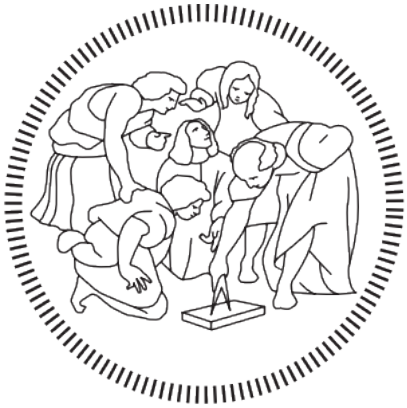


POLITECNICO
MILANO 1863

CodeKataBattle

Design Document

Software Engineering 2 project
Academic year 2023 - 2024

15 November 2023
Version 0.0

Authors:
Tommaso Pasini
Elia Pontiggia
Michelangelo Stasi

Professor:
Matteo Camilli

Revision History

Date	Revision	Notes
15/11/2023	v.0.0	Document creation
25/12/2023	v.1.0	First release

Contents

1. Introduction

1.1 Purpose

This document contains the complete design description of the CodeKataBattle platform. It provides a technical overview of the Requirement Analysis and Specification Document (RASD) for the system-to-be, describing the main architectural components, communication interfaces, and interactions. It also presents the implementation, integration, and testing plan. This document is mainly addressed to developers, testers, and project manager since it guides during the development process through an accurate vision of all parts of the software-to-be.

1.2 Scope

As explained in the RASD, CKB is a versatile platform designed to enhance students' software development skills through coding challenges. Educators can create coding battles in tournaments, where students solve programming exercises using a test-first approach. The system allows educators to upload Code Kata, set parameters like group size limits and deadlines, and define scoring criteria.

Teams, adhering to size constraints, participate in battles and can switch between different tournaments. After the registration deadline, teams access a GitHub repository for their Code Kata, working through an automated workflow.

Scoring ranges from 0 to 100, considering automated evaluations (test case pass rate, timeliness, source code quality) and optional manual evaluations by educators. Scores update in real-time as students commit code. Educators can perform manual evaluations before sharing the final rank.

Each student receives a personal score, the sum of their battle scores within a tournament. Educators can delegate battle creation and introduce gamification badges for student recognition. The platform provides a comprehensive system for coding skill improvement and competition.

The proposed system will take the form of a Web Application, accessible through any modern web browser. It will feature two distinct views, one for EDU and another for STU, each designed to implement the specified features effectively. For a comprehensive understanding of the functionalities available to end-users, please consult the RASD. The system architecture is structured into three physically separated layers, strategically

installed on different tiers to optimize performance and functionality.
These layers include:

1. **Presentation Layer:** responsible for managing the presentation logic and facilitating all interactions with end-users.
2. **Business Logic Layer:** tasked with overseeing the application functions provided by the system under development.
3. **Data Layer:** responsible for managing secure data storage and controlling access to the stored information.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Term	**Definition**
User / Actor	A person who uses the CKB platform, could be a Student or an Educator.
Educator	Identifies a person who provides instruction or education, such as a teacher.
Student	Identifies a person who is studying at school or college.
Kata	A training exercise system for karate where you repeat a form multiple times, making small improvements to each one.
Test-first approach	A software development process based on converting software requirements into test cases before creating the software, and then tracking the entire development process by repeatedly testing the software against those test cases.
Code Kata	Programming battles in which teams of students compete against each other.
Tier	Physical components or servers that constitute a system.
Layer	Logical separation of elements with associated functionality, such as presentation, business logic, and data access, each carrying out specific tasks within an application.
Frontend	Manages the visual appearance and interaction of the application with users, also called Presentation Layer.
Backend	Manages the logic and server operations, comprising the "Business Logic Layer" and the "Data Layer."

Table 1.1: Definitions

1.3.2 Acronyms

Acronyms	**Term**
CKB	CodeKataBattle
EDU	Educator
STU	Student
IDE	Integrated Development Environment
ESP	Email Service Provider
RASD	Requirement Analysis and Specification Document
UML	Unified Modeling Language
ER	Entity relationship
API	Application Programming Interface
UI	User Interface
UX	User Experience
OS	Operating System
REST	Representational State Transfer
SPA	Single Page Application
HTTPS	HyperText Transfer Protocol Secure
JSON	JavaScript Object Notation
DB	DataBase
DBMS	DataBase Management Systems
MVC	Model-View-Controller

Table 1.2: Acronyms

1.3.3 Abbreviations

Abbreviation	Term
$\mathbf{R}_i$	i-th Requirement
i.e.	in other words
e.g.	for example
iff	if and only if

Table 1.3: Abbreviations

1.4 Reference Documents

- CodeKataBattle RASD[1]
- Assignment RDD A.Y. 2023-2024[2]
- Course slides on WeeBeep[3]
- ISO/IEC/IEEE 29148 dated 2018,
  Systems and software engineering - Life cycle processes - Requirements engineering[4]

[1] https://github.com/pontig/sw-eng-2-PasiniPontiggiaStasi/blob/main/RASD/RASD.pdf
[2] https://webeep.polimi.it/mod/folder/view.php?id=219353
[3] https://webeep.polimi.it/mod/folder/view.php?id=207692
[4] https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29148:ed-2:v1:en

1.5 Document Structure

The structure of this DD document follows six main sections:

1. **Introduction:** provides a general overview of the system, its purpose, and its main components. It describes the design as a whole, highlighting key terms, and references to related documents, and giving an idea of the overall design.

2. **Architectural Design:** provides a high-level overview of how system responsibilities are distributed and assigned to subsystems. It identifies each high-level subsystem and the roles assigned to them. Additionally, it describes how these subsystems collaborate to achieve the desired functionality.

3. **User Interface Design:** illustrates the design of the UI of the system, including UX flowcharts.

4. **Requirements Traceability:** provides a cross-reference that connects system components to the requirements specified in the Requirements Analysis and Specification Document (RASD). This cross-reference is presented in a tabular format, which clearly shows which system components fulfill each of the functional requirements defined in the RASD.

5. **Implementation, Integration, and Test Plan:** details the strategy and sequence for deploying subsystems and components. It identifies the order in which sub-components are implemented, integrated, and tested. Additionally, it provides a comprehensive explanation of the methodologies and technologies to employ throughout the process.

6. **Effort Spent:** keeps track of the time spent to complete this document. The first table defines the amount of hours used by the whole team to make important decisions and to make reviews, the other tables contain the individual effort spent by each team member.

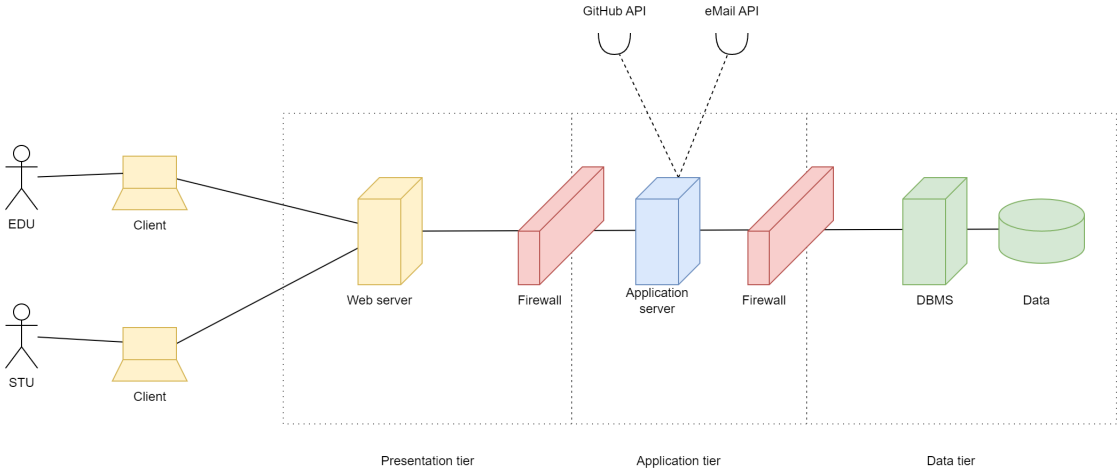
2. Architectural Design

This section provides a top-down exploration of the system's architecture. It starts with an overview of the overall architecture, accompanied by diagrams illustrating its components. Subsequently, it utilizes an ER diagram to elucidate the system's data structure and a deployment view to outline the layers and tiers involved. Sequence diagrams are employed to illustrate the main runtime perspectives, while the class diagram delineates component interfaces. The section concludes by discussing the architectural design choices and their underlying rationales.

2.1 Overview

The CKB platform software architecture is organized into three main logical layers, which are associated with an autonomous tier so that the system fits in a 3-tier architecture:

- the presentation tier, defines the user interface.
- the application tier, where data are processed.
- the data tier, where data are stored and managed.



Client => Web Application
Data => DB
Diagramma ad Alto o Basso Livello?
- Alto: aggiungerei email server, load balancer, firewall tra Client e Web server
a) 3 tier: Web e application server uniti, allora no firewall tra loro
b) 4 tier: Web e application server separati, firewall ok tra loro
- Basso: togliere tutti i firewall e li specifichiamo dopo nel deploy.

Figure 2.1: High level components diagram

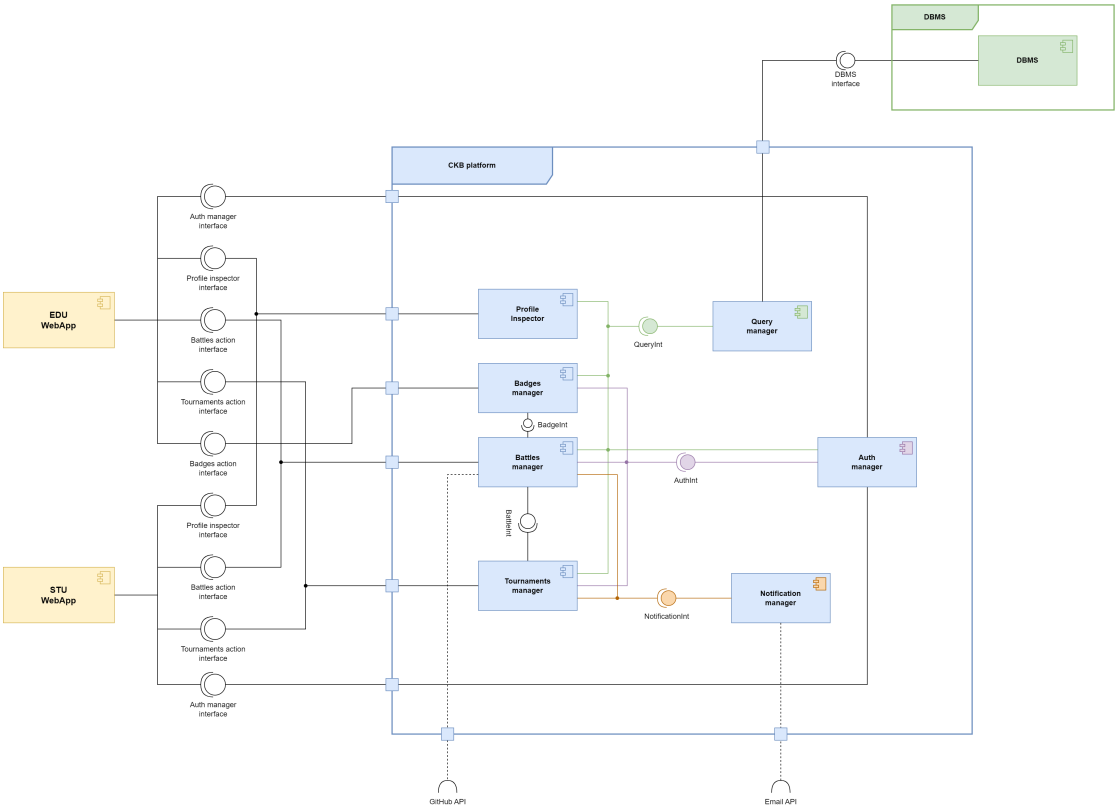
The system is accessible through a web interface and is developed as SPA. This web application type is ideal, as it facilitates extensive interaction without requiring frequent page reloads. In addition, this type of system allows for easy expansion of the services offered without major modifications to the system itself.

