



**POLITECNICO DI MILANO**

SOFTWARE ENGINEERING II

---

**CodeKataBattle**

**ITD Document**

Version 1.0

---

**sebaxe07/PereaRuiz: Project for SE2 (github.com)**

Daniel Mauricio Ruiz Suarez  
Sebastian Enrique Perea Lopez

*January 07, 2024*

# Contents

<b>1. INTRODUCTION</b>	<b>3</b>
1.1. Problem Description	3
1.2. Scope	3
1.3. Success Criteria	3
1.4. Revision history	3
1.4.1. Version 1.0 (05/02/2024):	3
1.5. Reference Documents	3
1.6. Document Structure	4
<b>2. SOFTWARE FUNCTIONS</b>	<b>5</b>
2.1. Accomplished Requirements	5
2.2. Implemented Functions	5
2.2.1. User Authentication and Registration:	5
2.2.2. Tournament and Battle Management Component:	15
2.2.3. Team Formation and GitHub Integration Component:	25
2.2.4. Notification Component:	26
2.3. Not Implemented Functions	28
2.4. Deployment view	30
2.5. Other design decisions	31
2.5.1. GitHub Integration	31
2.5.2. Automated Workflow (GitHub Actions)	31
2.5.3. Use of Static Analysis Tools	31
<b>3. DEVELOPMENT FRAMEWORKS</b>	<b>32</b>
Code Structure Overview:	34
Folder Structure:	34
Backend Structure:	34
Frontend Structure:	34
Shared Code:	38
Configuration Files:	38
Testing:	38
Unit Testing:	38
Automated Testing:	38
Code Version Control:	39
Version Control System:	39
Branching Strategy:	39
API Documentation:	40
Communications Inside the Application	40
Email Notification Services	40
GitHub's APIs Integration	40
<b>5. IMPLEMENTATION, INTEGRATION AND TEST PLAN</b>	<b>41</b>
5.1. Overview	41
5.2. System Testing	42

5.2.1. Testing Methodologies-----	42
5.3. Additional specifications on testing-----	43
5.3.1. Functional Testing-----	43
5.3.2. Non-functional Testing-----	43
5.3.3. Performance Testing-----	43
5.3.4. Usability and Compatibility Testing-----	44
<b>6. INSTALLATION INSTRUCTIONS-----</b>	<b>45</b>
6.1. Prerequisites:-----	45
6.2. Download the Source Code:-----	45
6.3. Backend Configuration:-----	45
6.4. Run the Application:-----	45
<b>7. EFFORT SPENT-----</b>	<b>47</b>
<b>8. REFERENCE-----</b>	<b>48</b>

# 1. INTRODUCTION

## 1.1. Problem Description

In the modern educational landscape, the importance of practical coding experience is undeniable. Programming challenges and coding competitions not only enhance the student's technical skills, but also nourish the creativity and problem solving abilities. Unlike more traditional methods that have a harder time creating exciting, competitive and engaging environments for the students.

So, our goal was to create a tool that makes life easier for modern teachers dealing with modern coding challenges. This tool will help set up and manage tournaments and battles where students can join in, test their skills, and learn in a fun and more immersive way. Our web application aims to be practical, simple, and appealing, while working well and consistently. We focused on developing software that is user friendly and minimalist, making the navigation through the app menus and various screens smooth and intuitive, grouping the UI components in a way that makes them easy to locate. Giving each role their respective level of access to their required components and screens (this is also imperative for the usability and security of the app). We build a platform that not only lets students code together and interact but also makes sure everyone gets a fair shot and gets their skills recognized.

## 1.2. Scope

The Implementation and Test Deliverable (ITD) will focus on transforming the conceptual design detailed in the Requirements and Architectural Specification Documents (RASD and DD) into a fully functional and tested Battle Code Management System. This includes translating the identified requirements and design elements into actual code, this includes a thorough examination of the software structure, programming languages chosen, and the integration of various components. Additionally, this document provides insights into the testing procedures performed to ensure the robustness and reliability of the Battle Code Management System.

## 1.3. Success Criteria

Our application will be successful if we are able to implement the system in a way that covers at least its main functionalities. It must be secure, reliable and usable, but overall it must allow the teachers to create their tournaments and battles, with their own deadlines, criterias and other details, while also allowing the students to join and participate in them.

## 1.4. Revision history

### 1.4.1. Version 1.0 (05/02/2024):

Initial release of the Implementation and Testing Document (ITD) for CodeKataBattle.

## 1.5. Reference Documents

- The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2023/2024.

## 1.6. Document Structure

In the document we kept a clear and simple structure, just looking to provide a brief introduction to each of the sections that we considered were crucial in the process of developing our application and looking to provide enough context for new users that are just interacting with it:

1. **Introduction:** The introduction provides a preliminary understanding of the Implementation and Test Deliverable. It outlines the purpose, scope, and overall structure of the document, serving as an entry point for readers seeking insights into the development and testing phases.
2. **Software Functions:** This section delves into the core functionalities of the CKB system, offering a detailed exploration of the major components. Through class diagrams, composite structure diagrams, and statecharts, it provides a comprehensive overview of the system's architecture.
3. **Development Frameworks:** Focused on the visual aspects, this section outlines design considerations for the user interface. It emphasizes creating an intuitive and user-friendly experience for both students and educators during the implementation phase.
4. **Structure of the Code:** Establishing a clear link between design decisions and initial requirements from the RASD, this section systematically maps design elements to corresponding requirements. It serves as a guide for developers, ensuring the alignment of the implementation with the specified requirements.
5. **Implementation, Integration and Test plan:** Outlining the roadmap for system implementation, integration, and testing, this section provides a detailed plan for developers. It guides the step-by-step process of bringing the CKB system to life, ensuring a methodical approach to development and testing.
6. **Effort spent**
7. **References:** Compiling external resources, standards, or documentation referenced in the DD, this section ensures transparency and credibility.

## 2. SOFTWARE FUNCTIONS

Here we will like to provide an in-depth mapping of the key features implemented within our application. From user authentication to tournament and battle management, the following subsections delve into the specific features that shape the system's capabilities.

The functionalities are in line with the defined requirements in the Requirements Analysis and Specification Document (RASD) and are supplemented by additional features that contribute to the platform's effectiveness.

### 2.1. Accomplished Requirements

- [R1] The system allows students to sign up and log in.
- [R2] The system allows educators to sign up and log in.
- [R3] Educators can create new tournaments, defining code battles with descriptions, deadlines, and scoring configurations.
- [R4] Educators can set minimum and maximum team sizes for battles within a tournament.
- [R5] Educators can manage and view the list of battles they've created within a tournament.
- [R9] Educators can set a registration deadline.
- [R10] Educators can set a final submission deadline.
- [R12] Students can form teams for battles, adhering to set team size limits.
- [R13] The platform automatically generates GitHub repositories for each battle upon creation.
- [R16] Educators can manually assess and assign personal scores to student submissions.
- [R17] After the submission deadline, educators review and provide manual evaluations based on code quality and adherence to the test-first approach.
- [R21] Automated notifications inform students about new tournaments and upcoming battles within subscribed tournaments.
- [R22] Real-time updates notify users (students and educators) about rank changes, deadlines, and significant tournament-related information.
- [R23] Email notifications alert users about critical updates, including new tournaments, battle results, and personal scores.
- [R24] Implement role-based permissions, allowing educators administrative access to manage tournaments, battles, and evaluations, while students have limited access as participants.
- [R25] Enable users (students and educators) to view their profiles, displaying personal information, ongoing tournaments, past performances, and badges earned.

### 2.2. Implemented Functions

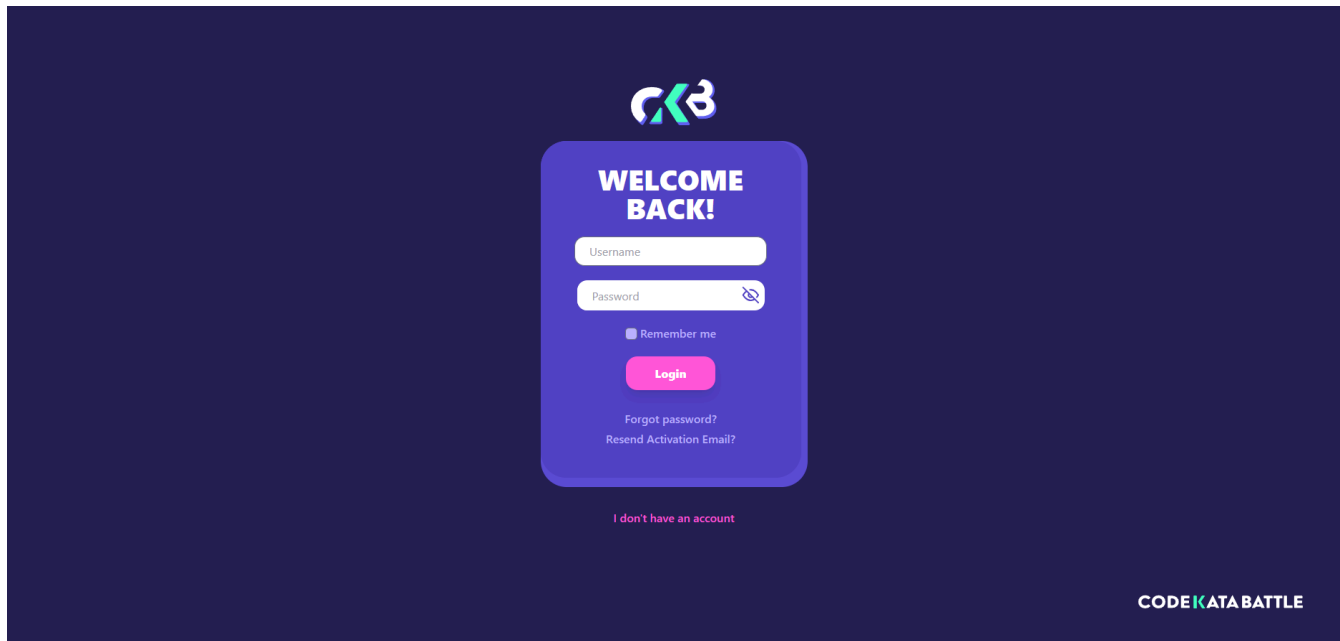
#### 2.2.1. User Authentication and Registration:

The Log-in and Sign-up feature were correctly implemented, this is an essential feature that provides a way for our users to register and access the application, while giving permissions to each user according to their selected role

- Responsible for user authentication, registration, and authorization.
- Interacts with the database to store user information and permissions.
- Interaction: Connected to various modules for access control and user-specific functionalities.

