



POLITECNICO DI MILANO

Software Engineering 2

---

**CodeKataBattle - Improve your  
programming skills online**

Design

Document

---

*Authors:*

Nicolò Giallongo - 10764261

Giovanni Orciuolo - 10994077

Giuseppe Vitello - 10766482

# Table of Contents

<b>1 - Introduction.....</b>	<b>3</b>
1.1 - Purpose.....	3
1.2 - Scope.....	3
1.3 - Definitions, Acronyms, Abbreviations.....	3
1.4 - Revision History.....	5
1.5 - Reference Documents.....	5
1.6 - Document Structure.....	5
<b>2 - Architectural Design.....</b>	<b>6</b>
2.1 - Overview.....	6
2.2 - Component View.....	6
2.2.1 - Component Diagram.....	7
2.2.2 - Components Description.....	8
2.3 - Deployment View.....	9
2.3.1 - Deployment View Description.....	9
2.4 - Component Interfaces.....	10
2.5 - Runtime View.....	12
2.6 - Selected Architectural Styles and Patterns.....	23
2.7 - Other Design Decisions.....	23
<b>3 - User Interface Design.....</b>	<b>24</b>
3.1 - Overview.....	24
3.2 - Mockups.....	25
<b>4 - Requirements Traceability.....</b>	<b>28</b>
<b>5 - Implementation, Integration and Test Plan.....</b>	<b>28</b>
5.1 - Implementation and Integration.....	28
5.2 - Test Plan.....	29
5.2.1 - Functional Testing (Unit + Integration).....	29
5.2.2 - End to End Testing (E2E).....	30
5.2.3 - MVP testing.....	30
<b>6 - Effort Spent.....</b>	<b>30</b>
<b>7 - References.....</b>	<b>30</b>

# 1 - Introduction

## 1.1 - Purpose

The purpose of the CKB application is to provide students and educators with a platform to improve their programming skills by taking part in friendly competitions (called “Tournaments”) where they can resolve various programming problems (called “Katas”) in their preferred language of choice (e.g. Java or Python).

The platform relies heavily upon GitHub in order to store solutions and to run tests and other activities on each push. As such, the users are required to use their GitHub account to access the application (or register one if they don’t have it).

## 1.2 - Scope

In this section of the document, we will discuss the main design choices that we have taken in relation to the domain of the product.

Given that CKB is an application that aims to allow its users to compete with other users in online code competition, it seems natural to implement it as a client-server application. More specifically, we agreed on using a three-tier architecture (presentation tier, business tier and data tier) to maintain a separation between the user interface, the logic of the system and the data, and in this way improve the maintainability, the stability and the security of the system. To further improve these aspects of the CKB application we decided to use the microservice approach to develop the system. In this way we also improve the scalability of the system, allowing it to adapt more easily and efficiently to different amounts of active users.

## 1.3 - Definitions, Acronyms, Abbreviations

In order to reduce ambiguity as much as possible, this document heavily uses acronyms and abbreviations to refer to entities and concepts in the CKB system. This section is aimed at defining each of these acronyms and abbreviations in a precise and concise way.

Acronym	Definition
IU	Interested Student: a Student <u>U</u> is Interested by a notification <u>N</u> : <ul style="list-style-type: none"><li>• if <u>U</u> has selected the option to receive notification from all the users.</li><li>• if <u>U</u> has selected the option to receive notification only from their friend and the user who has sent <u>N</u> is a friend of <u>U</u>.</li><li>• <u>N</u> is a notification of a Tournament in which <u>U</u> is subscribed.</li></ul>

<b>SST</b>	Students Subscribed to a Tournament.
<b>OTC</b>	Original Tournament Creator.
<b>TC</b>	Tournament Coordinator, appointed by the OTC.
<b>CDT</b>	Current DateTime.
<b>ETD</b>	Enrollment Tournament Deadline. After this deadline, it is not possible to join the Tournament as a Student.
<b>FTD</b>	Final Tournament Deadline. After this deadline, the Tournament is ended and the final leaderboard is made available.
<b>MNS</b>	Maximum Number of Subscribers. It is the maximum number of Students that can subscribe to a specific Tournament.
<b>EBD</b>	Enrollment Battle Deadline. After this deadline, it is not possible to join the Battle as a Student.
<b>FBD</b>	Final Battle Deadline. After this deadline, the Battle is ended and the final leaderboard is made available. Moreover, the Tournament related to the Battle is updated with the new leaderboard.
<b>OME</b>	Optional Manual Evaluation. Personal score assigned by the Educator, who checks and evaluates the work done by students (the higher the better).
<b>MAE</b>	Mandatory Automated Evaluation, which includes: <ul style="list-style-type: none"> <li>• Functional aspects, measured in terms of number of test cases that pass out of all test cases (the higher the better);</li> <li>• Timeliness, measured in terms of time passed between the registration deadline and the last commit (the lower the better);</li> <li>• Quality level of the sources, extracted through Static Analysis Tools that consider multiple aspects such as security, reliability, and maintainability (the higher the better). Aspects are selected by the educator at battle creation time.</li> </ul>
<b>DAS</b>	Directly Accepted Student: a Student <u>U</u> is directly accepted in a Tournament <u>I</u> if: <ul style="list-style-type: none"> <li>• <u>I</u> is public;</li> <li>• <u>I</u> is friends only and <u>U</u> is a friend of the OCT of <u>I</u>.</li> </ul>
<b>EB</b>	End Battle: <ul style="list-style-type: none"> <li>• If OME is not required, EB occurs when the FBD is reached.</li> </ul>

	<ul style="list-style-type: none"> <li>• If OME is required, EB is after the Educator performs it, or when the FTD is reached.</li> </ul>
<b>SAT</b>	Static Analysis Tools.

## 1.4 - Revision History

Version	Date	Changelog
<b>1.0</b>	03/01/2024	First draft of the document. Completed section 1, sections 2 and 3 are work in progress.
<b>1.1</b>	06/01/2024	Added Table of Contents. Completed section 2, 3 and 5. Section 4 WIP.

## 1.5 - Reference Documents

- The specification document Assignment RDD AY 2023-2024.pdf

## 1.6 - Document Structure

This document is divided in 5 main parts:

1. **Introduction:** introduces the main architectural and design choices made and briefly explains the motivations behind them.
2. **Architectural Design:** shows in detail the architectural decisions adopted and how they are developed and combined to create the global architecture of the system.
3. **User Interface Design:** shows the design adopted for the user interface, presenting a preview of the interface using the corresponding mockups.
4. **Requirements Traceability:** shows the mapping between the requirements defined in RASD and the design elements previously discussed.
5. **Implementation, Integration and Test Plan:** shows the order in which we plan to implement subsystems and components as well as the plan of the integration and test.
6. **Effort Spent:** contains information about the number of hours each group member has worked on this document.
7. **References:** contains information about the resources used to redact this document.

