

DD

Federico Valentino, Nicola Zarbo

October 2023

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Purpose . . . . .	3
1.2	Scope . . . . .	3
1.3	Definitions, Acronyms, Abbreviations . . . . .	4
1.4	Revision History . . . . .	4
1.5	Reference Documents . . . . .	4
1.6	Document Structure . . . . .	4
<b>2</b>	<b>Architectural Design</b>	<b>5</b>
2.1	Overview . . . . .	5
2.2	Component View . . . . .	7
2.2.1	Component Diagram . . . . .	7
2.2.2	Components description . . . . .	8
2.3	Deployment view . . . . .	11
2.4	Runtime view . . . . .	12
2.4.1	Sign Up . . . . .	12
2.4.2	Sign In . . . . .	13
2.4.3	Create Tournament . . . . .	14
2.4.4	Subscribe To Tournament . . . . .	15
2.4.5	Create Battle . . . . .	16
2.4.6	Create Repository . . . . .	17
2.4.7	Join Battle . . . . .	18
2.4.8	Score Commit . . . . .	19
2.4.9	View Battle Ranking . . . . .	20
2.4.10	Manual Evaluation . . . . .	21
2.4.11	Look at Tournament Leaderboard . . . . .	22
2.4.12	Close Tournament . . . . .	23
2.5	Component interfaces . . . . .	24
2.5.1	Message Broker API . . . . .	24
2.5.2	DBMS API . . . . .	24
2.5.3	RESTful API . . . . .	26
2.6	Selected architectural styles and patterns . . . . .	28
2.6.1	Microservices . . . . .	28
2.6.2	Hybrid architecture (REST and EBA) . . . . .	28

2.7	Other design decisions . . . . .	28
<b>3</b>	<b>User Interface Design</b>	<b>29</b>
3.1	General Overview . . . . .	29
<b>4</b>	<b>Requirements Traceability</b>	<b>37</b>
4.0.1	Functional requirement traceability . . . . .	37
<b>5</b>	<b>Implementation, Integration and Test Plan</b>	<b>41</b>
5.1	Implementation & Integration . . . . .	41
5.2	Test Plan . . . . .	41
<b>6</b>	<b>Effort spent</b>	<b>43</b>
<b>7</b>	<b>References</b>	<b>45</b>
7.1	Used tools . . . . .	45

# Chapter 1

## Introduction

### 1.1 Purpose

CodeKataBattle (CKB) is a new platform that helps students improve their software development skills by training with peers on code katas . Educators use the platform to challenge students by creating code kata battles in which teams of students can compete against each other, thus proving (and improving) their skills.

A code kata battle is essentially a programming exercise in a programming language of choice (e.g., Java, Python). The exercise includes a brief textual description and a software project with build automation scripts (e.g., a Gradle project in case of Java sources) that contains a set of test cases that the program must pass, but without the program implementation. Students are asked to complete the project with their code. In particular, groups of students participating in a battle are expected to follow a test-first approach and develop a solution that passes the required tests. Groups deliver their solution to the platform (by the end of the battle). At the end of the battle, the platform assigns scores to groups to create a competition rank.

### 1.2 Scope

The scope of this document is to provide a detailed description of how our system will be implemented to meet the requirements defined in the RASD. It outlines the overall architecture of CKB including all of its modules, interfaces and interactions. Furthermore it provides an integration and testing plan.

### 1.3 Definitions, Acronyms, Abbreviations

Acronym	Definition
CKB	CodeKataBattle
CK	Code Kata
UI	User Interface
SOA	Service Oriented Architecture
API	Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheet
JS	JavaScript
DBMS	DataBase Management System

Table 1.1: Acronyms used in the document

### 1.4 Revision History

### 1.5 Reference Documents

- RASD Document for CKB

### 1.6 Document Structure

The document is divided into 6 main sections:

- The first section is a brief introduction to the document, the purpose of the system, the scope and various definitions.
- The second section provides the chosen architectural design for the system. Here we describe every major component and how they interact with each other.
- The third section contains various mockups for the User Interface (UI).
- The fourth section describes how the current design maps to the requirements previously defined in the RASD document.
- The fifth section provides an implementation plan for the entire system.
- Lastly, the sixth section contains the effort spent by every group member.

## Chapter 2

# Architectural Design

### 2.1 Overview

The main aspects of the design are:

- **Microservices.** The system architecture is Service Oriented Architecture (SOA). In particular it follows the Microservices architecture.
- **Event-Based Architecture.** The service to service communication is thought with an event based fashion. Each component is either a producer or consumer of an event, for example when a tournament is created one module will advertise it and another module will see the event and notify users.
- **Restful API.** The user communicates with the system by using REST interfaces provided by the front end services.

The system also exploits a third party email service to communicate notification to the user and uses GitHub APIs for the management of battle repository. The following diagram represents how the system interacts with the world.

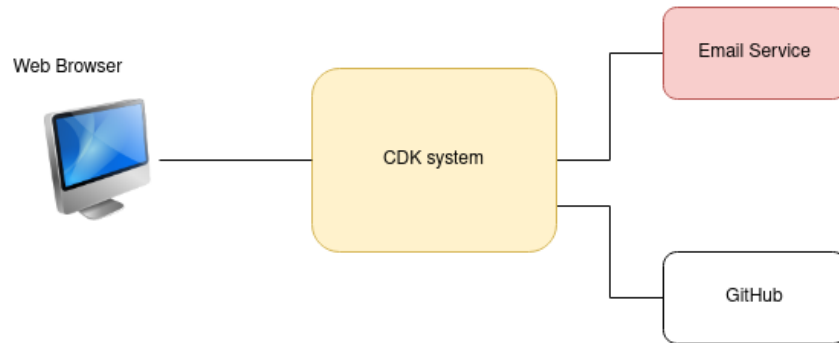


Figure 2.1: High Level View Diagram

## 2.2 Component View

The component diagram shows all the identified components and their interactions. We provide two different diagrams:

- In the first one we show how the various microservices interact with each other
- In the second one we show how the microservices interact with the external world.

After the diagrams a brief description of all the components will follow.