## Login

The login process in the Battle Code Management System is a straightforward yet secure procedure designed to ensure ease of access for both educators and students.



### ○ Functionality

Here's a step-by-step explanation of how the login process works for both user types:

#### User Input

- Users navigate to the application's login interface.
- Educators and students enter their respective usernames and passwords into the provided login form.

#### Data Validation:

- The application validates the entered credentials against the stored records in its database.
- The system checks if the provided username and password match the records associated with the user's account.

#### Authentication:

- If the entered credentials are valid, the application authenticates the user.
- The user is verified as a legitimate user of the system, granting access to their personalized account.

#### Session Management:

- Upon successful authentication, the system creates a session for the user.
- The session is a temporary state that allows users to access protected resources and perform specific actions without constant re-authentication.

#### User Context:

- The UserContext in the application manages the active user's information.
- Variables like activeUser and setActiveUser store and update the user's data during their session.

#### **Password Visibility:**

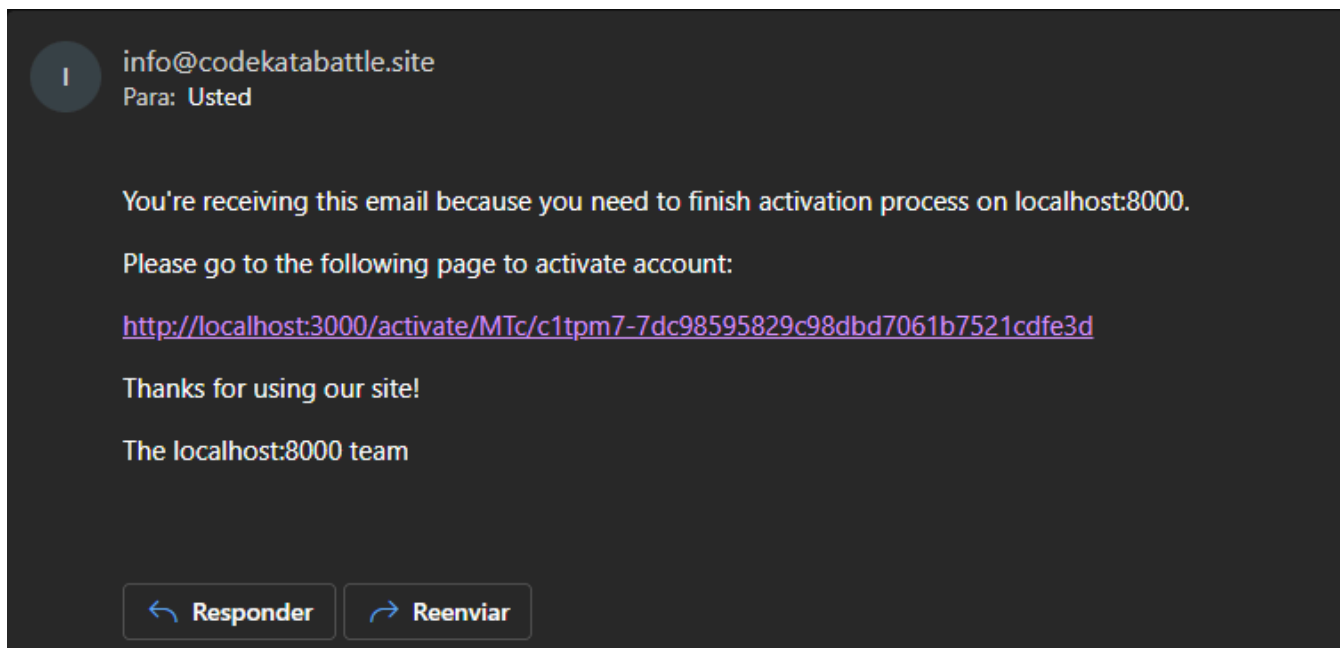
- The login interface includes a functionality to toggle the visibility of the password input field.
- This feature allows users to view or hide their entered password as they type.

#### **Remember Me:**

- Users can choose to have their login session remembered across multiple visits to the application, enhancing convenience.

#### **Redirect and Notifications:**

- After successful authentication, the system uses the useNavigate hook to redirect the user.
- Educators and students are directed to their respective dashboards or landing pages based on their roles.
- A toast notification is displayed to welcome the user back, providing a friendly and informative interaction.



The login process ensures a secure and user-friendly experience, contributing to the overall effectiveness of the Battle Code Management System.

### **Sign-Up**

Creating a new account in our system is a simple and guided experience.

#### ○ **Functionality**

Here's a step-by-step explanation of how the login process works for both user types:



### Select User Type:

- Start by choosing your user type (educator "SENSEI" or student "SEITO") on the first screen.

### Complete Sign-Up Form:

- Move to the next screen to fill out a form with basic details like email, first name, last name, and password.

### Validation Checks

- The form checks your inputs for validity, making sure things like passwords match and emails are correctly formatted.
- If there are any mistakes, an error message appears to guide you through corrections.

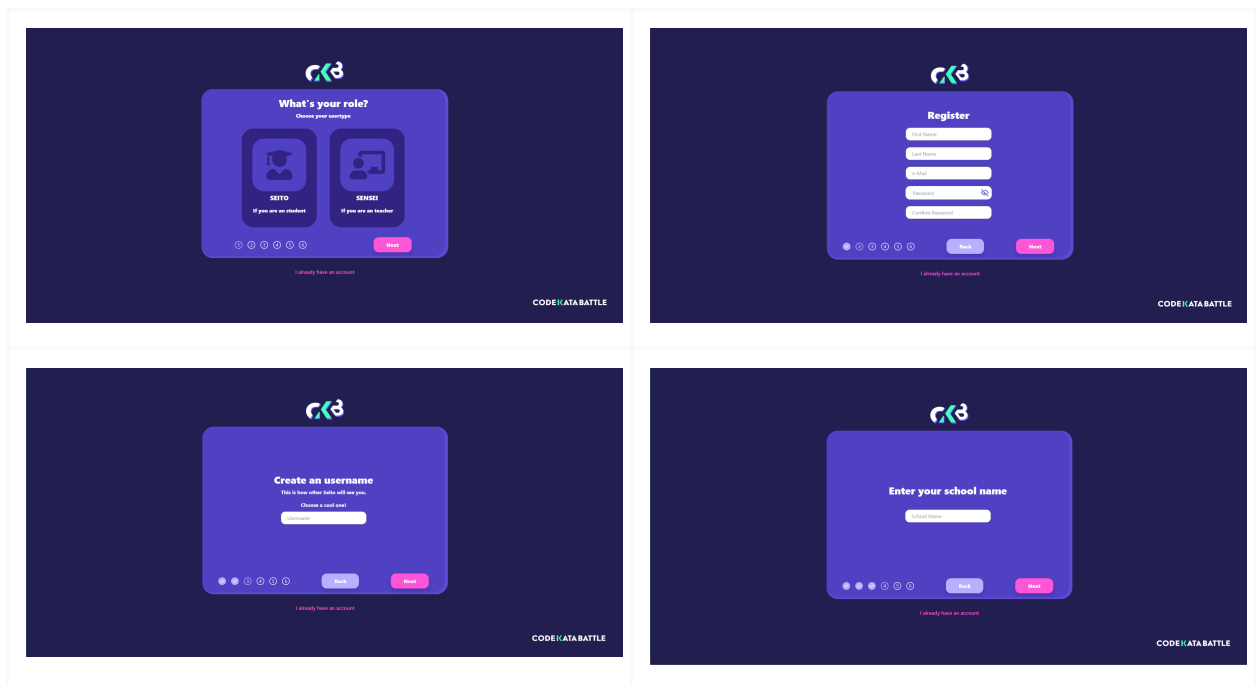
### User Related Fields:

The following screens just involve filling the required information

- Choose a unique username.
- Insert your school name.
- Add your valid GitHub username account.

### Select Avatar:

- Click "Next" after choosing your avatar to submit your details securely to our server.