## 2 - Architectural Design

### 2.1 - Overview

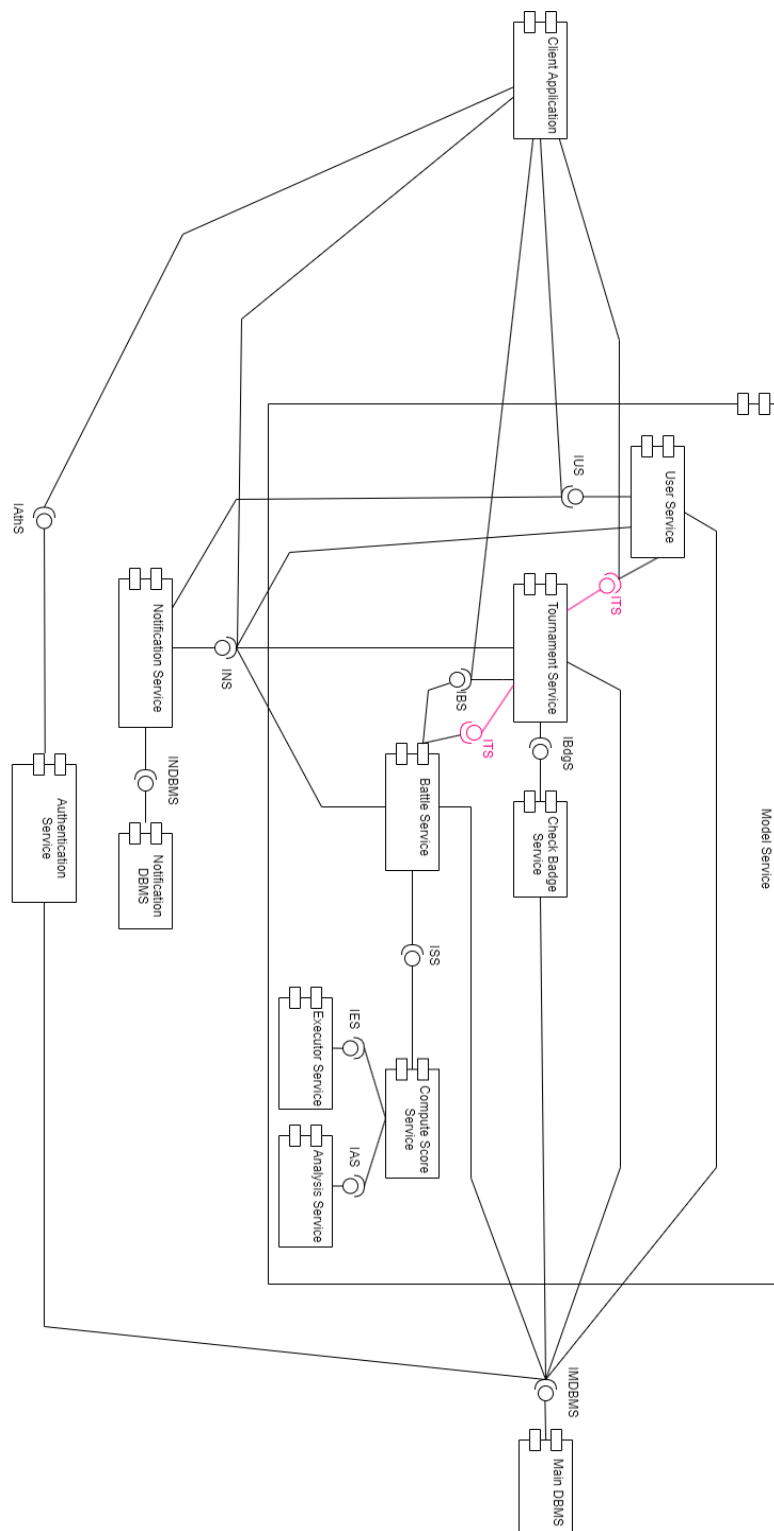
In this section, a general architecture of the CKB system is provided. The architectural choices are summarized in two main design decisions:

- **Client-Server architecture.** Consisting of a Presentation Tier, a Business Tier and a Data Tier. The Presentation Tier is the UI directly shown to the end user: its only objective is to graphically render and present the application to the user. The Business Tier handles everything related to the business logic of the system, by using the internal components (the services). The Data Tier persists the data through two databases: a PostgreSQL (Main DBMS) and a MongoDB instance (Notification DBMS). Further details are provided in section 2.3.1.
- **Microservices.** The Business Tier of the system is divided into several isolated services, in order to reduce coupling as much as possible and to improve scalability of the system as a whole.

### 2.2 - Component View

This section describes all the identified components and the relationships between them. The Component Diagram precisely shows the dependencies of all the interfaces. After that, a brief description of each component is shown.