### 2.2.1 Component Diagram

#### System Interaction

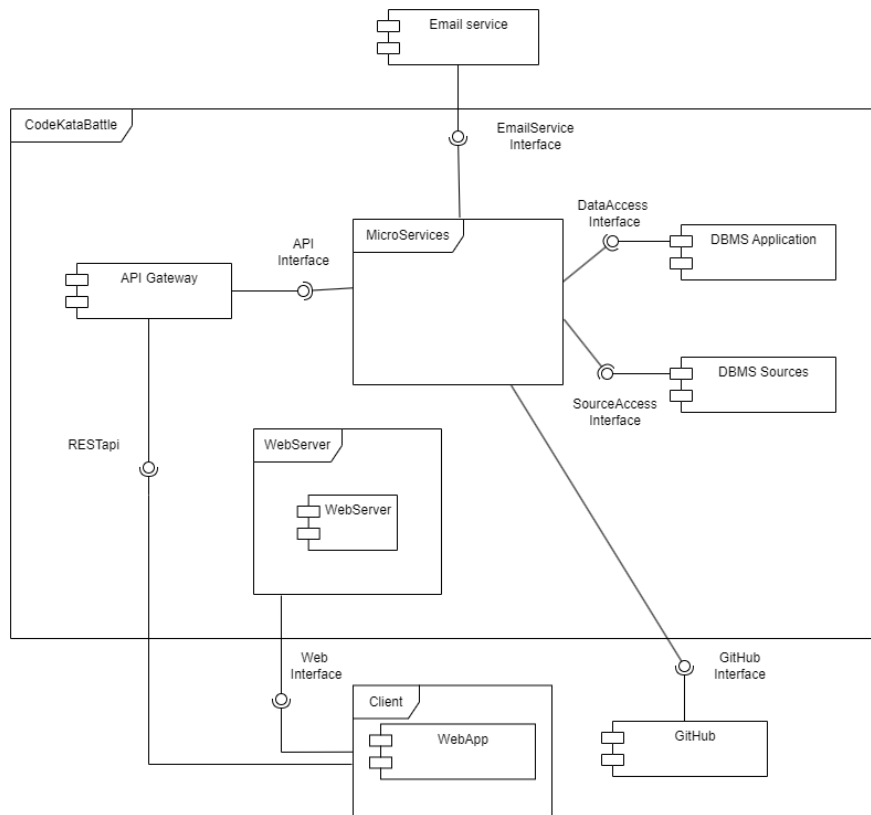


Figure 2.2: CKB component diagram



### Microservices Interaction

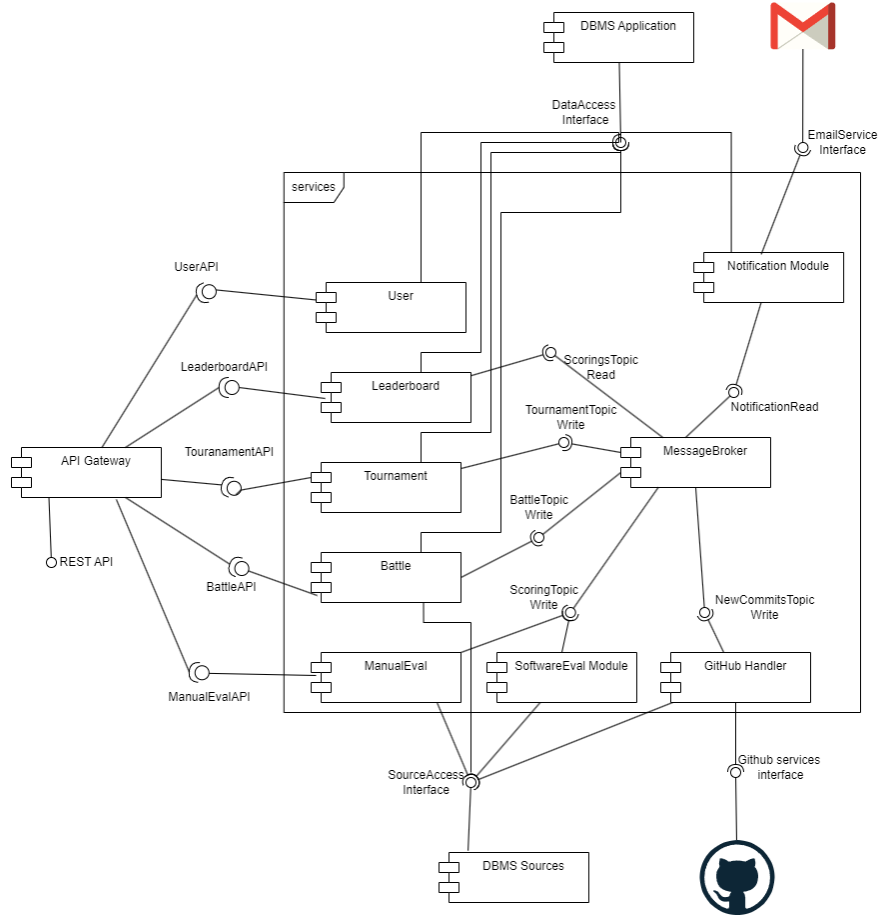


Figure 2.3: CKB Microservices Interactions

#### 2.2.2 Components description

The components are:

- GitHub handler : Interfaces with GitHub APIs in order to:
  - create a Repository for a Battle after the subscription deadline
  - retrieve the source code from a group after any commit and store it on a DataBase so that it can be automatically evaluated and then manually evaluated if needed during the consolidation phase

- Message broker : It is the component tasked with dealing with messages from other services to establish the event based communication between services. It keeps message queues for the various topics which some services produce and others are subscribed to.
- Email service : It is a third party service, used to send users various notifications.
- User service: this component handles all the login and registration logic but not the authentication one.
- API gateway : It handles client access to the various system services, it also handles the authorization aspects of the services use, to ensure system security.
- Web server : It interfaces with the client browser and responds to its requests with the needed web pages(html+css+js).
- WebApp : The part of the application that runs on the client browser. It is made of a set of web pages which are able to make requests to the web server and the services the system provides.
- Github : It is the third party application that handles the battle repositories.
  - The student users use its services outside the CKB scope to fork a CodeKata (CK) assignment and work on its solution.
  - The CKB system interfaces with some Github services in order to create the repository and retrieve the solutions committed by students for evaluation.
- DBMS (Application) : It is the DBMS that provides access to the database containing all the information about users, tournaments, battles and the scoring.
- DBMS (Sources) : The DBMS for the database that keeps the source code related to any groups in any battles. This database is separated from the other in order to :
  - Optimize its performance , since the dimension of its records can be possibly way bigger.
  - Have better control on the source codes to be stored, since they will run inside the CKB system for evaluation, making it a security concern.
- Tournament Service: The tournament service handles all aspects of a tournament, from the creation to the joining of students.
- Battle Service: The battle service handles all aspects of a battle, from the creation to the joining of students.

- Notification Service: The notification Service handles all the user notifications. Students and Educators are gonna get email notification from this service whenever: a new tournament is created, a new battle is created, a final rank for a battle is available and a final battle for a tournament is available.
- Leaderboard Service: The service handles all kinds of leaderboards present in the system. It exchanges messages with ManualEval and SoftwareEval and through those it updates the leaderboards of battles and tournaments inside the Application DBMS.
- ManualEval Service: The component handles the consolidation stage of a battle(if it was required). It gives access to a group's sources in order to add a new score for the battle the group is participating.
- SoftwareEval Service: The component handles the automatic scoring of new commits to the group's repository. Whenever a student commits to the repository GitHub sends a notification to the GithubHandler service which downloads the sources and signals to this component the neediness of an Evaluation

## 2.3 Deployment view

The following deployment diagram shows how all the components are distributed and how they interact with each other.

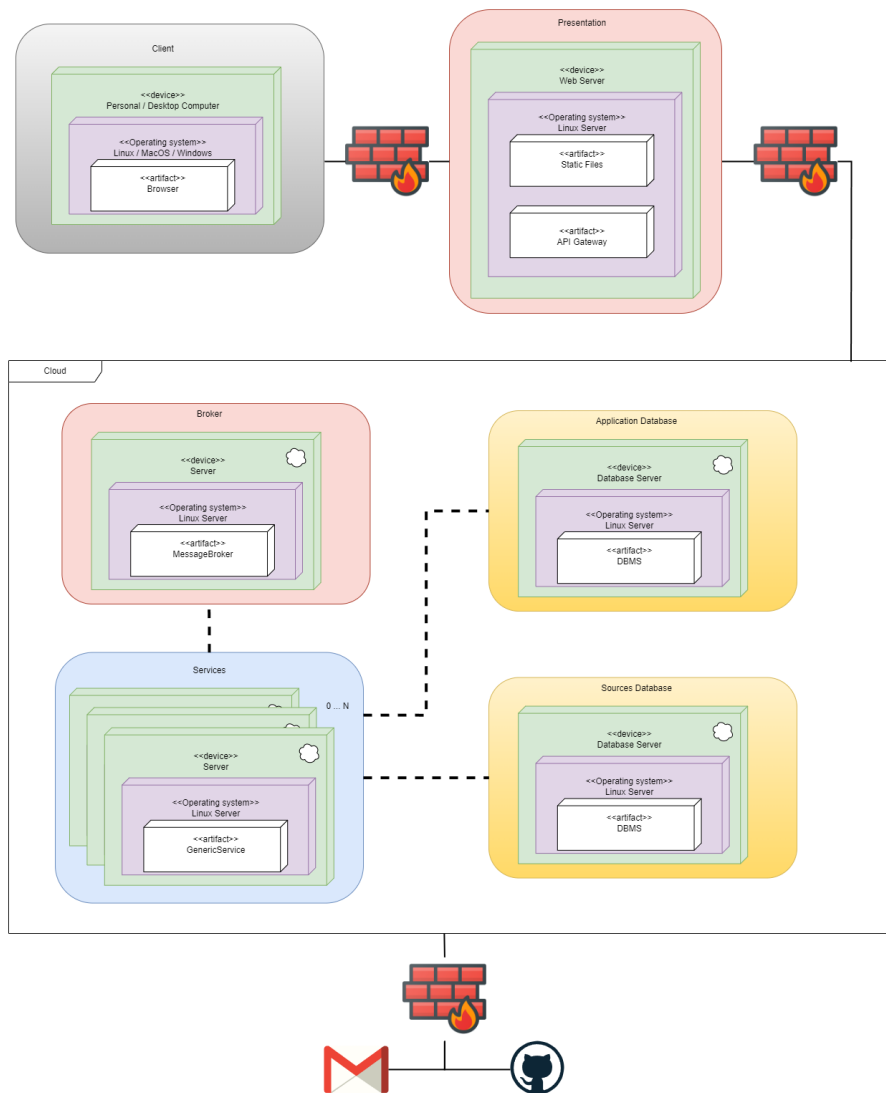


Figure 2.4: CKB Deployment diagram

## 2.4 Runtime view

Here we present the dynamics of our system through the use of sequence diagrams. All the interaction that use REST APIs start with the client request to an API that may contain a number of parameters, which are listed in the interface description of the API in the apposite document section, and end with either a positive response, eventually containing requested data and/or hyper-link for contextual page navigation (following REST's HATEOAS principle), or a negative response in case of any error. For reading simplicity only positive responses are shown in the diagrams, as any other error response sent to the client always results in an error alert in the client application page. In the interactions where the notification service is involved an email containing a message for the users is sent. The message content and destination is in the diagram description. In some complex diagrams where the message broker is involved, a more specific description of its interaction with the other services is provided.

### 2.4.1 Sign Up

After the user has signed up, the positive REST response contains the hypertext to redirect the client to the tournament's main page.