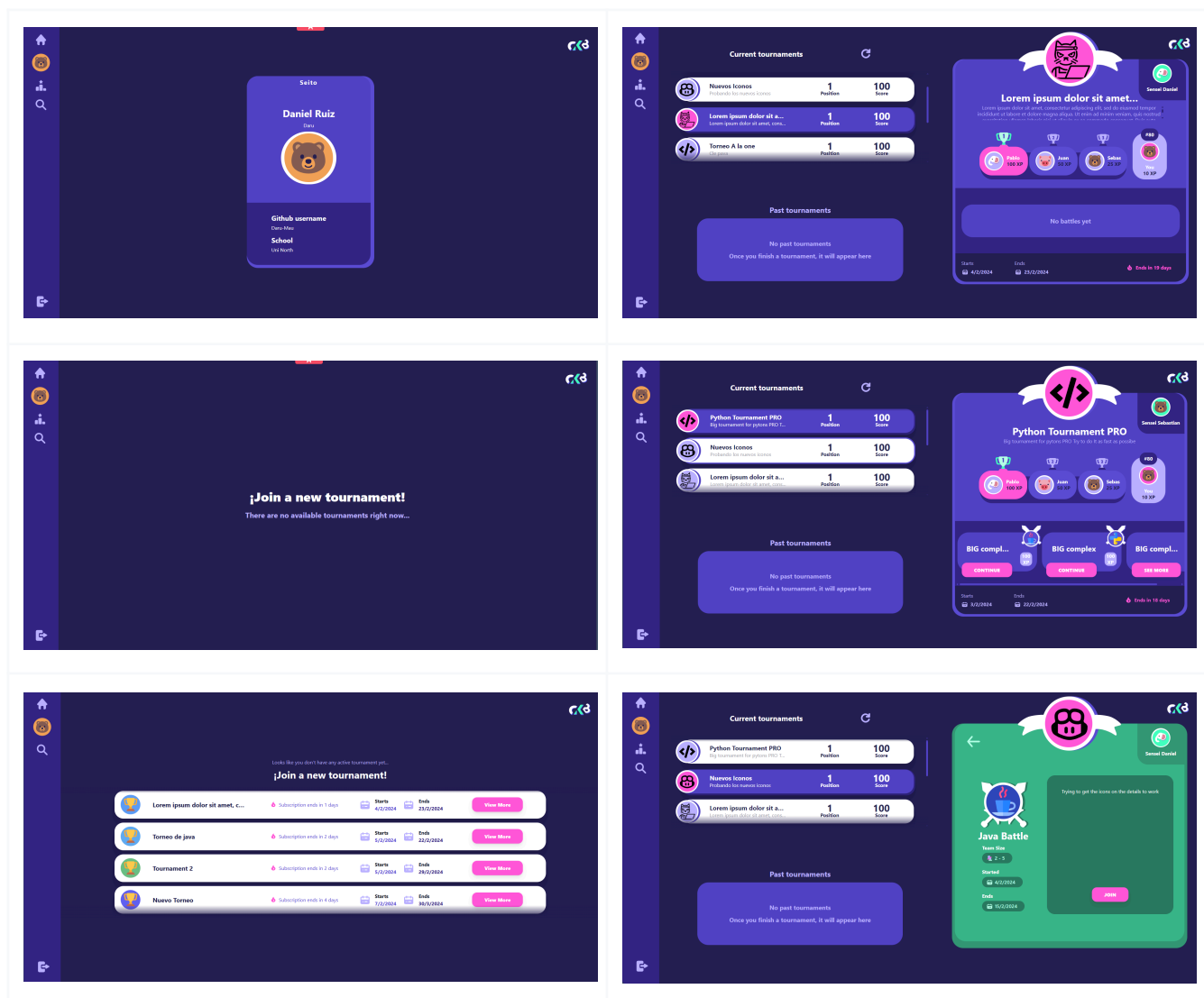


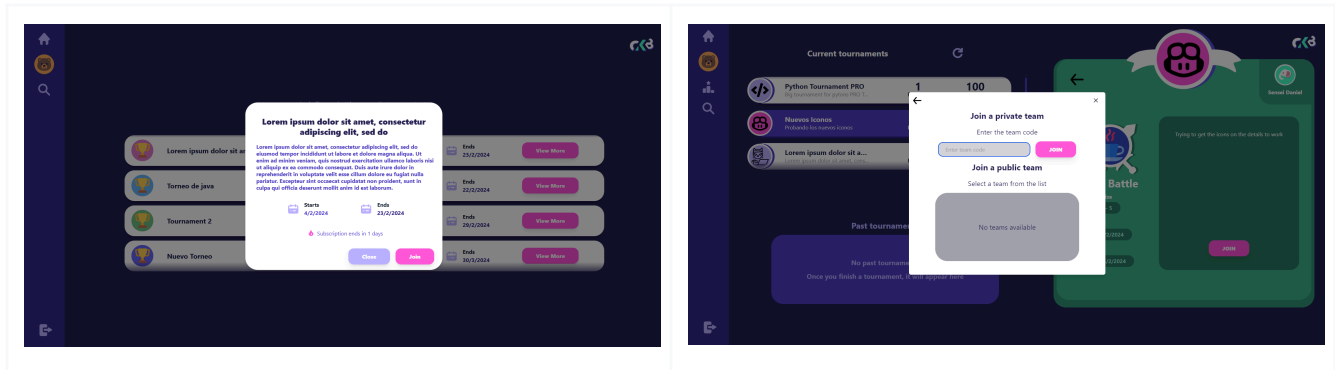


## Account Creation:

- If everything checks out, a success message appears, asking you to verify your new account via email, the one you previously provided, and you're redirected again to the login page.

You've completed the sign-up journey. Our system is designed to make this process smooth, interactive, and enjoyable for all users.



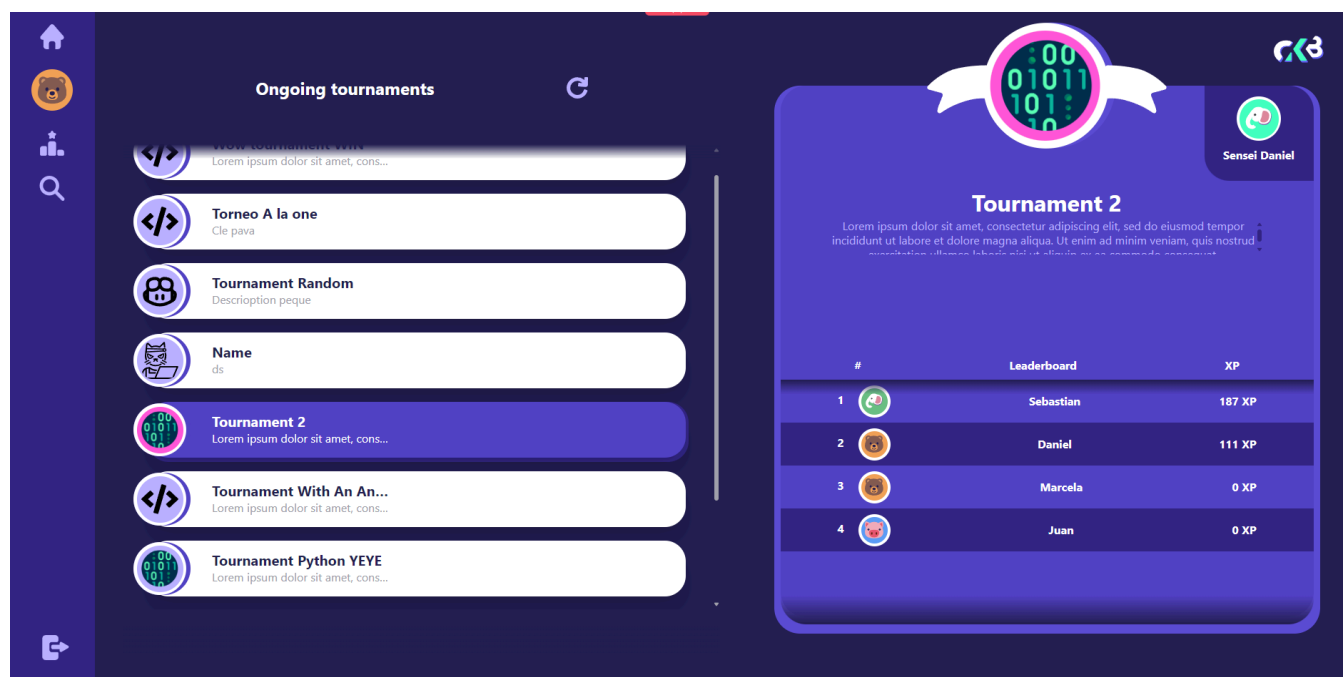


○ **Requirements accomplished:**

[R1] The system allows students to sign up and log in.

[R2] The system allows educators to sign up and log in.

[R24] Implement role-based permissions, allowing educators administrative access to manage tournaments, battles, and evaluations, while students have limited access as participants.



## 2.2.2. Tournament and Battle Management Component:

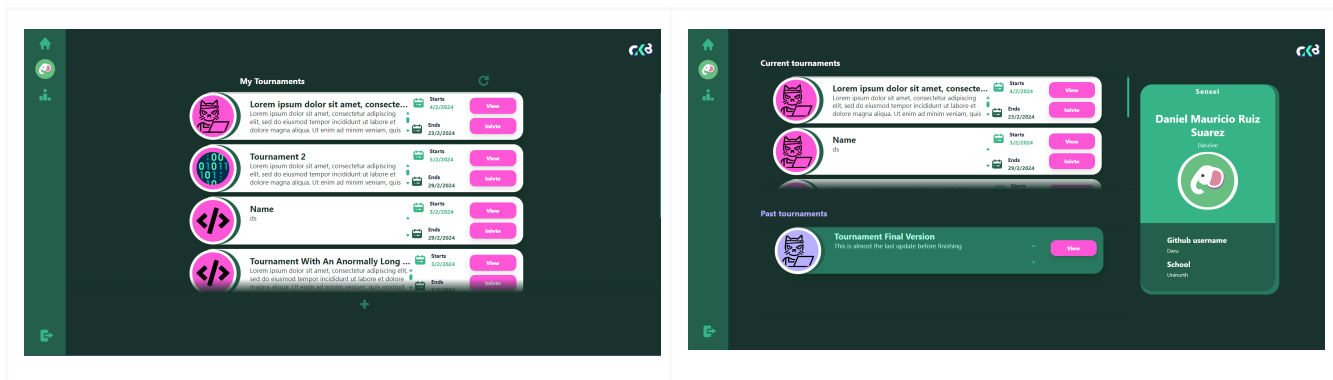
This feature is the backbone for educators to create the challenges in which the students can participate. We were able to implement all the Tournament-Battle related features that were previously considered.

- Handles the creation, subscription, and management of tournaments and battles.
- Interacts with the User Management Component for educator permissions and user subscriptions.

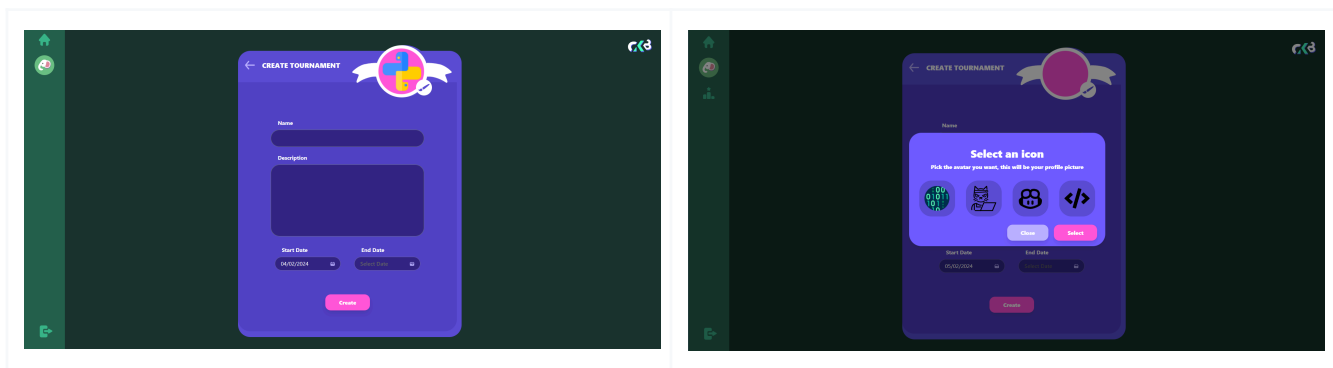
- Interaction: Interfaces with the Team Formation and GitHub Integration Component to facilitate battle setup and notifications.

- Tournament

Tournament Management serves as a foundational feature for overseeing coding competitions. It allows educators to administer tournaments effectively. While not subdivided, its functionalities might encompass tournament settings, participant monitoring, and progress tracking. Its significance lies in providing educators with control and oversight, ensuring organized and impactful coding events.

