POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering 2
Design Document

Author(s): **Filippo Balzarini - 10719101**
Christian Biffi - 10787158
Michele Cavicchioli - 10706553

07 January 2024 - Version 1.0
Academic Year: 2023-2024

Contents

1 Introduction

1.1. Purpose

This document contains the design description of the *CodeKataBattle* system. It includes the architectural design, the user interface design and the descrpition of all the operations that the system will perform. It also show how the requirements and use cases detailed in the RASD document are satisfied by the design of the system.

This document is intended to be read by the developers of the system, the testers and the project managers. It is also intended to be used as a reference for the future maintenance of the system.

1.2. Scope

The *CodeKataBattle* system is a web application that allows educators to create challenges for their students based on solving programming problems. In particular the system is based on the concept of *Code Kata* that is an exercise in programming which helps a programmer hone their skills through practice and repetition. The system will allow the educators to create competition and battle based on *Code Kata*. The students will be able to participate in the battles with a team or by themselves and solve the challenges in order to earn points. The system will also provide a leader board that will show the ranking of the students based on their scores.

A more detailed description of the system can be found in the RASD document, whilist in this document is provided a detailed description of the design of the system to implement the requirements and use cases described in the RASD document.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

User	Anyone interacting with the system, it can be both a Student or an Educator
Manage	Create, supervise and edit a certain element of the application.
Code Kata	A challenge intended to improve programming abilities, including description, test cases and build automation scripts.

Table 1.1: List of definitions

1.3.2. Acronyms

ST	Student
ED	Educator
CKB	CodaKataBattle
RASD	Requirements Analysis and Specification Document
SAT	Static Analyzer Tool
T	Team
MVC	Model View Controller

Table 1.2: List of Acronyms

1.4. Revision history

Date	Revision	Notes
07/01/2024	v1.0	First release

Table 1.3: Revision table

1.5. Reference Documents

- The specification document of the project: *Assignment RDD AY 2023-2024*
- The RASD document of the project

1.6. Document Structure

The document is divided in 6 sections:

- **Introduction**: provides a brief description of the purpose and the scope of the system. Morover it contains the definitions, acronyms and abbreviations used in the document and the reference documents.
- **Architectural Design**: this section provides a description of the architecture of the system, including the components and the interfaces between them. It also includes the runtime view of the most important operations of the system, the deployment view and the architectural styles and patterns used in the system.
- **User Interface Design**: includes the mockups of the user interface of the system.
- **Requirements Traceability**: this section shows how the requirements described in the RASD document are satisfied by the design of the system.
- **Implementation, Integration and Test Plan**: this includes the step-by-step plan for the implementation and testing of the system.
- **Effort Spent**: this section highlights the effort spent by each member of the group to redact this document.

2 | Architectural design

2.1. Overview: High-level components and their interaction

Inside the CKB application we are describing in this document we can distinguish five high-level components: the **CKB server**, the **Evaluator Server**, the **DBMS**, the **Mail Server**, and the **WebAppUI**. The **CKB server** is the main component of our system and contains every piece of business logic required to satisfy the goals we specified in the RASD. The **Evaluator server** has the purpose to run some analysis on the code written by a team participating in battles, and to compute a score using the results of such analysis. Since these two servers need to interact with each other they expose some APIs to do just so. The role of the **DBMS** is the usual one, interacting with a database by executing some operations on it (further information about the database is in the *Other Design Decisions section* [2.7.2]). The **Mail Server** is used to send notifications by e-mail to the platform's users. At last, the **WebAppUI** represents the web application from the client's perspective.

2.2. Component view

Client

- **WebAppUI**: represents the web application, which is reachable by any browser and usable only after a user has been authenticated using the ***AuthInterface***. It allows both STs and EDs to perform a certain set of actions based on the type of user by using the ***DashboardInterface***

Server

- **AuthenticationService**: provides the set of procedures required to handle the authentication of a user into the system (i.e., login, registration)
- **DashboardManager**: used as an intermediary between the *WebAppUI* and the other components of the server to provide the web application only the functionality strictly necessary for it to work properly.
- **CompetitionManager**: handler of all functionalities regarding competitions (for both STs and EDs) excluding badges, which management has been delegated to the *BadgeMonitor* component
- **Battle Manager**: same as the *CompetitionManager* but transposed to handle battles. Delegates the team creation/deletion to the *TeamManager*
- **TeamManager**: used to manage the teams used by the STs to participate to battles, it also includes the invite handling
- **BadgeManager**: its purpose is to deal with badges, create/remove a badge with its related rule, and perform checks to verify if a ST enrolled in a competition satisfies any badge rule defined in such competition
- **NotificationService**: its main goal is to implement procedures to send various types of notifications to the application's users; to send such notifications (mails) it uses the ***MailAPI*** provided by the *MailServer*
- **DataManager**: mediator between the model components and the *DBMS*; it uses the procedures provided by the ***DBMS_API*** to implement a set of functions, which have the sole purpose of manipulating the database or retrieving information from it
- **EvaluatorController**: it is called through the ***EvaluationAPI*** by *GitHub Ac-*

tions on each commit performed by a team. Its purpose is to control the evaluation process by calling the *StaticAnalyzer* and the *CodeEvaluator* to perform the proper checks on the last committed code. Moreover, it uses the functions provided by the ***EvaluatorInterface***, ***AnalyzerInterface***, ***ScoreInterface*** to change the configuration of the code evaluator and the static analyzer, with the last interface it sets the score functions used by the *PointManager*

- **CodeEvaluator**: used to execute the source code of a team's repository with the set of test cases provided by the EDs of the current battle. It exposes the ***EvaluatorInterface*** to be used to configure the evaluator for instance in terms of test cases or maximum execution time
- **StaticAnalyzer**: provides the ***AnalyzerInterface*** to configure the static analyzer, which will be used to analyze some input code. Such analysis will return some results, which can be sent to the *PointManager* through the proper interface still provided by the PointManager (this holds also for the *CodeEvaluator*)
- **PointManager**: as the name suggests, it manages the points assignment. In particular it provides two interfaces that are used by the *StaticAnalyzer* and the *CodeEvaluator* to send the results of their analysis. Such results are put in a score function designed by the EDs to compute the partial score of a single analysis; once the *PointManager* computes the score of both the evaluations (static analysis and code evaluation) it updates the final score of the team in the database

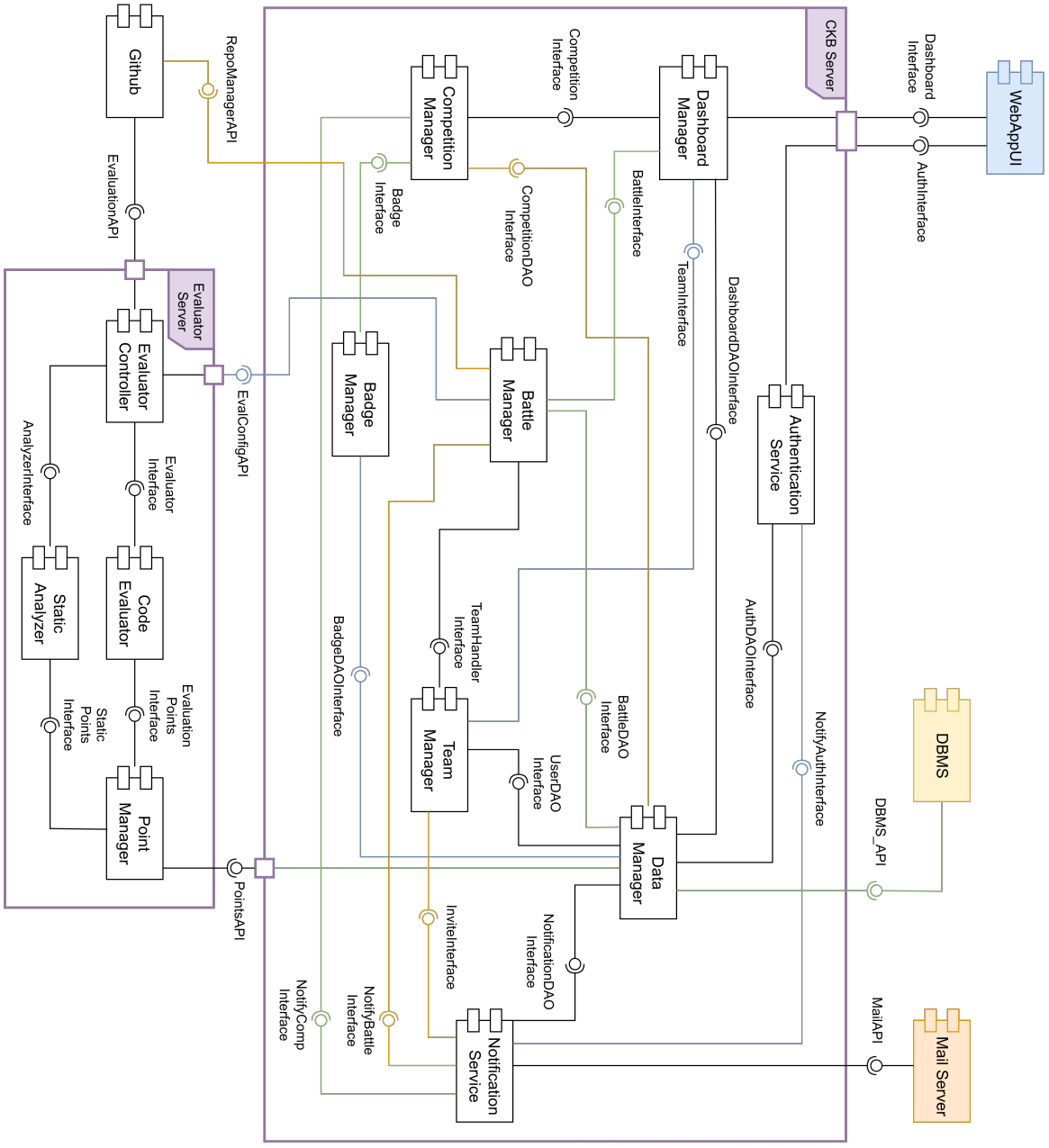

Figure 2.1: Deployment view

2.3. Deployment view

The following diagram shows the deployment view of the system. It illustrates the distribution of the components of the system on the different nodes and how they communicate bewteen them. The system is composed by four tiers: the client tier, the application tier, the data tier and the evaluation tier. The client tier is composed by the web browser, which is used by the users to access the application. The application tier is composed by the application servers, which are used to handle the requests from the users and to communicate with the database and with the evaluation servers. The data tier is composed by the database, which is used to store the data of the application. The evaluation tier is composed by the evaluation servers, which include both the static analyzer and the code evaluator.

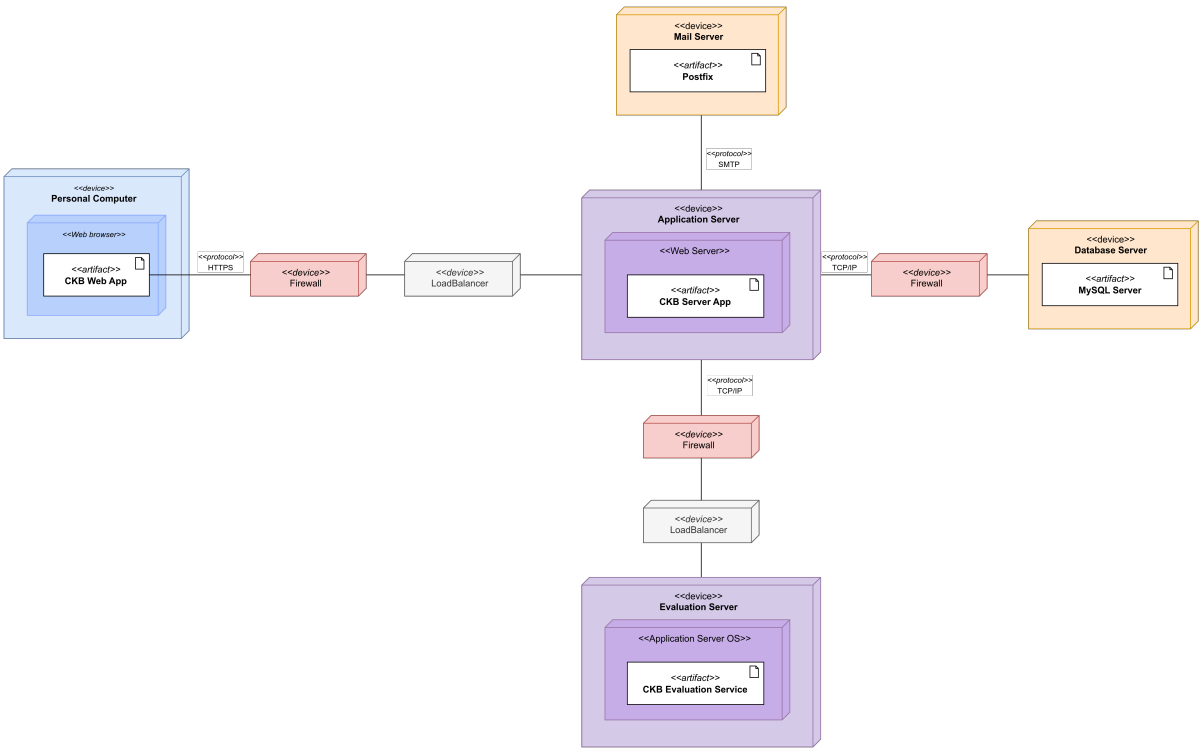

Figure 2.2: Deployment view

2.3.1. Load Balancing

Since the CKB will be potentially have a lot of concurrent users, we decided to use a load balancer to distribute the load among multiple servers. This allows us to have a more scalable application, because we can add more servers to the system to handle the increasing load.