## 2.2.1 - Component Diagram

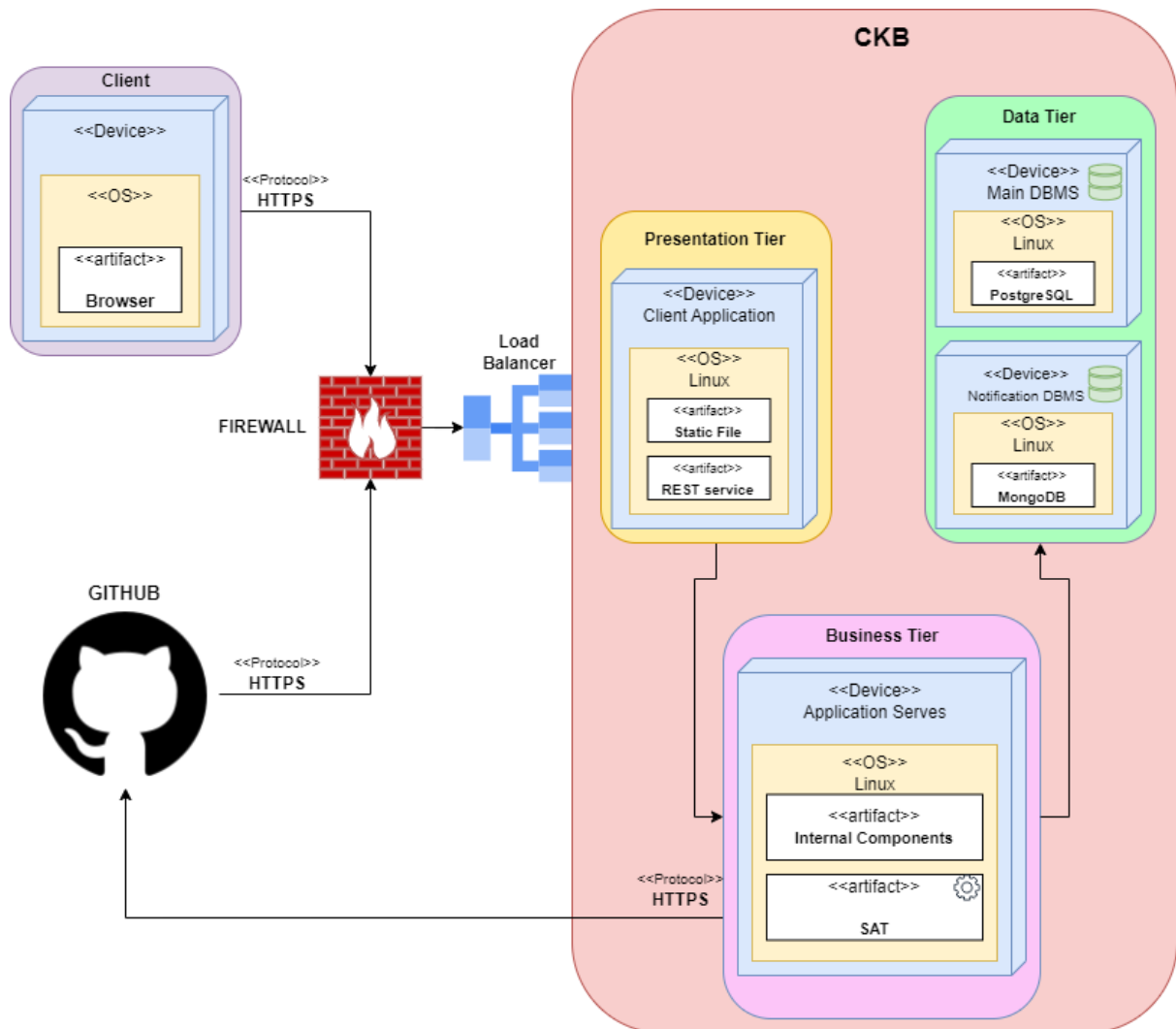


### 2.2.2 - Components Description

- **Client Application.** Represents the web application used by the final user to interface with the CKB system, so it reads the request and sends the html pages.
- **Authentication Service.** Handles authentication with GitHub following the OAuth 2.0 protocol and subsequent token verification processes.
- **User Service.** Handles registered users management, friends' list and privacy policy..
- **Tournament Service.** Handles everything related to tournaments, from creation to end of the Tournament, creation of battle.
- **Battle Service.** Handles everything related to tournament battles and their katas.
- **Executor Service.** Handles execution of code in an isolated sandbox. It is used to run kata submissions and check their output against the provided tests.
- **Analysis Service.** Handles execution of static analysis tools (SATs) in an isolated sandbox.
- **Notification Service.** Abstracts the process of delivering and storing notifications.
- **Main DBMS:** Holds and manages data about Tournaments, Battles, Users, Badges and the relationships between them. The DBMS of choice is PostgreSQL.
- **Notification DBMS:** Holds and manages data about notifications. The DBMS of choice will be a NoSQL solution (Redis).



## 2.3 - Deployment View



### 2.3.1 - Deployment View Description

**Presentation Tier:** It is the frontend layer in the 3-tier architecture, it is the user interface. Communication with the client using the REST standard. The other communication is only with the Business-Tier in order to reduce the dependence. The client sends a request via Browser, this request arrives in the Presentation-Tier it responds with html pages or messages.

It is a Nginx web server.

**Business Tier:** This is the middle tier between Presentation and Data. It receives requests by the Presentation-Tier, computes them, if needed communicate with Data-Tier in order to receive data and at the end send the information required at the Presentation-Tier.

It also can receive the code of the repository by GitHub in order to check them and communicate with the Data-Tier to update the data, and the Business-tier communicate with GitHub to authenticate the Client.

**Data Tier:** This is the tier that keeps track of all data in the system, so all Tournament, Battle, User, Notification and the relationship between them.

**Client:** It is the client that wants to interact with the system via HTTPS request.

**Firewall:** It is a device that monitors the packets incoming to the system, if a packet is potentially dangerous it is not forwarder. They are placed before the load balancers, in this way the only packets that enter the network are considered safe

**LoadBalancer:** A load balancer is a device which performs load balancing. This is the process of distributing a set of requests over a group of resources at the presentation-tier, with the intent of increasing performance, scalability and soundness of the system. It uses Nginx.

## 2.4 - Component Interfaces

In this section, for each component interface, there are listed methods with their parameter types and return value. Exceptions thrown are omitted as they are not of interest.

- **IUserService (IUS):**
  - getUser(userId: int): User
  - addFriend(notification: JSON): void
  - updateNotificationSetting(privacyLevel: PrivacyLevel): void
  - getTournamentsParticipating(userId: int): List<Tournament>
  - getTournamentsCoordinating(userId: int): List<Tournament>
  - getFriends(userId: int): List<User>
  - getBadges(userId: int): List<Badge>
  - getNotificationSetting(): PrivacyLevel
  - getInterestedUser(tournamentId: int): List<Int>
  - canSend(senderId: int, receiverId: int): bool
- **IAuthService (IAthS):**
  - getOAuthToken(accessCode: string): string
  - getUserProfile(token: string): GHUserProfile
- **ITournamentService (ITS):**
  - createTournament(tournament: Tournament): Tournament
  - joinTournament(notification: JSON): void
  - @overload joinTournament(tournamentId: int, userId: int): void ??
  - addCoordinator(notification: JSON): void
  - addBadge(badgeId: int, tournamentId: int): void
  - updateLeaderboard(leaderboard: List<Team>, tournamentId: int, battleId: int): void
  - getParticipants(tournamentId: int): List<User>
  - banParticipant(clientId: int, participantId: int, tournamentId: int): void
  - updateETD(date: DateTime, tournamentId: int, userId: int): void
  - updateFTD(date: DateTime, tournamentId: int, userId: int): void
  - Getters/Setters

- **IBattleService (IBS):**
  - createBattle(battle: Battle): Battle
  - addKata(battleId: int, kata: Kata): Battle
  - joinBattle(notification: JSON): void
  - @overload joinBattle(battleId: int, userId: int): void ??
  - getTeams(battleId: int): List<Team>
  - getParticipants(battleId: int): List<User>
  - performOME(battleId: int, teamId: int, userId: int, score: float): void
  - updateEBD(date: DateTime, battleId: int, userId: int): void
  - updateFBD(date: DateTime, battleId: int, userId: int): void
  - forceEndBattles(battleIds: List<int>): List<userId,score>
- **IScoreService (ISS):**
  - computeScore(kata: Kata, code: String, evaluationSetting: EvaluationSetting): float
- **IExecutorService (IES):**
  - runTests(test: List<KataTest>, code: String): List<KataTestResult>
- **IAnalysisService (IAS):**
  - runAnalysis(code: String, evaluationSetting: EvaluationSetting): List<float>
- **INotificationService (INS):**
  - sendFriendRequestNotify(senderId: int, receiverId: int): void
  - interactFriendRequestNotify(receiverId: int, notificationId: int, accepted: bool): void
  - sendTeamInvitationNotify(battleId: int, teamId: int, senderId: int, receiverId: int): void
  - interactTeamInvitationNotify(receiverId: int, notificationId: int, accepted: bool): void
  - sendBattleCreationNotify(battle: Battle): void
  - interactBattleCreationNotify(receiverId: int, notificationId: int, accepted: bool): void
  - sendBattleOMENotify(battle: Battle): void
  - sendBattleEndNotify(battle: Battle): void
  - sendTournamentCreationNotify(tournament: Tournament): void
  - interactTournamentCreationNotify(receiverId: int, notificationId: int, accepted: bool): void
  - sendTournamentCoordinatorNotify(senderId: int, receiverId: int): void
  - interactTournamentCoordinatorNotify(receiverId: int, notificationId: int, accepted: bool): void
  - sendTournamentEndNotify(tournament: Tournament): void
  - openNotification(userId: int): List<Notification>
  - errorTournamentFull(userId: int, tournamentId: int): void
  - sendEditTournamentDeadlineNotify(userIds: List<Int>, tournamentId: int): void
  - sendEditBattleDeadlineNotify(userIds: List<Int>, battleId: int): void

- **IBadgeService (IBdgS):**
  - getBadge(badgelId: int): Badge
  - createBadge(badge: Badge): Badge
  - checkRules(tournamentId: int): void
- **INotificationDBMS (INDBMS):**
  - writeNotification(data: JSON):void
  - writeNotification(data: JSON):void **aggiungere??**
  - getAllNotification(userId: int): List<JSON>
  - deleteNotification(notificationId: int): void
  - checkNotification(notificationId: int, receiverId:int) JSON

## 2.5 - Runtime View

In this section, there is a list of important use cases and the corresponding sequence diagram to perform them using the previously described component interfaces.