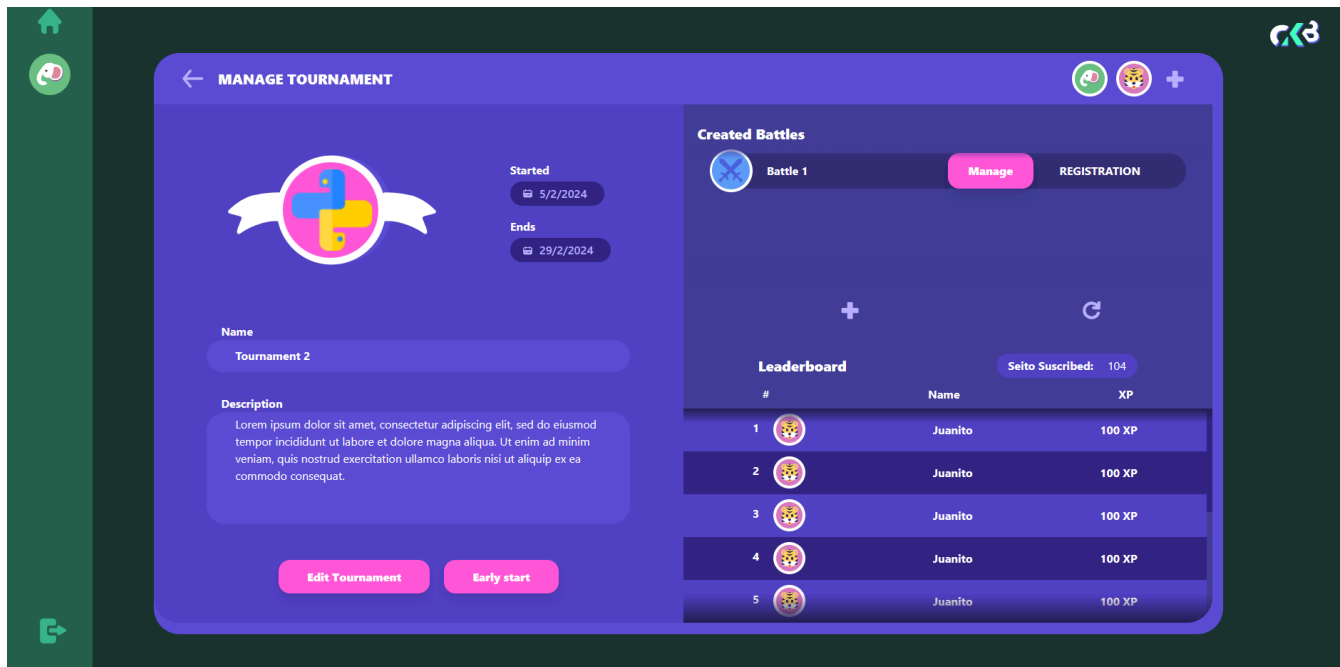


## Tournament Creation



This is the screen the educator can access when looking to create a new tournament, it requests the basic data to correctly generate a tournament, name, description and start/end dates.

## Tournament Management



### ○ Functionality

Here's a step-by-step explanation of how the login process works for both user types:

#### Dependencies Integration:

- The component initiates by importing necessary dependencies from React, React Router, Chakra UI, and axios, establishing the foundation for the tournament management system.

#### Functional Component Definition:

- "ManageTournament" is implemented as a functional component using arrow function syntax, providing a structured blueprint for the tournament management interface.

#### State and Context Initialization:

- The component's state is configured, encompassing variables for modal visibility, loading indicators, tournament data, battles, and toast notifications. The "useContext" hook is employed to access the global user context.

#### Implementation of useEffect Hook:

- The "useEffect" hook orchestrates three distinct operations:
  - a. On component mount, it retrieves tournaments from local storage, filters by ID, and fetches battles if not locally available.
  - b. It logs tournament data upon changes to the tournament state variable.
  - c. Similarly, battle data is logged when the battle state variable undergoes changes. Similarly, battle data is logged when the battle state variable undergoes changes.

### Battles Retrieval Process:

- The "fetchBattles" function takes precedence, facilitating the fetching of battles. It incorporates loading indicators, executes an API request, and updates relevant state variables.

### Refresh Mechanism for Battles:

- The "refresh" function is introduced to maintain battle data currency. Triggered by a user-initiated refresh, it calls the fetching function.

### Early Start Functionality:

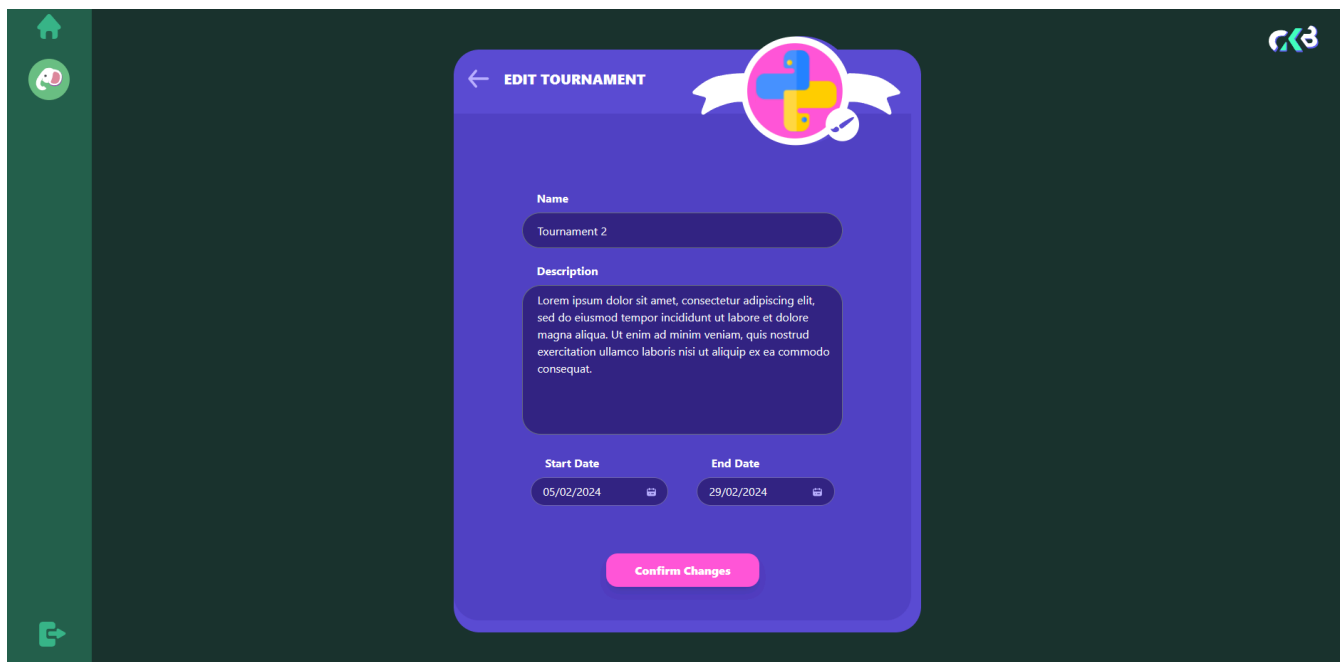
- "handleEarlyStart" manages the early start functionality. It initiates a PATCH request, updates the tournament's status, and communicates outcomes through notifications.

### Rendering UI Elements:

- The component culminates in JSX rendering, incorporating buttons, modals, text fields, and specialized components like "TopDecorator," "MiniDetails," "CreatedBattle," and "TeamLeaderboard." This presentation encompasses tournament details, created battles, and the leaderboard.

In essence, "ManageTournament" serves as the key to access the essential operations required for effective tournament administration.

## Tournament Edit

The image shows a mobile application interface for editing a tournament. A dark green sidebar on the left contains a home icon, a profile icon, and a right-pointing arrow. The main content area is a purple modal titled 'EDIT TOURNAMENT' with a back arrow and a puzzle piece icon. The form includes a 'Name' field with 'Tournament 2', a 'Description' field with placeholder text, and 'Start Date' and 'End Date' pickers set to 05/02/2024 and 29/02/2024 respectively. A pink 'Confirm Changes' button is at the bottom.

### ○ Functionality

Here's a step-by-step explanation of how the login process works for both user types:

**Dependencies Integration:**

- The component initiates by importing necessary dependencies from React, React Router, Chakra UI, and axios, establishing the foundation for the tournament management system.

**Functional Component Definition:**

- "ManageTournament" is implemented as a functional component using arrow function syntax, providing a structured blueprint for the tournament management interface.

**State and Context Initialization:**

- The component's state is configured, encompassing variables for modal visibility, loading indicators, tournament data, battles, and toast notifications. The "useContext" hook is employed to access the global user context.

**Implementation of useEffect Hook:**

- The "useEffect" hook orchestrates three distinct operations:
  - a. On component mount, it retrieves tournaments from local storage, filters by ID, and fetches battles if not locally available.
  - b. It logs tournament data upon changes to the tournament state variable.
  - c. Similarly, battle data is logged when the battle state variable undergoes changes. Similarly, battle data is logged when the battle state variable undergoes changes.

**Battles Retrieval Process:**

- The "fetchBattles" function takes precedence, facilitating the fetching of battles. It incorporates loading indicators, executes an API request, and updates relevant state variables.

**Refresh Mechanism for Battles:**

- The "refresh" function is introduced to maintain battle data currency. Triggered by a user-initiated refresh, it calls the fetching function.

**Early Start Functionality:**

- "handleEarlyStart" manages the early start functionality. It initiates a PATCH request, updates the tournament's status, and communicates outcomes through notifications.

**Rendering UI Elements:**

- The component culminates in JSX rendering, incorporating buttons, modals, text fields, and specialized components like "TopDecorator," "MiniDetails," "CreatedBattle," and "TeamLeaderboard." This presentation encompasses tournament details, created battles, and the leaderboard.

In essence, the screen to Manage Tournaments serves as the key to access the essential operations required for effective tournament administration.