A load balacing is also applied for the Evaluation Servers to handle the queue of the new commits to evaluate.

2.3.2. Firewall

To protect the application from external attacks, we decided to use multiple firewall to filter the traffic. In particular, we have a firewall between the client and the load balancer, a firewall between the application servers and the load balancer before the Evaluation Server and a firewall between the application servers and the database. This allows us to have a more secure application, because the firewall will filter the traffic before it reaches the different components of the system.

2.4. Runtime view

This section contains the sequence diagrams of the most important operations of the system. The diagrams include the component that we have already described in the previous section and the external components that are involved in the operations.

User Registration

When a user wants to register to the system, he/she has to fill in the registration form and submit it. The difference between a ST and an ED is in the information passed with *UserInfo* object, where the ED has to insert also the information about the institution he/she works for.

The whole process is mainly handled by the *Authentication Service* component, that interact with the *Data Manager* component to validate the information and insert the new user into the DBMS.

The system will check if the information inserted are valid and if the user is not already registered. This check is done internally from CKB and if the information are valid and the user is not already registered, the system will insert it into the DBMS and sends an email to the user with a link to confirm the registration, using the *Notification Manager* component. The user will click on the link and the *Authentication Service* will confirm the registration.

If the information inserted are not valid, the user is already registered or the confirmation link is expired, the system will send an error message to the user.

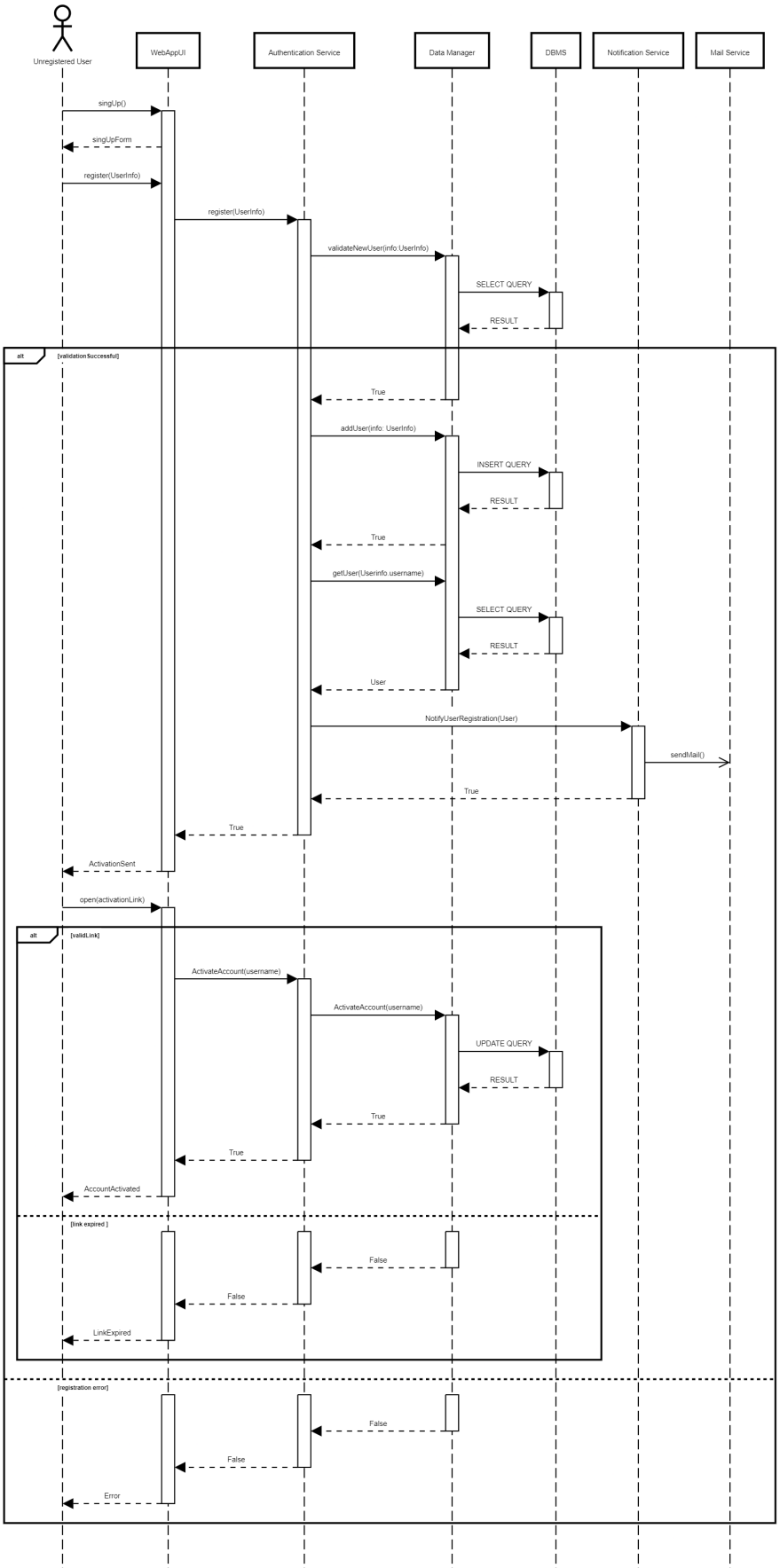

Figure 2.3: Registration sequence diagram

User Login

When a user wants to login to the system, he/she has to fill in the login form and submit it. This process is equal for both the ST and the ED. As the User Registration, the whole process is handled by the *Authentication Service* component, that interact with the *Data Manager* component to validate the information. Once the user is logged in, the *Authentication Service* will generate a token for the user, send it to the client and the user can finally access the dashboard.

If the login information are not valid, the system will show an error message to the user.

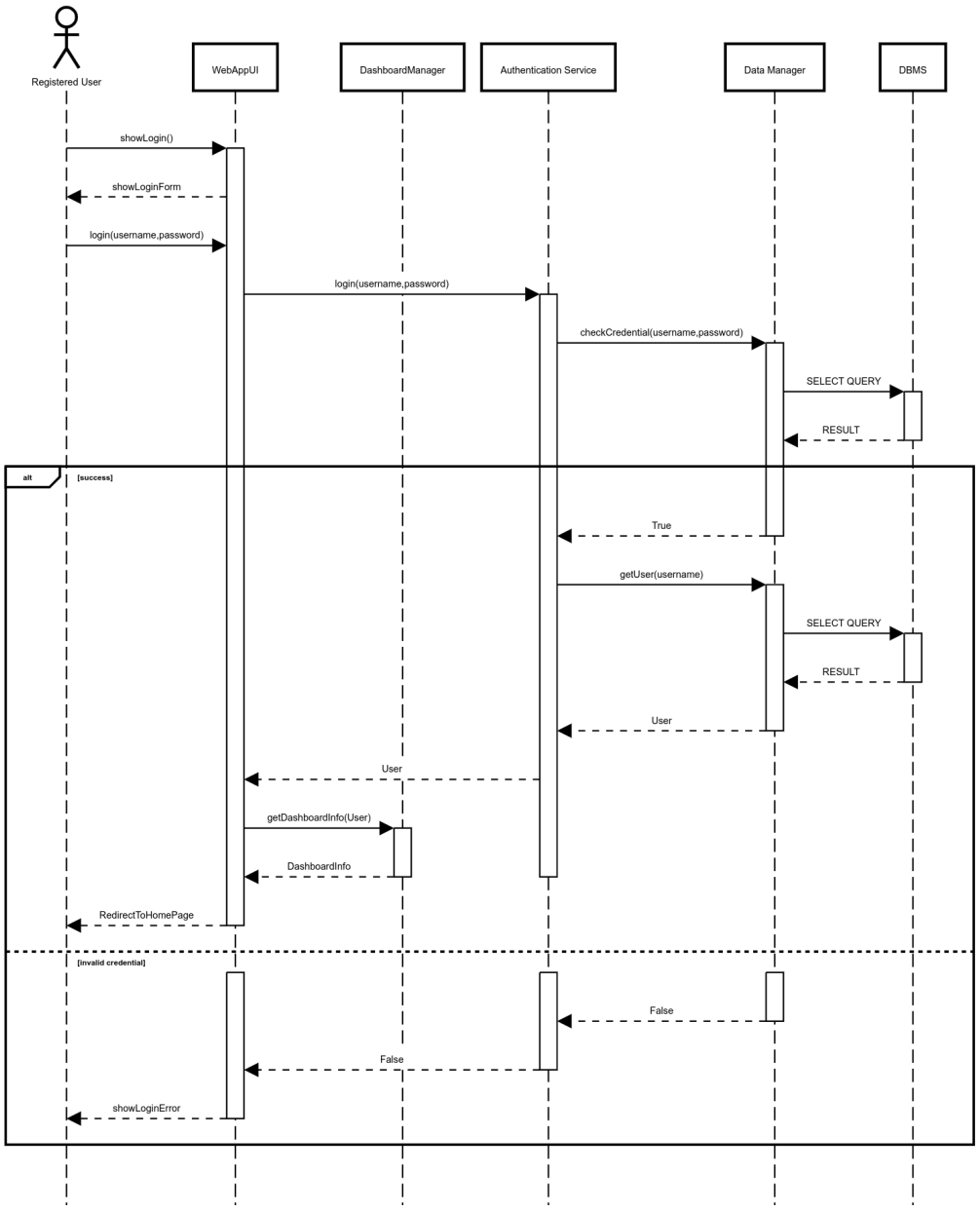

Figure 2.4: Login sequence diagram

ED Creates Competition

Here is shown how a competition is created by an ED. The ED has to fill in the form with the information about the competition and submit it. The *Competition Manager* component will check if the name is available and if so the competition will be created and inserted into the DBMS using the *Data Manager* component, otherwise a new name will be requested. The system will then returns the competition and will be shown the competition page.