All sequence diagrams are made using Mermaid. As such, it is possible to render them graphically using the appropriate command line application.

*UC: Calculate Total Score*

```
sequenceDiagram
    participant GHA as GitHub Action
    participant ISS as IScoreService
    participant IES as IExecutorService
    participant IAS as IAnalysisService
    participant DBMS as Main DBMS

    GHA->>ISS: computeScore(kata, code, evalSettings)
    activate ISS
    ISS->>IES: runTests(tests, code)
    activate IES
    alt OK
        IES-->>ISS: List<KataTestResult>
        ISS->>IAS: runAnalysis(code, evalSettings)
        activate IAS
        alt OK
            IAS-->>ISS: SATResult
            ISS->>DBMS: persistScore(teamId, score)
        else ERR
            IAS-->>ISS: Error
            ISS-->>GHA: Error
        end
    end
    deactivate IAS
    else ERR
        IES-->>ISS: Error
    end
```

```

ISS-->>GHA: Error
deactivate IES
end
deactivate ISS

```

### UC: Registration

#### sequenceDiagram

```

participant CA as Client Application
participant GH as GitHub
participant IAUTH as IAuthService
participant DBMS as Main DBMS
CA->>GH: requestGitHubLogin()
GH->>GH: requestUserPermission()
alt ACCEPTED:
GH-->>CA: accessCode
CA->>IAUTH: getOAuthToken(accessCode)
IAUTH-->>CA: JWT
IAUTH->>GH: getUserProfile(jwt)
GH-->>IAUTH: GHUserProfile
IAUTH->>DBMS: storeUser(userProfile)
else REFUSED:
GH->>CA: showMessage("Permissions refused")
end
end

```

### UC: Visualize User Profile

#### sequenceDiagram

```

participant CA as Client Application
participant IUS as IUserService
CA->>+IUS: getUser(userId)
IUS-->>-CA: User
CA->>+IUS: getBadges(userId)
IUS-->>-CA: Badges
CA->>+IUS: getFriends(userId)
IUS-->>-CA: List<User>

```

### UC: Send Friend Request

#### sequenceDiagram

```

participant CAs as Client Application
participant INS as INotificationService
participant INDBMS as INotificationDBMS
CAs->>INS: sendFriendRequestNotify(senderId, receiverId)
INS->>INS: data = friendNotification(senderId, receiverId)
INS->>INDBMS: writeNotifcation(data)

```

### UC: Send team request

#### sequenceDiagram

participant CAs as Client application (sender)

participant INS as INotificationService

participant INDBMS as INotificationDBMS

CAs->>+INS: sendTeamInvitationNotify(battleId, teamId, senderId, receiverId)

INS->>+INS: data = teamNotification(battleId, teamId, senderId, receiverId)

INS->>+ INDBMS: writeNotifcation(data)

### UC: Accept/Reject team request

#### sequenceDiagram

participant CAr as Client application (receiver)

participant INS as INotificationService

participant INDBMS as INotificationDBMS

participant IBS as IBattleService

participant IMDBMS as IMainDBMS

CAr->>+INS: openNotification(userId)

INS->>+ INDBMS: getAllNotification(userId)

INDBMS->>+INS: List<JSON>

INS->>+ CAr: List<Notification>

CAr->>+INS: interactTeamInvitationNotify(reciverId, notificationId, accepted)

alt accepted == true:

INS->>+ INDBMS: getNotification(notificationID)

INDBMS->>+ INS: notification = JSON

INS->>+IBS: joinBattle(notification)

IBS->>+IBS: team = isInBattle(battleId, receiverId)

alt team != NULL:

IBS->>+IMDBMS: deleteTeamPartecipant(team.getId(), battleId, receiverId)

IMDBMS->>+IMDBMS: execute the delete

end

IBS->>+IMDBMS: addTeamPartecipant(teamId, battleId, reciverId)

IMDBMS->>+IMDBMS: execute the add

end

INS->>+INDBMS: deleteNotification(notificationId)

INDBMS->>+INDBMS: execute the delete

### UC: Join battle via notification

```
sequenceDiagram
    participant CA as Client application
    participant INS as INotificationService
    participant INDBMS as INotificationDBMS
    participant IBS as IBattleService
    participant IMDBMS as IMainDBMS

    CA->>+INS: openNotification(userId)
    INS->>+ INDBMS: getAllNotification(userId)
    INDBMS->>+INS: List<JSON>
    INS->>+ CA: List<Notification>
    CA->>+INS: interactBattleCreationNotify(receiverId, notificationId,
    accepted)

    alt accepted == true:
        INS->>+ INDBMS: getNotification(notificationId)
        INDBMS->>+ INS: notification = JSON
        INS->>+IBS: joinBattle(notification)
        IBS->>+IBS: team = newTeam(battleId, reciverId)
        IBS->>+IMDBMS: addTeam(team.getId(), battleId)
        IMDBMS->>+IMDBMS: execute the add
        IBS->>+IMDBMS: addTeamPartecipant(teamId, battleId, reciverId)
        IMDBMS->>+IMDBMS: execute the add
    end

    INS->>+INDBMS: deleteNotification(notificationId)
    INDBMS->>+INDBMS: execute the delete
```

### UC: Join battle via webapp

```
sequenceDiagram
    participant CA as Client application
    participant IBS as IBattleService
    participant IMDBMS as IMainDBMS

    CA->>+IBS: joinBattle(userId)
    IBS->>+IBS: team = newTeam(battleId, reciverId)
    IBS->>+IMDBMS: addTeam(team.getId(), battleId)
    IMDBMS->>+IMDBMS: execute the add
    IBS->>+IMDBMS: addTeamPartecipant(teamId, battleId, reciverId)
    IMDBMS->>+IMDBMS: execute the add
```

### UC: Join tournament via notification

```
sequenceDiagram
```

```

participant CA as Client application
participant INS as INotificationService
participant INDBMS as INotificationDBMS
participant ITS as ITournamentService
participant IMDBMS as IMainDBMS

CA->>+INS: openNotification(userId)
INS->>+ INDBMS: getAllNotification(userId)
INDBMS->>+INS: List<JSON>
INS->>+ CA: List<Notification>
CA->>+INS: interactTournamentCreationNotify(receiverId, notificationId,
accepted)
alt accepted == true:
    INS->>+ INDBMS: getNotification(notificationID)
    INDBMS->>+ INS: notification = JSON
    INS->>+ITS: joinTournament(notification)
    ITS->>+IMDBMS: addTournamentPartecipant(tournamentId, reciverId)
    IMDBMS->>+IMDBMS: execute the add
end
INS->>+INDBMS: deleteNotification(notificationId)
INDBMS->>+INDBMS: execute the delete