- **Requirements accomplished:**

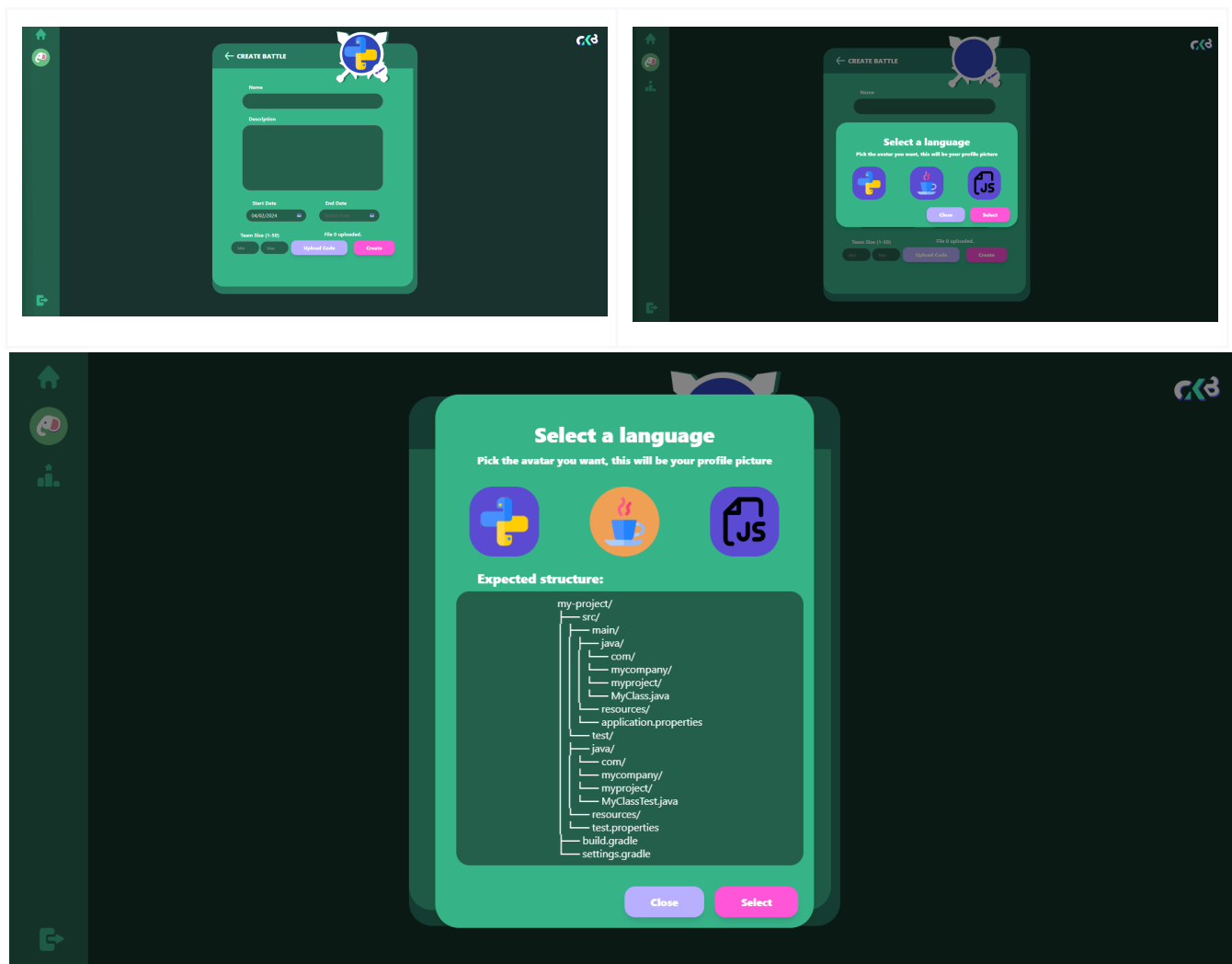
[R3] Educators can create new tournaments, defining code battles with descriptions, deadlines, and scoring configurations.

[R9] Educators can set a registration deadline.

- Battle

## Battle Creation

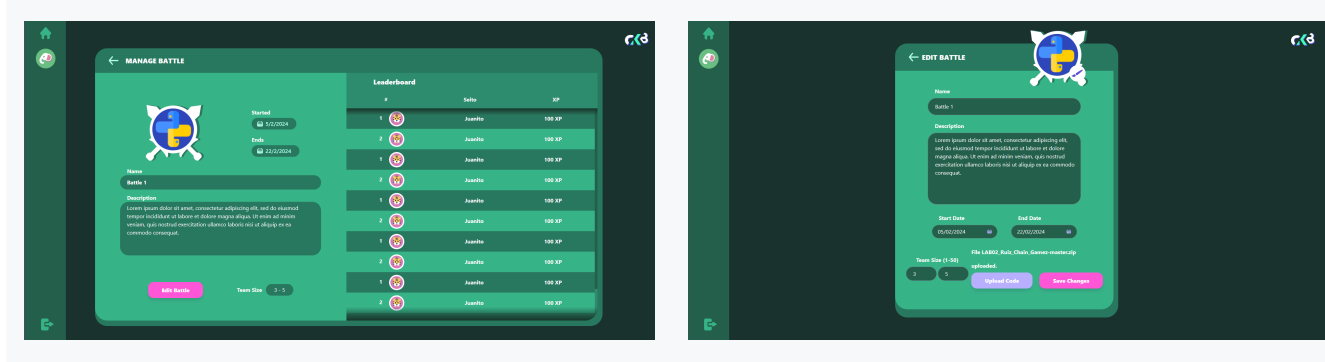
Battle Creation is fundamental for educators to craft specific coding scenarios within tournaments. It acts as a core feature influencing the learning objectives by tailoring challenges. It could be divided into sub-features like task creation, test case setting, and deadline management. Its significance lies in providing precise and varied coding challenges, essential for students to enhance their problem-solving skills.





## Battle Management

Battle Management ensures precise oversight of individual coding challenges within tournaments. It might include sub-features like task-specific configurations and submission deadlines. Its significance lies in tailoring and monitoring coding challenges, ensuring alignment with learning objectives and assessment precision.



### ○ Functionality

Here's a step-by-step explanation of how the login process works for both user types:

#### Dependencies Integration:

- The component begins by importing necessary libraries, including React, hooks like useState and useEffect, UI components (e.g., Text, TextField), and icons.

#### State Management:

- Utilizes the useState hook to create state variables, storing user input for tournament details (name, description, dates, etc.). Also employs hooks like useNavigate and showToast for navigation and displaying notifications.

#### Toast Notifications:

- Implements showToast and closeAll functions. showToast displays error messages as toast notifications using the chakra-ui library, offering a non-intrusive way to communicate issues.

#### Form Submission:

- The handleSubmit function, triggered on form submission, performs form validations. Upon successful validation, it sends a POST request to the server (via axios) for tournament creation. Manages the isCreated state variable based on successful

#### Early Start Functionality:

- "handleEarlyStart" manages the early start functionality. It initiates a PATCH request, updates the battle status, and communicates its start through notifications.

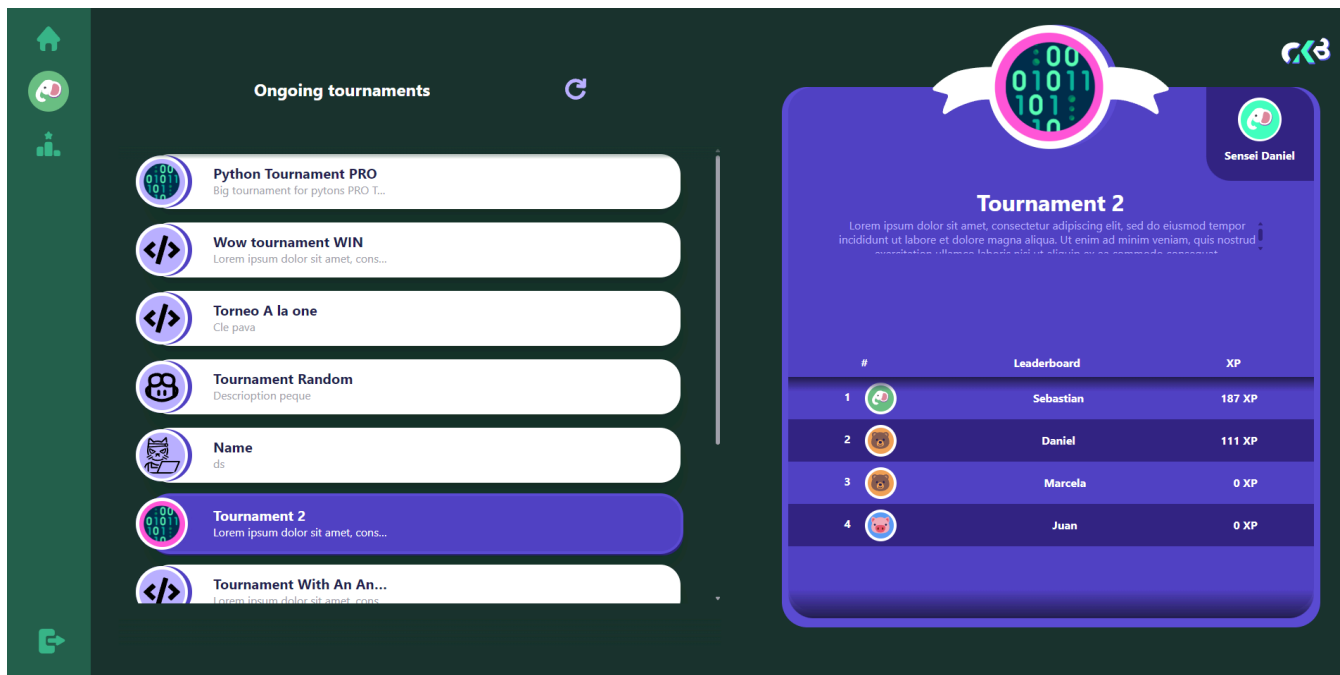
#### Conditional Rendering:

- Employs conditional rendering in the return statement, adapting UI components based on isLoading and isCreated states. For example, displays a loading screen during form submission and a completion component upon successful creation.

#### Successful Battle Creation:

- The "refresh" function is introduced to maintain battle data currency. Triggered by a user-initiated refresh, it calls the fetching function.

In essence, the screen serves as a controlled form component for capturing user input, executing form validations, communicating with the server that manages tournaments to create and add new battles to it.



