evaluation

[F6] Tournament Consolidation

For what concerns this feature all the component has been tested in the previous diagrams.The only thing that to be tested is the piece of code in the Competition Manager that is in charge of consolidating the tournament.

[F7] Badge Creation and assignment

The same things must be asserted for the badge creation and assignment but what has to be tested are the pieces of code that are in charge of doing this things in the respective managers

5.4. System testing

CKB platform has to be tested for being sure that what the different teams coded has the correct functionalities which are consistent with what they expected to obtain.Therefore, During the development phase, each component that has been created has to be tested but not all the implemented component can be tested separately from the whole architecture so they need some stub and drivers which are helpful in replacing the missing ones.If a single thread unit has been tested and validated then it could be added to the software architecture.When the whole architecture has been coded then it could be tested entirely. The main purpose of this test is to verify if the system fulfills the functional and non-functional requirements that have been specified in the RASD document.In this process every actor that is involved in the software development is necessary so not only developers have to test the software but also the other stakeholders can contribute to the testing of the platform. The steps that will be followed during the testing are the following:

- **Functional Testing**:For performing this kind of test it is important to verify if the functional requirements are fulfilled.The most effecting way to achieve this test is to run the software as described in the use cases in the RASD document and verify if they are fulfilled
- **Performance Testing**:The main objective of this type of test is to detect bottlenecks which could affect response time, utilization, throughput, you can also detect inefficient algorithms hardware/network issues or it's possible to find out optimization possibilities.To perform this type of verification it's necessary to load the system with the expected workload and also measure and compare the performance and identify optimization possibilities
- **Usability Testing**: it is a method of testing the features of a website, app, or other digital product by observing real users as they attempt to complete tasks on it.
- **Load Testing**:you can expose your system to some bugs such as memory leaks,buffer overflow or memory mismanagement.This type of test is useful for identifying the upper bound of components.You can also compare different architectural options.To perform this test it's compulsory to test the system with increasing workload until it can support it for a long period of time that is established before starting the test.
- **Stress Testing**: This test aim at verifying if the system recovers gracefully after failure. you can try increasing the system resources of you can reduce the them.

5.5. Additional specifications on testing

During the system development it is also important to have continuous feedback from user and stakeholders. This should happen every time a new characteristics is implemented.On the other hand during the alpha test, it is important to get the level of satisfaction of the people that are chosen for this phase for obtaining this is helpful to have the opinion of other working figure as an example psychologist, which could choose the correct question to dispense to the tester involved. The alpha test is also of a great importance for finding out malfunctions before the beta testing. Beta testing is useful for real users to use a product in a production environment to uncover any bugs or issues before a general release. The testing is important also when the application will be released because some logs should be sent to the developers that have to use them to debug

6 | Effort Spent

In this part there is an overview of the time effort spent by each member of this team. Everyone have spent some time writing each section of this document and here its visible the amount of time.

- [redacted]

chapter	Effort(In hours)
1	2
2	10
3	3
4	7
5	10
appendix	0

- [redacted]

chapter	Effort(In hours)
1	5
2	9
3	0
4	9
5	2
appendix	5

- [redacted]

chapter	Effort(In hours)
1	3
2	20
3	0
4	4
5	2
appendix	1

Bibliography

[1] Hyperskill. Three-tier architecture. URL `https://hyperskill.org/learn/step/25083`. Last accessed 21 Dec 2023.

[2] IBM. Three tier architecture. URL `https://www.ibm.com/topics/three-tier-architecture`. Last accessed 21 Dec 2023.

A | Appendix A

For creating the rules for the badges in a simple and extendable way, we created the following grammar:

A.1. Grammar

$< \{E, T\}, \{and, or, (,), >, \geq, <, \leq, =, -, boolean\_var, numerical\_var,$

$NUMBER\}, E, P >$

where P is:

$E \rightarrow E\ (and|or)\ T \mid T$

$T \rightarrow (E) \mid (not)?\ boolean_var \mid numerical_var\ (>|\geq \mid < \mid \leq \mid =)$NUMBER

A.1.1. Examples

Example of *boolean_var*: PARTICIPATED_IN_ALL_BATTLES

Example of *numerical_var*: NUMBER_OF_COMMITS

Rule example:

PARTICIPATED_IN_ALL_BATTLES and NUMBER_OF_COMMITS > 0

A.1.2. Parsing and evaluation

First upon creation of the badge, the rule will be parsed to check if it is valid. Afterwards, when the tournament ends, it will be evaluated for every enrolled user with the appropriate values of all the ..._*var* tokens.

A.2. Disclaimer

We have not established ways to create custom variables for users because it would be difficult to do so without violating encapsulation. However, we created this flexible grammar that will allow developers to easily add new variables. We will provide users with a large number of variables which will hopefully be enough, but of course any suggestion of adding new variables will be considered.

List of Figures

List of Tables