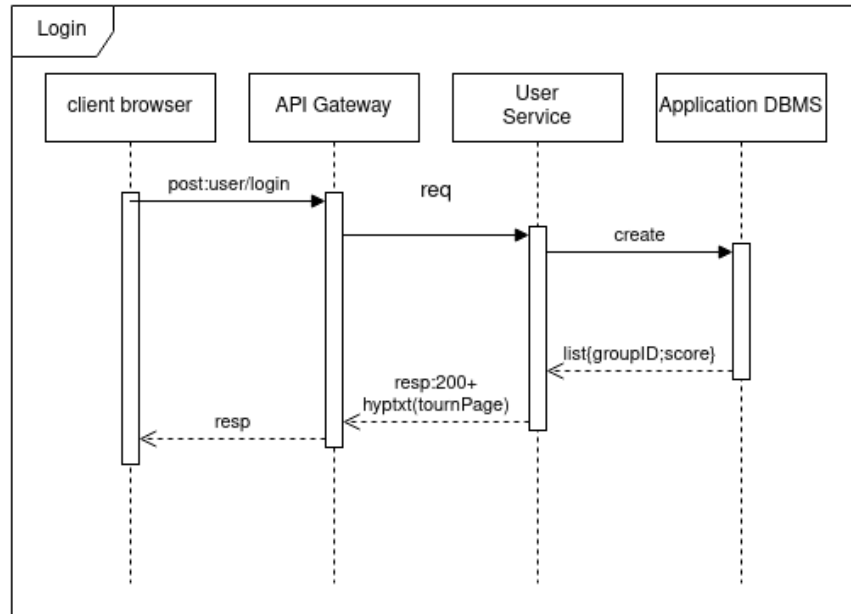


Figure 2.5: Sign up sequence diagram

### 2.4.2 Sign In

After the user has signed in, the positive rest response contains the hypertext to redirect the client to the tournaments main page.

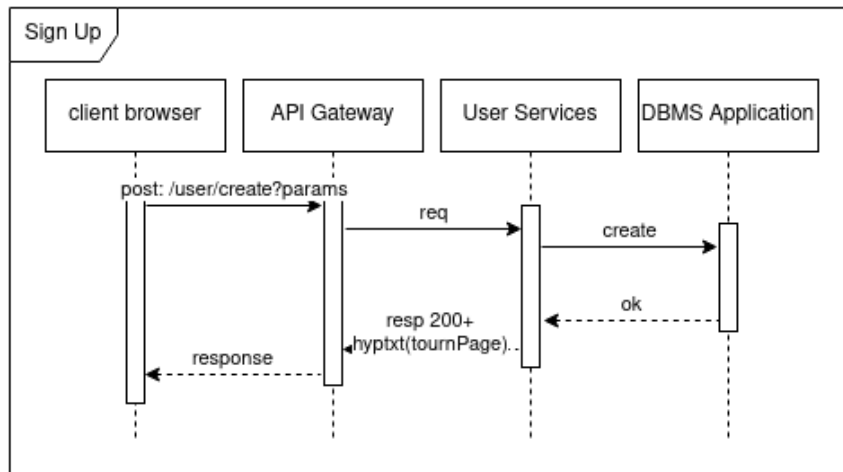


Figure 2.6: Sign in sequence diagram

### 2.4.3 Create Tournament

Once the tournament has been created, the client is redirected to the new tournament page and an email is sent to all CDK student users notifying them of the new tournament existence.

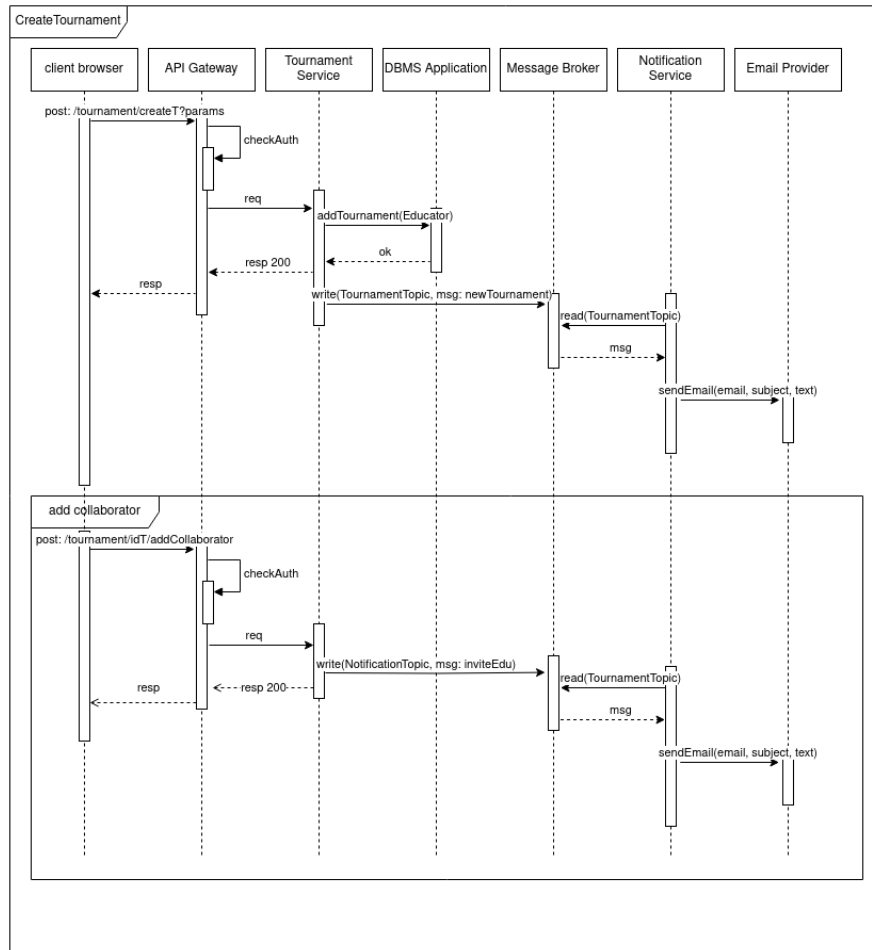


Figure 2.7: Create Tournament sequence diagram

### 2.4.4 Subscribe To Tournament

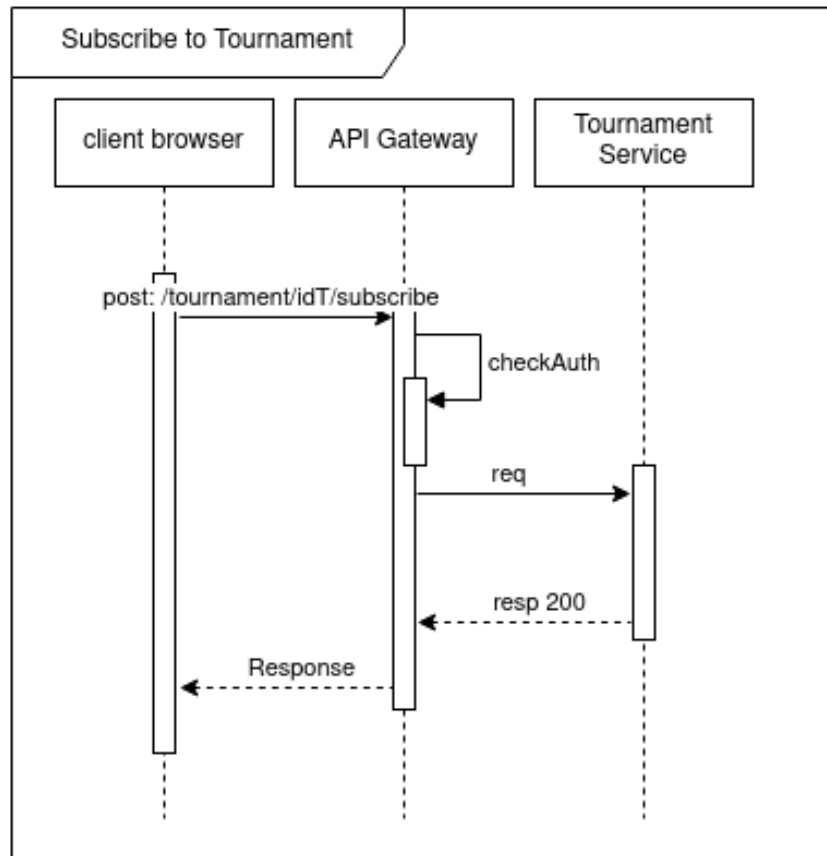


Figure 2.8: Subscribe To Tournament sequence diagram



### 2.4.5 Create Battle

At the end of the interaction the systems response contains the hyperlink to the newly created battle resource. An email is also sent to all the student users that are subscribed to the tournament, notifying them of the new available battle.