```

### *UC: Join tournament via notification*

```

sequenceDiagram
participant CA as Client application
participant ITS as ITournamentService
participant IMDBMS as IMainDBMS
participant INS as INotificationService
participant INDBMS as INotificationDBMS

CA->>+ITS: joinTournament(tournamentId,userId)
ITS->>+ITS: outcome = isFullTournament(tournamentId)
alt outcome == false
    ITS->>+IMDBMS: addTournamentPartecipant(tournamentId, reciverId)
    IMDBMS->>+IMDBMS: execute the add
else
    ITS->>+INS: errorTournamentFull(tournamentId, reciverId)
    INS->>+INS: data = errorTournamentFullNotification(tournamentId,
reciverId)
    INS->>+INDBMS: writeNotificaTion(data)
end
end

```



### UC: createTournament

#### sequenceDiagram

```
participant CA as Client application
participant ITS as ITournamentService
participant IMDBMS as IMainDBMS
participant INS as INotificationService
participant IUS as IUserService
participant INDBMS as INotificationDBMS
CA->>+ITS: createTournament(Tournament)
ITS->>+IMDBMS: addTorunament(Tournament)
activate IMDBMS
alt OKTournament
IMDBMS-->>ITS: Tournament
ITS->>+INS: sendTournamentCreationNotify(Tournament)
INS->>+IUS: getInterestedUsers(TounramentId)
IUS-->>INS: List<Int>
INS->>+INDBMS: writeNotifications(Data)
alt OKNotification
INDBMS-->>+INS: OK
INS-->>+ITS: OK
ITS-->>+CA: OK
else ErrorNotification
INDBMS-->>+INS: Error
INS-->>ITS: Error
ITS-->>CA: Error
end
else ErrorTournament
IMDBMS-->>ITS: NULL
ITS-->>CA: Error
end
end
```

### UC: AddingCoordinator

#### sequenceDiagram

```
participant CA as Client application
participant INS as INotificationService
participant IUS as IUserService
participant INDBMS as INotificationDBMS
CA->>+INS: sendTournamentCoordinatorNotify(sender, reciver)
INS->>+IUS: canSend(sender, reciver)
alt SendNotification
IUS-->>+INS: True
```

```

INS->>INDBMS:writeNotification(Data)
alt OK
INDBMS-->>INS:OK
INS-->>CA:OK
else ERROR
INDBMS-->>INS:Error
INS-->>CA:Error
end
else NotSendNotification
IUS-->>+INS:False
INS-->>+CA:OK
end

```

### UC: Accept/Reject TC request

```

sequenceDiagram
    participant CA as Client application
    participant INS as INotificationService
    participant INDBMS as INotificationDBMS
    participant ITS as ITournamentService
    participant IMDBMS as IMainDBMS
    CA->>+INS: interactTournamentCoordinatorNotify(receiverId,
notificationId, accepted)
    INS->>+INDBMS:checkNotification(reciverId,notificationId)
    alt OK
    INDBMS-->>+INS:Notification
        alt Accepted==TRUE
        INS->>+ITS:addCoordinator(notification)
        ITS->>+IMDBMS:writeCoordinator(tournamentId,coordinatorId)
            alt WriteOK
            IMDBMS-->>+ITS:OK
            ITS-->>+INS:OK
            INS-->>+CA:OK
        else WriteError
            IMDBMS-->>+ITS:Error
            ITS-->>+INS:Error
            INS-->>+CA:Error
        end
    else Accepted==False
    INS-->>+CA:OK
    end
end

```

### UC: CreateBadge

```
sequenceDiagram
    participant CA as Client application
    participant IBdgS as IBadgeService
    participant IMDBMS as IMainDBMS
    CA->>+IBdgS: createBadge (Badge)
    IBdgS->>IMDBMS: writeBadge (Badge)
    alt WriteOK
        IMDBMS-->>IBdgS: OK
        IBdgS-->>CA: OK
    else WriteError
        IMDBMS-->>IBdgS: Error
        IBdgS-->>CA: Error
    end
end
```

### UC: CreateBattle

```
sequenceDiagram
    participant CA as Client application
    participant IBS as IBattleService
    participant IMDBMS as IMainDBMS
    participant INS as INotificationService
    participant ITS as ITournamentService
    participant INDBMS as INotificationDBMS
    CA->>+IBS: createBattle (Battle)
    IBS->>+IMDBMS: addBattle (Battle)
    alt OKBattle
        IMDBMS-->>IBS: Battle
        IBS->>+INS: sendBattleCreationNotify (Tournament)
        INS->>+ITS: getParticipant (TournamentId)
        ITS-->>INS: List<Int>
        INS->>+INDBMS: writeNotifications (Data)
        alt OKNotification
            INDBMS-->>+INS: OK
            INS-->>+IBS: OK
            IBS-->>+CA: OK
        else ErrorNotification
            INDBMS-->>+INS: Error
            INS-->>IBS: Error
            IBS-->>CA: Error
        end
    end
    alt ErrorBattle
        IMDBMS-->>IBS: NULL
        IBS-->>CA: Error
    end
end
```

end

### UC: Perform OME (part 1)

#### sequenceDiagram

```
participant CA as Client application
participant IBS as IBattleService
participant ITS as ITournamentService
participant IMDBMS as IMainDBMS
participant INS as INotificationService
CA->>+IBS: performOME(battleId, teamId, userId, score)
IBS->>+IMDBMS: updateScoreOME(battleId, teamId, userId, score)
alt UpdateOK
alt CompleteForEachStudents
IMDBMS-->>+IBS: List<Team>,tournamentId
IBS->>+ITS: updateLeaderboard(leaderboard,tournamentId,battleId)
ITS->>+IMDBMS:updateLeaderboard(leaderboard,tournamentId,battleId)
alt UpdateLeaderboardOK
IMDBMS-->>+ITS: OK
ITS->>+INS: EndBattle
INS->>+ITS: ...
ITS-->>+IBS: OK\Error
IBS-->>+CA: OK\Error
else UpdateLeaderboardError
IMDBMS-->>+ITS: Error
ITS-->>+IBS: Error
IBS-->>+CA: Error
end
else NotCompleteForEachStudents
IMDBMS-->>+IBS: OK
IBS-->>+CA: OK
end
else UpdateError
IMDBMS-->>+IBS: Error
IBS-->>+CA: Error
end
end
```

### UC:EndBattle

#### sequenceDiagram

```
participant CA as Client application
participant IBS as IBattleService
```

```

participant ITS as ITournamentService
participant IMDBMS as IMainDBMS
participant INS as INotificationService
participant INDBMS INotificationDBMS
IBS->>+ITS: updateLeaderboard(leaderboard,tournamentId,battleId)
ITS->>+IMDBMS:updateLeaderboard(leaderboard,tournamentId,battleId)
alt UpdateLeaderboardOK
IMDBMS-->>+ITS: OK
ITS->>+INS:sendBattleEndNotify(Battle)
INS->>+INDBMS:writeNotifications(Notifications)
alt WriteOk
INDBMS-->>+INS:OK
INS-->>+ITS:OK
else WriteError
INDBMS-->>+INS:Error
INS-->>+ITS:Error
end
ITS-->>+IBS: OK\Error
IBS-->>+CA: OK\Error
end