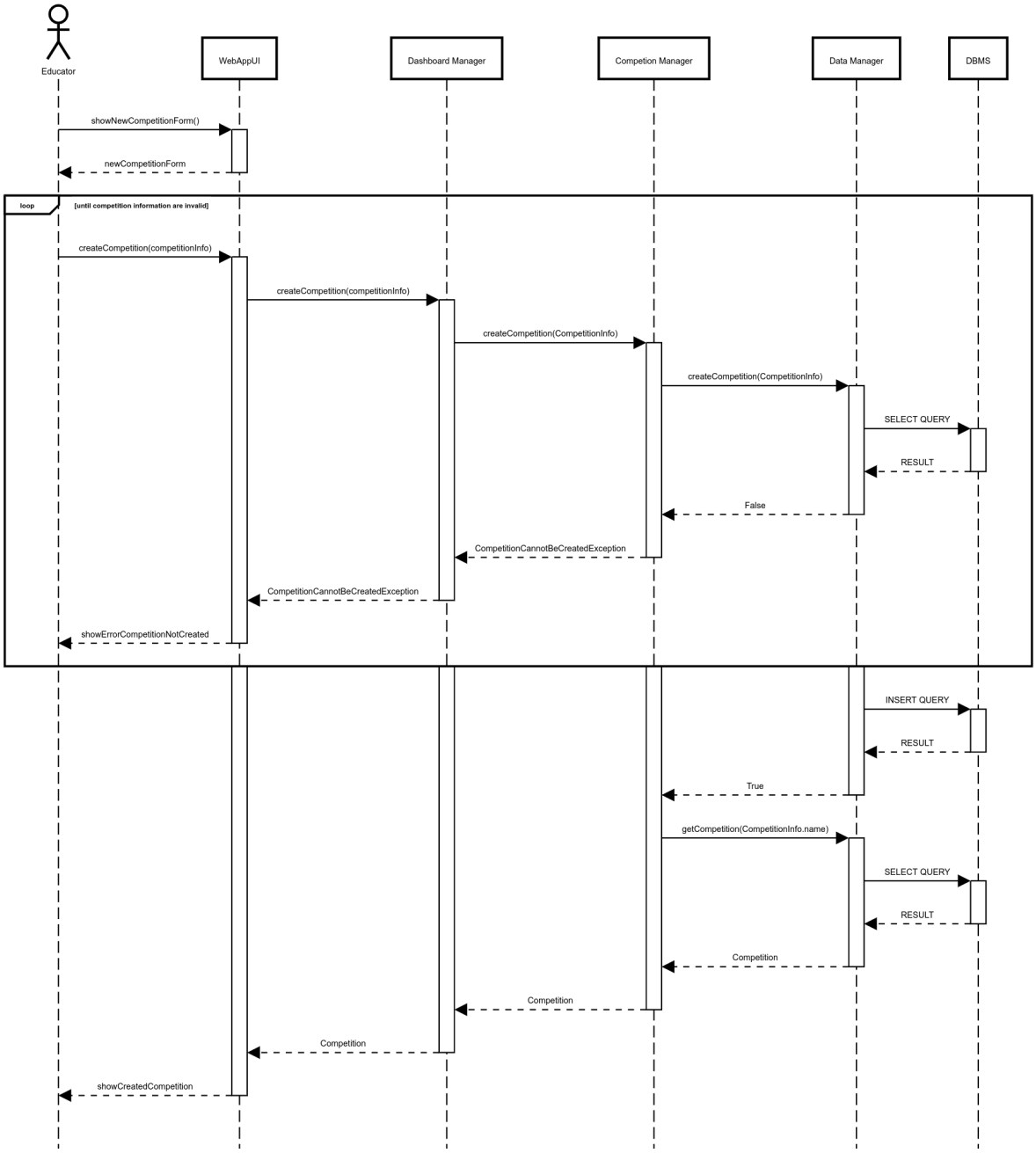

Figure 2.5: Create competition sequence diagram

ST Joins Competition

The ST firstly visualize all the available competitions in the platform, then he/she can choose one of them and join it. The *Competition Manager* component will handle both the search of the available competitions and the insertion into the DBMS using the *Data Manager* component. The system will then show a success message to the user.

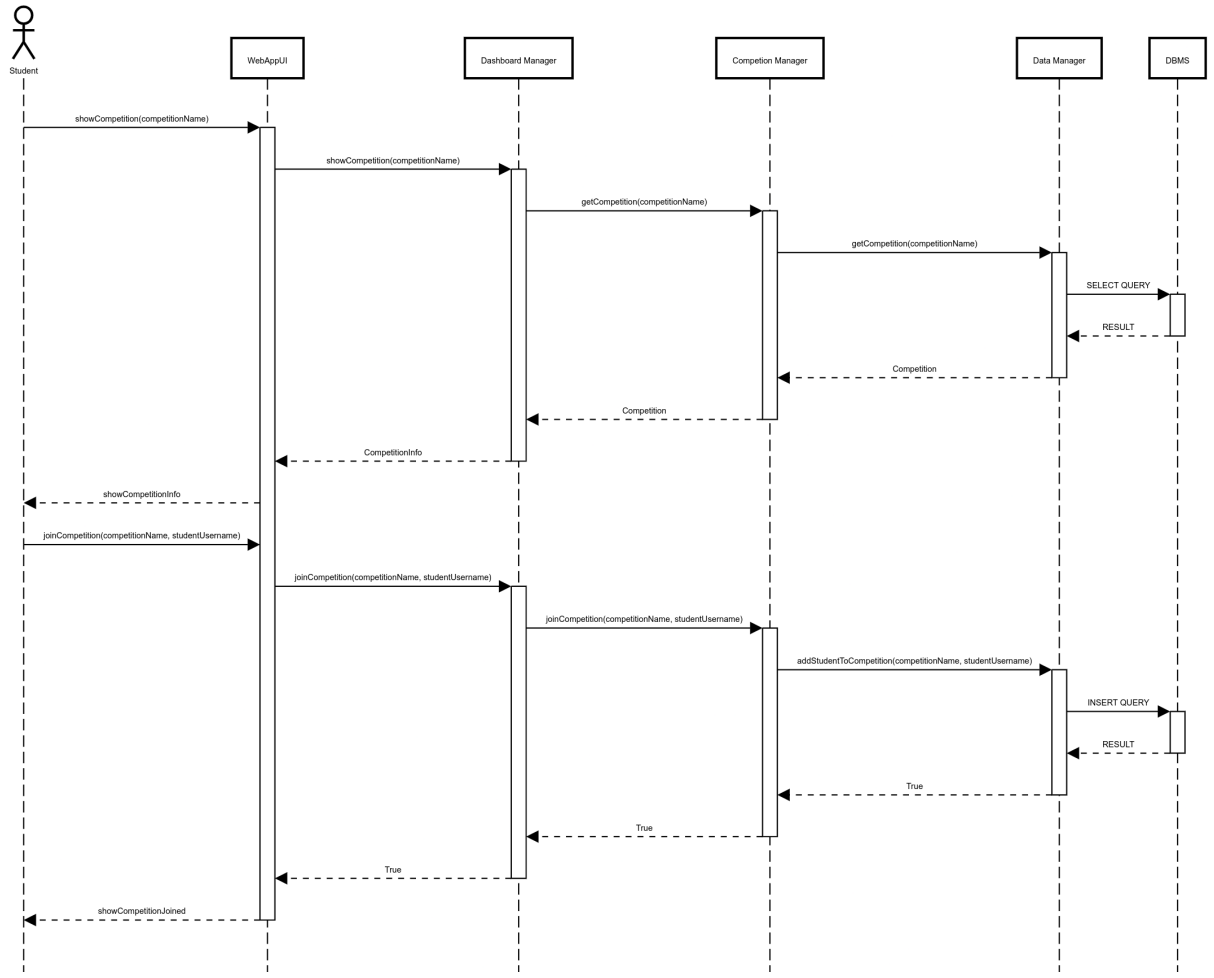

Figure 2.6: Join competition sequence diagram

ED Creates Battle

When a ED wants to create a battle, he/she needs to be in a competition page that he/she manages. The interaction is divided in two parts: the first one is where the ED inserts the general information about the battle, the name is then validated by the *Battle Manager* component and if it is valid the system will show the second part of the form where the ED can insert the information about the automatic evaluation and static analysis. The *Battle Manager* component will check if the information are valid and if so the battle will

be created and inserted into the DBMS using the *Data Manager* component.

If the battle is successfully created, the system will send a notification, through the *Notification Manager* to all the ST enrolled in the competition where the battle has been created.

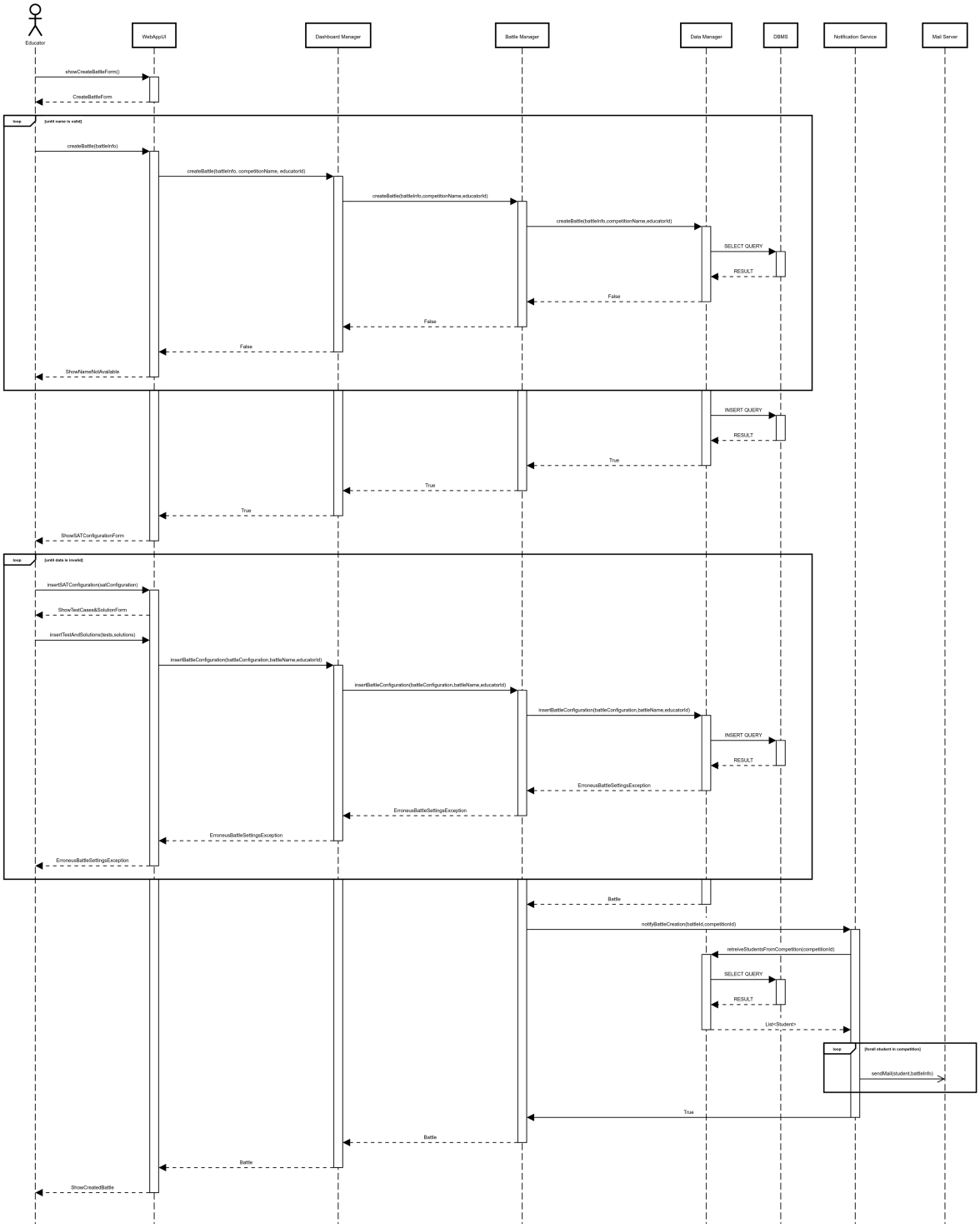

Figure 2.7: Create battle sequence diagram

ST Joins Battle

To be able to join a battle, a *ST* must be enrolled in a competition and in such competition there has to be at least 1 battle available (subscription deadline not expired). Assuming all those things, when a ST wants to join a battle, he/she has two options: create a team (with the possibility, if enabled by the EDs of the battle, to be a singleton) or to join an already existing team. In the first case the ST has to provide the system all the information needed for the team registration, and if the team has been created correctly he/she can invite other STs to join his/her team (if the ST, owner of the team, wants to participate alone he/she must skip the invite part). In the latter case instead, the ST looks at the list of existing and available teams and if he/she wants to join one of those team they can simply click on the join button.