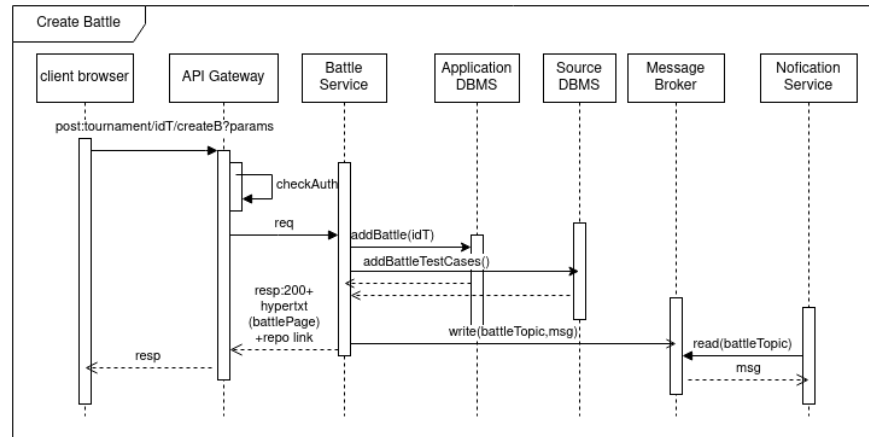


Figure 2.9: Create Battle sequence diagram

2.4.6 Create Repository

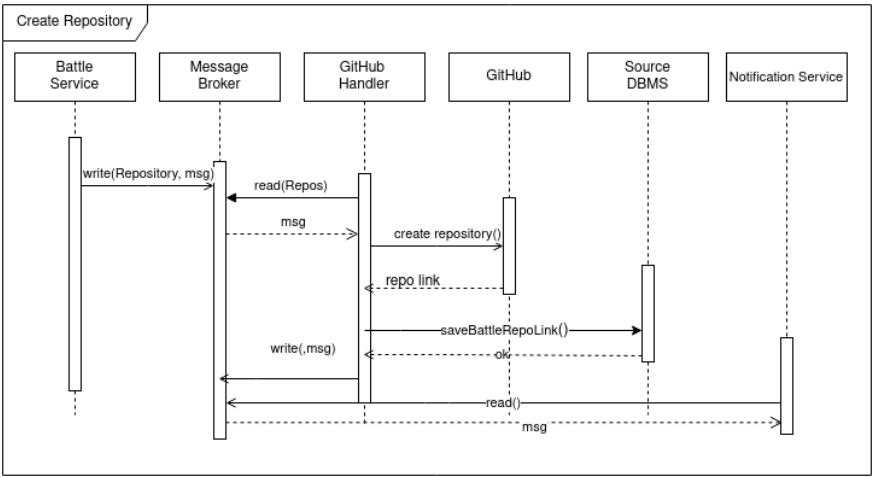


Figure 2.10: Create Repository sequence diagram

### 2.4.7 Join Battle

During the interaction the notification service sends an email the student listed from the user when joining the battle, notifying them of their position as group members in the battle. If the battle group rules allows it and the user decide to join alone, the notification service won't read any message from the message broker, and no email will be sent.

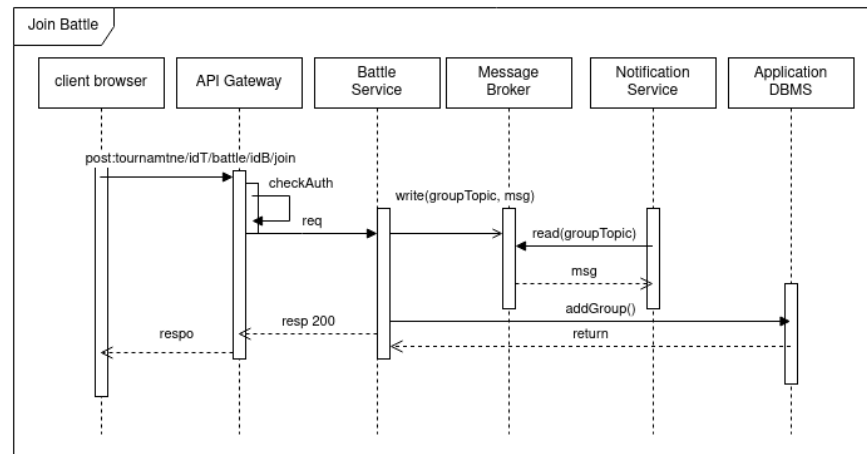


Figure 2.11: Join Battle sequence diagram

### 2.4.8 Score Commit

The interaction starts from GitHub notifying the system. Once the source code has been pulled it is stored in the source DBMS and a message notifying the new sources is sent to the message broker. The software evaluation service is subscribed to the source topic and sees the notification, therefore it reads the sources from the source DBMS and starts to score the source code. Then it notifies the Leaderboard service with the new score through the message broker, which stores it as the group score in the application DBMS.

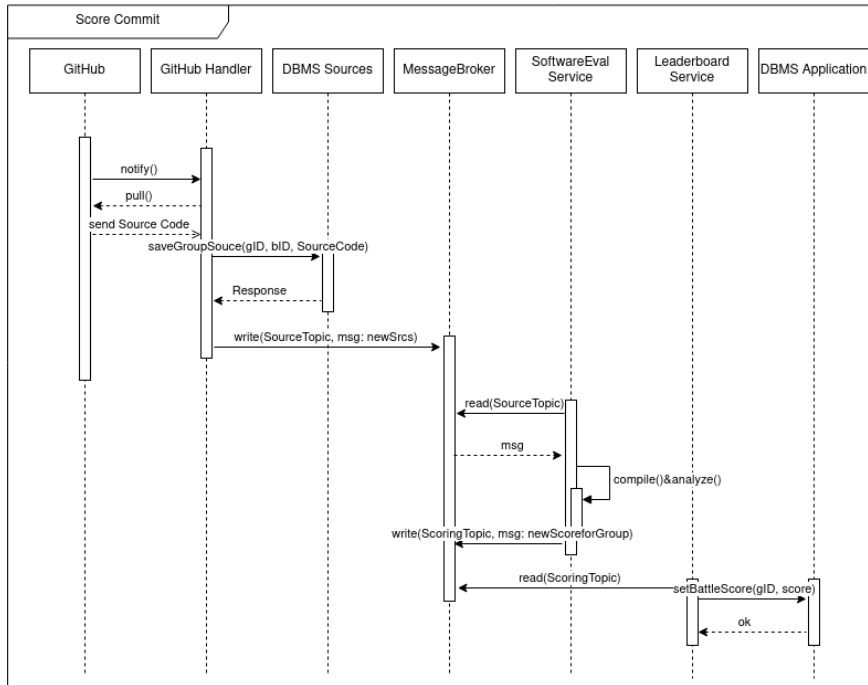


Figure 2.12: Score Commit sequence diagram

### 2.4.9 View Battle Ranking

The response received from the client contains the Battle Leaderboard data which will be visualized.

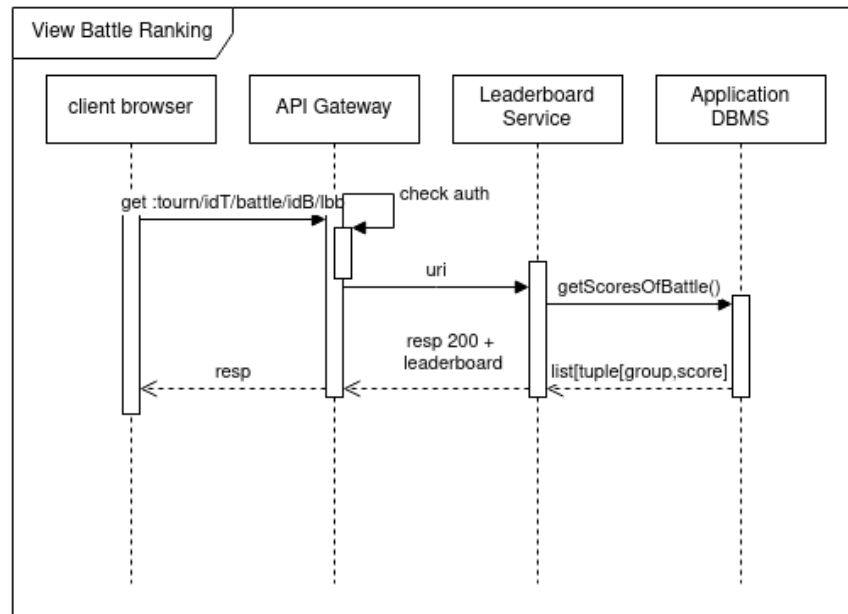


Figure 2.13: View Battle Ranking sequence diagram

## 2.4.10 Manual Evaluation

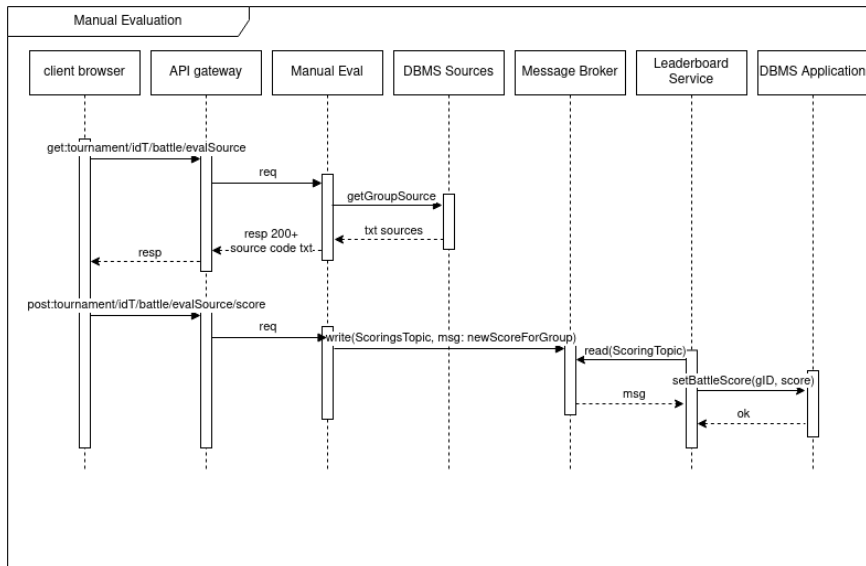


Figure 2.14: Manual Evaluation sequence diagram

### 2.4.11 Look at Tournament Leaderboard

The response received from the client contains the Tournament Leaderboard data which will be visualized.