```

### UC: Edit deadline

(This is ETD is equal for FTD, for EBD and FBD change only ITournamentService with IBattleService)

```

sequenceDiagram
    participant CA as Client application
    participant ITS as ITournamentService
    participant IMDBMS as IMainDBMS
    participant INS as INotificationService
    CA->>+ITS: updateETD(date,tournamentId,userId)
    ITS->>+IMDBMS: updateETD(date,tournamentId,userId)
    alt UpdateOk
    IMDBMS-->>+ITS: OK
    ITS->>+INS:sendEditTournamentDeadlineNotify(userIds,tournamentId)
    alt WriteOK
    INS-->>ITS:OK
    ITS-->>CA:OK
    else WriteError
    INS-->>ITS:Error
    ITS-->>CA:Error
    end
    end
    else UpdateError
    IMDBMS-->>+ITS: Error

```

```
ITS-->>CA:Error
end
```

### UC: Ban Student

```
sequenceDiagram
    participant CA as Client application
    participant ITS as ITournamentService
    participant IMDBMS as IMainDBMS
    CA->>ITS: banParticipant(clientId,participantId,tournamentId)

    ITS->>IMDBMS: deleteParticipant(clientId,participantId,tournamentId)
    alt DeleteOk
        IMDBMS-->>ITS: OK
        ITS-->>CA: OK
    else DeleteError
        IMDBMS-->>ITS: Error
        ITS-->>CA: Error
    end
end
```

### UC: FBD expired

```
sequenceDiagram
    participant ModelService
    participant IBS as IBattleService
    participant ITS as ITournamentService
    participant INS as INotificationService
    participant INDBMS as INotificationDBMS
    ModelService-->>IBS: FBDEexpired
    alt NotOME
        IBS-->>ITS: EndBattle
        ITS-->>IBS: ...
        IBS-->>ModelService: OK\Error
    end
    alt OME
        IBS->>INS: sendBattleOMENotify(Battle)
        INS->>INDBMS: writeNotifications(Notifications)
        alt WriteOK
            INDBMS-->>INS: OK
            INS-->>IBS: OK
            IBS-->>ModelService: OK
        else WriteError
            INDBMS-->>INS: Error
            INS-->>IBS: Error
        end
    end
```

```
IBS-->>+ModelService:Error  
end  
end
```

*UC: FTD expired so EndTournament*

## 2.6 - Selected Architectural Styles and Patterns

The main architectural style adopted for the system is Microservices Architecture.

It was chosen for the following main reasons:

- **Scalability:** Microservices can be independently deployed on the cloud. This means that it is possible to dynamically scale the number of instances of each services, depending on the current demand of the users.
- **Availability:** If one or more services fail, the entire system keeps working while autoheal procedures are running to restore the failed services.
- **Maintainability:** Since each service is implemented separately in its own repository, this increases maintainability by reducing coupling and allows multiple teams to work in parallel.
- **Security:** Each microservice communicates with the others only through methods exposed on the interface, in this way a microservice can't access inner methods and parameters of other microservices, and can communicate with DBMSs only through the corresponding microservices.

## 2.7 - Other Design Decisions

The ERD for the Main DBMS is available at the following URL:

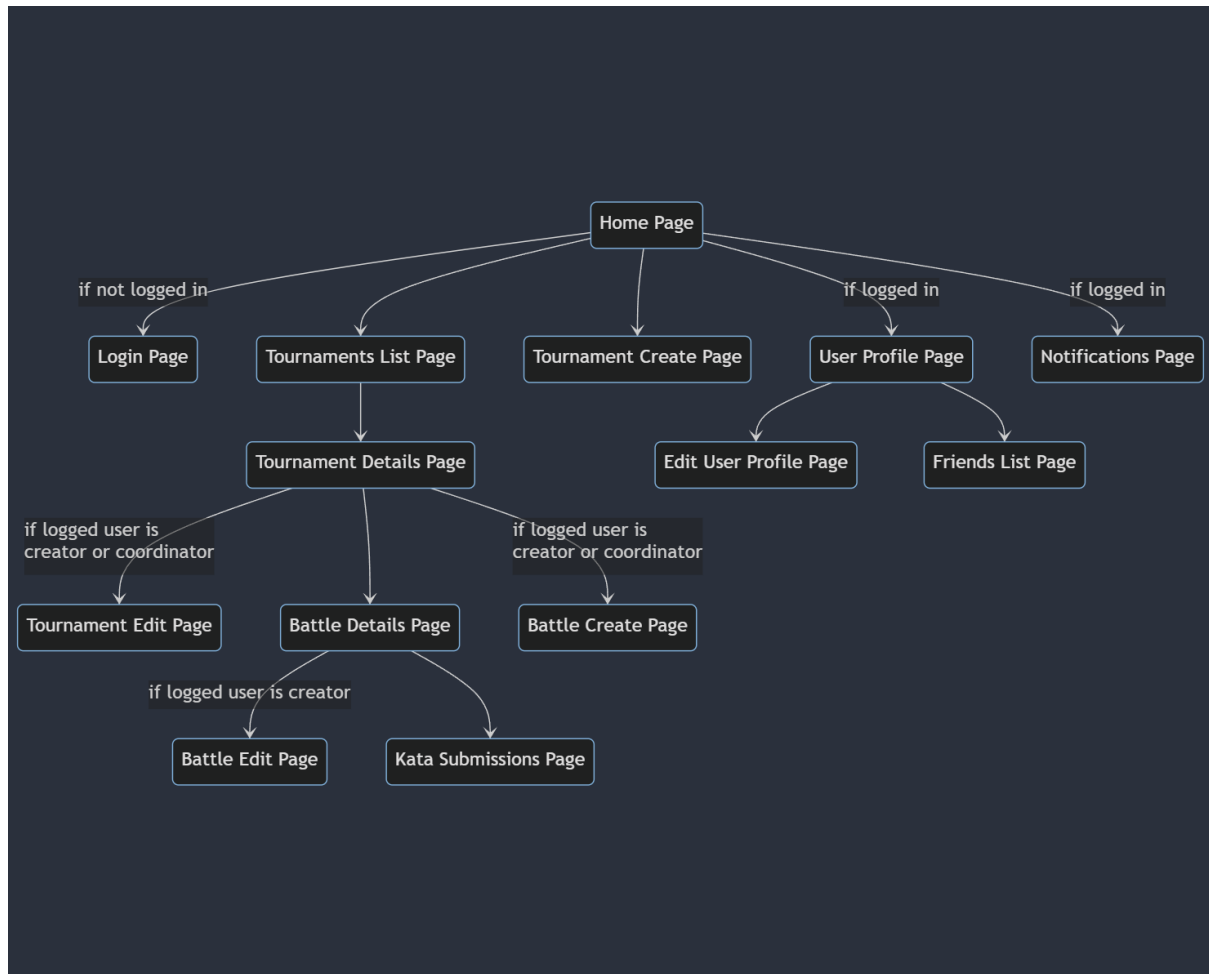
[https://editor.ponyorm.com/user/giovanni\\_or2/CodeKataBattle](https://editor.ponyorm.com/user/giovanni_or2/CodeKataBattle)

## 3 - User Interface Design

This section is dedicated to presenting a general overview of the user interface of the CodeKataBattle system, as well as showing some mockups mainly focused on UX rather than UI, since that aspect will be handled by focus groups.

### 3.1 - Overview

The following is a state diagram indicating a general sitemap of the platform.