Since the sequence diagram below was pretty long we decided to split it in two so that it was easier to read it; this means that the two following diagrams are to be understood as consecutive (the [*alt 'ST wants to join an existing public T'*] fragment, in the second part of the diagram, is the [*else*] of the [*alt 'ST selects to create a new T'*] fragment in the first part of the diagram)

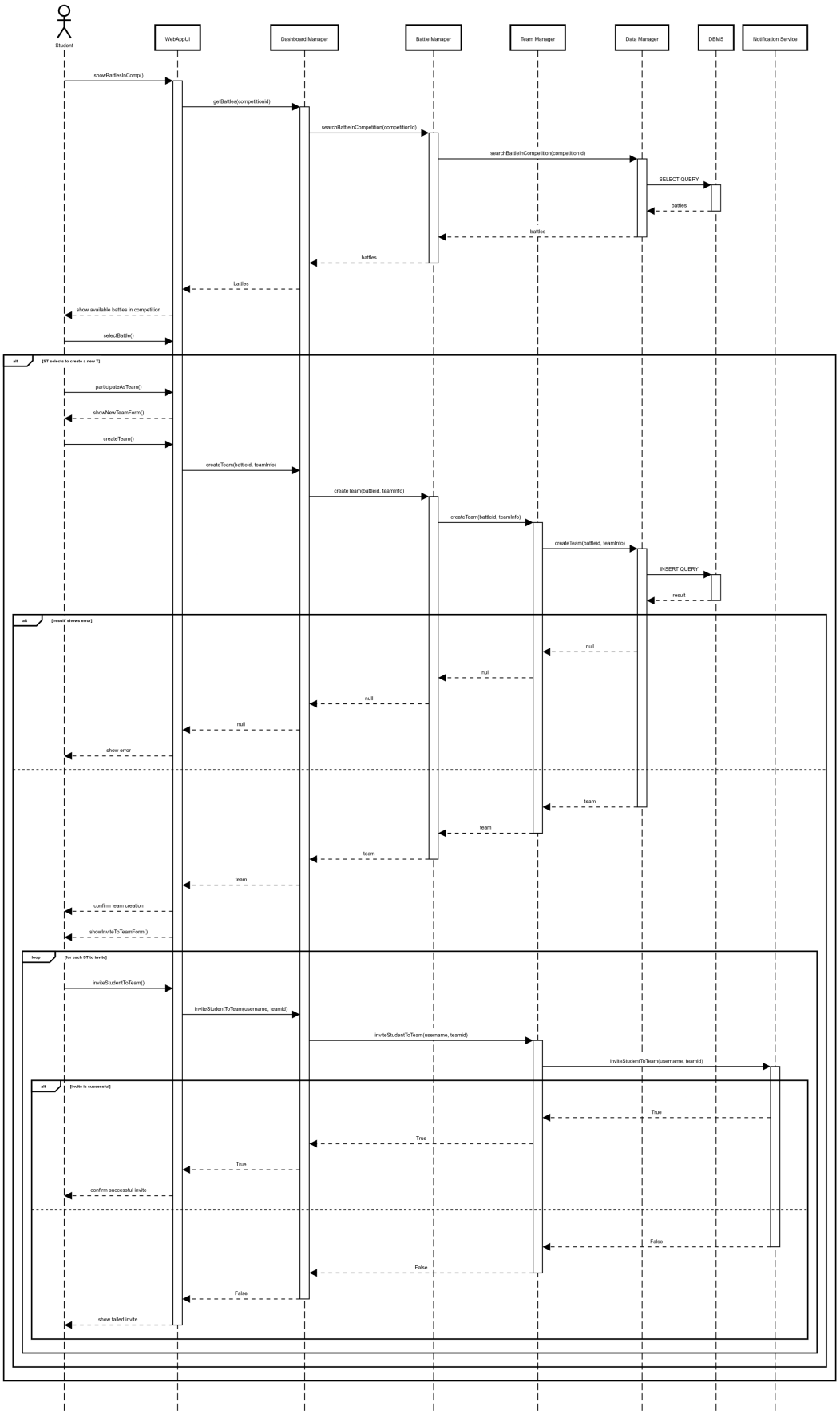

Figure 2.8: ST joins battle sequence diagram - Part 1

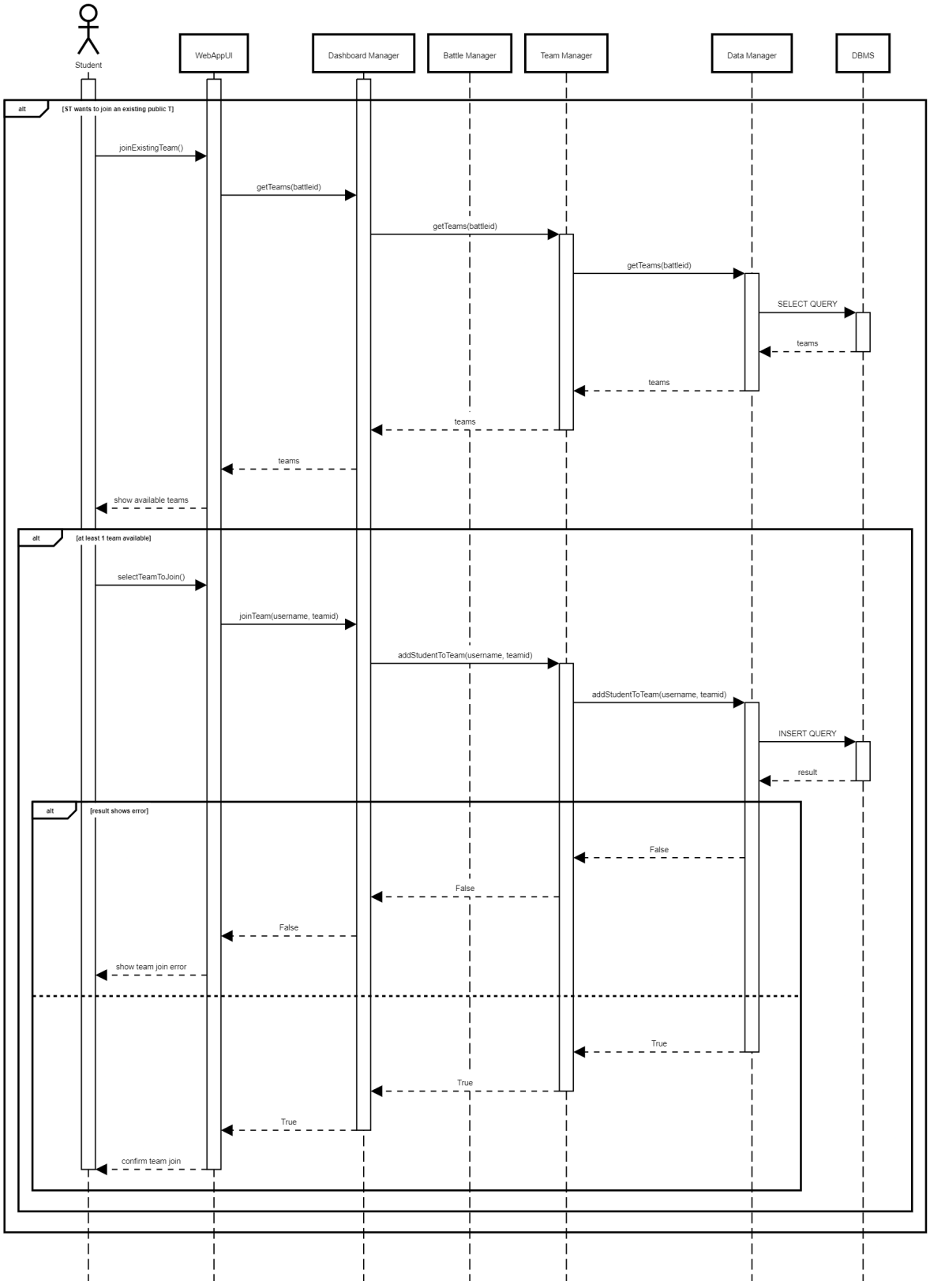

Figure 2.9: ST joins battle sequence diagram - Part 2

User Visualizes battle Rankings

Assuming that a User (ST or ED) is enrolled firstly in a competition and secondly subscribed to a battle that is still ongoing, such User can see the partial leaderboard of the battle. The sequence diagram below describe exactly this process, starting from the page of the competition.

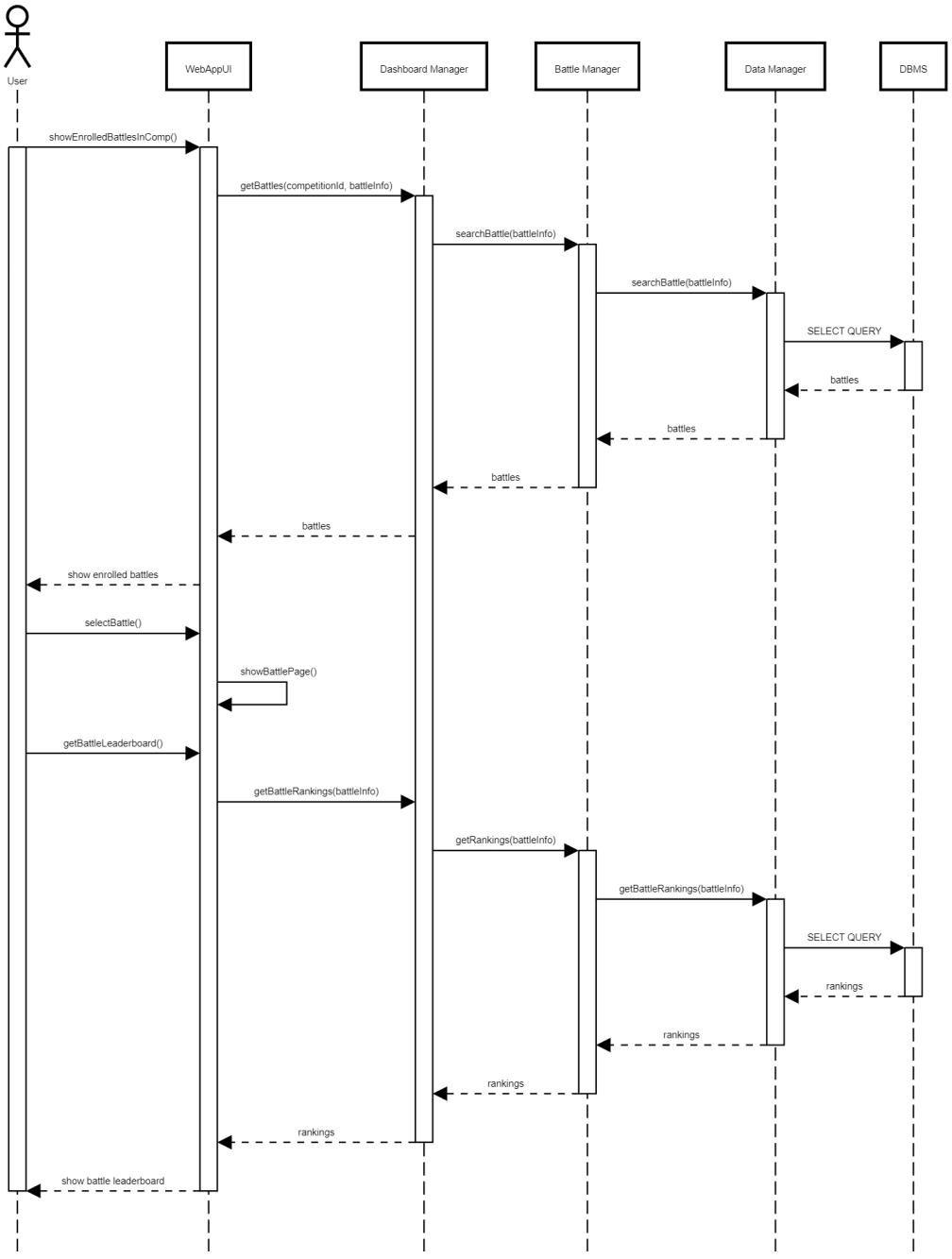

Figure 2.10: Create battle sequence diagram

ED Evaluates Code

An ED can select a T from the list of teams participating to a battle and evaluate its last commit. In particular the ED after selecting the team, he/she can visualize some brief information about the latest commit of the T, from the visualization is possible to navigate to the Github page of the commit. After reviewing the code on Github the ED can return to the CKB page and insert the evaluation of the commit. The process is handled by the *Team Manager* component, which interacts with the *Data Manager* component to insert the evaluation into the DBMS.