The system architecture is organized into separate layers, where application servers communicate with a database management system and employ APIs for data retrieval and storage. Following REST standards, the application servers are deliberately designed as stateless, managing user login sessions through caching and adhering to web application best practices.

Moreover, the system will incorporate multiple firewalls to enhance security.

2.2 Component view

The system is composed of the following components:



Badges Manager non deve collegarsi a Battles Manager, ma a Tournaments Manger
AuthInit non va collegato anche a Profile Inspector? Perchè alla fine è meglio controllare sempre se l'user può eseguire le azioni ed esiste

Figure 2.2: Component diagram

To maintain the readability of this diagram, the interfaces have been grouped according to their functionality. A complete set of endpoints is available later in the document.
Non mi è chiara la storia delle proper APIs per comunicare con la WebApp di EDU o STU

2.2.1 Client components

The client components are represented only by a single one, a user-friendly WebApp, that behaves differently depending on the type of user using it, an EDU or a STU.
The only logic incorporated is the ability to perform basic checks, such as the detection of incomplete forms or wrong data format. The WebApp is interfaced with the server components through all the APIs offered by the server, a detailed description is in the following sections.

2.2.2 Server components

Query manager

The query manager is the component that handles the queries made by the other components that need to access the database.
It is responsible for the execution of the queries and for the communication with the database.
It is interfaced with all the internal models of the system that need to access the database, i.e. all the other components of the system except for the notification manager.
It is interfaced with the database through the DBMS API, external to the system.

Auth manager

The auth manager is the component that handles the authentication of the users and the authorization of the requests made by the other components that need to access the database concerning the user who made the request.
It is interfaced with all the internal models of the system that behave differently depending on the level of the user that made the request, i.e. the badges manager, the tournament manager, and the battle manager.
It isn't interfaced with any external component.

Notification manager

The notification manager is the component that handles the need of the system to notify the users of some events, such as the start of a tournament or the end of a battle.
It is interfaced with all the internal models of the system that need to notify the users, i.e. the tournament manager and the battle manager.
It is interfaced with the Email API, external to the system.

Badges manager

The badges manager is the component that handles the gamification badges.
It allows:

- the creation of new badges;
- the assignment of badges to the STUs;
- Aggiungere badges ad un torneo, i badge sono del torneo non delle battle

Qui devo interfacciare anche con battle e user? Tommi: Secondo me no
It is interfaced with the auth manager since the creation of a badge is admissible only by the EDUs.
It is interfaced with the query manager to access the database to store the badges.
It is interfaced with the EDU WebApp through the proper APIs, external to the system.

Tournaments manager

The tournament manager is the component that handles the management of the tournaments.
It allows:

- the creation of new tournaments;
- the closure of the tournaments;
- the visualization of the tournaments;
- the exchange of admin permission between EDUs;
- the subscription of the STUs to the tournaments;
- the visualization of the STUs' score in the tournaments, and so the overall ranking;
- the visualization of the battles within a tournament.

Si interfaccia con i badge che si possono specificare per la battaglia
It is interfaced with the auth manager to allow and perform different actions depending on the level of the user that made the request.
It is interfaced with the query manager to access the database to store or retrieve data.
It is interfaced with the notification manager to notify the users about the start and the end of a tournament.
It is interfaced with the WebApp, both EDU and STU, through the proper APIs, external to the system.

Battles manager

specificare meglio => Tommi: non mi sembra male! Solo l'ultima frase non ho capito bene
The battles manager is the component that handles the management of the battles.
It allows:

- the creation of new battles;
- the subscription of the STUs to the battles;
- the visualization of the scores of the teams in the battles, and so of the ranking
- the visualization of the battles;
- the automatic evaluation of the battles;
- the eventual manual evaluation of the battles by the EDUs

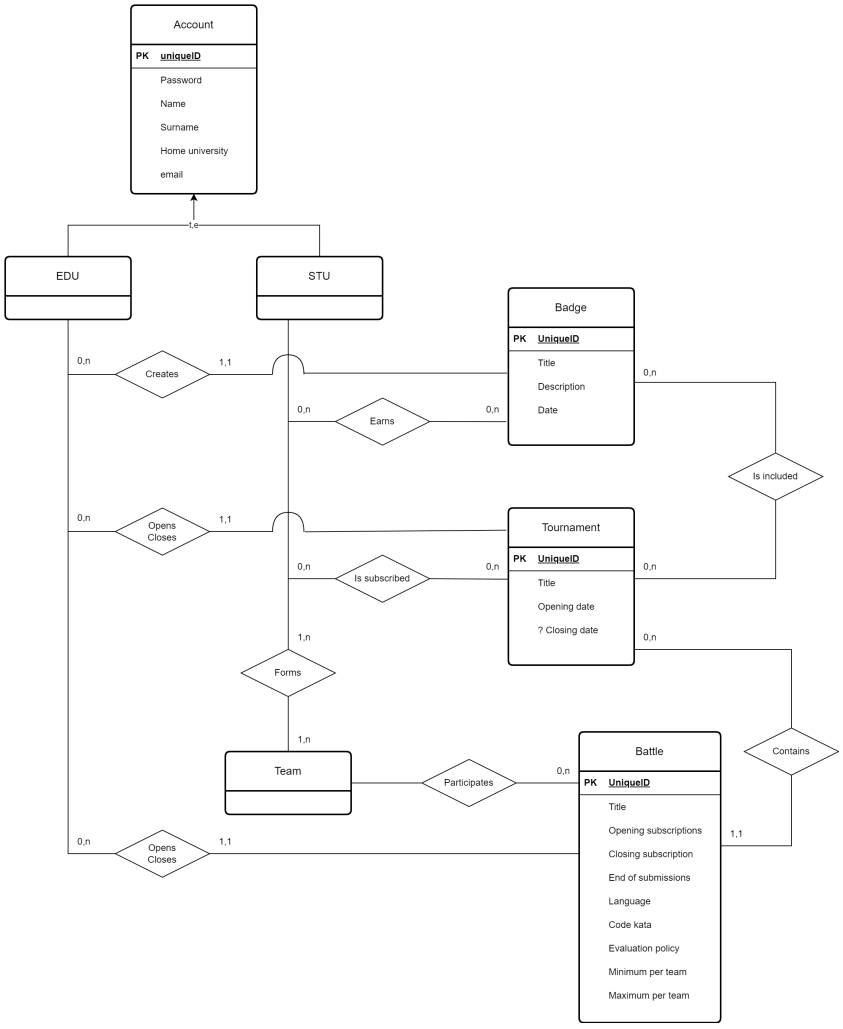
It is interfaced with the auth manager to allow and perform different actions depending on the level of the user that made the request.
It is interfaced with the query manager to access the database to store or retrieve data.
It is interfaced with the notification manager to notify the users of the opening of subscriptions, the start and the end of a battle.
It is interfaced with the WebApp, both EDU and STU and the proper APIs and with GitHub (to perform the manual evaluation), all external to the system.

Profile Inspector

The profile inspector is the component that handles the visualization of the profiles of the STUs and, consequently, of the badges that it has earned during the tournaments.
It is interfaced with the query manager to access the database to retrieve data.
It is interfaced with the WebApp, both EDU and STU, through the proper APIs, external to the system.

2.2.3 Logical description of the data

The data of the system is organized in a relational database, following the below ER diagram:



Da aggiungere tabella per le richieste di join team [IDTEAM - IDBATTLE - IDTOURNAMENT - IDSTU]

Figure 2.3: Entity-relationship diagram

Vogliamo magari scrivere il progetto logico?

2.3 Deployment view

The diagram 2.4 shows the deployment view of the system.
Our system comprises two essential components: a static web server and an application server. The static web server serves as the entry point for clients to access the SPA, while the application server furnishes the necessary APIs for the SPA's functionality. To optimize performance, these components follow distinct solutions.

The static web server is hosted on a CDN (Content Delivery Network) on the cloud, exploiting its edge location caches and reverse proxies to ensure rapid response times.
On the other hand, the application server, containing both the business logic and the data layer, is home-based on a cloud provider.
This decision offers numerous advantages over traditional in-house hosting, including:

- **Scalability and Flexibility**: The cloud infrastructure allows for the dynamic addition or removal of resources like virtual machines, performance cores, or memory as per the evolving needs. Load balancing services further enable the application server to adapt seamlessly to changes in traffic or workload.
- **Security**: Enhanced security features, such as live monitoring and firewalls, contribute to safeguarding the application server against potential data breaches, cyberattacks, and other security threats.
- **Cost-efficiency**: The cloud provider's pay-as-you-go model ensures cost efficiency by charging only for the utilized resources. This approach helps in reducing overall costs, making it a financially prudent choice.

These attributes position a cloud provider as an ideal hosting solution for large, high-traffic applications. The selected cloud provider must respect all these features to effectively meet our system requirements.