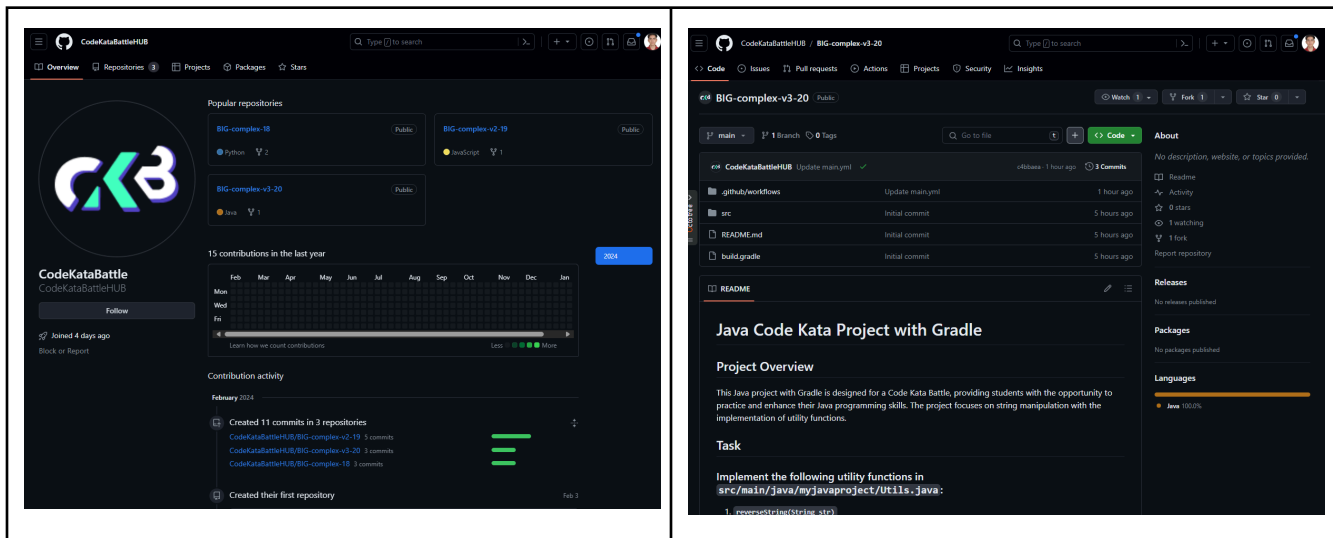
#### ○ Requirements accomplished:

- [R4] Educators can set minimum and maximum team sizes for battles within a tournament.
- [R5] Educators can manage and view the list of battles they've created within a tournament.
- [R6] Educators can grant permission to colleagues to create battles within a specific tournament.
- [R7] Educators can upload the code kata with the respective description and software project, including test cases and build automation scripts.
- [R9] Educators can set a registration deadline.
- [R10] Educators can set a final submission deadline.

### 2.2.3. Team Formation and GitHub Integration Component:

GitHub Integration is pivotal for seamless code submission and management. While seemingly independent, it plays a critical role in code version control and submission, making it an integral part of the platform. It may encompass sub-features like repository linking, version control setup, and automated code retrieval. Though not directly visible, its implementation significantly impacts user experience and collaborative coding practices.

- Integrate with GitHub for code repository management.
- Interaction: Communicates bidirectionally with the Automated Evaluation Component for code submission triggering and updates.



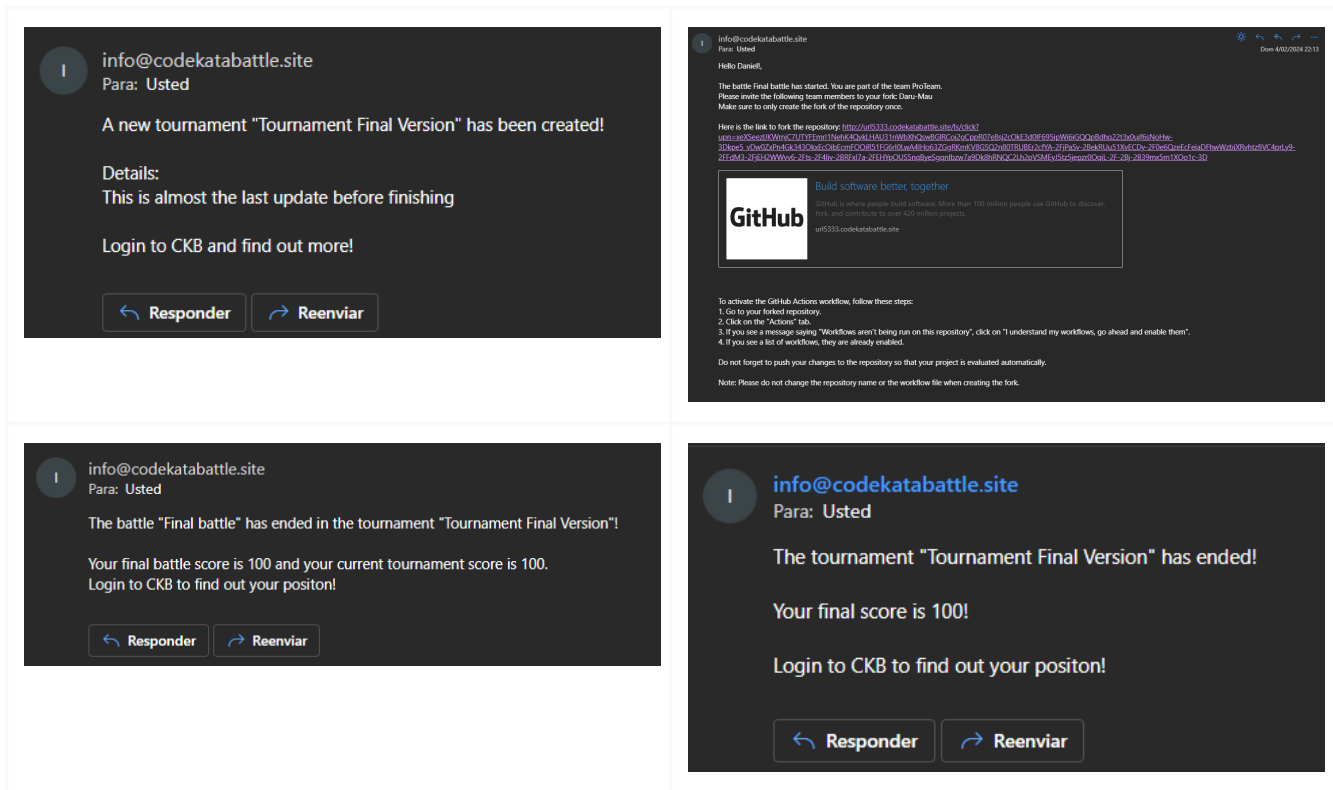
- **Requirements accomplished:**
  - [R13] The platform automatically generates GitHub repositories for each battle upon creation.
  - [R18] The system continuously updates battle scores during the battle period, displaying evolving rankings to students and educators.
  - [R20] The platform updates personal tournament scores for each student, presenting the cumulative performance across battles within a tournament.

### 2.2.4. Notification Component:

We were able to implement the notification system that manages timely communication across the platform. It is able to inform the student users about any updates, deadlines, and critical information regarding the tournaments and battles they are subscribed to. While not subdivided, its implementation ensures effective information flow, enhancing user engagement.

- Manages notifications regarding tournament/battle updates, deadlines, and score availability.
- Sends notifications to subscribed users and educators based on system events.

- Interaction: Triggered by events from various components, particularly Tournament and Battle Management and Scoring System Components.



- **Requirements accomplished:**

- [R21] Automated notifications inform students about new tournaments and upcoming battles within subscribed tournaments.
- [R22] Real-time updates notify users (students and educators) about rank changes, deadlines, and significant tournament-related information.
- [R23] Email notifications alert users about critical updates, including new tournaments, battle results, and personal scores.

- **Automated Evaluation Component:**

Automated Evaluation expedites the assessment process for submitted code solutions. This feature automates grading, executes test cases, analyzes code quality, and generates automated scores. While not subdivided, its role in providing timely feedback and code quality improvement among participants.

- Conducts automated evaluation of submitted code for functional aspects, timeliness, and code quality.
- Utilizes static analysis tools and test case execution to assess code quality and functionality.
- Interaction: Receives code submissions triggered by GitHub integration and updates the scoring system accordingly.

- **Requirements involved:**

- [R14] The platform performs automated evaluation of submitted code based on test cases, calculating the functionality score.

- [R15] Integration with static analysis tools for quality evaluation, considering security, reliability, and maintainability of code.

- **Scoring System Component:**

The Scoring System quantifies performance and progress within coding challenges. It calculates and assigns scores based on automated evaluations and potential manual scores provided by educators. While not divided, its implementation ensures fair and transparent evaluation.

- Calculates scores based on automated evaluation factors such as test case passes and timeliness.
- Allows for the incorporation of educator-assigned manual scores.
- Interaction: Integrates with all evaluation components to compute battle scores and updates user profiles and rankings.

- **Requirements involved:**

- [R11] Educators can set additional configurations for scoring.

- [R16] Educators can manually assess and assign personal scores to student submissions.

- [R17] After the submission deadline, educators review and provide manual evaluations based on code quality and adherence to the test-first approach.

### 3. DEVELOPMENT FRAMEWORKS

In this section, we will like to introduce to you the foundational choices that shape our CKB management systems architecture and functionality. Covering the selection of the programming languages and APIs play a pivotal role in determining the systems performance, scalability and efficiency in general. The development framework we choose aim to strike balance between the industry-standard tools, ease of development for the application and satisfy the specific requirements outlined in the Design Document

#### **Frontend: React Native (Javascript)**

For our frontend, we wanted a library that simplifies UI development. With our previous experience in another subject working with React Native to build a mobile application, we looked into React's declarative approach and its component-based structure, and went with it to make our application, because the React library enables us to build dynamic and efficient user interfaces.

##### **Advantages**

- We choose this framework in particular not just because we had previous practice with it, but also because its component-based architecture allows for modular development, promoting code reusability and maintainability, property that we appreciate given that in our design most of the components can be reutilized in different contexts and screens, so this feature saved us some programming time
- We aimed to build a modern, dynamic web application. React's compatibility with React Router made it an ideal choice for developing single-page applications, enhancing navigation and user interactions.

#### **Backend: Django (Python)**

We chose to work with python mainly because we were already familiar with it, considering that it is a language quite simple, yet powerful, that is vastly supported by its community and has all types of libraries and frameworks for various purposes. As stated in the Design Document, we went with Django for the backend part of our project because of all the advantages and possibilities that it provided and would have been favorable for us in the development of our project.

##### **Advantages**

- We wanted a backend framework that allowed us to develop quickly without sacrificing maintainability. Django's built-in features and "batteries included" philosophy helped us achieve just that.
- Python's large and active community means plenty of resources, support, and third-party packages. It translates to a smoother development process and readily available solutions to challenges we might encounter.
- Django's modular design ensures scalability and maintainability. We can build reusable components, reducing redundancy and easing future updates.

##### **Disadvantages**

- Like we already had our fair share of experience working with python, it was quite not that difficult to learn how Django worked, but normally developers unfamiliar with Python or Django may experience a quite harsh learning curve.

Using Python for the backend and JavaScript for the frontend helped us to maintain a consistent development environment. This synergy simplifies the development process, allowing our team to focus on delivering features rather than grappling with language disparities. Django, coupled with Django REST Framework, facilitates smooth communication between the frontend and backend. This helps us build robust APIs for efficient data exchange.