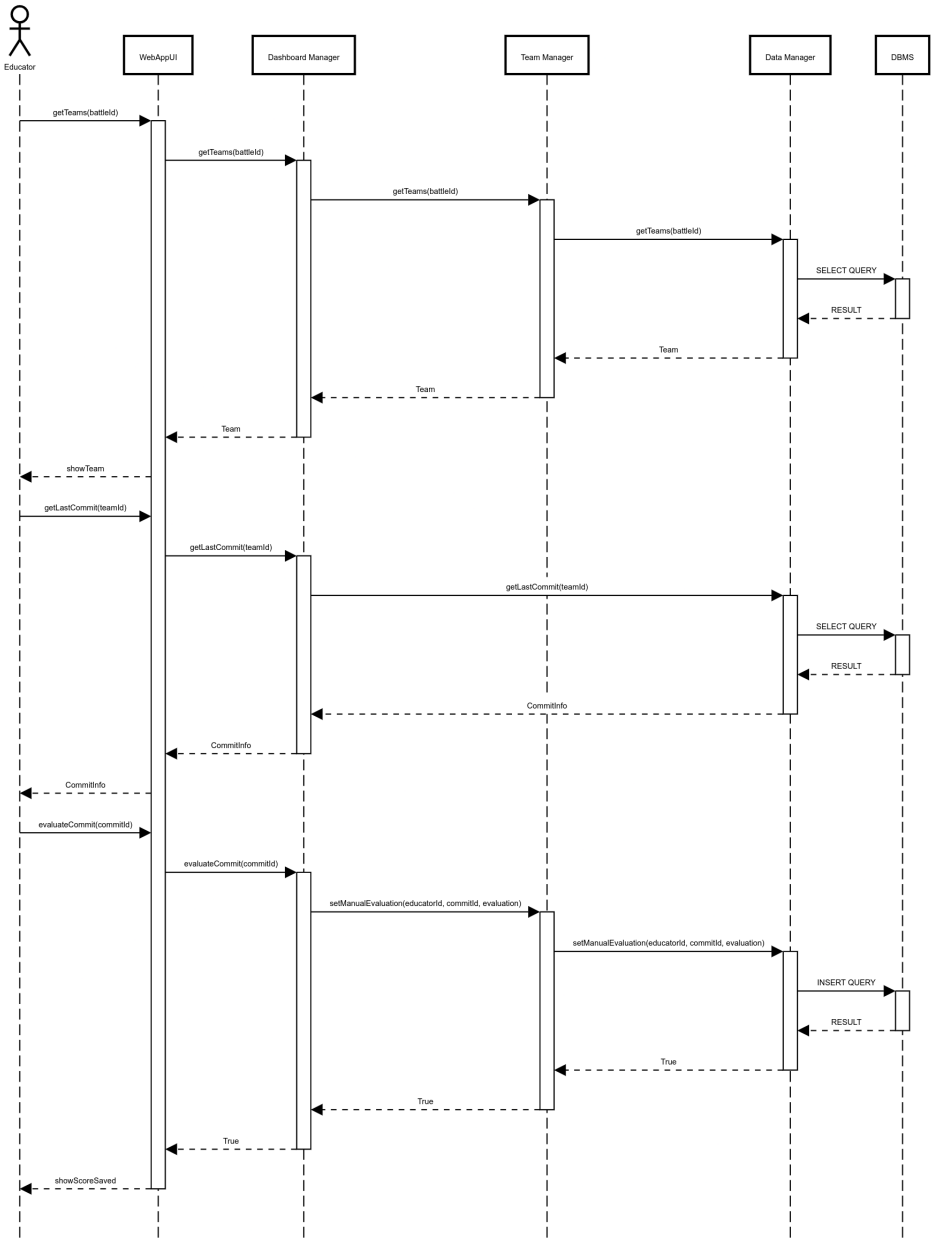

Figure 2.11: ED evaluates code sequence diagram

ST Accepts invite

When a ST clicks on the link in the email received from the system, he/she will be added to the T. The *Team Manager* component will handle the request and will insert the ST into the DBMS using the *Data Manager* component. In case the invitation link is expired, the system will show an error message to the user.

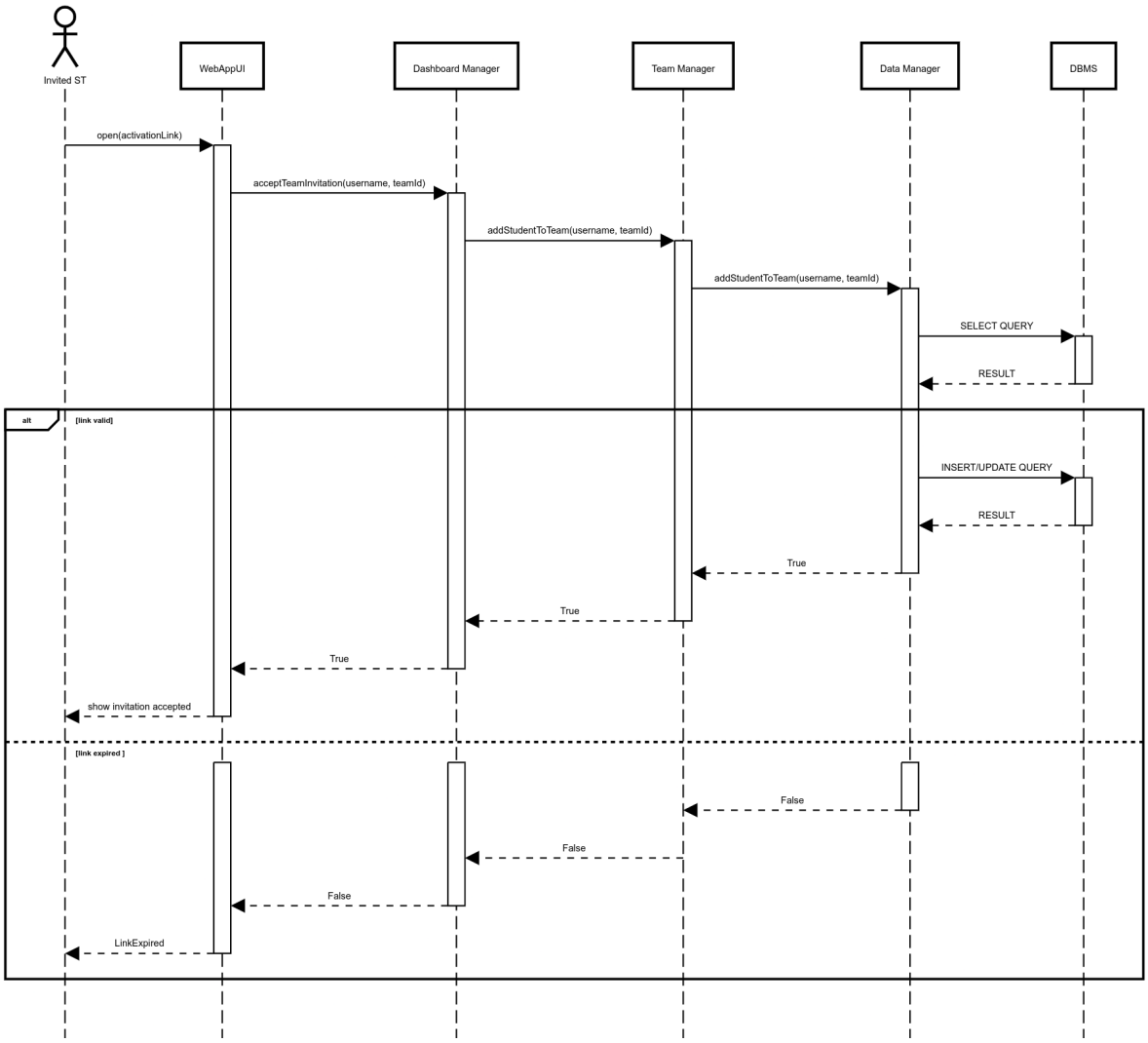

Figure 2.12: ST accepts invite sequence diagram

ST Visualizes other ST Profile

STs can visualize the profile of other STs. The ST can search for a specific ST using its username, the system will show the list of the corresponding STs and the ST can select one of them to visualize its profile. In the profile page the ST can see some information about the ST and the list of the badges that he/she has earned. The *Dashboard Manager*

component will handle the request and will retrieve the information from the DBMS using the *Data Manager* component.

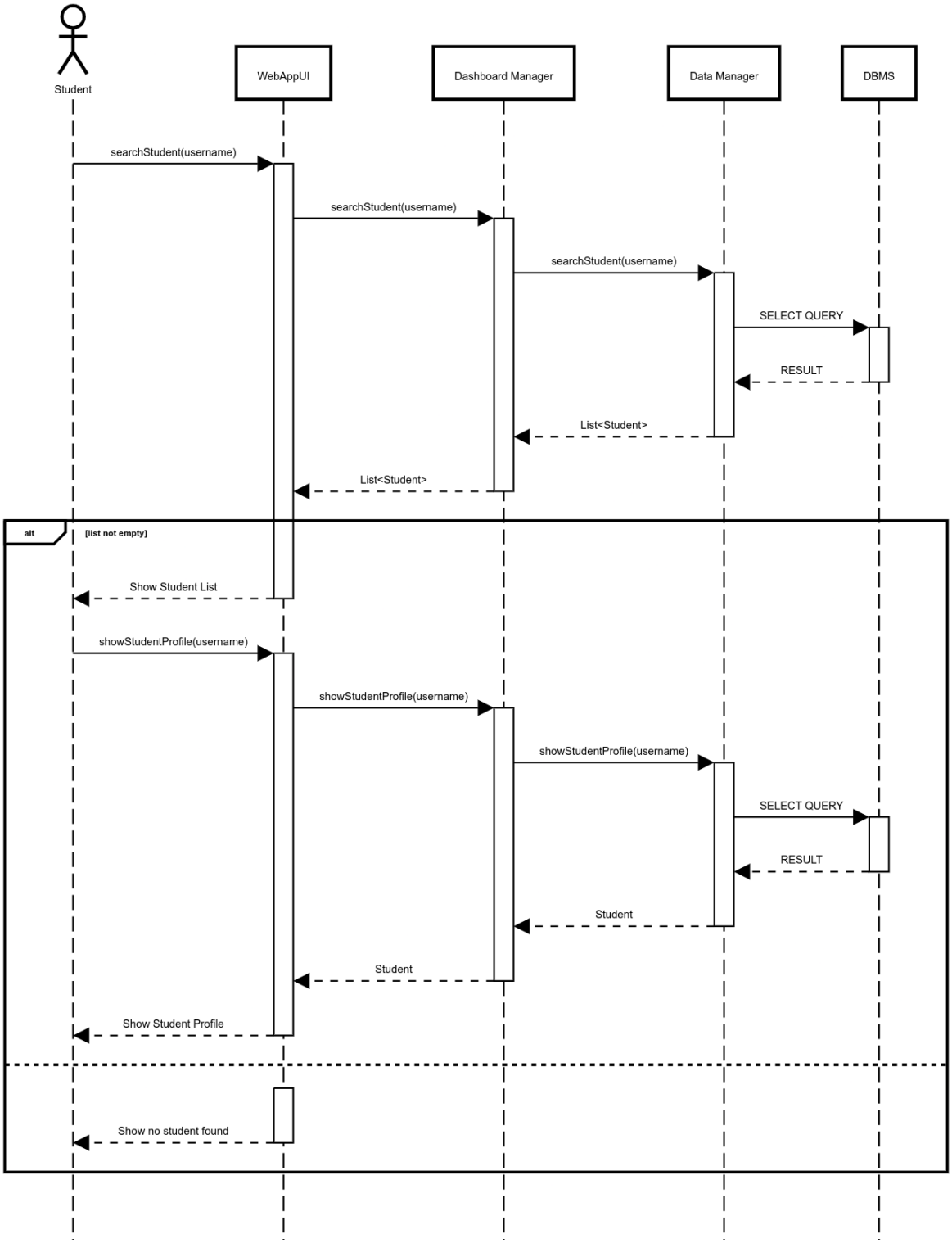

Figure 2.13: ST visualizes other ST profile sequence diagram

2.5. Component interfaces

2.5.1. Authentication Service

AuthInterface

- `User login(username:String,password:String)`
- `bool register(info:UserInfo)`

2.5.2. Data Manager

AuthDAOInterface

- `bool addUser(info: UserInfo)`
- `bool checkCredential(username:String,password:String)`
- `bool validateNewUser(info:UserInfo)`
- `User getUser(username:String)`

CompetitionDAOInterface

- `bool createCompetition(info:CompetitionInfo)`
- `bool addStudentToCompetition(competitionName:String, studentUsername:String)`
- `bool checkEdCanbeInvited(competitionName:String,educatorId:String)`
- `Competition getCompetition(name:String)`
- `List<Team> getCompRankings(competitionid: String)`

BattleDAOInterface

- `bool createBattle(competitionName:String, battleInfo:BattleInfo, educatorId:String)`
- `Set<Battle> searchBattle(battleInfo:BattleInfo)`
- `List<Battle> searchBattleInCompetition(competitionId:String)`
- `Battle showBattle(battleName:String)`
- `List<Team> getBattleRankings(battleid: String)`

- bool insertBattleConfiguration(battleConfiguration:BattleConfiguration, battleName:String,educatorId:String)

UserDAOInterface

- Team createTeam(battleId:String, teamInfo:TeamInfo)
- bool removeTeam(teamId:String)
- bool addStudentToTeam(username:String,teamId:String)
- bool setManualEvaluation(educatorId:String,commitId:String,evaluation: Evaluation)
- Team getTeams(battleid: String)

BadgeDAOInterface

- Set<User> retrieveUsersFromCompetition(competitionName:String)
- BadgeRule getBadgeRule(badge: Badge)
- bool removeBadge(badge:Badge)
- bool assignBadge(username:String, badge:Badge)
- bool createBadge(educatorId:String,competitionId:String, badgeInfo:BadgeInfo)

NotificationDAOInterface

- Set<Student> retrieveStudentsFromTeam(teamId:String)
- Set<User> retrieveUsersFromCompetition(competitionId:String)
- Set<Student> retrieveStudentsFromCompetition(competitionId:String)
- Set<Educator> retrieveEducatorsFromCompetition(competitionId:String)
- Set<User> retrieveUsersFromBattle(battleId:String)
- Set<Student> retrieveStudentsFromBattle(battleId:String)
- Set<Educator> retrieveEducatorsFromBattle(battleId:String)
- Educator retrieveEducatorInfo(educatorId:String)

PointsAPI

- bool addEvaluationToTeam(teamId:string, evaluation:Evaluation)