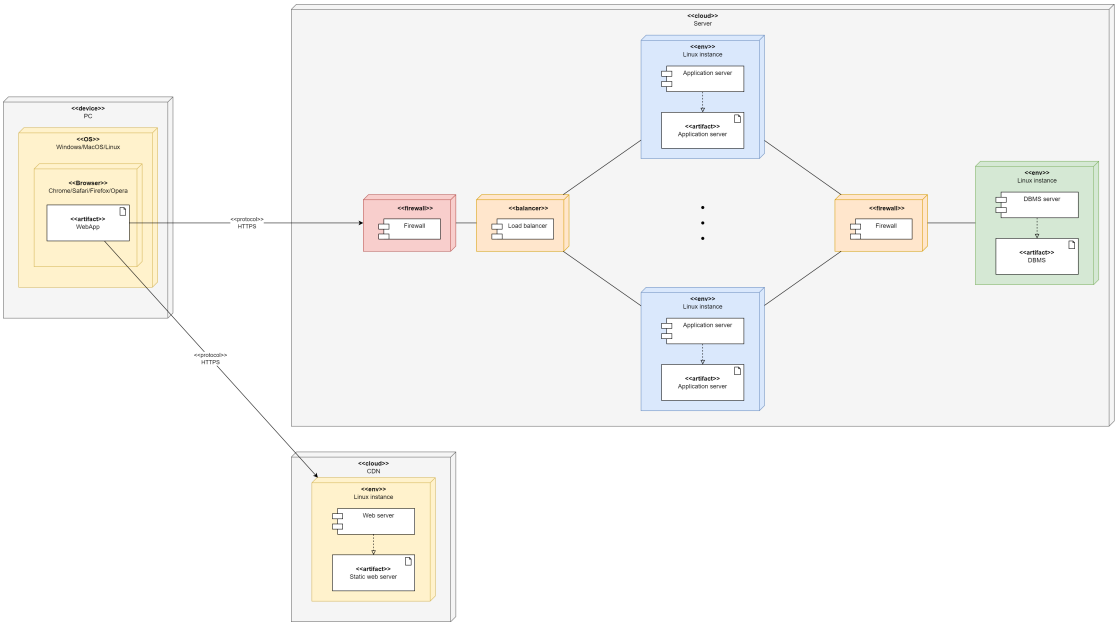


Figure 2.4: Deployment diagram

Se Web e Application Server son server diversi, quindi due tier, e il sistema diventa 4-tier al posto di 3.

3 tier: Client -> Firewall -> Load Balancer -> (CDN) Web Server -> Application server -> Firewall -> DB
4 tier: Client -> Firewall -> Load Balancer -> (CDN) Web Server -> Firewall -> Load Balancer -> Application server -> Firewall -> DB
Client -> HTTPS -> Apllicazion Server -> HTTPS / TCP-IP -> DB
Mettere la CDN in mezzo, come sulle slides di tiw

The components of the system are explained in the following:

- **PC**: Personal computer of the user, it suffices to have a working OS and a Browser installed that supports JavaScript and HTML5 to use the system.
- **CDN**: The CDN is used to host the static web server, that serves as the entry point for clients to access the SPA. It allows to download the static web pages without affecting the performance of the main application server. The SPA is static and all of its code is run on the client's machine, so there is no need for any logic to be implemented on the CDN side.
- **Cloud provider**: The cloud provider is used to host the application server. anche il web server hosta no? It allows the system to be scalable, flexible, secure, and cost-efficient.
  It is composed by:
  - **Load balancer**: The load balancer is used to distribute the traffic between the different instances of the application server. It is used to make the system scalable and flexible.
  - **Application servers**: The application servers are used to host the application server. They are in an array of instances so that the load balancer can distribute the traffic between them. The number of instances can be changed dynamically so that the system can adapt to the traffic. They are used to make the system scalable and flexible.
  - **DBMS server**: The DBMS server is used to host the database.
  - **Firewalls**: Firewalls are used to make the system secure, and are placed between the load balancer and the external world and between the application servers and the DBMS server. They provide an additional layer of security by blocking or allowing traffic based on predetermined rules. This helps to protect the system from unauthorized access or malicious attacks.

Da qui in avanti dobbiamo prima parlarne un secondo su cosa metter (secondo me la sezione *API endpoints*, visto che c'è prevalentemente server-side, dovrebbe scriverlo chi si occuperà del backend)

2.4 Runtime View

Da contorllare se i nomi dei metodi usati sono consistenti con quelli del RASD
This section describes the most important components interactions of the system.

2.4.1 User login

At the beginning the end user must log in to use the main functions of the application. The login is done by enetring the email and password. If the credentials are present in the database and correct the process will be successful and it will be able to open its dashboard, otherwise it will have to repeat the procedure. (Figure 2.5)

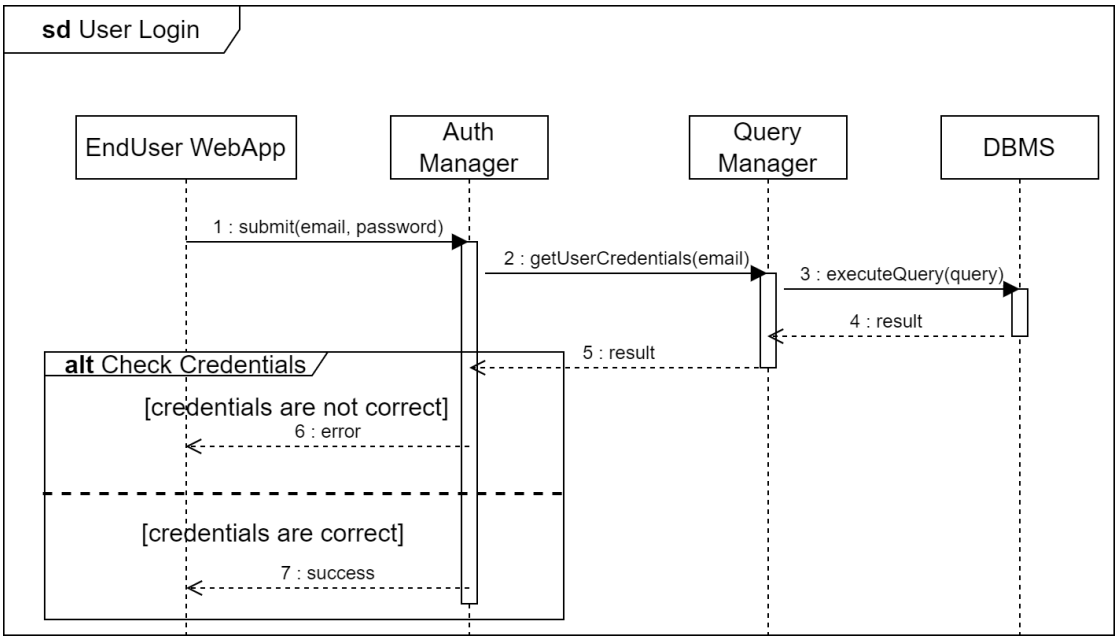


Figure 2.5: Runtime View of User Login Event

2.4.2 Create a tournament

The following sequence diagram is used to explain how to create a tournament. The end user from its device can create a tournament entering the correlated details through the related component, which are sent to the database. If the tournament has been created successfully, the correlated page is created.(Figure 2.6)

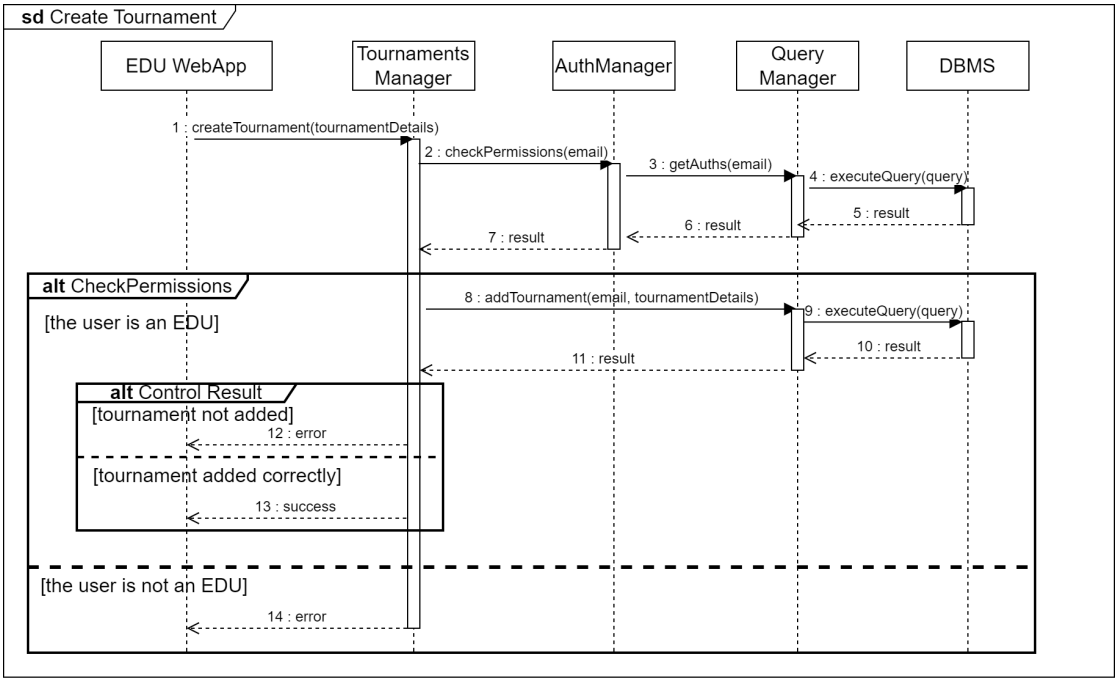


Figure 2.6: Runtime View of Create a Tournament Event

2.4.3 Create a battle

The following sequence diagram is used to explain how to create a battle within a tournament. The end user from its device can create a battle entering the correlated details through the related component. If the user is a granted educator for the current tournament, then the details of the new battle are sent to the database and if the battle has been created successfully, the correlated page is created and all the users subscribed to the correlated tournament are notified. (Figure 2.7)

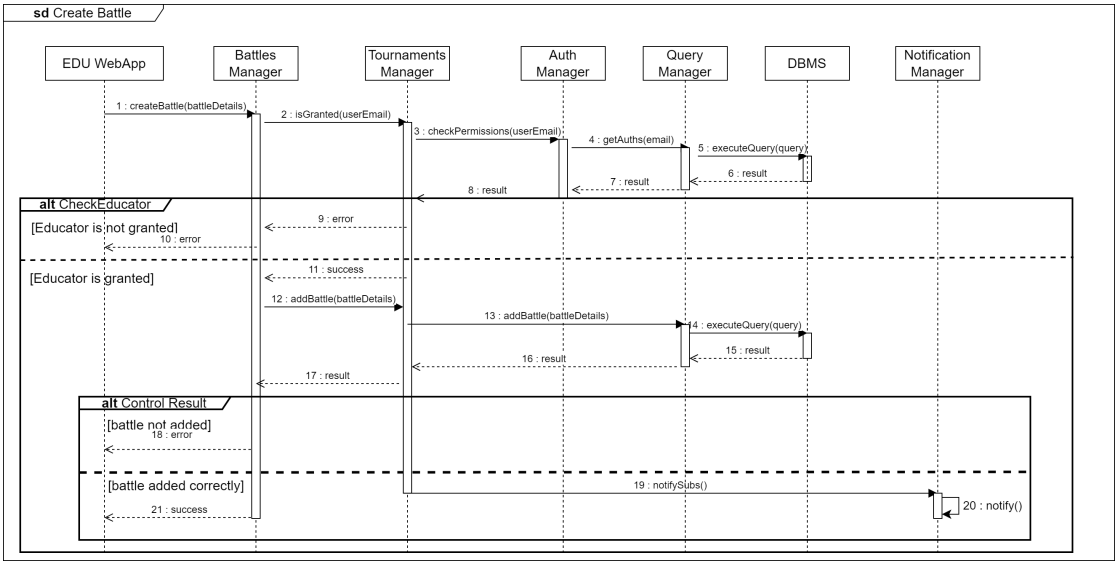


Figure 2.7: Runtime View of Create a Battle Event

2.4.4 Join a Battle Solo

In this sequence diagram is shown how an user can subscribe to a battle. By joining the battle, the system add to the database the email of the user into the subscribed email of that battle. If the user decides to invite other members, the inserted emails are notified. (Figure 2.8)