#### stateDiagram-v2

```
home: Home Page
login: Login Page
tournament_list: Tournaments List Page
tournament_create: Tournament Create Page
tournament_details: Tournament Details Page
tournament_edit: Tournament Edit Page
battle_details: Battle Details Page
battle_edit: Battle Edit Page
battle_create: Battle Create Page
kata_submission: Kata Submissions Page
```



```
user_profile: User Profile Page
edit_profile: Edit User Profile Page
notifications: Notifications Page
friends_list: Friends List Page
home --> login: if not logged in
home --> tournament_list
home --> user_profile: if logged in
home --> notifications: if logged in
home --> tournament_create
tournament_list --> tournament_details
tournament_details --> battle_details
    tournament_details --> tournament_edit: if logged user is\ncreator
or coordinator
    tournament_details --> battle_create: if logged user is\ncreator or
coordinator
    battle_details --> battle_edit: if logged user is creator
    battle_details --> kata_submission
user_profile --> edit_profile
user_profile --> friends_list
```

## 3.2 - Mockups

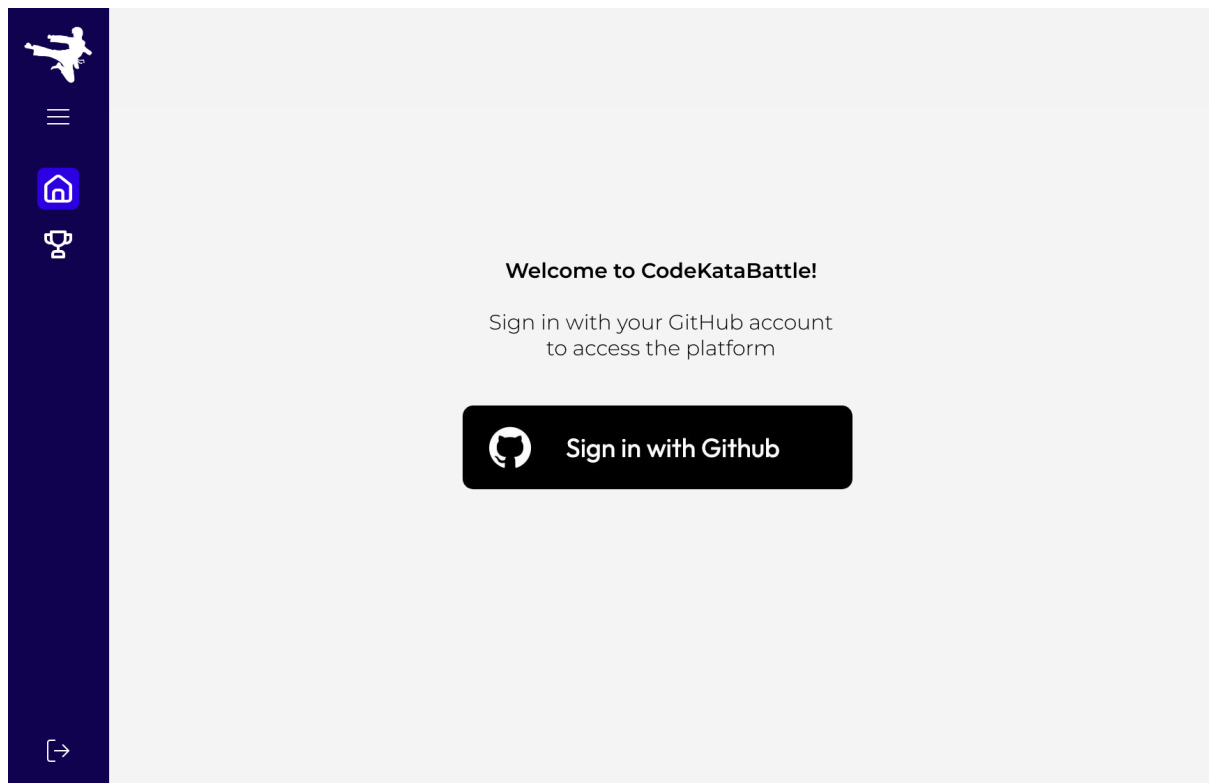
The mock-ups serve to indicate to the team developing the Client Application a general direction for the UX and style of the application (regarding inputs and typography). Not all pages are shown, as those will be developed by focus groups.

<https://www.figma.com/file/T3paTQUdTwdw49rkhTvXBL/Frontend-Mockups?type=design&node-id=0%3A1&mode=design&t=t3KtYBauaHz5l9eE-1>

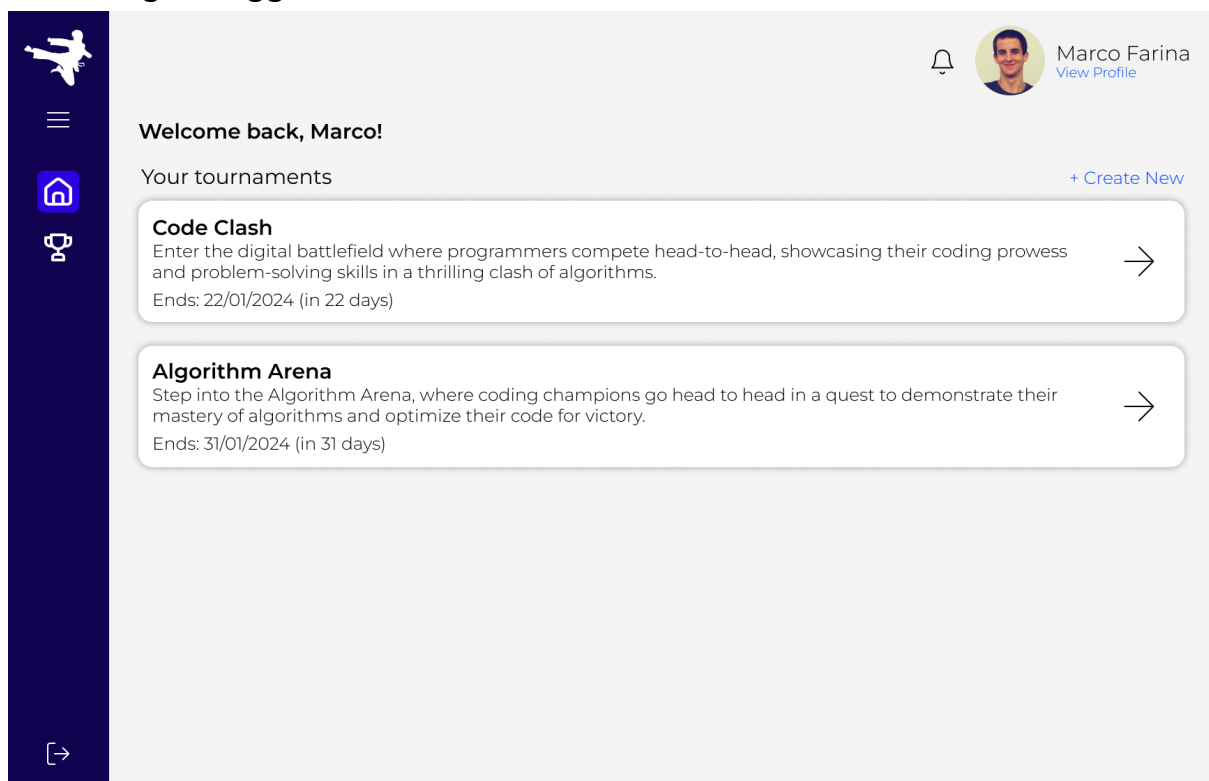
The structure of the application is composed of a simple sidebar and navbar. The sidebar allows users to navigate between different sections of the application, while the navbar allows users to visit the logged in user profile and view unread notifications.