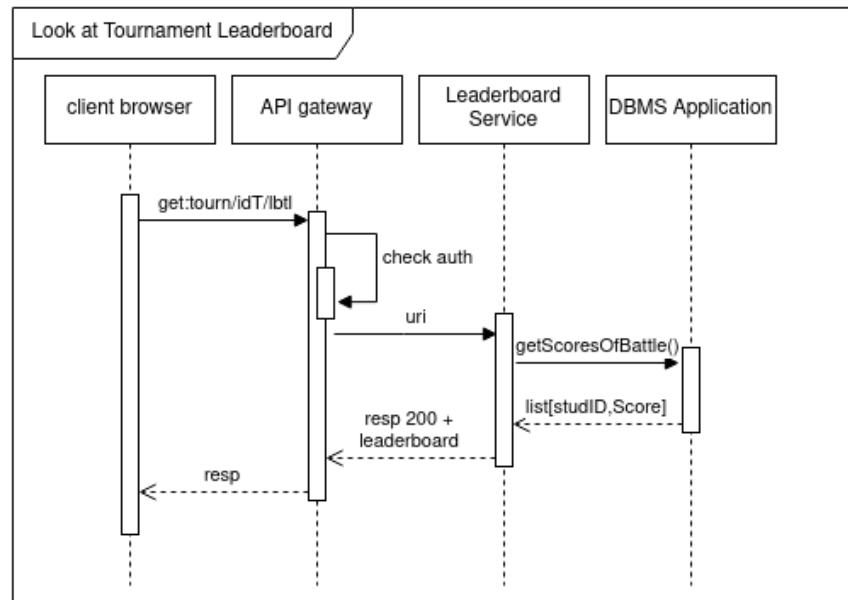


Figure 2.15: Look at Tournament Ranks sequence diagram

2.4.12 Close Tournament

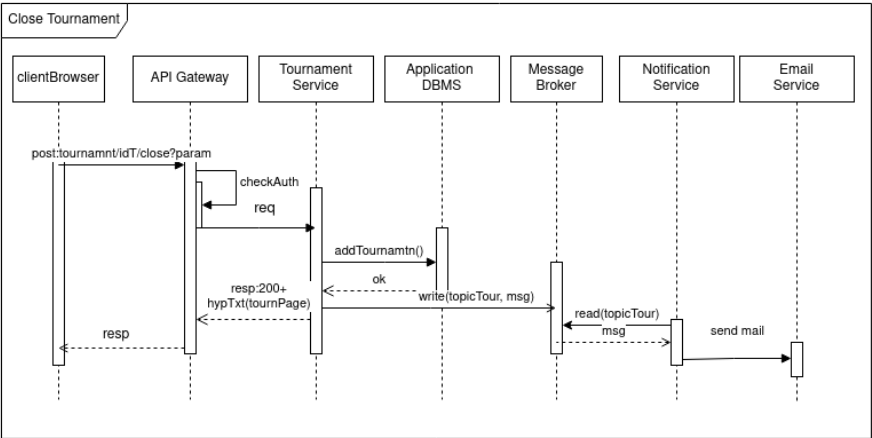


Figure 2.16: Close Tournament sequence diagram



## 2.5 Component interfaces

### 2.5.1 Message Broker API

The message broker exposes two methods:

- read(Topic: int): Message: String
- write(Topic: int, Message: String)

All the microservices write/read on a particular topic exchanging JSON strings:

Topic	Publisher	Subscriber	Content
Tournaments	Tournament Service	Notification Service	int: TournamentID, bool: TournamentStatus
Battles	Battle Service	Notification Service	int BattleID, int: BattleStatus
InvitationsBattle	Battle Service	Notification Service	int: groupID, list[]: userID
InvitationsTournamet	Tournament Service	Notification Service	int: userID int: TournamentID
Commits	GitHubHandler	SoftwareEval Service	int: groupID
Scores	ManualEval Service, SoftwareEval Service	Leaderboard Service	int: groupID int: Score
Repository	Battle Service	GitHubHandler	int: battleID
RepoLinks	GithubHandler	Notification Service	int: battleID String: LinkToRepository

Table 2.1: Topics table

### 2.5.2 DBMS API

#### Data Access Interface

Exposed by Application DBMS, used by Leaderboard, tournament, battle, group and user services.

- getSubscribedStudents(String : IDT): list[string studentID]

#### Used by Leaderboard Service

- getScoresOfTournament(String : IDT): Map[string: studenID; int: score]
- setScoresUserTournament(String:IDStud, String : IDT, int :score): void
- setScoresGroupBattle(String:IDGroup, String : IDB, int :score): void
- getScoresOfBattle(String: IDB): Map[ string: groupID, int: score]

**Used by Notification Service**

- getAllSignedStudent(): list[ string studID]
- getSubscribedStudent(string IDT): list[ string studID]
- getGroups(string IDB): map[string groupID; list[string studID]]
- getInvolvedEDUBattle(string IDB): list[ string eduID]

**Used by Tournament Service**

- addTournament(String : EduCreator):void
- grantBattleCreation(string : IDT, string : grantedEDU): void
- getCurrentTournamtn(): list[ string :tournamentID]
- getBattlesOfTourn(String : IDT):list[ string :battleID]
- checkEducatorPermission(String : IDT, String: IDEDU ): boolean response

**Used by Battle Service**

- getDeadlinesBattle(String: IDB): tuple(subsDL; submDL)
- addBattle(String TournamentID, string assignment, int submDL, int subsDL, int maxsize, int minsize): string IDB
- getBattleAssigment(string IDB): string assignment
- getBattleGroupRules(string IDB): tuple(int maxsize,int minsize)
- getBattleAssigment(string IDB): string assignment
- getBattleDeadlines(string IDB): tuple(string submDL,string subsDL)
- addGroup(list[string studID], String IDB): void

**Used by User Service**

- addStudent(String: UserName): void
- addEducator(String UserName): void

**Source Access Interface**

Exposed by Source DBMS, used by ManualEval, SoftwareEval, GithubHandler, Battle Service

- addBattleTestCases(string IDB,list[ string] testcase):void
- getGroupSource(String: groupID, String: battleID): string SourceCodetxt
- saveGroupSource(String: groupID, String: battleID, String: SourceCode-txt): void

### 2.5.3 RESTful API

These are all external API's exposed by the API Gateway and used by the client application. They follow rest principle, each service handles some resources revolving around a particular aspect of the application i.e.: battles, tournaments, etc. Here follows a three of the uri path tree of the exposed resources:

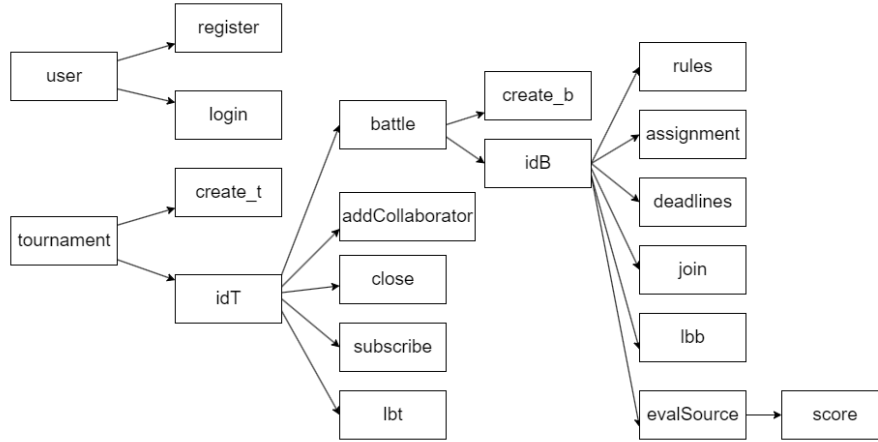


Figure 2.17: Resource Tree path

#### UserAPI

- **POST /user/register**  
registerUser(StringU:UserName, String : UserType):void
- **POST /user/login**  
login(String : psw, String : UserID):void