DashboardDAOInterface

- List<Student> searchStudent(studentName:String)
- Student showStudentProfile(username:String)
- CommitInfo getLastCommit(teamId: String)

2.5.3. Dashboard Manager

DashboardInterface

- DashboardInfo getDashboardInfo(User)
- createCompetition(competitionInfo:CompetitionInfo)
- bool createBattle(battleInfo:BattleInfo,competitionName:String, educatorId:String)
- bool joinCompetition(competitionName:String, studentUsername:String)
- CompetitionInfo showCompetition(name:String)
- List<Battle> getBattles(competitionid: String)
- Battle getBattle(competitionid: String, battleId: String)
- bool insertBattleConfiguration(battleConfiguration:BattleConfiguration, battleName:String,educatorId:String)
- bool inviteStudentToTeam(username: String, teamid: String)
- List<Team> getTeams(battleid: String)
- bool joinTeam(username: String,teamid: String)
- List<Team> getCompRankings(competitionid: String)
- List<Team> getBattleRankings(battleInfo: BattleInfo)
- List<Student> searchStudent(studentName:String)
- Student showStudentProfile(username:String)

- CompetitionSetting showCompetitionSettings(educatorId:String, competitionId:String)
- BadgeCreation showBadgeCreation(educatorId:String, competitionId:String)
- InviteEdPanel showInviteEd(educatorId:String,competitionId:String)
- bool inviteED(educatorId:String,competitionId:String,invitedEducatorId:String)
- bool createBadge(educatorId:String,competitionName:String, badgeInfo: BadgeInfo)
- CommitInfo getLastCommit(teamId: String)
- bool acceptInvitation(educatorId:String,competitionId:String)
- bool acceptTeamInvitation(username:String,teamId:String)
- Team createTeam(battleid: String, username: String)
- bool evaluateCommit(commitId: String)

2.5.4. Competition Manager

CompetitionInterface

- Competition createCompetition(info:CompetitionInfo)
- Set<Competition> searchCompetition(info:CompetitionInfo)
- bool deleteCompetition(name:String)
- Competition getCompetition(name:String)
- bool addManager(competitionName:String,username:String)
- bool removeManager(competitionName:String,username:String)
- bool endCompetition(competitionName:String)
- bool joinCompetition(competitionName:String, studentUsername:String)
- bool createBadge(educatorId:String,competitionName:String, badgeInfo: BadgeInfo)
- bool removeBadge(badgeId:String)
- List<Team> getRankings(competitionid: String)

- `bool acceptInvitation(educatorId:String,competitionId:String)`
- `bool inviteED(educatorId:String,competitionId:String,invitedEducatorId:String)`

2.5.5. Badge Manager

BadgeInterface

- `bool createBadge(competitionName:String, badgeInfo: BadgeInfo)`
- `bool assignBadges(competitionName:String)`
- `bool removeBadge(badgeId:String)`

2.5.6. BattleManager

BattleInterface

- `bool createBattle(competitionName:String, battleInfo:BattleInfo, educatorId:String)`
- `Team createTeam(battleId:String, teamInfo:TeamInfo)`
- `bool removeTeam(teamId:String)`
- `List<Battle> searchBattle(battleInfo:BattleInfo)`
- `List<Battle> searchBattleInCompetition(competitionId:String)`
- `Battle showBattle(battleName:String)`
- `bool deleteBattle(battleName:String)`
- `bool insertBattleConfiguration(battleConfiguration:BattleConfiguration, battleName:String,educatorId:String)`
- `List<Team> getRankings(battleid: String)`

2.5.7. Team Manager

TeamHandlerInterface

- `Team createTeam(battleId:String, teamInfo:TeamInfo)`
- `bool removeTeam(teamId:String)`
- `Team getTeams(battleid: String)`

TeamInterface

- `bool addStudentToTeam(username:String,teamId:String)`
- `bool inviteStudentToTeam(username:String,teamId:String)`
- `bool setManualEvaluation(educatorId:String,commitId:String,evaluation:Evaluation)`

2.5.8. Notification Service

InviteInterface

- `bool inviteStudentToTeam(username:String,teamId:String)`

NotifyBattleInterface

- `bool notifyBattleCreation(battleId:String, competitionId:String)`
- `bool notifyStartedBattle(battleId:String,username:String)`
- `bool notifyEndBattle(battleId:String,username:String)`
- `bool notifyManageBattle(battleId:String,username:String)`

NotifyCompInterface

- `bool notifyNewCompetion(competitionId:String,username:String)`
- `bool notifyNewBattle(competitionId:String, battleId:String,username:String)`
- `bool notifyEdInvitation(competitionId:String,invitedEd:String)`
- `bool notifyManageCompetition(competitionId:String,username:String)`

NotifyAuthInterface

- `bool NotifyUserRegistration(User)`

2.5.9. Evaluator Controller

EvaluationAPI

- `bool pullCode(authorId:String, commitId:String)`

2.5.10. Code Evaluator

EvaluatorInterface

- `bool evaluateCode(authorId:String, commitId:String)`
- `void setConf(conf: EvalConfig)`

2.5.11. Static Analyzer

AnalyzerInterface

- `bool evaluateCode(authorId:String, commitId:String, params:String)`
- `void setConf(conf: StatConfig)`

2.5.12. Point Manager

EvaluationPointsInterface

- `bool assignPoint(authorId:String, evalResults: EvalResults)`

StaticPointsInterface

- `bool assignPoint(authorId:String, statResults: StatResults)`

ScoreInterface

- `void setEvalScoreFunction(conf: EvalScoreFunction)`
- `void setStatAnalysisScoreFunction(conf: StatScoreFunction)`

2.6. Selected architectural styles and patterns

2.6.1. Client-Server

Since the CKB is a platform offered through the web, the client-server architecture is the most suitable for this kind of application. In particular we are using a 3-tier architecture, where the client is the web browser, the server is the application server and the database is the data server. This architecture is the most suitable for our application because it allows us to separate the presentation layer from the business logic and the data layer. Moreover, this separation allows us to have a more maintainable and scalable application.

2.6.2. REST

The communication between the client and the server is based on the REST architectural style. The REST architectural style is stateless, this means that the server does not need to store any information about the client session, this allows us to have a more scalable application. Moreover, in combination with the HTTP protocol, it allows us to provide a uniform interface to the client, which is based on the HTTP methods (GET, POST, PUT, DELETE).

2.6.3. MVC

The MVC pattern is used to separate the presentation layer from the business logic. In particular, the *Model* contains the definition of all the elements of our application, the *View* is represented by the *WebAppUI* component and the *Controller* is represented by all the other *Managers* component. This pattern allows us to have a more maintainable application, because the changes in one layer do not affect the other layers.

2.7. Other design decisions

2.7.1. SandBox

To protect the application from malicious code, we decided to use a sandbox. In particular, the code evaluator and the static analyzer will be executed in a sandbox. This allows us to have a more secure application, because the code will be executed in a controlled environment.

2.7.2. Database

We decided to use a relational database to store the data of the application given its structure. In particular, we decided to use PostgreSQL, because it is an open source database and it is one of the most used relational database. Moreover, it is ACID compliant, this means that it guarantees the atomicity, consistency, isolation and durability of the transactions.

3 | User interface design

In this section we will describe the user interface design of the system. We will provide a mockup of the main pages of the system and a description of the main functionalities.

The user interface of the system is designed to be simple and intuitive. As the system is intended to be used with a desktop browser, the interfaces presented here are based on a desktop browser, but the interface is thought to be responsive and consequently usable also on mobile devices.

Common Interfaces

Some pages of the platform are common, or very similar, for both ST and ED, so in this section we will show only once the mockup of the pages and we will describe the functionalities of the pages for both ST and ED.

Login Page

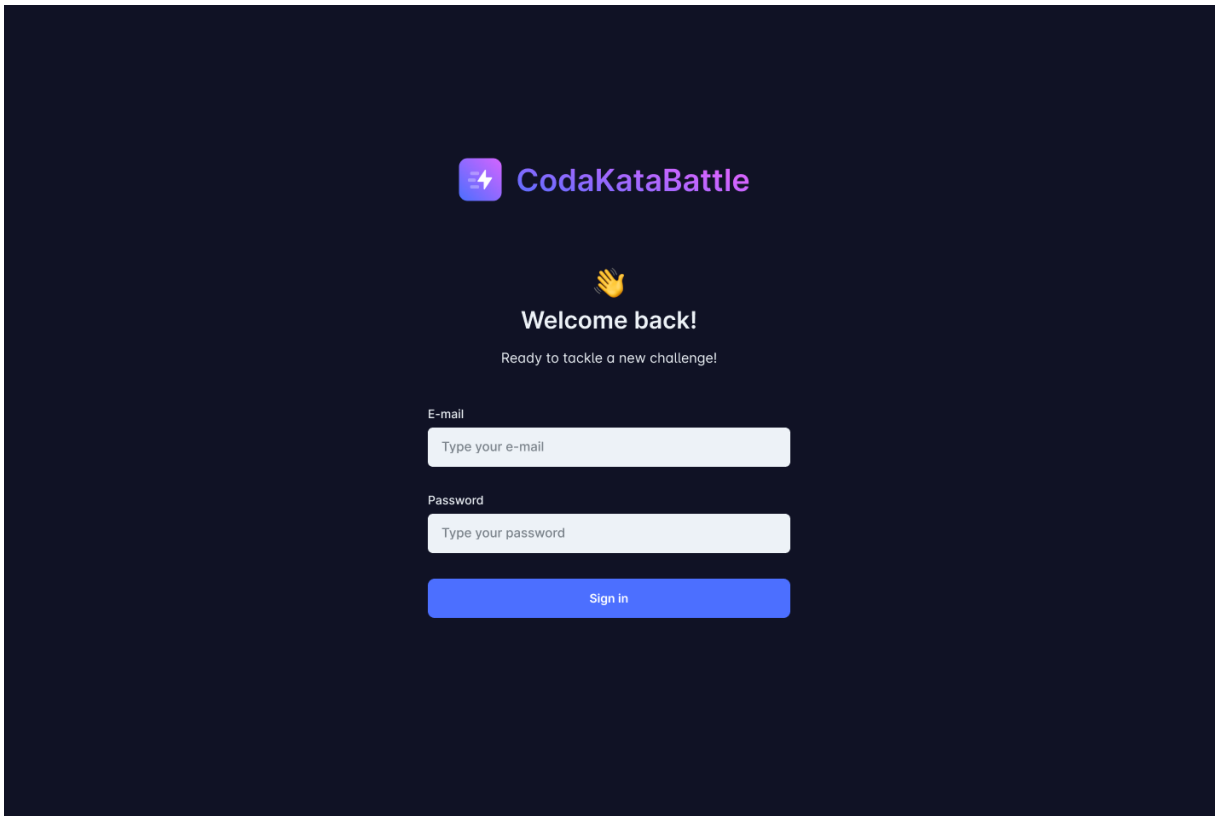

Figure 3.1: CKB login page

Registration Page

Here for simplicity we show only the registration page for a ST, but the registration page for a ED is equal to this one with the only difference that the ED is required to insert also information about the institution he/she works for.

