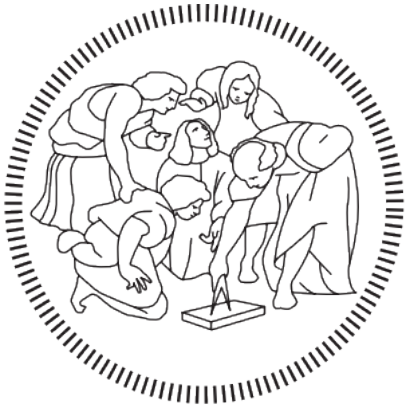

POLITECNICO
MILANO 1863

CodeKataBattle

Design Document

Software Engineering 2 project
Academic year 2023 - 2024

15 November 2023
Version 0.0

Authorss:
Tommaso Pasini
Elia Pontiggia
Michelangelo Stasi

Professor:
Matteo Camilli

Revision History

Date	Revision	Notes
15/11/2023	v.0.0	Document creation
TBD	v.1.0	First relese

Contents

1. Introduction

Questa parte qui è mooooolto simile a quella che c'è nel RASD, quindi aspettiamo di avere il RASD definitivo per fare il merge

2. Architectural Design

2.1 Overview

The CKB platform system is composed by a 3-tier architecture.
Its software application architecture is organized into three logical tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.

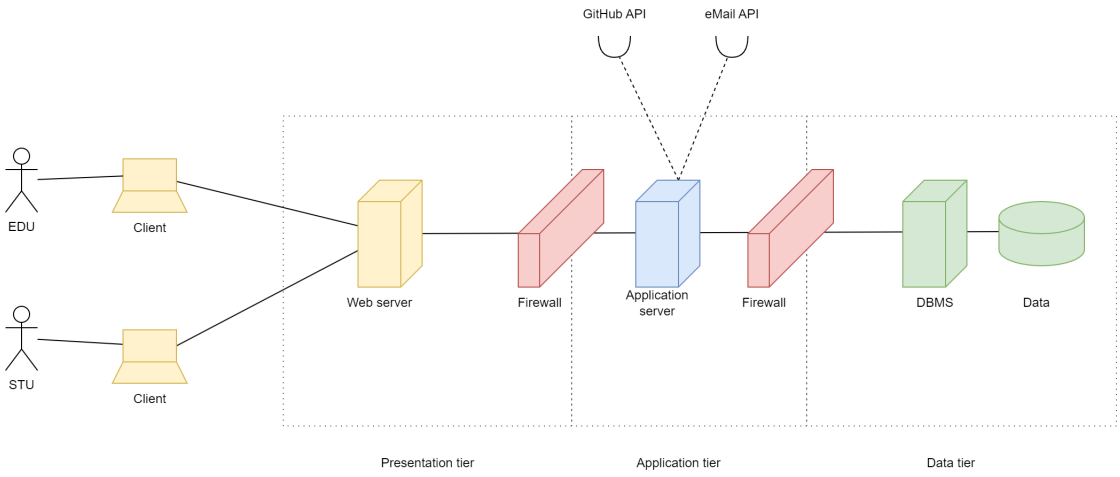

Figure 2.1: High level components diagram

The service will be accessed through a web interface, employing a Single Page Application (SPA). Utilizing an SPA is ideal for this application, as it facilitates extensive interaction without necessitating frequent page reloads.
The system's architecture is structured into distinct layers, with application servers interacting with a database management system and utilizing APIs for data retrieval and storage. Adhering to REST standards, the application servers are intentionally designed to be stateless, handling the login sessions for user thanks to the caching, following the best practices for web applications.
The system will include more than one firewall to ensure security.

2.2 Component view

The system is composed by the following components:

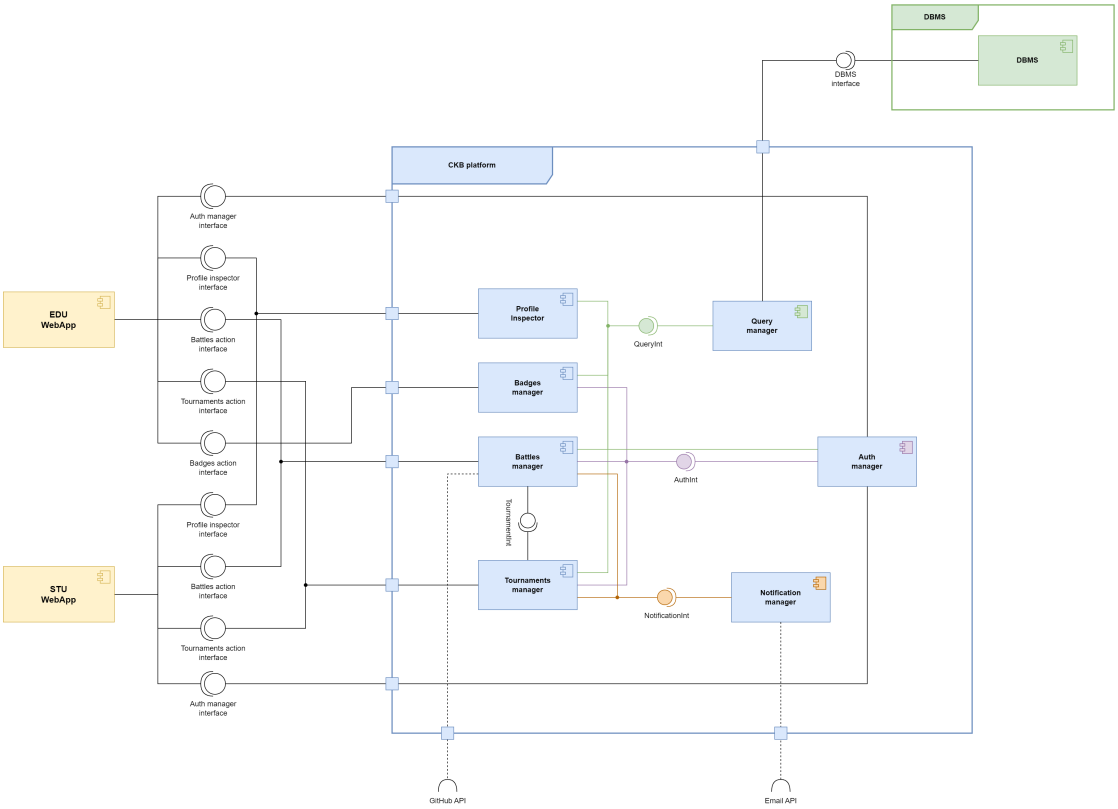

Figure 2.2: Component diagram

In order to maintain the readability of the diagram, the interfaces have been grouped with respect to their functionality; the complete set of endpoints is available later in the document.

2.2.1 Client components

The client components are represented by only a single component, a user-friendly WebApp, that will behave differently depending on the type of the user that is using it (whether it is an EDU or a STU).
The WebApp is interfaced with the server components through all the APIs offeded by the server (see later for a detailed description of the APIs).

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 exept 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 with respect to the user that 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;
- the visualization of the badges assigned to a user (?)

Qui devo interfacciare anche con battle e user? It is interfaced with the auth manager (since the creation of a badge is admissible only for the EDUs) and with the query manager (since it needs to access the database to store the badges).
It is interfaced with the EDU WebApp through the proprer, 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 torunaments
- the visualization of the tournaments;
- the excange of admin permission between EDUs;
- the subscription of the STUs to the tournaments;
- the visualization of the scores of the EDUs in the tournaments, and so of the ranking

It is interfaced with the auth manager (to allow and perform different actions depending on the level of the user that made the request), with the query manager (since it needs to access the database) and the notification manager (since it needs to notify the users of 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

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), with the query manager (since it needs to access the database) and the notification manager (since it needs to notify the users of the opening of subscriptions, the start and the end of a bettle).
It is interfaced with the WebApp, both EDU and STU, through 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 a STU has earned during the battles and the tournaments.
It is interfaced with the query manager (since it needs to access the database). 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, with the following entity-relationship diagram:

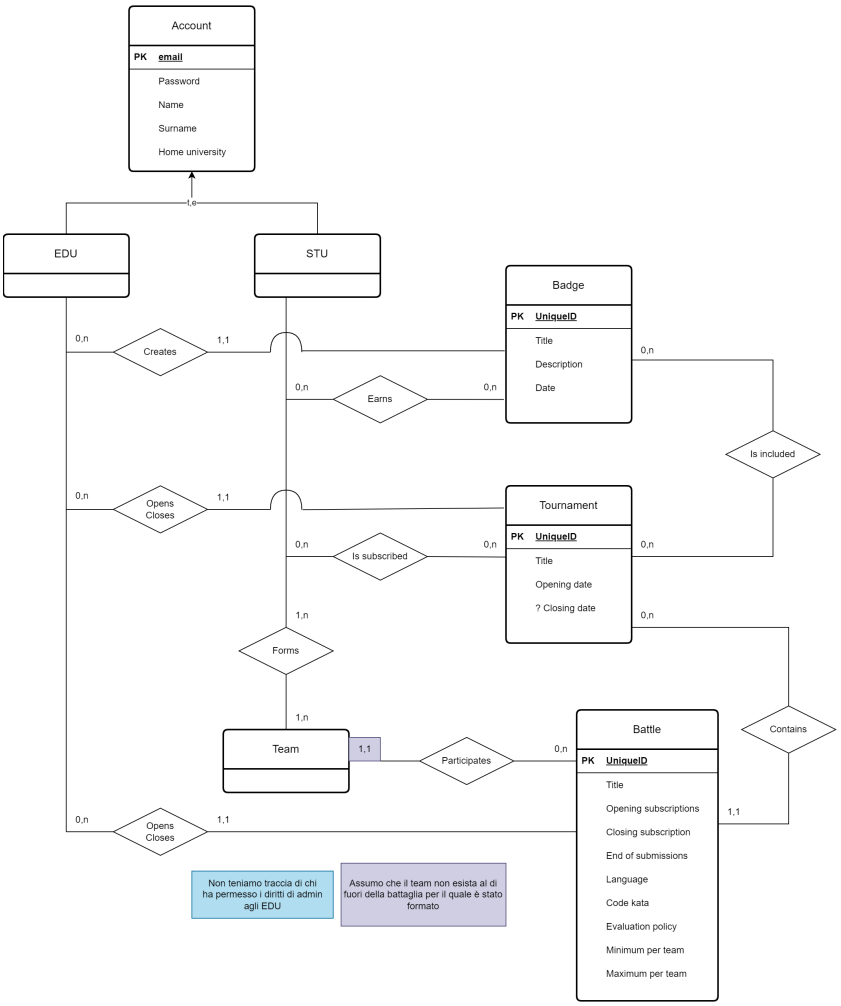

Figure 2.3: Entity-relationship diagram

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, we have opted for distinct solutions for these components.
The static web server will be hosted on a CDN (Content Delivery Network) on cloud, exploiting its edge location caches and reverse proxies to ensure rapid response times. On the other hand, the application server, containing both a business logic layer and a data tier, will find its home 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.

The components of the system are explaineed in the following:

- **PC**: Personal computer of the user, it suffices to have a wordking O.S. and a browser installed that supports JavaScript and HTML5 in order to use the system.
- **CDN**: As said before, the CDN is used to host the static web server, that serves as the entry point for clients to access the SPA. It will allow it to be downloaded 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. It will allow the system to be scalable and flexible, to be secure and to be cost-efficient. It is composed by:

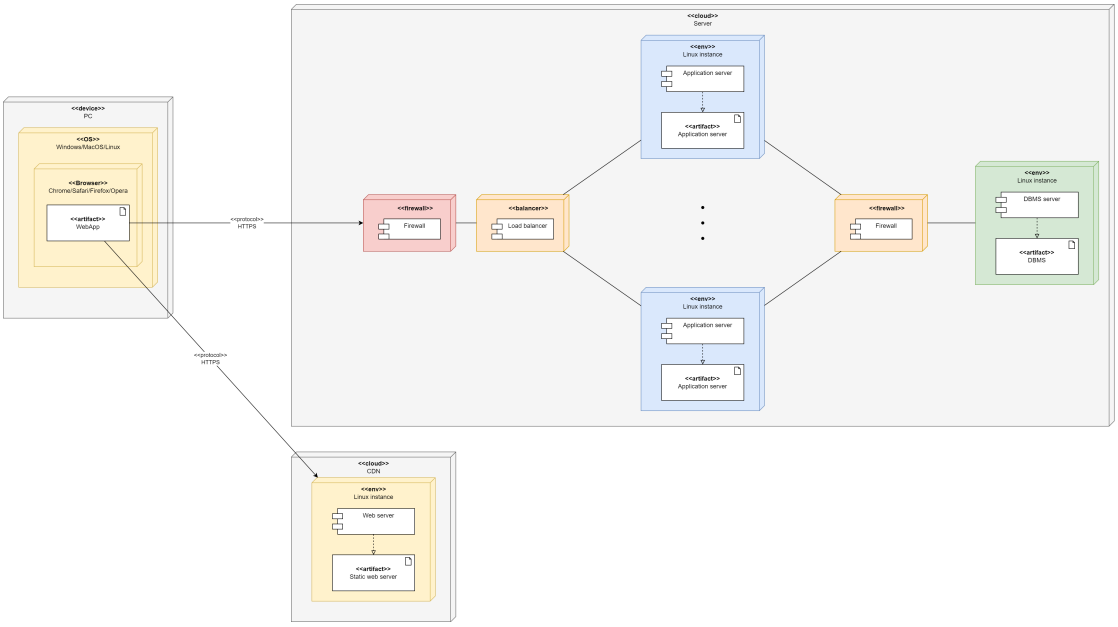

Figure 2.4: Deployment diagram

- **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 will be 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**: They 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 in questo capitolo 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 Other design decisions

2.4.1 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 being accessed by the users.

2.4.2 Single Page Application

The system will be developed as a Single Page Application (SPA), that 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 will allow the system to be more responsive and to be more similar to a desktop application, since the user will not have to wait for the entire page to be reloaded every time he performs an action. Furthermore, it will allow the system to be more efficient, since the server will 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.

2.4.3 Relational Database

We selected a relational database for our system design 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

2.4.4 RESTful API

We have chosen to implement a RESTful API for our system because it is a simple, lightweight, and flexible architecture that is easy to understand and use. It is also scalable and reliable, making it ideal for our application.

3. User Interface Design

Since the mockups of the user interface have already been presented in the RASD document, we will not repeat them here. Instead, we will present the UX diagram

4. Effort Spent

Team

Topic	Time

Table 4.1: Effort Spent during team meetings

Tommaso Pasini

Topic	Time

Table 4.2: Effort Spent by Tommaso Pasini

Elia Pontiggia

Topic	Time
Architectural design	4h

Table 4.3: Effort Spent by Elia Pontiggia

Michelangelo Stasi

Topic	Time

Table 4.4: Effort Spent by Michelangelo Stasi