#### TournamentAPI

- **GET /tournament/**  
getCurrentTournament(String: UserId,String: UserType) : list<Tournament>
- **POST /tournament/create\_t**  
createTournament(String: UserId,String: UserType, String : TournamentName): void
- **POST /tournament/idT/addCollaborator**  
addCollaborator(String: UserId,String: UserType, String : CollaboratorID):void
- **POST /tournament/idT/close**  
closeTournament(String: UserId,String: UserType, String : TournamentID): void

- **POST /tournament/idT/subscribe**  
subscribeTournament(String: UserId,String: UserType, String : TournamentID) : void
- **GET /tournament/idT/battle/**  
getTournamentsBattles(String: UserId,String: UserType, String : TournamentID): List<Battle>

### BattleAPI

- **POST /tournament/idT/battle/create\_b**  
createBattle(String: UserId,String: UserType, String : BattleName, tuple(int maxsize, int minsize): groupRule, string: assignemtent, tuple(date: subs, date: subm): deadline, list[ string]: testcases): void
- **GET /tournament/idT/battle/idB/rules**  
getGroupRules(String: UserId, String : BattleID): Tuple (int :MaxSize, int: MinSize)
- **GET /tournament/idT/battle/idB/assignment**  
getAssignemntText(String: UserId, String : BattleId): String AssignmentText
- **GET /tournament/idT/battle/idB/deadlines**  
getDeadlines(String: UserId, String : BattleID):Tuple (int :SubscriptionDL, int: SubmissionDL)
- **POST /tournament/idT/battle/idB/join**  
joinBattle( String: UserId, String : BattleID, String: UserType, OPTIONAL List< StudentID> ): void

### LeaderBoardAPI

- **GET /tournament/idT/lbt**  
getLeaderBoardTournament(): LeaderBoard
- **GET /tournament/idT/battle/idB/lbb**  
getLeaderBoardBattle():Leaderboard

### ManualEvaluationAPI

- **GET /tournament/idT/battle/evalSource**  
getSourcesForEval(String: UserId, String : BattleID, String: UserType ): String SourceCode
- **POST /tournament/idT/battle/evalSource/score**  
addManualScore(String: UserId, String : BattleID, String: UserType, int : score) : void

## 2.6 Selected architectural styles and patterns

### 2.6.1 Microservices

A microservice architecture is needed mainly in order to freely scale some component independently from others, like the **software evaluation** service, which at time may need much more computational resources, meanwhile other services are less likely to require as much, as they don't perform heavy activities like code analysis and execution. Other than that, this style of architecture provides many more general advantages, making easy to choose.

#### API Gateway

The API gateway provides many benefits as the only entry point between the client and services. Here are the ones which were more valued: Security and authentication; Load balancing; Caching.

### 2.6.2 Hybrid architecture (REST and EBA)

#### Event Based architecture

Used in the system backend for the communication between microservices. The use of asynchronous communication allows for more flexibility and scalability, both important for some of the microservices, in particular the **software evaluation** service, which needs to scale independently from other services because of its higher computational needs. Moreover the system interfaces with Github APIs which have also an EBA.

#### Rest(ful APIs)

The services which provide functionalities for the front end expose RESTful APIs. This way there is a greater separation of concern between front end and back-end, as when developing the client application a team doesn't need to concern themselves with the internal structure of the system made of services, but only need to know the URI tree path used to access resources and their operations.

## 2.7 Other design decisions

There were no other design decisions to note.

## Chapter 3

# User Interface Design

In this section will be presented an overview of the user's interface of the CKB system. The application, as previously discussed will be a web app, accessible from a desktop browser. The access from mobile will be enabled and the interface will be scaled appropriately.

### 3.1 General Overview

The image below shows a map of the pages accessible by Educators and Students.

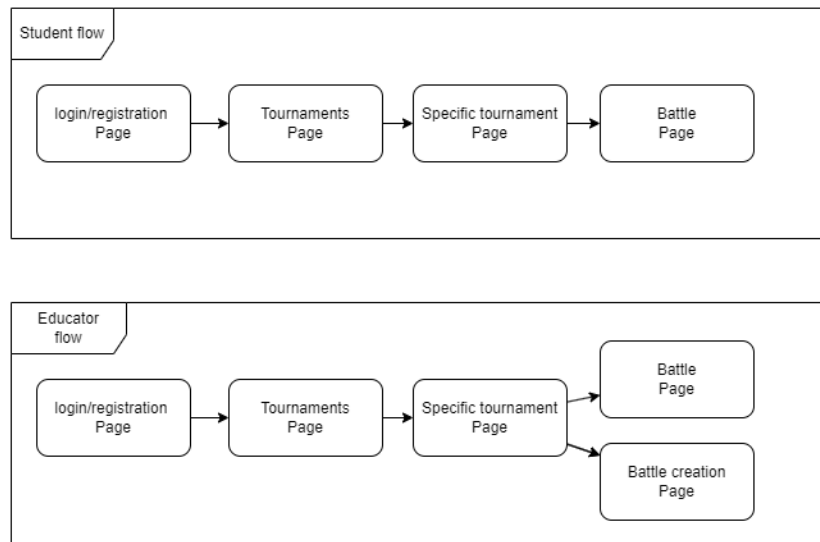


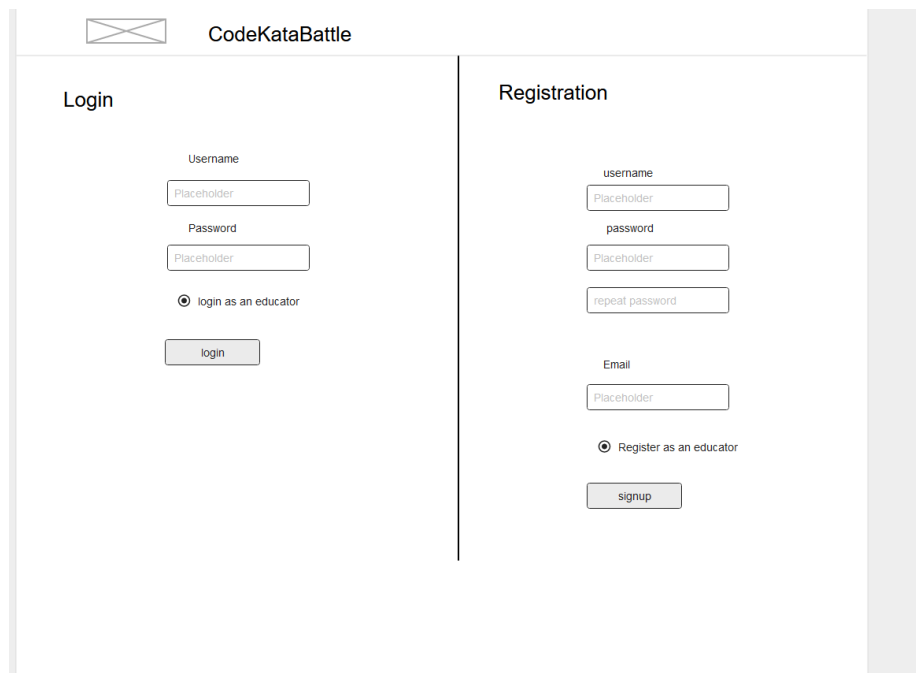
Figure 3.1: Web pages map

## Login Page

This page is the first one viewed by the user when accessing the application. It contains two forms, one for the login of the user and one for the registration. For the login the user can input their Username, password and select whether the user is an Educator or not.

For the registration the user is asked to insert their desired username, the password, and the type of user he desires to register as. Also the user has to insert the email where he will receive notification.

Once the user has created the account or logged into his own account he is directed to the tournaments page.



The image shows a web interface for 'CodeKataBattle'. At the top, there is a logo (a square with an 'X') and the text 'CodeKataBattle'. Below this, the page is split into two main sections: 'Login' on the left and 'Registration' on the right.

**Login Section:**

- Label: Username
- Input field: Placeholder
- Label: Password
- Input field: Placeholder
- Radio button (selected): login as an educator
- Button: login

**Registration Section:**

- Label: username
- Input field: Placeholder
- Label: password
- Input field: Placeholder
- Input field: repeat password
- Label: Email
- Input field: Placeholder
- Radio button (selected): Register as an educator
- Button: signup

Figure 3.2: Login Page

## Main Page

In this page a user can see the list of ongoing tournaments and the list of tournaments he/she is involved in, from which a specific tournament can be chosen to be redirected to. If the user is an Educator then the page will also display a form for the creation of a tournament.