The following images are rendered from Figma:






## Home Page - Not logged in





## Home Page - Logged in



## Upcoming Tournaments







Marco Farina  
[View Profile](#)

### Upcoming Tournaments


**Advanced Python Trials**  
Show off your Python skills!  
Enrollment Deadline: 02/01/2024 (in 2 days)  
104 students subscribed (max 150)




**The Ultimate Go Challenge**  
Improve your Go skills by competing with others  
Enrollment Deadline: 05/01/2024 (in 5 days)  
55 students subscribed (max 100)








**Problem Solving Trials**  
Solve problems with various languages and compete to improve your programming skills  
Enrollment Deadline: 10/01/2024 (in 10 days)  
21 students subscribed (max 200)





**10 Days with JavaScript**  
Compete with other students to improve your knowledge of JavaScript  
Enrollment Deadline: 14/01/2024 (in 14 days)  
10 students subscribed (max 500)



## Create New Tournament





Marco Farina  
[View Profile](#)

### Create Tournament




Title  
Code Clash


Description  
Enter the digital battlefield where programmers compete head-to-head, showcasing their coding prowess and problem-solving skills in a thrilling clash of algorithms.

Enrollment Deadline  
25/01/2024

Closing Deadline  
26/02/2024






Privacy Level  
Public



Coordinators:  [+ Invite coordinators](#)

Badges:  [+ Add badges](#)

Create

## Tournament Details



**Marco Farina**  
[View Profile](#)

### Code Clash

Enter the digital battlefield where programmers compete head-to-head, showcasing their coding prowess and problem-solving skills in a thrilling clash of algorithms.

**Battles** **Leaderboard**

**Battle #1 - Status: In Progress**  
Language: Python  
Final Deadline: 02/01/2024 (in 2 days)  
20 teams subscribed (min 2 per team, max 4 per team) →

**Battle #2 - Status: Enrolling**  
Language: JavaScript  
Enrollment Deadline: 10/01/2024 (in 10 days)  
30 teams subscribed (min 1 per team, max 5 per team) →

**Battle #3 - Status: Enrolling**  
Language: JavaScript  
Enrollment Deadline: 20/01/2024 (in 20 days)  
45 teams subscribed (min 1 per team, max 5 per team) →

## 4 - Requirements Traceability

## 5 - Implementation, Integration and Test Plan

### 5.1 - Implementation and Integration

In the implementation phase it is important that the Product Owner follows each step of the implementation in order to guarantee that requirements are being fulfilled correctly.

Regarding the implementation itself, first it is needed to deploy MainDBMS and NotificationDBMS (as they are external applications), after that the implementation can start.

The approaches for implementation are Bottom-up and Thread.

Some components are independent of others, so they are possible to implement using the Thread approach.

–Spiegare Thread–

For the Bottom-up it is possible to implement a few parts of the system in order to reduce the complexity.

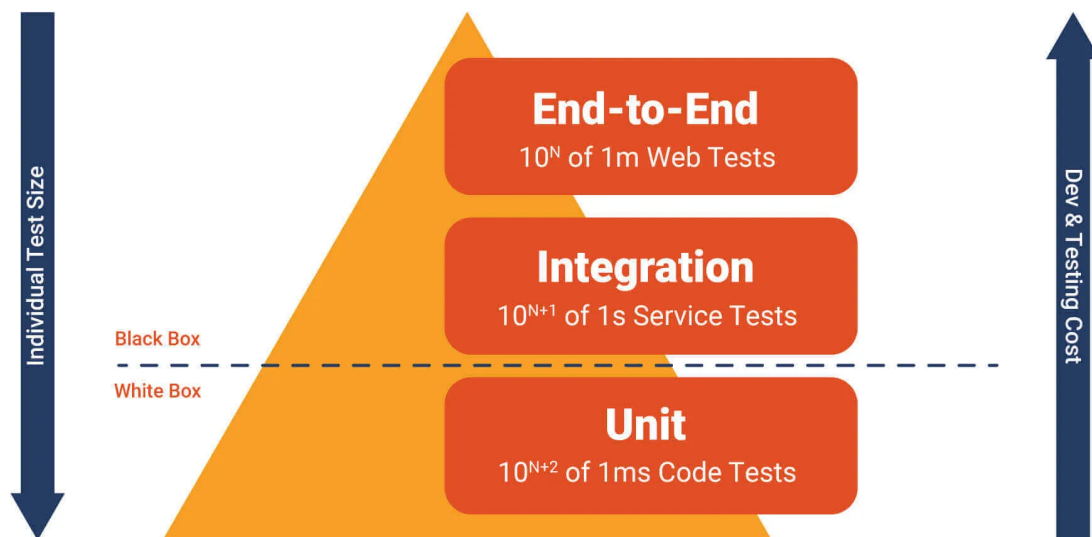
### –Bottom-up–

Order of services implementation (In the same level it is possible using Thread):

- 1) ExecutorService, AnalysisService, BadgeService, AuthenticationService, NotificationService;
- 2) UserService, TournamentService + BattleService, CalculateScoreService;
- 3) ClientApplication.

## 5.2 - Test Plan

The approach that will be followed for writing tests is the famous pyramidal approach, where there are many unit tests followed by fewer integration tests and, lastly, only a very small amount of E2E tests.



### 5.2.1 - Functional Testing (Unit + Integration)

First, the CKB system will be tested by using the properly written unit tests for each module. These tests aim to show the exactness and soundness of the codebase at the level of singular functions, by mocking every dependency. These tests are very short and their execution should be as fast as possible, in order to let the developer iterate quickly without waiting for unit tests to execute.

The next step will be integration testing. These tests span across multiple components, are longer and slower compared to unit tests, and aim to show that components are communicating correctly (following the correct format of inputs and outputs) and no unexpected errors are being thrown.

### 5.2.2 - End to End Testing (E2E)

The last type of tests to be implemented are End to End tests (E2E), which aim to test end user interactions with the entire system through the UI (Client Application component). These tests require the emulation of an entire browser environment and simulation of user interactions. As such, they are very slow and consume a lot of resources, so there must be a small number of them implemented. They show the correctness of entire application flows at once.

### 5.2.3 - MVP testing

Lastly, the MVP of the application will be tested by a small, selected group of schools interested in adopting the solution for their students. These tests will be performed manually and will aim to identify possible sources of confusion in the UI/UX of the program, and solve them before releasing the final version of the product.

## 6 - Effort Spent

Group Member	Effort Spent (hours)
Nicolò Giallongo	Introduction: 3h Architectural Design: 6h User Interface Design: 1h Requirements Traceability: Implementation, Integration and Test Plan:
Giovanni Orciuolo	Introduction: 1h Architectural Design: 5h User Interface Design: 6h Requirements Traceability: Implementation, Integration and Test Plan: 1h
Giuseppe Vitello	Introduction: 1h Architectural Design: 6h User Interface Design: 1h Requirements Traceability: Implementation, Integration and Test Plan: 1h

## 7 - References

- Version Control System: GitHub (<https://github.com>)
- State diagrams made with: Mermaid (<https://mermaid.js.org>)
- Sequence diagrams made with: Mermaid (<https://mermaid.js.org>)
- Mock-ups made with: Figma (<https://www.figma.com/>)
- Document written with: Google Documents (<https://docs.google.com>)