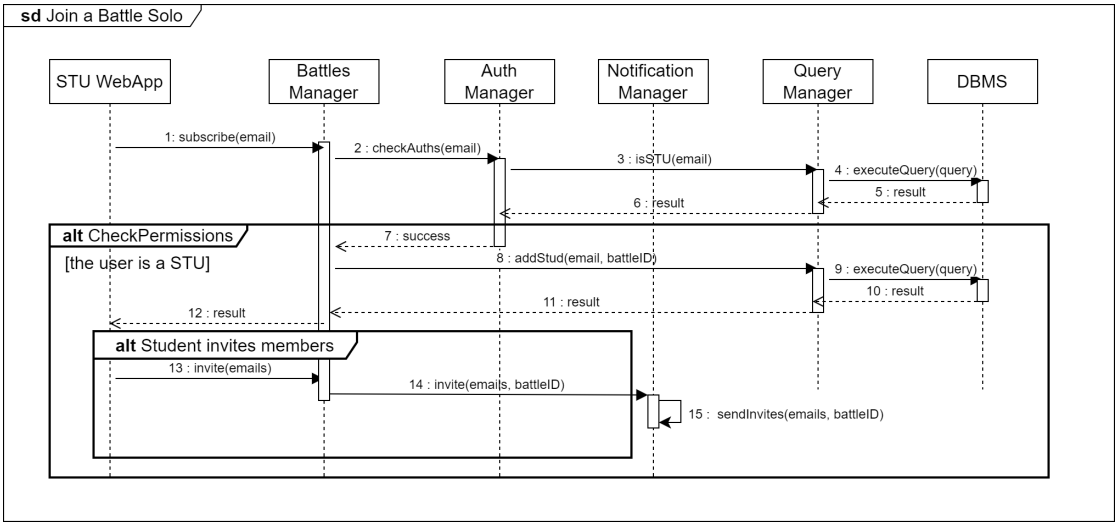


Figure 2.8: Runtime View of joining a Battle Solo

2.4.5 Upload code

In this case, the GitHub API notify the system which through the related component computes the new score and update it by sending the new score to the database. Finally, the system updates the battle ranking with the new score. (Figure 2.9)

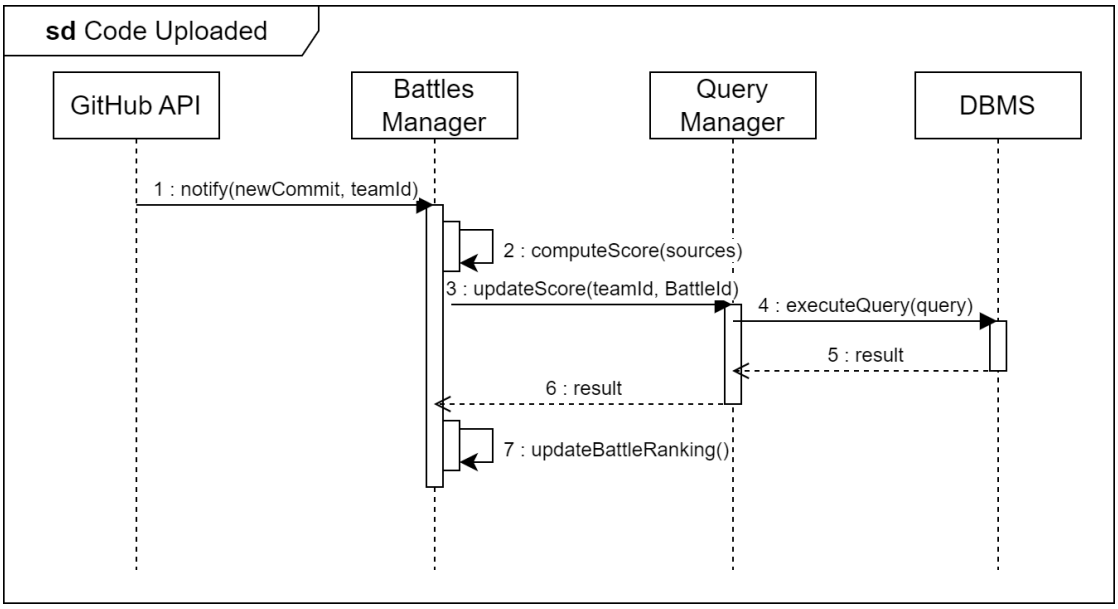


Figure 2.9: Runtime View of Notification of a new commit of a team

2.5 API endpoints

The following section describes one by one the endpoints of the APIs offered by the system.
For the "battle" object, the "phase" attribute is intended to be as an enumeration of 4 possible values:

- 1 - Registration
- 2 - Code Submission
- 3 - Manual evaluation
- 4 - Finished, closed

Qui vi lascio tutte le api con le relative informazioni che mi sevono, poi i path per raggiungerle gestiteveli voi come vi è più comodo e fatemi sapere

GET endpoints - General User

Search for a student

This endpoint allows the anyone who writes in the searchbox to search for a STU by name, surname or email.
Method : GET
EndPoint : api/search/student/:query
URL Parameters :

- `query` : the string that the user has written in the searchbox

Response :

- `200` : JSON object

```
[
    {
        "id": Number,
        "name": String,
        "surname": String,
    },
    ...
]
```

- `400` : error String("The searchbox is empty, at least 1 character is required")
- `401` : error String("Unauthorized")

Inspect a STU's profile

This endpoint allows the anyone who clicks on a STU's name in the search results to inspect its profile.
Method : GET
EndPoint : api/search/student/id/profile
URL Parameters :

- `id` : the id of the student the user is looking for

Response :

- `200` : JSON object

```
{
    "name": String,
    "surname": String,
    "tournaments": [
```

```
            {
                "id": Number,
                "name": String
            },
            ...
        ],
        "badges": [
            {
                "id": Number,
                "name": String,
                "rank": Number[1 - 7],
                "obtained": [
                    {
                        "date": String(Date),
                        "tournament": String,
                        "tournament_id": Number,
                        "battle": String,
                        "battle_id": Number
                    },
                    ...
                ]
            },
            ...
        ]
    }
```

- `401` : error String("unauthorized")

GET endpoints - EDU

Get all tournaments

This endpoint allows the EDU to get the list of all the tournaments on the CKB platform, it will be accessed whenever the EDU clicks on the "Show all tournaments" button on the dashboard.
Method : GET
EndPoint : api/search/tournaments
URL Parameters: None
Response:

- `200` : JSON object

```
[
    {
```

```
            "id": Number,
            //name of the tournament
            "name": String,
            // first name of the tournament creator
            "first_name": String,
            // last name of the tournament creator
            "last_name": String,
            // if the tournament is open (true) or closed (
               false)
            "active": Boolean,
        },
        ...
    ]
```

- `401`: error String("unauthorized)

Get owned tournaments

This endpoint allows the EDU to get the list of the tournaments that it owns, it will be accessed every time the EDU opens the dashboard.

Method : GET
EndPoint : api/search/tournaments/:parameters
URL Parameters:

- `id`: the id of the EDU that is requesting the list of the tournaments

Response:

- `200` : JSON object

```
[
    {
        "id": Number,
        //name of the tournament
        "name": String,
        // first name of the tournament creator
        "first_name": String,
        // last name of the tournament creator
        "last_name": String,
        // if the tournament is open (true) or closed (
           false)
        "active": Boolean,
    },
    ...
]
```

- `401` : error String("Unauthorized")

Get tournament details

This endpoint allows the EDU to get the details of a tournament, it will be accessed when an EDU selects a tournament from the list of the tournaments in the dashboard.
Method : GET
EndPoint : api/search/tournaments/:parameters
URL Parameters:

- `id` : id of the tournament
- `edu_id`: the id of the EDU that is requesting the details of the tournament

Response:

- `200` : JSON object

```
[
    {
        "id": Number,
        "name": String,
        "active": Boolean,
        "admin": Boolean,
        "battles": [
            {
                "id": Number,
                "name": String,
                "language": String,
                "participants": Number,
                "phase": Number[1 - 4],
                // Time remaining in the current phase
                "remaining": String(Date)
            },
            ...
        ],
        "ranking": [
            {
                "id": Number,
                "name": String,
                "points": Number
            },
            ...
        ],
    }
]
```

- `401` : error String("Unauthorized")

Get battle details

This endpoint allows the EDU to get the details of a battle, it will be accessed when an EDU selects a battle from the list of the battles in a tournament view.