In essence, our decision to choose Python with Django and JavaScript with React was driven by a desire for efficiency, scalability, and practicality for us. This combination allowed our team to deliver a feature-rich, secure, and user-friendly application tailored to our project's unique requirements.

## 4. STRUCTURE OF THE CODE

In this section we will like to provide a brief overview as to how we decided to manage our project, the distribution that we applied to each section and a general explanation of the code

### Code Structure Overview:

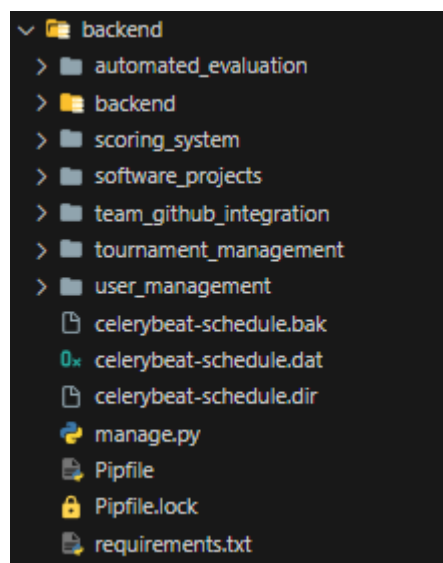
#### Folder Structure:

Our code is build inside a main source folder, and composed in its core by two folders corresponding to the backend and frontend sections of the project

**CKBApp/:** PereaRuiz\ITD\CKBApp  
**backend/:** PereaRuiz\ITD\CKBApp\backend  
**frontend/:** PereaRuiz\ITD\CKBApp\frontend

#### Backend Structure:

The backend of the project is currently built with all the necessary code to satisfy the features previously described. It ensures that the frontend interacts seamlessly with the database and facilitates the overall functioning of the software.



#### Main Components:

In our backend folder are contained all the sub-folders containing the modules and other key components for the definition of the database in Django and other key features that are needed for the complete and correct functionality of our program. We would like to highlight some essential components or modules, that form part of our backend:

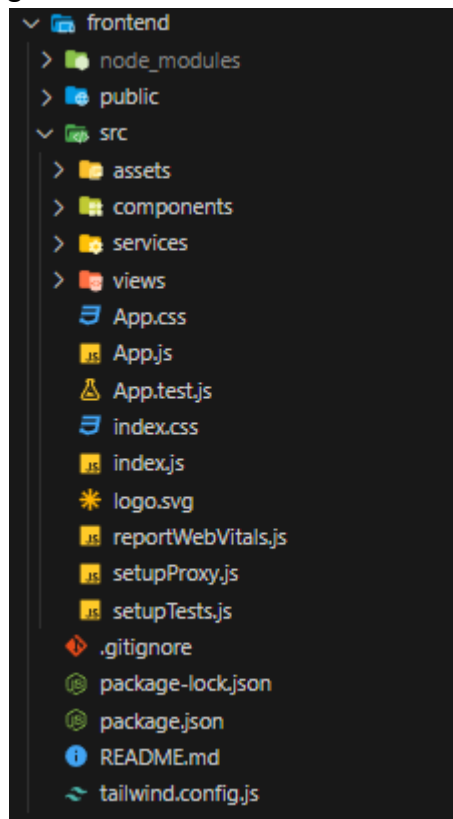
- \backend\backend\urls.py
- \backend\team\_github\_integration\workflow\_templates\
  - In this folder are contained the standard templates that the files and projects uploaded by the educator must follow for it to be valid and able to be evaluated by the automatic scoring system



- \backend\software\_projects\
  - This folder is the one that store all the zip files uploaded by the educator and that are an essential part of the creation of a battle

### Frontend Structure:

On the other end, the frontend is divided between four main folders, one containing all the assets for the project, another that have all the components created for the project and that can be used between each other, a service folder that integrates our frontend with the backend, and the folder containing all the main screens that form the application.



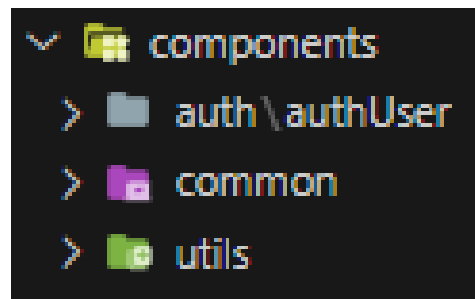
### Main Components:

Here we would like to highlight some essential components or modules, that form part of our frontend:

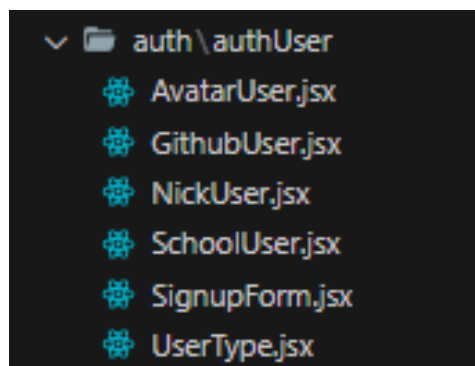
- \App.js
  - The App.js file serves as a pivotal entry point and orchestrator for the entire front-end application. Here's why it's important:
    - Different routes are mapped to specific components, ensuring that the correct views are rendered based on the URL.
    - It distinguishes between routes for student and educator functionalities, each having its own set of views and routes.
    - Protected routes are implemented to ensure that certain pages are only accessible to authenticated users.
    - The GlobalRankings component is included as a route for both students and educators, emphasizing a shared view of global rankings across user types.

- `\views\auth\`  
In the auth folder are contained all the views related to the user registration, login and authentication, as well as the component for the account activation and password reset functionalities
- `\views\educator\EducatorHome`  
It's the component responsible of rendering the home page for educators and its a main part of our project, because its from where the educator user can access all its functions, like manage, create and edit tournaments and battles
  - Here we define the UI for the educators when they log in to the application. It presents a clean and organized layout where educators can view and manage their tournaments.
  - We fetch the tournament data when the component mounts. The data is fetched from the server and stored in the local storage. This enhances the performance by reducing redundant API calls to provide a smoother user experience.
- `\views\student\StudentHome`  
This file plays a main role in the CKB system, serving as the frontend component responsible for managing and displaying essential information to students.
  - In this view we display a user-friendly interface that displays both the current tournaments the student is subscribed to and the past ones that he has participated in.
  - We implemented dynamic fetching to present an updated list of tournaments, ensuring that the students always have the latest information about the ongoing tournaments
  - Students can easily navigate through tournaments, view details, and join battles using buttons
  - Incorporates local storage to cache tournament and team data, optimizing the user experience by reducing the need for redundant data fetching.
  - Utilizes a context to access and manage the active user's information, allowing for personalized interactions based on the user's role and personal data.
- `\services\`  
Here are included all of the context related components, like the register context and user context, that manage the user information across some of the screens and the backend too; the different providers for the application and the component that fetches the data for the users, and even some visual effects like a loading effect.
- `\components\`

In the components folder are contained all of the elements that can be, and will be used for the different views, or other components per se, are divided into 3 main categories, in which we will highlight some of the most used or important ones

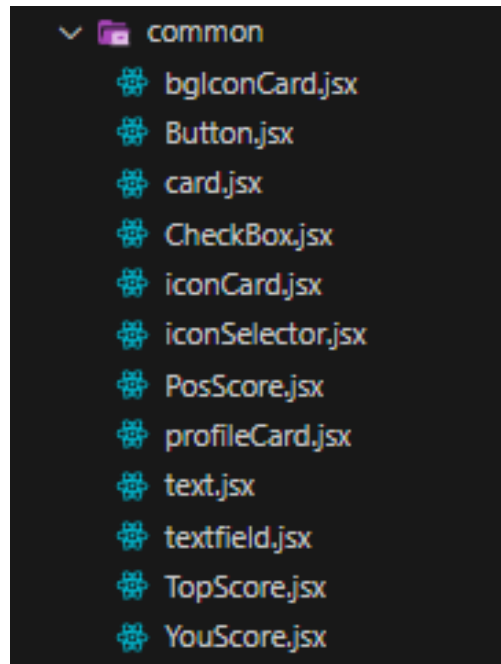


- auth\
  - Here are contained all of the different forms loaded when a user is getting registered in the application
  -



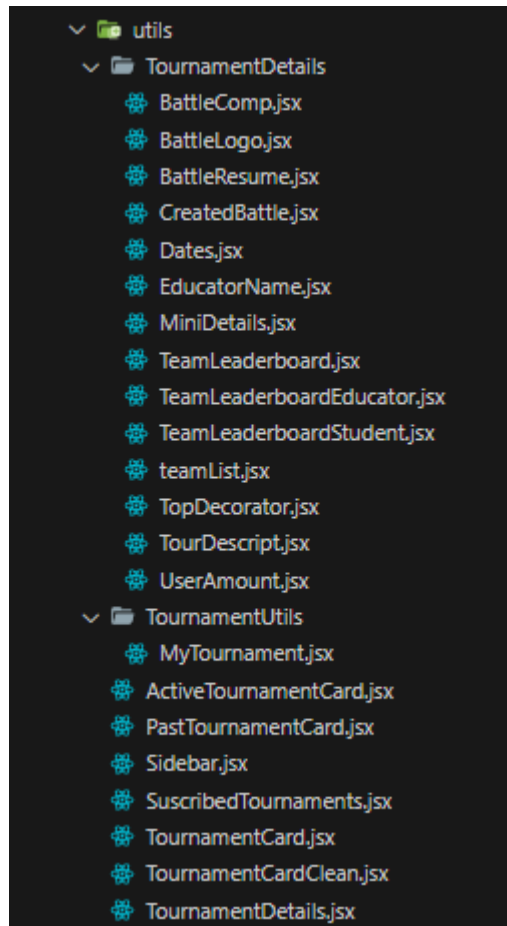
- common\

In this folder we placed the most used components between all the application, mostly re-implementations of common components like text, button and textfield, and other custom made that are being used across all the application



- utils\

This folder can be considered an extension of the main views for the tournament and battles related content for the educator, because the majority of its components are elements that are used in this specific views, like the one that allows us to display the information related to a battle, or load and display the tournaments created or subscribed to the educator profile



### Configuration Files:

Our project has some key files that are in charge of the specific configurations, both in the backend and the frontend, these are:

- **CKBApp\backend\backend\settings.py**

This file serves as a centralized configuration hub, allowing developers to customize and fine-tune the behavior of their application. It consolidates various settings related to database connections, middleware, security, templates, and more, providing a convenient way to manage the project's overall configuration.