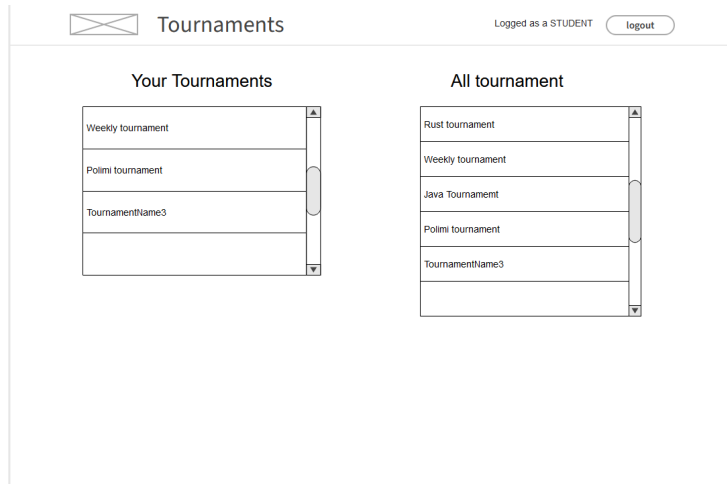


Figure 3.3: Student Main Page

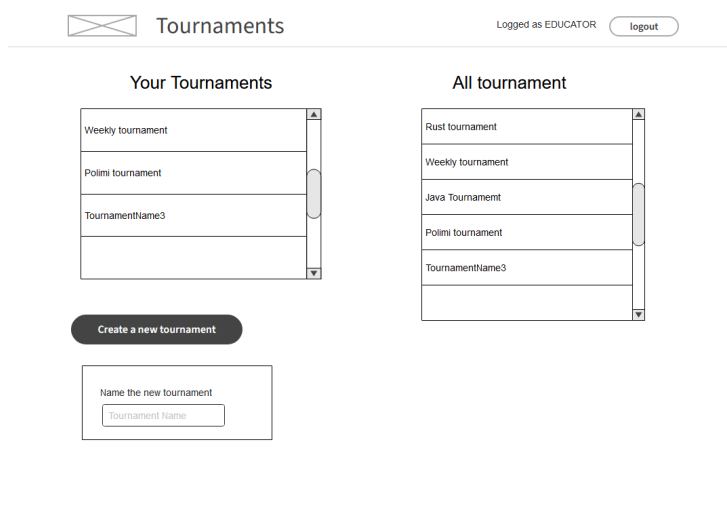


Figure 3.4: Educator Main Page



### Tournament Page

This page contains the list of battle in the context of the current tournament as well as the tournament leaderboard and 3 buttons. The buttons allow the user to either create a battle in the tournament context, add a collaborator to the tournament or close the tournament. The battles in the list can be selected to be directed to the selected battle page. If the user is a student then instead of the three buttons there will be only one, allowing the student to join the tournament.

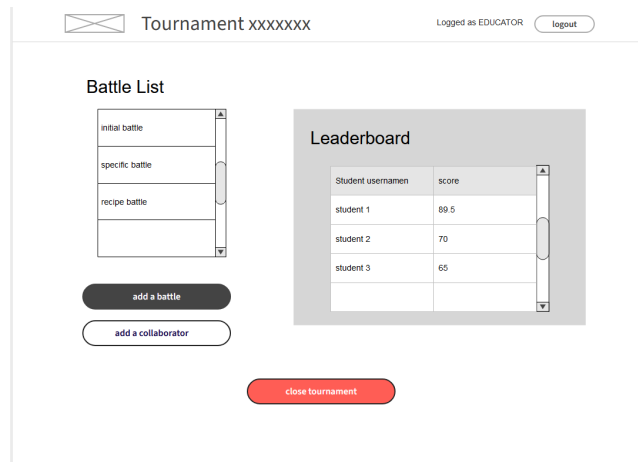


Figure 3.5: Educator Tournament Page

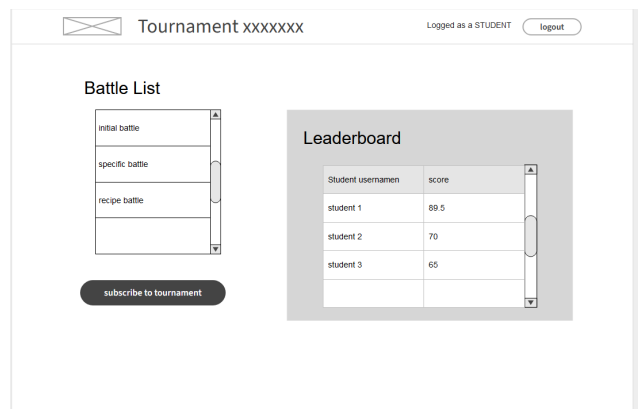


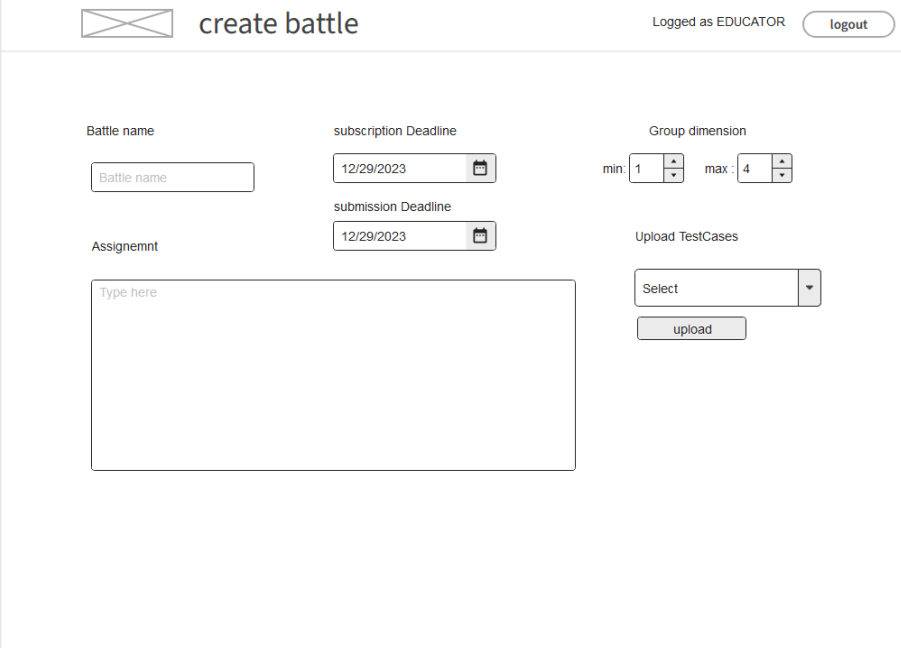
Figure 3.6: Student Tournament Page

### Battle Creation Page

The page presents a form where an educator can insert:

- Battle name;
- Text assignment for the CK;
- The two deadlines,
- The group rules.

There is also an upload form for the CK testcases.



The screenshot shows a web interface for creating a battle. At the top, there's a header bar with a logo on the left, the text "create battle" in the center, and "Logged as EDUCATOR" with a "logout" button on the right. Below the header, the form is organized into several sections. On the left, there's a "Battle name" label above a text input field. To its right, there's a "subscription Deadline" label above a date picker showing "12/29/2023". Further right, there's a "Group dimension" label above a range selector with "min: 1" and "max: 4". Below the "Battle name" field is an "Assignemnt" label above a large text area with the placeholder "Type here". To the right of the text area is a "submission Deadline" label above another date picker showing "12/29/2023". On the far right, there's an "Upload TestCases" label above a dropdown menu with "Select" and an "upload" button below it.

Figure 3.7: Battle Creation

### Battle Page

For both type of users the page presents the text of the assignment, the rules for the group size, the deadlines and finally the battle leaderboard. In the student version during the battle subscription phase a button for joining the battle is present. When clicked the user is asked to insert a list of other student to join the battle with( following the group rules) in a form.

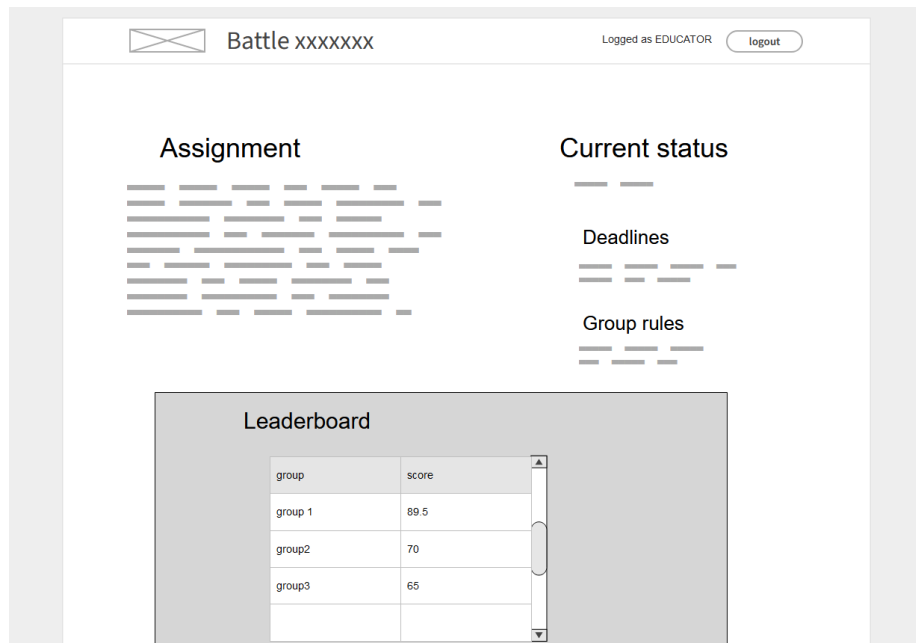


Figure 3.8: Battle Page

Manual Evaluation Page

The page contains the source code of the group to be manually reviewed and an input where the user can put the decided score to be sent after clicking the 'score group' button.