Method : GET
EndPoint : api/search/tournament/:id/battles/:parameters
URL Parameters:

- `id`: the id of the battle that the EDU wants to get the details of
- `edu_id`: the id of the EDU that is requesting the details of the battle

Response:

- `200` : JSON object

```
[
    {
        "id": Number,
        "title": String,
        "description": String,
        "language": String,
        "opening": String(Date),
        "registration": String(Date),
        "closing": String(Date),
        "min_group_size": Number,
        "max_group_size": Number,
        "link": String,
        "phase": Number[1 - 4],
        "tournament_name": String,
        "tournament_id": Number,
        "admin": Boolean,
        // Could be reduntant with phase, but it's easier
           to check
        "manual": Boolean,
        "ranking": [
            {
                "id": Number,
                "name": String,
                "score": Number
            },
            ...
        ],
```

```
    }
]
```

- `401` : error String("Unauthorized")

Get the list of groups for the manual evalation

This endpoint allows the EDU to get the list of the groups subscribed to a battle in the "manual evaluation" stage, and for each group that has already been evaluated it provides the score that it has obtained.
Method : GET
EndPoint : api/search/tournament/:tournament_id/battles/:battle_id/:manualEvaluation=true&phase=
URL Parameters:

- `tournament_id` : the id of the tournament of the correlated battle
- `battle_id` : the id of the battle that the EDU wants to get the list of the codes to evaluate of
- `manualEvaluation` : it is a parameter that represent the boolean variable "manual" of the battle
- `phase` : it is a parameter that represents the phase in which a specific battle is. In this case, the battle must be in phase 3, which corresponds to the manual evalation

Response:

- `200` : JSON object

```
[
    {
        "id": Number,
        "team": String,
        "score"?: Number
    },
    ...
]
```

- `401` : String("unauthorized")

Evaluate code

code.js
This endpoint allows the EDU to evaluate a code, it will be accessed when an EDU clicks on the "Evaluate" button from the table of the codes to evaluate.
Method : GET
EndPoint : api/search/tournament/:tournament_id/battles/:battle_id/:team_id
URL Parameters:

- `tournament_id` : the id of the tournament of the correlated battle
- `team_id`: the id of the group that the EDU wants to evaluate
- `battle_id`: the id of the battle that the EDU wants to evaluate

Response:

- `200` : JSON object

```
{
    "group_id": Number,
    "battle_id": Number,
    "language": String,
    // Name of the group
    "name": String,
    // Score of the group, null if not evaluated yet
    "score"?: Number,
    // Code of the group. The url links to the file
    "code": String(url)
}
```

- `401` : error String("unauthorized")

GET endpoints - STU

Get subscribed tournaments

This endpoint allows the STU to get the list of the tournaments that it is subscribed to, it will be accessed every time the STU opens the dashboard.
Method : GET
EndPoint : api/search/tournament/:subscribed=email
URL Parameters:

- `email` : the email of the user actually logged into the platform

Response:

- `200` : JSON object

```
[
    {
        "id": Number,
        "name": String,
        // first name of the tournament creator
        "first_name": String,
        // last name of the tournament creator
```

```
            "last_name": String,
            // if the tournament is open (true) or closed (
               false)
            "active": Boolean,
        },
        ...
    ]
```

- `401` : error String("unauthorized")

Get unsubscribed tournaments

This endpoint allows the STU to get the list of the tournaments that it is not subscribed to, it will be accessed every time the STU opens the dashboard.
Method : GET
EndPoint : api/search/tournament/:subscribed!=email
URL Parameters:

- `email` : the email of the user actually logged into the platform

Response:

- `200` : JSON object

```
[
    {
        "id": Number,
        "name": String,
        "daysLeft": Number,
    },
    ...
]
```

- `401` : error String("unauthorized")

Get tournament details

This endpoint allows the STU to get the details of a tournament, it will be accessed when a STU selects a tournament from the list of the tournaments in the dashboard.
Method : GET
EndPoint : api/search/tournament/STU=true&id=tournament_id
URL Parameters:

- `STU` : it is needed to retrieve the correct object for a STU
- `tournament_id` : this is the id of the tournament the user is looking for

Response:

- `200` : JSON object

```
[
    {
        "id": Number,
        "name": String,
        "active": Boolean,
"canSubscribe": Boolean,
        "subscribed": Boolean,
        "battles": [
            {
                "id": Number,
                "name": String,
                "language": String,
                "participants": Number,
                "subscribed": Boolean,
                "score"?: Number,
                "phase": Number[1 - 4],
                // Time remaining in the current phase
                "remaining": String(Date)
            },
            ...
        ],
        "ranking": [
            {
                "id": Number,
                "name": String,
                "points": Number
            },
            ...
        ]
    }
]
```

- `401` : error String("unauthorized")

Get battle details

This endpoint allows the STU to get the details of a battle, it will be accessed when a STU selects a battle from the list of the battles in a tournament view.
Method : GET
EndPoint : api/search/tournament/STU=true&id=battle_id
URL Parameters:

- `STU` : it is needed to retrieve the correct object for a STU
- `battle_id` : this is the id of the battle the user is looking for

Response :

- `200` : JSON object

```
[
    {
        "id": Number,
        "title": String,
        "description": String,
        "language": String,
        "opening": String(Date),
        "registration": String(Date),
        "closing": String(Date),
        "min_group_size": Number,
        "max_group_size": Number,
        "link": String,
"canSubscribe": Boolean,
        "canInviteOthers": Boolean,
        // If the user's team satisfies the min members
           constraint
        "minConstraintSatisfied": Boolean,
        "subscribed": Boolean,
        "phase": Number[1 - 4],
        "tournament_name": String,
        "tournament_id": Number,
        "ranking": [
            {
                "id": Number,
                "name": String,
                "score": Number
            },
            ...
        ]
    }
]
```

- `401` : error String("unauthorized")

POST endpoints - General User

Registration

This endpoint allows the user to register to the platform.
Method : POST
EndPoint : TODO **Body Parameters** :

- `name` : the name of the user
- `surname` : the surname of the user
- `email` : the email of the user
- `uni` : the university of the user
- `role` : the role of the user (with value "STU" or "EDU")
- `password` : the password of the user
- `password2` : the password of the user, repeated

Response :

- `200` : message String("")
- `400` : error String("")
- `409` : error String("")

Login

This endpoint allows the user to login to the platform.
Method : POST
EndPoint : TODO **Body Parameters** :

- `email` : the email of the user
- `password` : the password of the user

Response :

- `200` : message String("")
- `400` : error String("")
- `401` : error String("")

POST endpoints - EDU

Create Tournament

This endpoint allows the EDU to create a new tournament.
Method : POST
EndPoint : TODO **URL Parameters**:

- `EDU_id` : Integer

Body Parameters :

- `tournemant_name` : String
- `registration_deadline` : String (date)
- `badges` : Array of Objects (badge_id)

Response :

- `200` : message String("")
- `400` : error String("")
- `403` : error String("")
- `404` : error String("")

Share Tournament

This endpoint allows the EDU to share a tournament with other colleagues.
Method : POST
EndPoint : TODO **URL Parameters**:

- `EDU_id` : Integer
- `tournament_id` : Integer

Body Parameters :

- `invited_EDU_id` : Integer

Response :

- `200` : message String("")
- `400` : error String("")
- `403` : error String("")
- `404` : error String("")

Create Battle

This endpoint allows the EDU to create a new battle within the context of a tournament.
Method : POST
EndPoint : TODO **URL Parameters**:

- `EDU_id` : Integer
- `tournament_id` : Integer

Body Parameters :

- `battle_name` : String
- `programming_language` : String
- `registration_deadline` : String (date)
- `end_battle` : String (date)
- `manual_evaluation` : Boolean
- `min_STU` : Integer
- `max_STU` : Integer
  Gradle
- `test_cases` : ?
- `build_script` : ?
- `problem_spec` : File PDF

Response :

- `200` : message String("")
- `400` : error String("")
- `403` : error String("")
- `404` : error String("")

Create Badge

This endpoint allows the EDU to create a new badge insisde the CKB platform.
Method : POST
EndPoint : TODO **URL Parameters**:

- `EDU_id` : Integer

Body Parameters :

- `badge_name` : String
- `rules` : Array of String

Response :

- `200` : message String("")
- `400` : error String("")
- `403` : error String("")
- `404` : error String("")

Manual Evaluation

This endpoint allows the EDU to evaluate perform a manual evaluation on team's code.
Method : POST
EndPoint : TODO **URL Parameters**:

- `EDU_id` : Integer
- `tournament_id` : Integer
- `battle_id` : Integer

Body Parameters :

- `given_grades` : Array of Tuple (team_id - score)

Response :

- `200` : message String("")
- `400` : error String("")
- `403` : error String("")
- `404` : error String("")

POST endpoints - STU

Join Tournament

This endpoint allows the STU to join a tournament.
Method : POST
EndPoint : TODO **URL Parameters**:

- `STU_id` : Integer

Body Parameters :

- `tournament_id` : Integer

Response :

- `200` : message String("")
- `400` : error String("")
- `403` : error String("")
- `404` : error String("")
- `409` : error String("")

Join Battle

This endpoint allows the STU to join a battle forming a team respection the boundaries imposed.
Method : POST
EndPoint : TODO **URL Parameters**:

- `STU_id` : Integer
- `tournament_id` : Integer

Body Parameters :

- `battle_id` : Integer
- `team_name` : String
- `STU_invited` : Array of STU (STU_id)

Response :

- `200` : message String("")
- `400` : error String("")
- `403` : error String("")
- `404` : error String("")
- `409` : error String("")

Join Team

This endpoint allows the STU to join a team to compete in a battle.
Method : POST
EndPoint : TODO **URL Parameters**:

- `STU_id` : Integer

Body Parameters :

- `team_id` : String
- `tournament_id` : Integer
- `battle_id` : Integer
- `choice` : Boolean

Response :

- `200` : message String("")
- `400` : error String("")
- `403` : error String("")
- `404` : error String("")
- `409` : error String("")

2.6 Component interfaces

Da contorllare se i nomi dei metodi usati sono consistenti con quelli del RASD e delle Runtime View

In this diagram are displayed all the functions that each component uses to communicate with other components. In particular, the blue interfaces are external APIs in which at this level of abstraction do not require to be detailed as the other interfaces. The interfaces contains both the operations for STU users and EDU users.