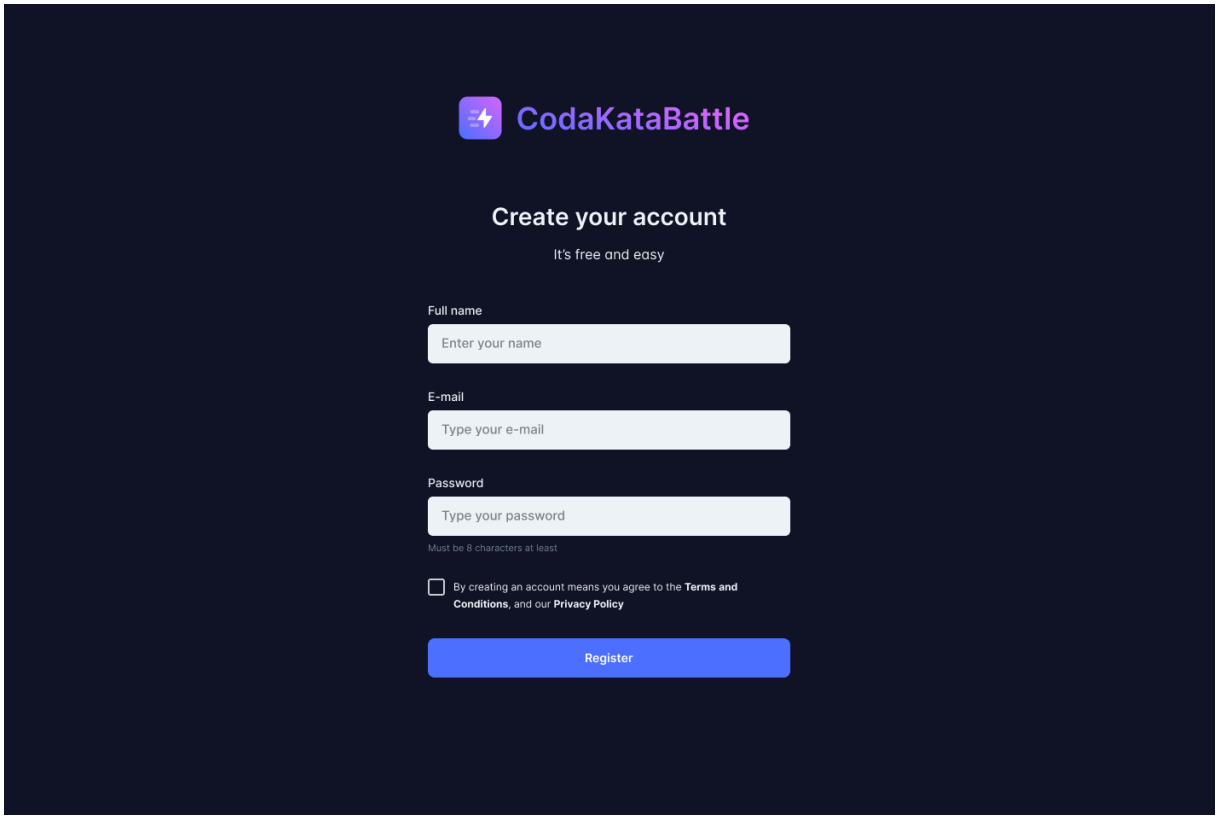

Figure 3.2: CKB registration page

Home Page

This is a mockup of the homepage of a ST. ED would see a very similar home page with statistics about the competition and battle created.

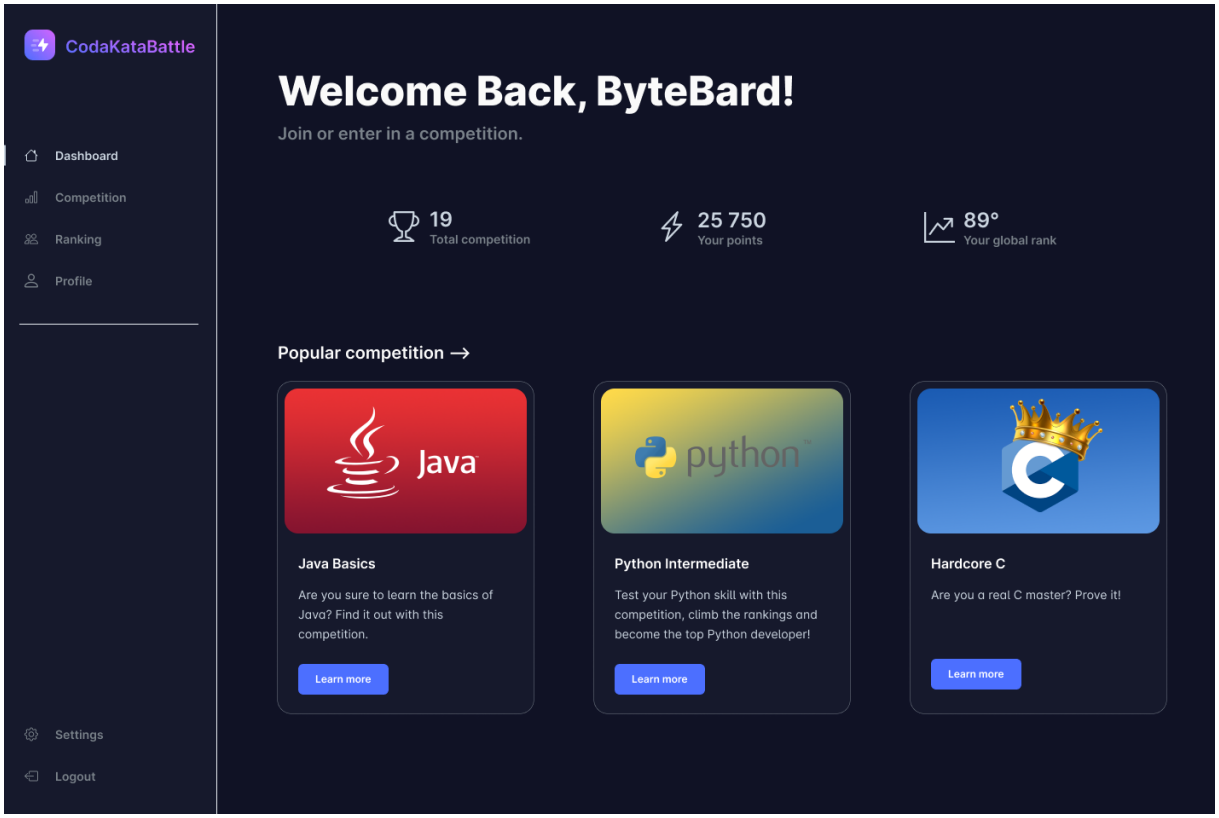

Figure 3.3: CKB student home page

Competition Page

In this page the ST can see the list of all the competitions he/her is currently enrolled in. The ED can see the list of all the competitions he/her has created.

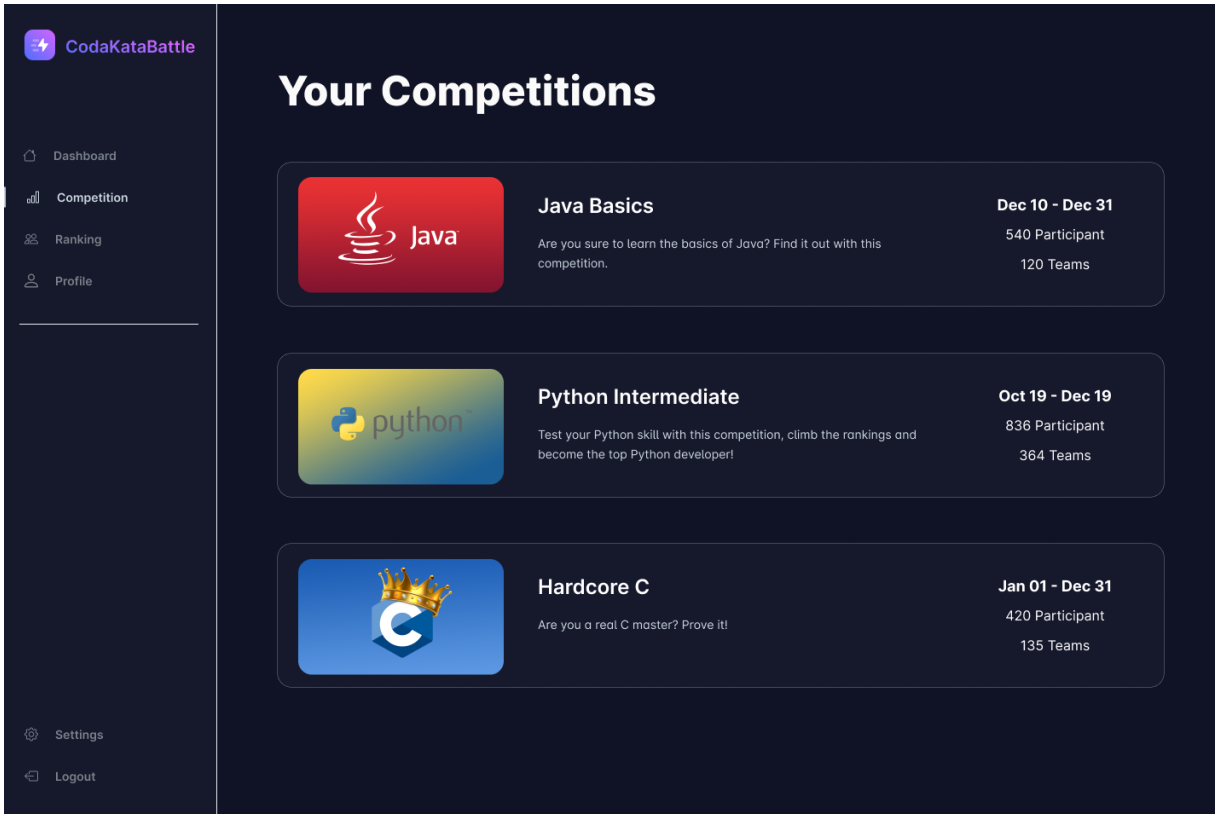

Figure 3.4: CKB competition page

Ranking Page

This is a the mockup of all the rankings present in the system. In particular, this is equal for the global ranking, competition raning and battle ranking pages. Both the ST and the ED are presented with the same interface and functionalities when consulting the ranking pages.

Figure 3.5: CKB global ranking page

ST Profile Page

Also this page is equal for both ST and ED. In particular in this page it is possible to see all the badges earned by the ST, other than the information about the competition he/her has partecipated in and their statistics.

Figure 3.6: CKB ST profile page

ST Interfaces

Now are presented the interfaces that are specific for the ST, in particular are shown the pages relative to the join of a battle by the ST.

Join Battle Pages

To create a more pleasant experience for the ST, the join battle pages are divided in different steps. In particular, the first step is to choose to join with a T or as a single ST. In the second step, if the ST has decided to join as a T he/her has to choose if he/her wants to create a new T or join an existing one. In case the ST has decided to create a new T is presented with the relative page, otherwise he/her is presented with the page to join an existing public T.