- **CKBApp\backend\backend\requirements.txt**

This file contains all the necessary libraries that are needed to be installed for the backend to run correctly, there is a section ahead that will explain an easy way to do it.

### Code Version Control:

For our software development process, we implemented a consistent, steady and robust version control system that would help us manage and track the changes done to our code effectively. This provided an efficient way to collaborate, ensured the code stability and provided an easy navigation through the changes history

## Version Control System:

We have chosen GitHub as our version control system. GitHub is a widely adopted platform that leverages the Git version control system. Git allows us to keep a detailed history of our project, including all modifications made to the source code, enabling collaboration among team members, rollback to previous versions, and seamless integration with various development tools. This choice aligns with industry best practices and promotes a streamlined and organized approach to code management.

## Branching Strategy:

During all the process of the development of this code, we followed the strategy previously proposed in the DD, parting the project in 4 main branches:

- 1-login-and-signup
  - Created base components and DB connection
  - Feat: Backend Register and Login
  - Feat: Login and Signup sequence
  - Feat: Main Student Page
  - Feat: Signup + Login data
  - Feat: Student Profile screens
  - Feat: Battle Resume and Details
  - Feat: Login and Signup
- 2-tournament-and-battle-creation
  - Update Sidebar.jsx
  - Feat: Login, Main screens
  - Feat: Base models creation
  - Feat: Educator home and profile Screen
  - Feat: Educator Main Screens
  - Feat: All creation-related screens
  - Update: Creation and Management screens
  - Feat: Backend models and view
  - Feat: Battles & Tournament creation + teams
  - Fix: AutoLogin, Battle Creation
  - Feat: Full team creation, and joining
- 3-evaluation-and-scoring-mechanisms
  - Feat: Battle creation on GitHub
  - Feat: Visual bugs in educator screens
  - Feat: Evaluation and scoring system with GitHub
  - Feat: Creation and Edit Screens, Global Ranking
- 4-management-and-ranking-implementation
  - Fix: Joining ongoing battles validation
  - Feat: Battle ranking
  - Update: Visual bugs and condition screens
  - Feat: Consolidate and end battle

- Fix: Team leaderboard with students pending

## **API Documentation:**

### **Communications Inside the Application**

Our application relies completely in the Django Rest Framework (DRF) to provide a seamless communication between the frontend and the backend components. It serves as the foundation for creating robust and standardized APIs that allows us to exchange data, ensuring a responsive and dynamic experience.

### **Email Notification Services**

For the email notification services, we integrated SMTP from SendGrid into our application. With this combination we are able to efficiently manage email communications with the users, from user authentication and verification, to notifications on relevant information about the tournaments and battles. SendGrid is a reliable infrastructure that, in addition to the Django APIs, ensures that the users receive important information in a timely manner.

### **GitHubs APIs Integration**

For the management of the battles and the teams, we relied on the GitHubs API to dynamically generate and allow access to the repositories. This enables us to create a collaborative and controlled environment for the students during the battles. By utilizing GitHubs native features, such as repository creation, branching, workflow and merging, we enhance the collaborative and competitive aspects of our application.

These integrations collectively contribute to the robustness and functionality of our application, providing users with a seamless experience while ensuring effective communication, secure authentication, and collaborative development through version control.

## 5. Testing:

The Testing phase is crucial in ensuring the reliability and functionality of the CodeKataBattle (CKB) platform. It consists of various testing levels and methodologies to validate the correctness, performance, and integration of different components.

### 5.1. Component Testing

Unit testing involves evaluating individual units or components of the software to ensure they function as intended. This was relevant because thanks to our considerable number of different components that may work within other, it was bound to present some compatibility issues with the different properties, with that in mind, this kind of testing was a fundamental practice that enhanced our software development experience by promoting early bug detection, maintaining code quality, and supporting a more efficient and agile development process.

- **Backend**

For Django, the component testing is an essential aspect of ensuring the correctness and functionality of each component, like the views, models and form, there are various ways in which Django allow us to test its components, and we implemented the following:

- **Test Classes and Methods**

Django provides a testing framework with a base `TestCase` class that developers can extend to create test classes that we write to evaluate some specific component or functionality

- **Assertions**

We used assertions to check if the output given matched the result we were expecting, this was particularly useful in handling all the information that we were bringin back and forth between the backend, frontend and the different components of our application

- **Database Handling**

The `TestCase` class also rolls back database changes after each test to maintain a clean slate for subsequent tests, this function was useful to us because it allowed us to handle the database transactions during the testing phases.

- **Client for Simulating Requests**

Django has a test function that provides a test-client that allows us to simulate HTTP requests and receive responses. Thanks to this, we were able to simulate the behavior of our application when deployed, making sure everything was working as intended, especially the email notifications and the github integrations.



- **Frontend**

On the other hand, component testing in a React application involves verifying the behavior and functionality of individual React components. Here's how JavaScript React tests work for component testing:

- **Rendering Components:**

React testing libraries provide functions for rendering React components within a test environment. This allows developers to simulate the rendering of components in an isolated context, and it's a simple yet effective way to try if our components are behaving the way we intended them to.

- **Simulating User Interactions:**

React testing libraries allow simulating user interactions with components, such as clicking buttons, entering text, or triggering events. This ensures that components respond correctly to user input.

- **Async Testing:**

For components with asynchronous behavior, testing libraries support asynchronous testing patterns to handle promises, callbacks, or async/await functions.

## **5.2. Integration Testing:**

Integration testing ensures that different modules or components of the system work cohesively when implemented together.

To achieve comprehensive system testing, a range of methodologies, including functional, non-functional, end-to-end testing and usability testing, will be employed. Each method plays a crucial role in assessing different aspects of the system, from its core functionalities to its responsiveness, security, and scalability.

### **Testing Methodologies**

As explained in the Design Document (DD), the system testing phase encompasses several testing methodologies:

- **Functional Testing:**

- **Description:** Evaluates system functions against specified requirements.
- **Approach:** Conducts validation of features like user authentication, tournament and battle creation, GitHub integration, scoring, and notifications.

- **Non-functional Testing:**

- **Description:** Assesses system characteristics beyond functional aspects.

- **Approach:** Includes security testing, usability assessment, reliability checks, and compatibility testing to ensure the system's robustness under diverse conditions.
- End-to-End Testing:
  - **Description:** Measures system performance under varying load conditions.
  - **Approach:** Conducts stress testing, load testing, and scalability checks to assess system responsiveness and stability.
- Usability Testing:
  - **Description:** Evaluates user interface intuitiveness and user experience.
  - **Approach:** Conducts user-centric tests to ensure the platform's ease of use and accessibility for diverse users.

Through these tests, we aim to do a full assessment of our CKB platform, covering all its possible functionalities and scenarios, ensuring its correct and consistent functionality, security and performance, and being able to provide the users with a leisure and smooth experience while using it.

### 5.3. Additional specifications on testing

These additional specifications detail the focused testing approaches for each methodology, aiming to comprehensively evaluate the CodeKataBattle platform. The diverse set of tests ensures adherence to functional requirements, security standards, performance benchmarks, and user-centric design principles.

#### Functional Testing

- We validated the login, registration, password recovery, and role-based access control functionalities for all users.
- All the educator features related to the creation, management and edition of battles and tournaments were tested and secured their correct functionality
- We make sure that the code repository linking, commit tracking, and submission functionalities are working correctly after various tries securing and analyzing the correct way to implement and test the files provided by the educator.
- The notifications system delivers the updates and score availability in a timely manner and we have tested its correct functionality and the content of the messages.

## 6. INSTALLATION INSTRUCTIONS

### 6.1. Prerequisites:

Before proceeding with the installation, ensure you have the following prerequisites installed on your system:

**Python3:** Download Python | Python.org

**Node.js:** Node.js (nodejs.org)

### 6.2. Download the Source Code:

Download or clone the GitHub repository using the following link:

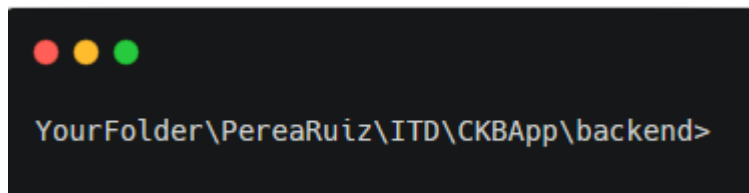
**Download link:** sebaxe07/PereaRuiz: Project for SE2 (github.com)

**Clone repository:** `gh repo clone sebaxe07/PereaRuiz`

Save the repository in your preferred location.

### 6.3. Backend Configuration:

Open your preferred code editor and navigate to the "backend" folder of the GitHub repository you just installed, the path should look something like this



Install the required libraries by running the following command:

**`pip install -r .\requirements.txt`**

This command will automatically install the necessary backend dependencies.

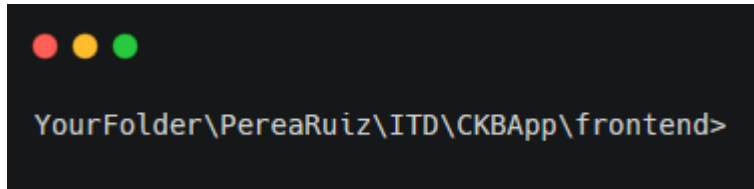
After the installation are correctly completed, you now can start the backend by running:

**`py manage.py runserver`**

This command initiates the backend of the application.

### 6.4. Run the Application:

Now navigate to the "frontend" folder and access a new terminal.



Install the frontend libraries by executing the following command:

### **npm install**

This command installs all the required frontend dependencies.

Once the installation is finished, run the following command to open the application:

### **npm start**

This command launches the application, automatically opening it in your default web browser.

Alternatively, you can access it manually by entering the provided localhost link in your browser.

## 7. EFFORT SPENT

Daniel Mauricio Ruiz Suarez

Chapter	Effort(in hours)
1	7
2	13
3	17
4	16
5	9
6	2

Sebastian Enrique Perea Lopez

Chapter	Effort(in hours)
1	
2	
3	
4	
5	
6	

## 8. REFERENCE

### **Diagrams:**

Lucidchart. (2023). Diagrams. Retrieved from <https://www.lucidchart.com>

### **Mockups:**

Figma. (2023). Mockups. Retrieved from <https://www.figma.com>

### **IEEE Standards:**

IEEE. IEEE Standards. Retrieved from <https://grouper.ieee.org/groups/802/22/Documentation/format-rules.html#Document>

### **Software Engineering 2 course material:**

Politecnico di Milano 2023-2024