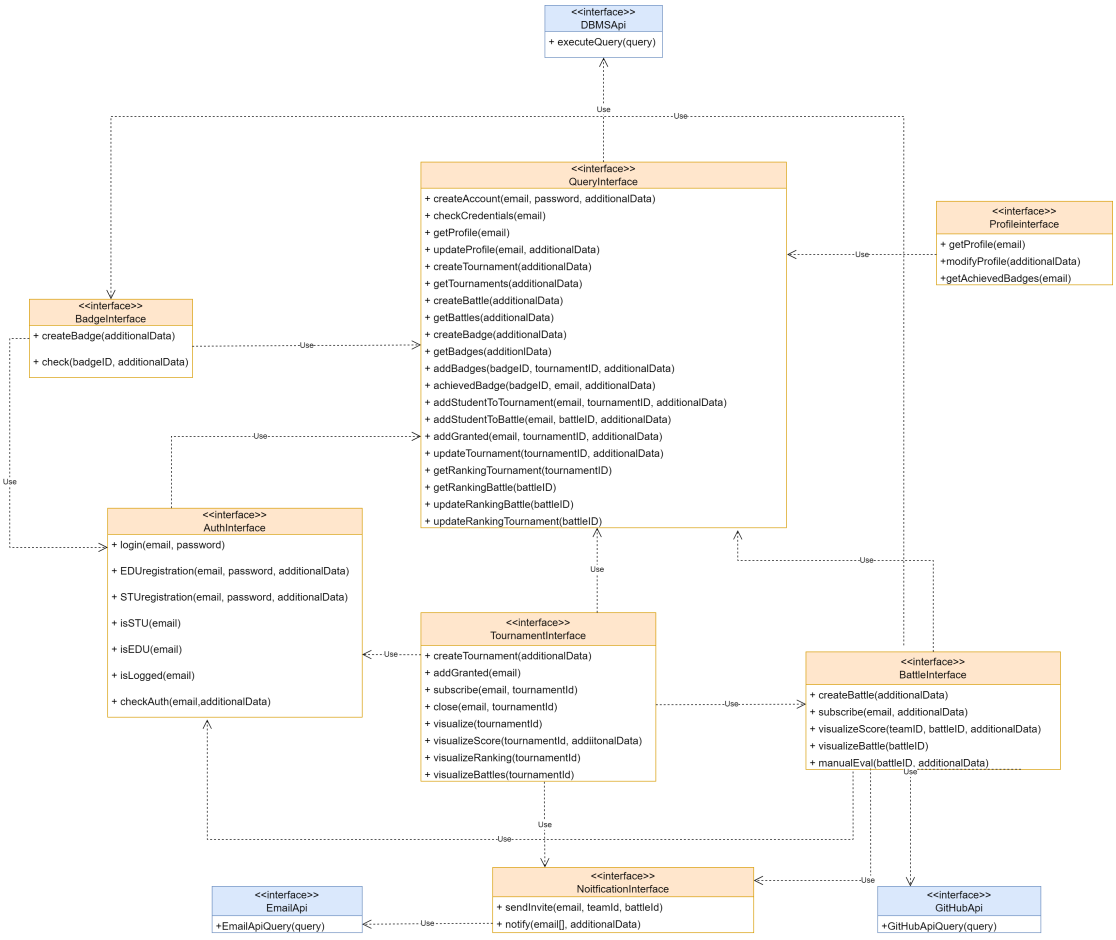


Figure 2.10: Component interfaces diagram

2.7 Selected Architectural Styles and Patterns

2.7.1 Client - Server Paradigm

The client-server paradigm represents a software architecture model in which two distinct components, identified as the client and server, interact to deliver a service. The client is the component that requests the service, while the server is responsible for its provision. Communication between them is based on the concept of asynchrony, allowing the client to send requests without waiting for an immediate response. The server, in turn, can process the request at a later time and send a response to the client. This highly flexible paradigm finds extensive use in implementing various applications, including web, network, and distributed contexts. Key advantages include flexibility, scalability, and the ability to configure the system to ensure an adequate level of security.

2.7.2 N - tiers Architecture

Modificare in base a se teniamo una 3 o 4 tier application
The 4-tier architecture is a software architecture model that divides an application into four distinct layers: Client, Web Server, Application Server, and Database Server. In its implementation for web applications, users send requests to the web server through the browser, which provides static content and forwards dynamic requests to the application server. The application server processes the requests, interacts with the database server, and sends responses to the web server, which, in turn, transmits them to the client. The advantages of this architecture include scalability, enhanced security through data isolation, and increased ease of maintenance and testing due to the separation of various layers. The 4-tier architecture is particularly suitable for complex and high-traffic web applications, offering flexibility and scalability to meet a wide range of application needs.

2.7.3 MVC Design Pattern

The MVC is a widely adopted architectural pattern for the development of UI, separating the logic of the user interface from the business logic to enhance maintainability and scalability. Comprising three main components:

- **Model:** manages data and business rules.
- **View:** displays data to the user.
- **Controller:** handles user requests.

Advantages of the MVC pattern include the separation of concerns, ease of scalability, and maintainability of the application. MVC is a versatile model suitable for various applications, particularly for web applications.

2.7.4 RESTfull API

The RESTful API architecture is a software model for creating APIs based on the HTTP protocol. RESTful APIs consist of resources identified by URLs and operate through HTTP methods such as GET, POST, PUT, and DELETE. They adhere to the principles of REST, ensuring flexibility, scalability, and ease of use. Key principles include the use of unique URIs for each resource, the adoption of HTTP methods to define operations, and the standardized representation of data (such as JSON or XML). RESTful APIs have been widely embraced for web API development due to their flexibility, scalability, and ease of implementation.

2.8 Other design decisions

2.8.1 Web Application

As the platform's primary functionality is closely tied to coding activities, it is expected that users will predominantly employ personal computers, so there is no need to develop a mobile application, which would require a significant amount of additional work.
Instead, the system will be accessible through a web application, that is much easier to develop, maintain, and access by the users.

2.8.2 Single Page Application

The system is developed as a SPA, which is a web application that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages.
This approach allows the system to be more responsive and similar to a desktop application since the user does not wait for the entire page to be reloaded every time it performs an action. Furthermore, it will allow the system to be more efficient, since the server does not have to send the entire page every time the user performs an action, but only the data that has changed, leaving the client to render the page.

2.8.3 Relational Database

The system is designed to use a relational database because it is effective at storing structured data, granting data integrity, and providing fast query performance. It can also be easily scaled to handle large amounts of data and support many concurrent users. The database allows us to store and retrieve information efficiently, while also ensuring that the data is accurate and consistent.

Non abbiamo mai parlato di Gradle secondo me nel design è da mettere, non so bene dove, ma bisonga mostrare che è parte del sistema

3. User Interface Design

Since the mockups of the user interface have already been presented in the RASD document, they are not repeated here; instead, are presented the UX diagrams.

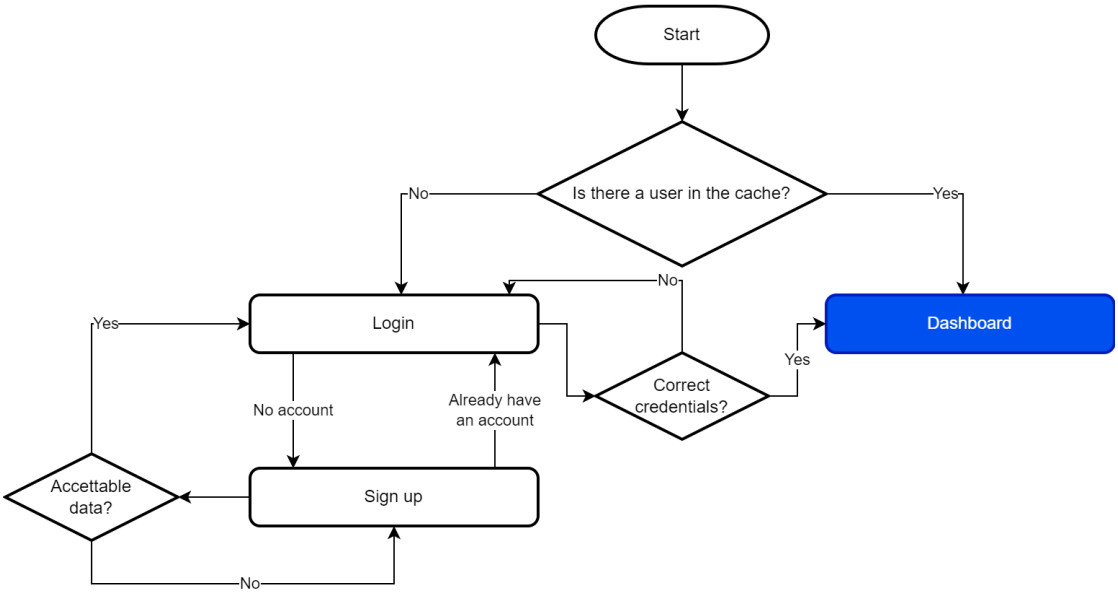


Figure 3.1: Undifferentiated: Entry point i.e. the login page

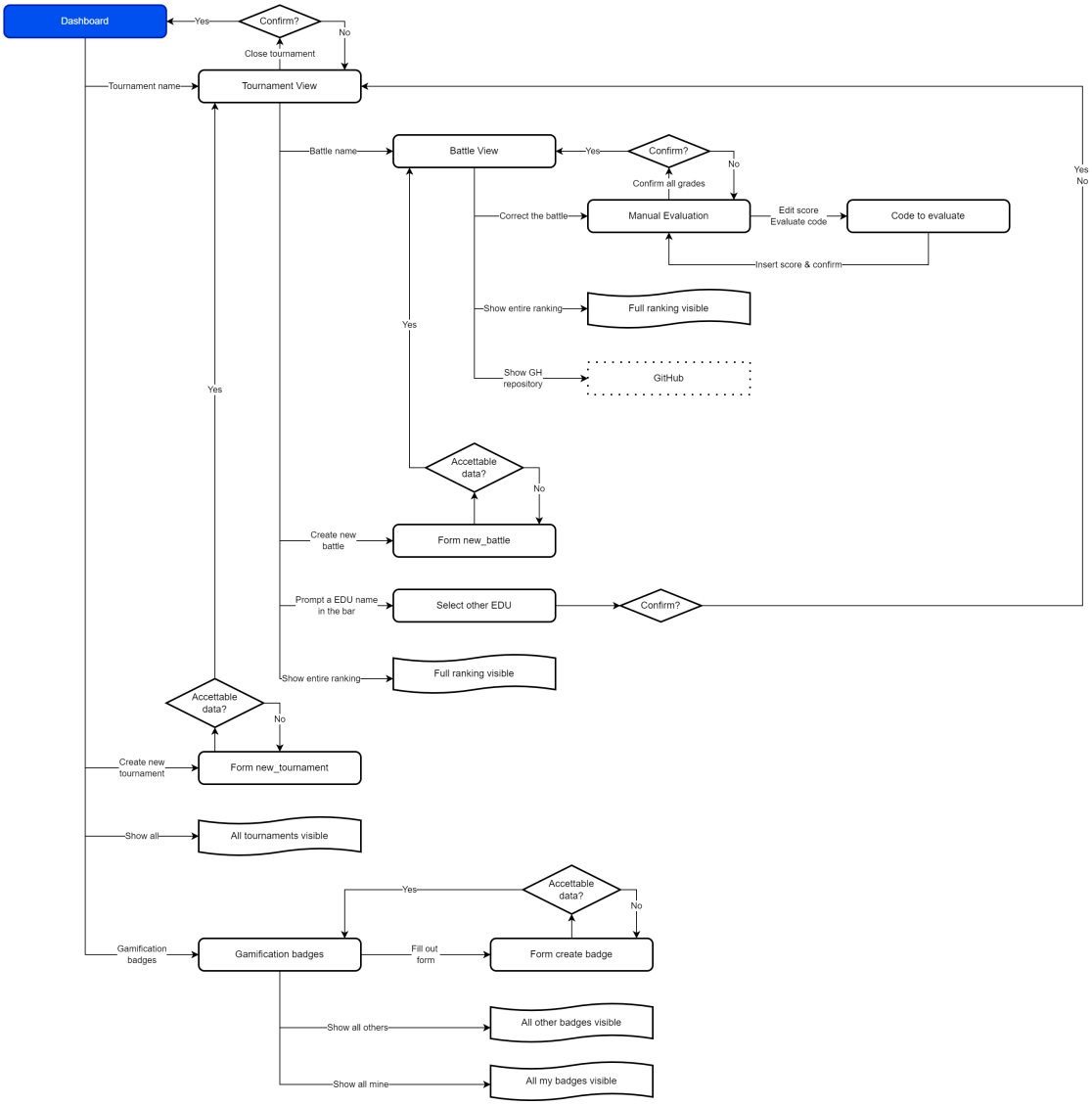
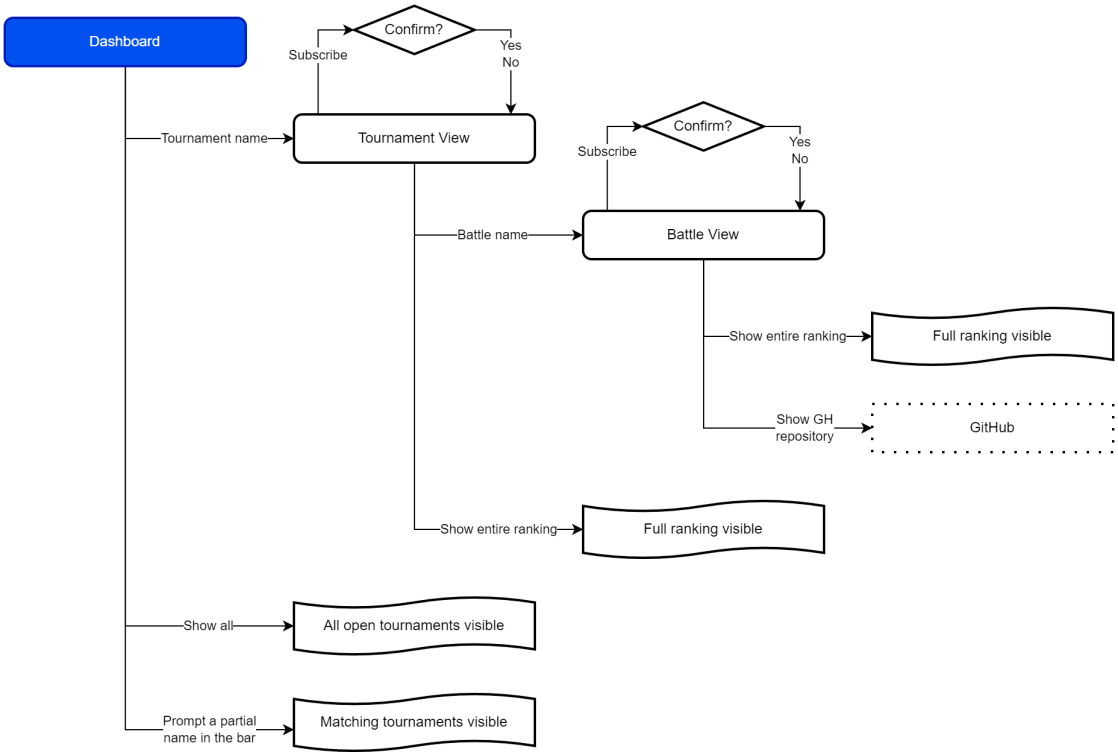