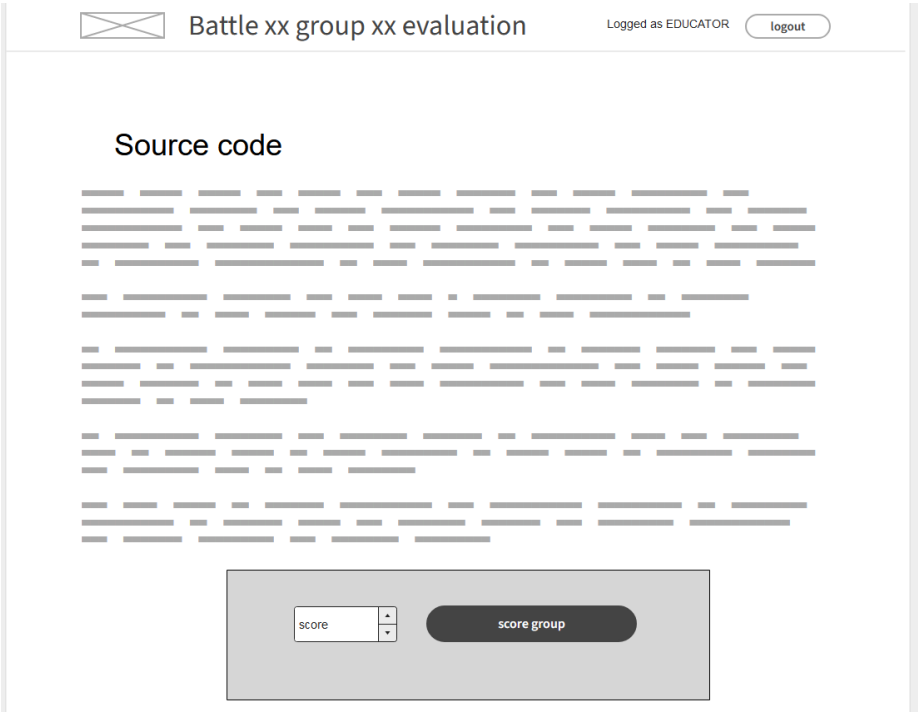


Figure 3.9: Battle Page



## Chapter 4

# Requirements Traceability

This Chapter shows how the functional and non-functional requirements of the CKB system described in the RASD are met. We show the mapping between each requirement and which components that meet that.

### 4.0.1 Functional requirement traceability

- R1: The platform allows a signed in educator to create tournaments
  - API Gateway: sign in
  - Tournament Service: creation of tournaments
- R2: Educators can create battles in the context of a specific tournament they are involved in (Either by creation or by invitation)
  - Battle Service: creation of battles
- R2.1: The platform allows an educator that created a tournament, to invite other educator and to grant them permission to create battles in the tournament context
  - Tournament Service: Invitation to collaborate to a tournament
- R2.2: The platform allows an educator to upload the codekata (description and software project, including test cases and build automation scripts) when creating a battle
  - Battle Service: creation of battles
- R2.3: The platform allows an educator to set subscription and submission deadlines when creating a battle
  - Battle Service: creation of battles
- R2.4: The platform allows an educator to set minimum and maximum group size when creating a battle

- Battle Service: creation of battles
- R2.5: The platform allows an educator creating a battle to include a manual evaluation stage
  - Battle Service: creation of battles
  - Manual Evaluation Service: Allows an educator to add a manual score
- R3: The platform allows students to subscribe to a tournament
  - Tournament Service: allows students to subscribe to new tournaments
- R4: The platform allows a student subscribed to a tournament to join a battle in that tournament context
  - Battle Service: allows a student to subscribe to a new battle
- R4.1: The platform allows students to create a group by inviting other students when joinin a battle
  - Battle Service: allows students to create and join groups
- R5: The platform allows Students to login
  - User Service: allows login
- R6: The platform allows Educators to login
  - User Service: allows login
- R7: If manual evaluation is required during consolidation stage the platform allows an educator to go through sources and add a score to the group score
  - Manual Evaluation Service: Allows an educator to add a manual score
- R8: The platform allows all users to view student ranks from previous and current tournaments
  - Leaderboard Service: Retrives from ApplicationDB all info to rebuild leaderboards
- R8.1: For every tournament the platform maintains a ranking for students based on the sum of their battle scores
  - Leaderboard Service: Holds all tournament information in ApplicationDB

- R9: The platform pulls group sources from Github when it receives a notification within the submission deadline
  - GitHub Handler: receives notifications from Github and stores each group source in SourceDB
- R9.1: The platform analyzes, runs testcases and scores the source of a group solution for a codekata battle and updates the group score accordingly
  - Software Evaluation Service: Whenever a pull is performed by Github Handler the service compiles and analyzes sources to then submit a new score
  - Leaderboard Service: Updates score in ApplicationDB
- R10: The platform allows all user to sign in
  - API Gateway: allows register
- R11: The platform allows educators to close tournaments
  - Tournament Service: allows educators to manage tournaments
- R12: When a battle deadline expires the platform sets the battle status to consolidation stage
  - Battle Service: Managed all battle related deadlines
- R13: The platform allows all users who are involved in a battle to look at group ranks for that battle
  - Leaderboard Service: Retrives from ApplicationDB all info to rebuild leaderboards
- R13.1: For every battle the platform maintains a ranking of the groups based on their battle score
  - Leaderboard Service: Holds all battle information in ApplicationDB
- R14: The platform will notify signed students of a newly created tournament
  - Tournament Service: Will write a new message on message broker every time a new tournament is created
  - Notification Service: Will read from message broker any new message regarding the creation of a tournament and notify students through Email Service
- R15: The platform shall notify students who are subscribed into a tournament that a new battle is available in that tournament context



- Battle Service: Will write a new message on message broker every time a new battle in a tournament is created
  - Notification Service: Will read from message broker any new message regarding the creation of a battle and notify subscribed students through Email Service
- R16: The platform shall notify students who are participating in a battle that the final rank for that battle is available
  - Battle Service: Will write a new message on message broker every time a battle in a tournament has ended
  - Notification Service: Will read from message broker any new message regarding the end of a battle and notify subscribed students through Email Service
- R17: The platform shall notify students who are subscribed in a tournament that the final rank for that tournament is available
  - Battle Service: Will write a new message on message broker every time a tournament has ended
  - Notification Service: Will read from message broker any new message regarding the end of a tournament and notify subscribed students through Email Service
- R18: The platform shall create a new repository with Github for every new battle in any tournament after the submission deadline expires
  - Battle Service: Will write a new message on message broker when the registration deadline expires
  - GitHub Handler: Will read a create repository message from the message broker and communicate with GitHub to create a new repository
- R19: The platform shall send the battle repository link to students who joined that battle after the submission deadline expires
  - GitHub Handler: Will write a message everytime a new repository is created
  - Notification Service: Will read a message containing the link to the repository and send an email through the email service to every student that has joined the repository's battle

## Chapter 5

# Implementation, Integration and Test Plan

### 5.1 Implementation & Integration

The implementation and integration plan will follow a very simple strategy:

- First we start by developing the various Microservices of our system: Tournament Service, Battle Service, Leaderboard Service, Invitation Service, Software Evaluation Service, Manual Evaluation Service, Github Handler Service.
- Next we make sure the communication with the various DBs works
- Then after testing the basic functionalities we will start making our services communicate with the external world by connecting the notification service to an email provider and the GitHub Handler Service to GitHub
- Lastly we implement the User side of things, the API gateway and the WebApp

### 5.2 Test Plan

The system should be tested statically during development and dynamically with particular dedication to ensure that no vulnerabilities are left open. For example one of the main security weak point is the Software Evaluation Service as it runs any external code which could eventually be malicious. Regarding the UI/UX side of things it is crucial that the application will be intuitive enough to ensure a good user experience.



## Chapter 6

### Effort spent

Chapter	Valentino	Zarbo
1	3	2
2	16	20
3	3	5
4	7	2
5	4	4

Table 6.1: Time Spent in hours for each chapter



## Chapter 7

# References

### 7.1 Used tools

- GitHub for project versioning
- Draw.io for UML diagrams
- Overleaf as  $\text{\LaTeX}$  editor
- Moqups as UI Mockups editor