Figure 3.2: Flow graph for the EDU



Da aggiungere Accettare o Declinare le richieste per i team

Figure 3.3: Flow graph for the STU

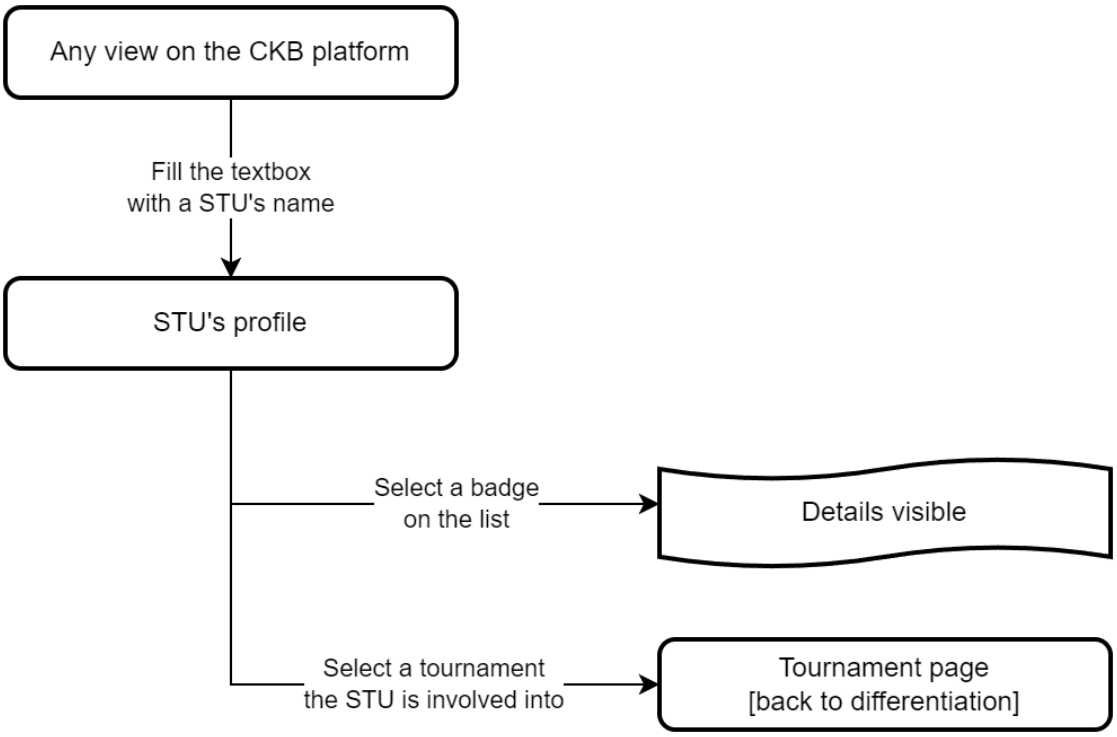


Figure 3.4: Undifferentiated: viewing a STU's profile

4. Requirements Traceability

The table below reports the mapping of the design components on the requirements. This is useful to understand the role of each component to satisfy a certain functional requirement. All the components are mapped, so each component is important to satisfy at least one functional requirement.

Requirement	Description	Design Element(s)
R1	The software shall allow the unregistered EDUs to create an account.	EDU WebApp, Auth Manager, Query Manager, DBMS
R2	The software shall allow the registered EDUs to log in.	EDU WebApp, Auth Manager, Query Manager, DBMS
R3	The software shall allow the authenticated EDUs to create new tournaments.	EDU WebApp, Auth Manager, Tournaments Manager, Query Manager, DBMS
R4	The software shall allow an authenticated EDU to permit other authenticated colleagues to create battles in its tournament, those become owners of the tournament as much as who added them.	EDU WebApp, Auth Manager, Tournaments Manager, Query Manager, DBMS

R5	The software shall allow the authenticated EDUs to create coding battles in their tournaments by letting them upload the Code Kata, setting the minimum and maximum number of STUs per group, registration and final submission deadlines, and the score configurations.	EDU WebApp, Auth Manager, Battles Manager, Query Manager, DBMS
R6	The software shall allow the authenticated EDUs to manually evaluate the work done by STUs subscribed to their own Code Kata battle.	EDU WebApp, Auth Manager, Battles Manager, Query Manager, DBMS
R7	The software shall allow the authenticated EDUs to see the sources produced by each team participating in their tournaments.	EDU WebApp, Auth Manager, Battles Manager, Query Manager, DBMS
R8	The software shall allow the authenticated EDUs to see the personal tournament score of each STU (which is the sum of all battle scores received in that tournament).	EDU WebApp, Tournaments Manager, Profile Inspector, Query Manager, DBMS
R9	The software shall allow the authenticated EDUs to see a rank that measures how a STU's performance compares to other STUs in the context of that tournament.	EDU WebApp, Tournaments Manager, Profile Inspector, Query Manager, DBMS
R10	The software shall allow the authenticated EDUs to see the list of ongoing and finished tournaments as well as the corresponding tournament rank.	EDU WebApp, Tournaments Manager, Query Manager, DBMS
R11	The software shall allow the authenticated EDUs to see the list of ongoing and finished battles as well as the corresponding battle rank within a tournament.	EDU WebApp, Tournaments Manager, Battles Manager, Query Manager, DBMS

R12	The software shall allow an authenticated EDU to close a tournament iff it is one of the owners of that tournament.	EDU WebApp, Tournaments Manager, Query Manager, DBMS
R13	When the authenticated EDU creates a tournament, the software shall allow it to define gamification badges concerning that specific tournament.	EDU WebApp, Auth Manager, Badges Manager, Query Manager, DBMS
R14	The software shall allow the authenticated EDU to create new badges and define new rules as well as new variables associated with them.	EDU WebApp, Auth Manager, Badges Manager, Query Manager, DBMS
R15	The software shall allow all authenticated EDUs to visualize the badges created in CKB by other EDUs.	EDU WebApp, Badges Manager, Query Manager, DBMS
R16	The software shall allow all authenticated EDUs to visualize STUs' profile where they can see their collected badges, and some personal information.	EDU WebApp, Profile Inspector, Query Manager, DBMS
R17	The software shall allow the unregistered STUs to create an account.	STU WebApp, Auth Manager, Query Manager, DBMS
R18	The software shall allow the registered STUs to log in.	STU WebApp, Auth Manager, Query Manager, DBMS
R19	The software shall allow the authenticated STUs to form teams by inviting other STUs respecting the minimum and maximum number of STUs per group set for that battle.	STU WebApp, Battles Manager, Notification Manager, Auth Manager
R20	The software shall allow the authenticated STUs to join a team by an invite.	STU WebApp, Battles Manager, Notification Manager, Auth Manager, Query Manager, DBMS

R21	The software shall allow the authenticated STUs to subscribe to a tournament until a certain deadline.	STU WebApp, Tournaments Manager, Auth Manager, Query Manager, DBMS
R22	The software shall allow the authenticated STUs subscribed to a coding battle to upload their work until the final submission deadline of that Code Kata battle.	Battles Manager
R23	When the registration deadline of a battle expires, the software shall create a GitHub repository containing the Code Kata.	Battles Manager
R24	The software shall send to all authenticated STUs who are members of subscribed teams to a battle the link to a GitHub repository containing the Code Kata.	Battles Manager, Notification Manager, Query Manager, DBMS
R25	The software shall run the tests on executables pushed by a team, it shall also calculate and update the battle score of the corresponding team.	Battles Manager, Query Manager, DBMS
R26	At the end of the consolidation stage of a specific battle *'b'*, the software shall send a notification to all authenticated STUs participating to *'b'* when the final battle rank becomes available.	Battles Manager, Notification Manager, Query Manager, DBMS
R27	The software shall allow the authenticated STUs to see the list of ongoing and finished battles as well as the corresponding battle rank, iff they are part of the tournament.	STU WebApp, Battles Manager, Query Manager, DBMS

R28	The software shall allow all authenticated STUs to see the personal tournament score of each STU (which is the sum of all battle scores received in that tournament).	STU WebApp, Profile Inspector, Tournaments Manager, Query Manager, DBMS
R29	The software shall allow all authenticated STUs to see a rank that measures how a STU's performance compares to other STUs in the context of that tournament.	STU WebApp, Tournaments Manager, Query Manager, DBMS
R30	The software shall allow all authenticated STUs to see the list of ongoing and finished tournaments as well as the corresponding tournament rank.	STU WebApp, Tournaments Manager, Query Manager, DBMS
R31	The software shall notify all authenticated STUs involved in a closed tournament when the final tournament rank becomes available.	Tournaments Manager, Notification Manager
R32	The software shall allow all authenticated STUs to visualize other STUs' profile where they can see their collected badges, and some personal information.	STU WebApp, Profile Inspector, Query Manager, DBMS

Table 4.1: Requirements traceability

5. Implementation, Integration and Test Plan

The application is based on three logical layers (data, logic, and presentation) that can be implemented in parallel.
The whole system can be tested in parallel following a bottom-up approach, this is because all the components can be developed independently and then integrated, and so the testing can be done in an incremental way to evaluate the dependencies between the subcomponents.

5.1 Development plan

Since the front-end relies on REST APIs provided by the back-end, the focus is more on the back-end development, to provide the front-end developers with the APIs needed.

5.1.1 Front-end

Even if it relies on the back-end APIs, the front-end can be developed in parallel with the back-end, since the APIs are well defined and documented. It is sufficient to provide a mock-up of the JSON objects that will be returned by the APIs to correctly develop the front-end.

5.1.2 Back-end

Back-end development is the most important part of the project since it provides the front-end with the APIs it needs.
The order of development of the subcomponents is given by the dependencies between them, so the order is the following:

1. **DBMS**: it is the core of the data layer, so it is the first component to be developed. It includes the implementation of the ER diagram given in Figure 2.3
2. **Query manager**: it is the component that provides the APIs to the logic layer that will be used to query the database. It is the second component to be developed

since it relies on the DBMS and it is the only component that can access the database.

3. **Auth manager, Notification manager**: they shall be implemented before the other subcomponents because they are used by the other subcomponents to authenticate the users and to send notifications to them. Since they are independent from each other, they can be developed in parallel.

4. **All other subcomponents**: they can be developed in parallel since they are sufficiently independent from each other. If, for any reason, a subcomponent needs to interact with another one, it can see the other one as a black box

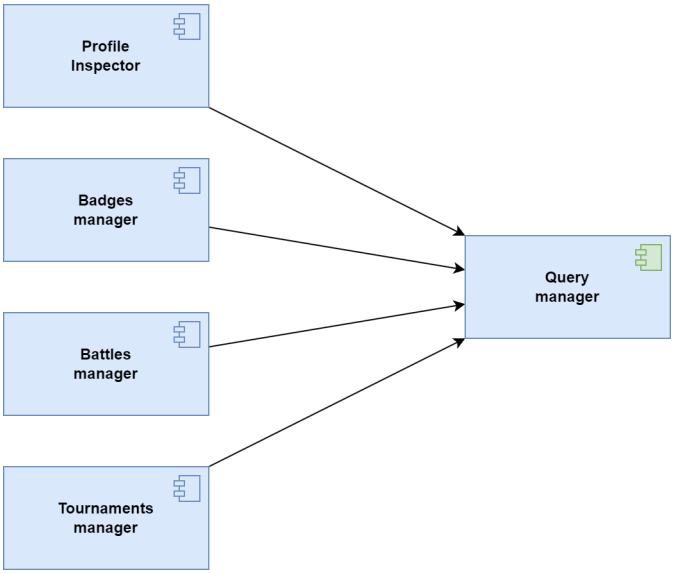
5.2 Integration plan

The following section shows the order in which the subcomponents are integrated to constitute the whole system.
Each component must be uint tested before being integrated with the others. The integration testing process occurs incrementally to allow for bug tracking. Every time a component of a dependency level is completed, it is integrated with the modules of the other levels to test the behavior of the developed subsystem. When the whole component is fully integrated is finally tested.
The order of the integration is given by the dependencies between the subcomponents, so the order is shown by the following graphs (for the sake of simplicity, the graphs show only the dependencies between the subcomponents omitting the interfaces between them):

The first components to integrate are the Query Manager and the DBMS since they make a connection between the application layer and the data layer.

Figure 5.1: Integration plan for the data layer

The Query Manager is the only component that can access the database and nearly all the other components rely on it to query the database.



Se query manager è l'unico componente che si interfaccia al DB, allora Auth Manager deve star collegato ad esso

Figure 5.2: Integration plan for the query manager

The second component to integrate is the Auth Manager since it makes the system behave differently based on the user's permissions and so it is crucial to ensure that it works correctly before integrating other components (a concrete example is to ensure that the query to the databases are done by an authenticated user at the right level).

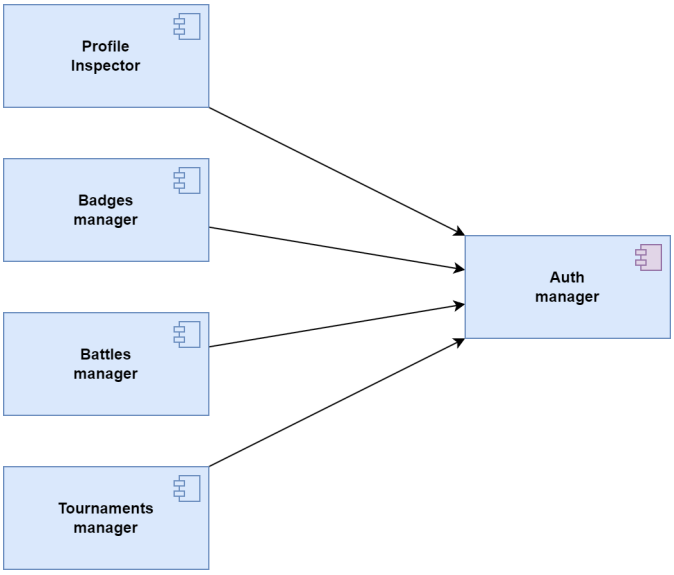
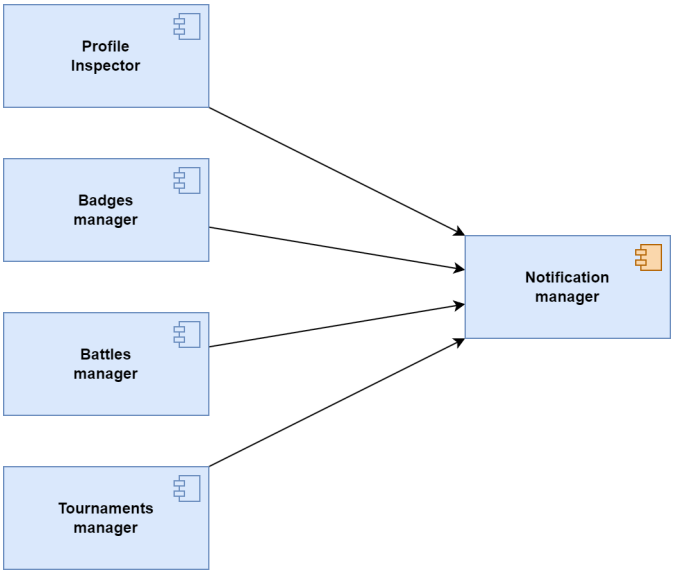


Figure 5.3: Integration plan for the auth manager

Another component to integrate is the Notification Manager, which is used to send notifications to the users.
It can be integrated in parallel with the Query Manager since it does not need to access the database.



Non credo si debbano mettere
Profile inspector e Badges Manager

Figure 5.4: Integration plan for the notification manager

Profile Inspector, Badge Manager, Battle Manager, and Tournament Manager are components that need to be integrated with both the Query and the Auth Manager.
Some of them rely on the Notification Manager as well, so they need an integration with it.

Finally, once all the unit tests are passed and the components are integrated, the presentation layer can be integrated with the other components and a full testing phase can be performed on the whole system.

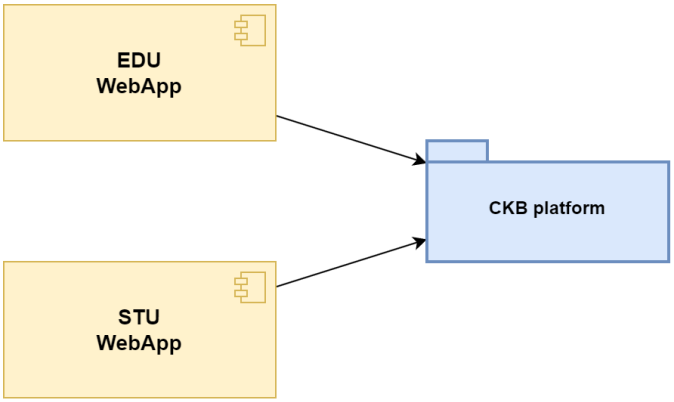


Figure 5.5: Integration plan for the presentation layer

Vogliamo mettere un diagramma riassuntivo di tutto l'Application server per mostrare le dipendenze?

5.3 System Testing

The testing phase aims to verify the functional and non-functional requirements and must take place in a testing environment that is as close as possible to the production environment.
To ensure the minimum probability of failure of the released software, these testing performed are:

- **Functional testing**: to verify that the system meets the functional requirements, in particular, the ones specified on the RASD document.
- **Performance testing**: to verify that the system meets the non-functional requirements, it helps to identify and eliminate performance bottlenecks affecting response time, utilization, and throughput. in particular, whether the software remains functional with increased demand and various environmental conditions.
- **Load testing**: to detect memory leaks, mismanagement of memory, and buffer overflows, it also identifies the maximum operating capacity of the application.
- **Stress testing**: to measure software robustness and error handling under heavy load conditions, it verifies stability and reliability.

6. Effort Spent

Team

Topic	**Time**
Division of work	1h

Table 6.1: Effort Spent during team meetings

Tommaso Pasini

Topic	**Time**
Chapter 1	1h30m
Revised Chapter 1, 2 and 3 (Grammar and consistency)	3h
Architectural Styles and Patterns	30m
Revised Chapter 4 and 5 (Grammar and consistency)	1h
API EndPoint POST	1h30m

Table 6.2: Effort Spent by Tommaso Pasini

Elia Pontiggia

Topic	**Time**
Architectural design	4h
User Interface Design	2h
Implementation, integration and test plan	3h

Table 6.3: Effort Spent by Elia Pontiggia

Michelangelo Stasi

Topic	**Time**
Runtime Views	5h
API EndPoints	1h
Component Interfaces	2h
Requirements Traceability	2h

Table 6.4: Effort Spent by Michelangelo Stasi