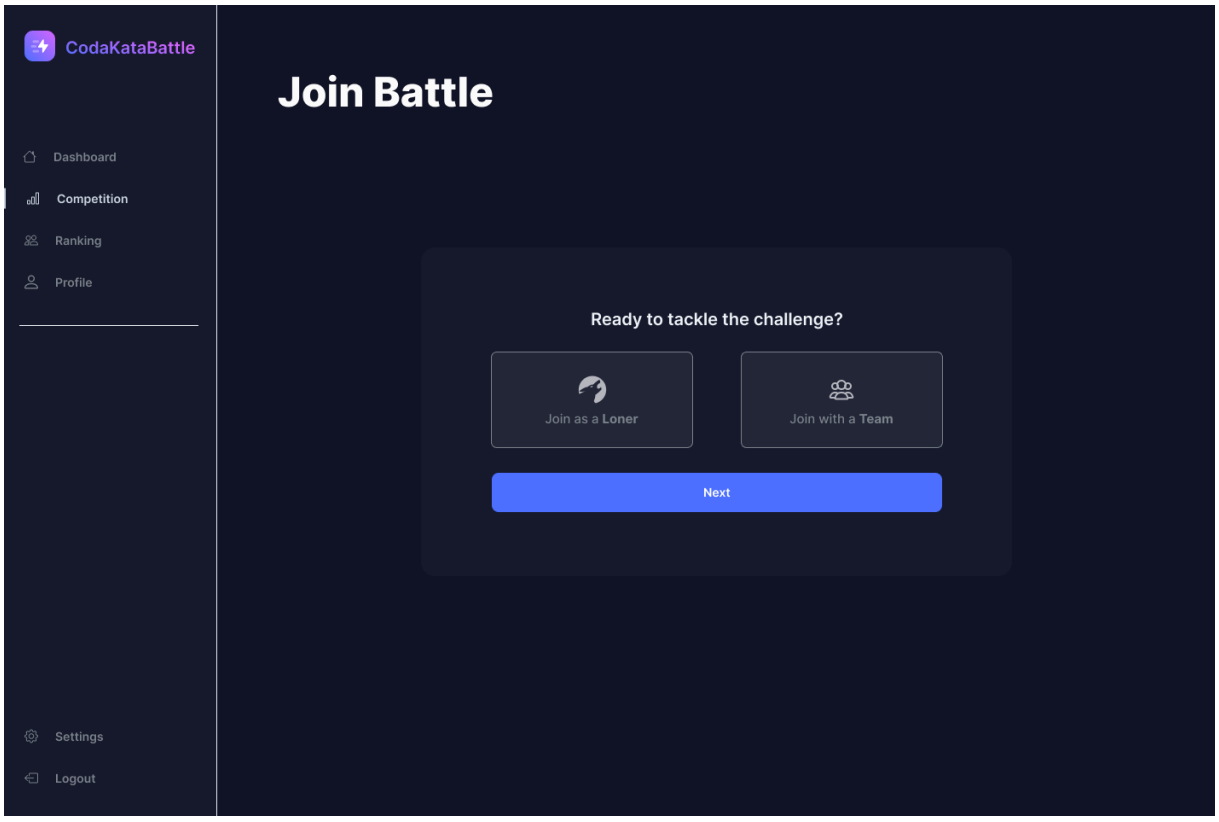

Figure 3.7: Join battle page - step 1

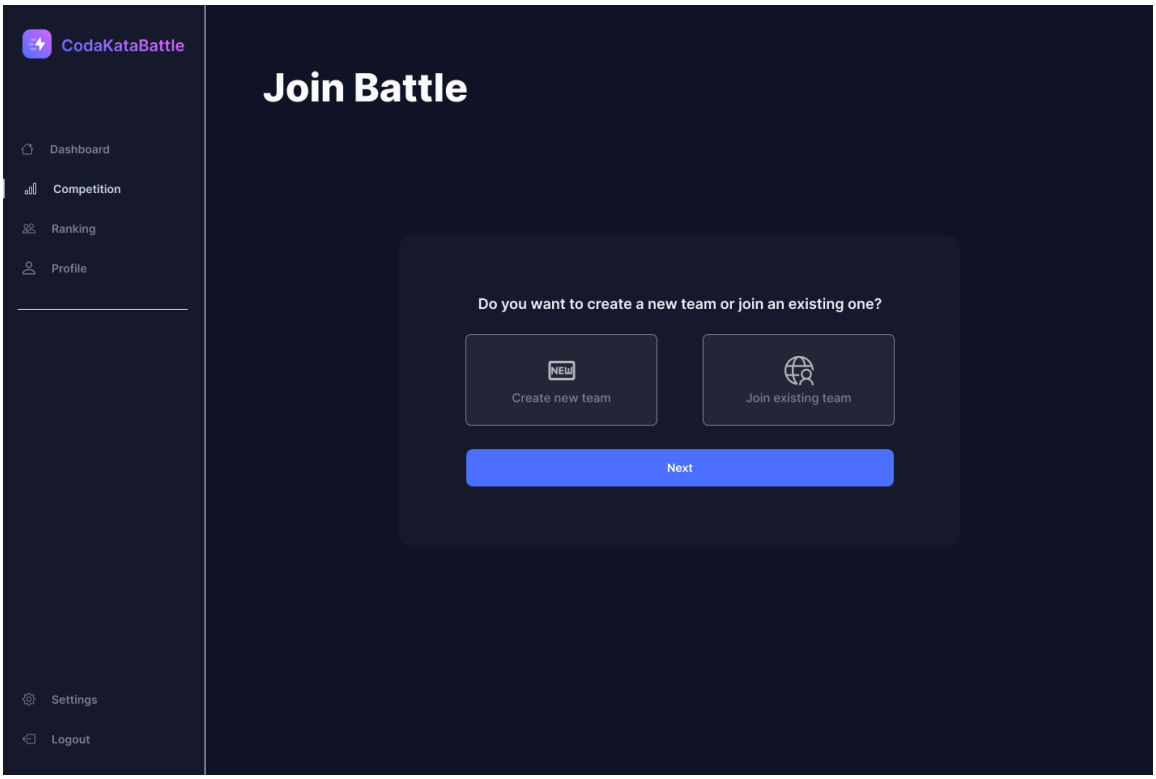

Figure 3.8: Join battle page - step 2

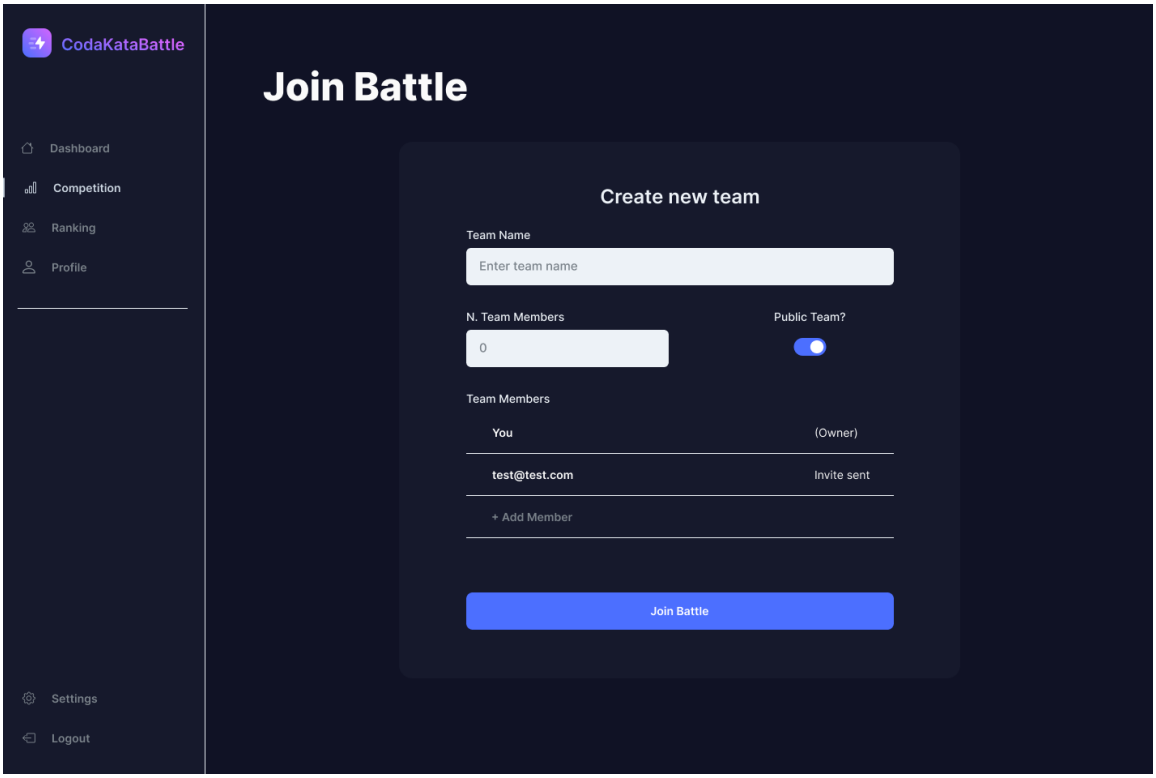

Figure 3.9: Create a new team

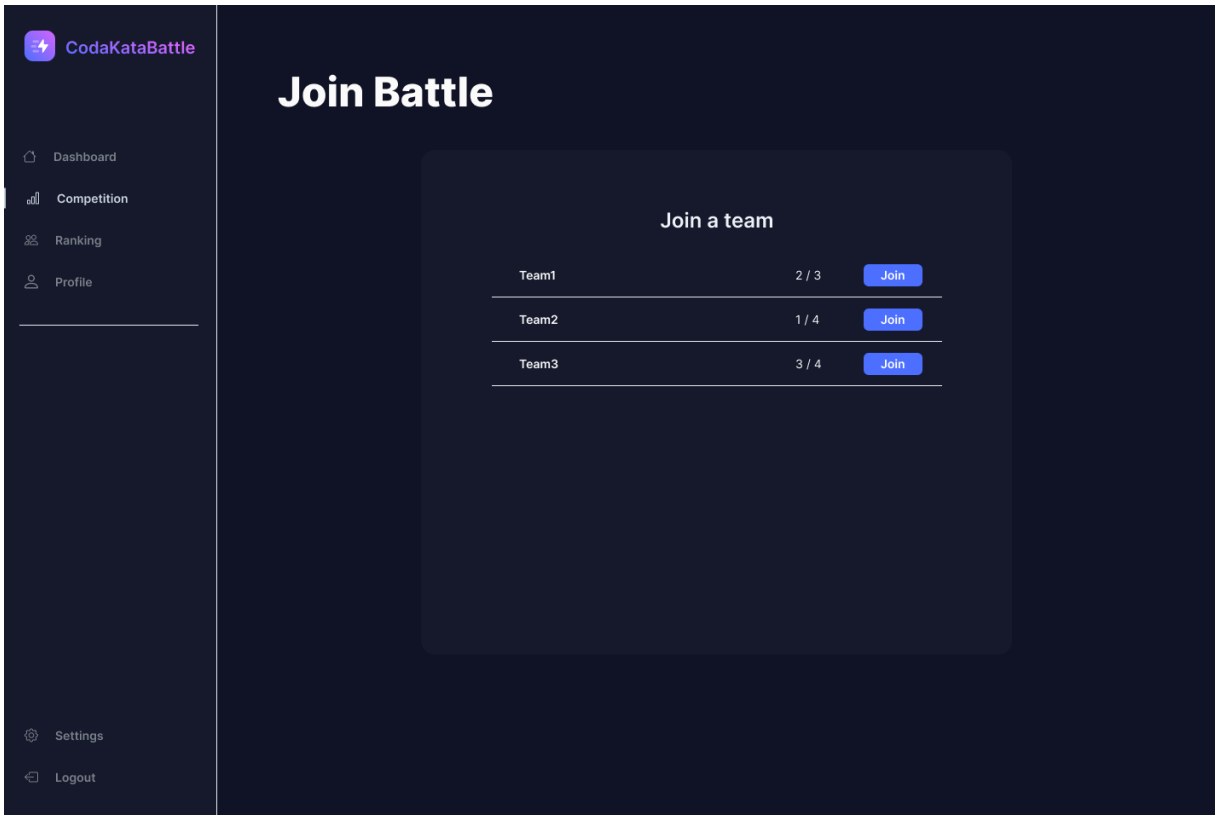

Figure 3.10: Join a public team

ED Interfaces

In this section are presented the interfaces that are specific for the ED, in particular are shown the pages relative to the creation of a competition, the creation of a battle and the creation of a badge.

Create Competition Page

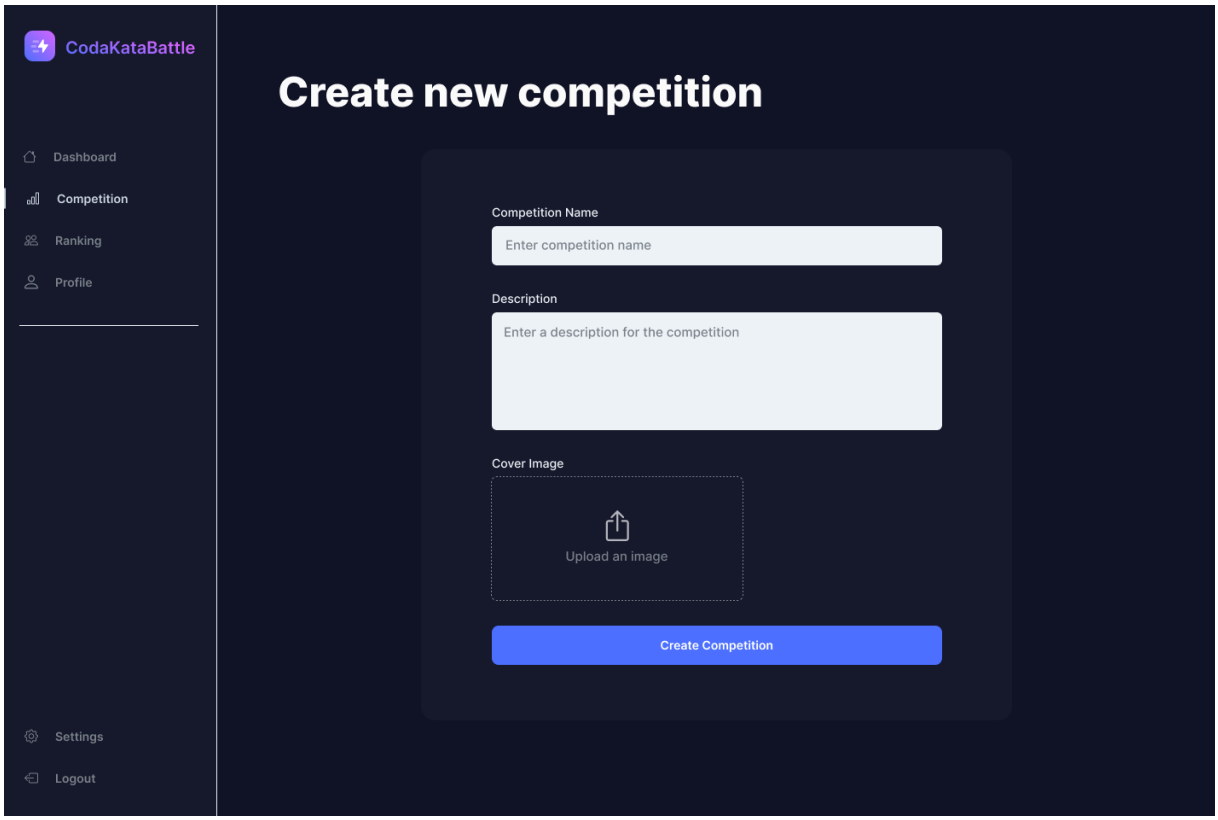

Figure 3.11: CKB create competition page

Create Battle Pages

As for the join of a battle for a ST also the creation of a battle is divided in two different steps. In the first step the ED has to insert all the general information about the battle, while in the second step he/her has to insert the code and settings for the automatic evaluation of the code.

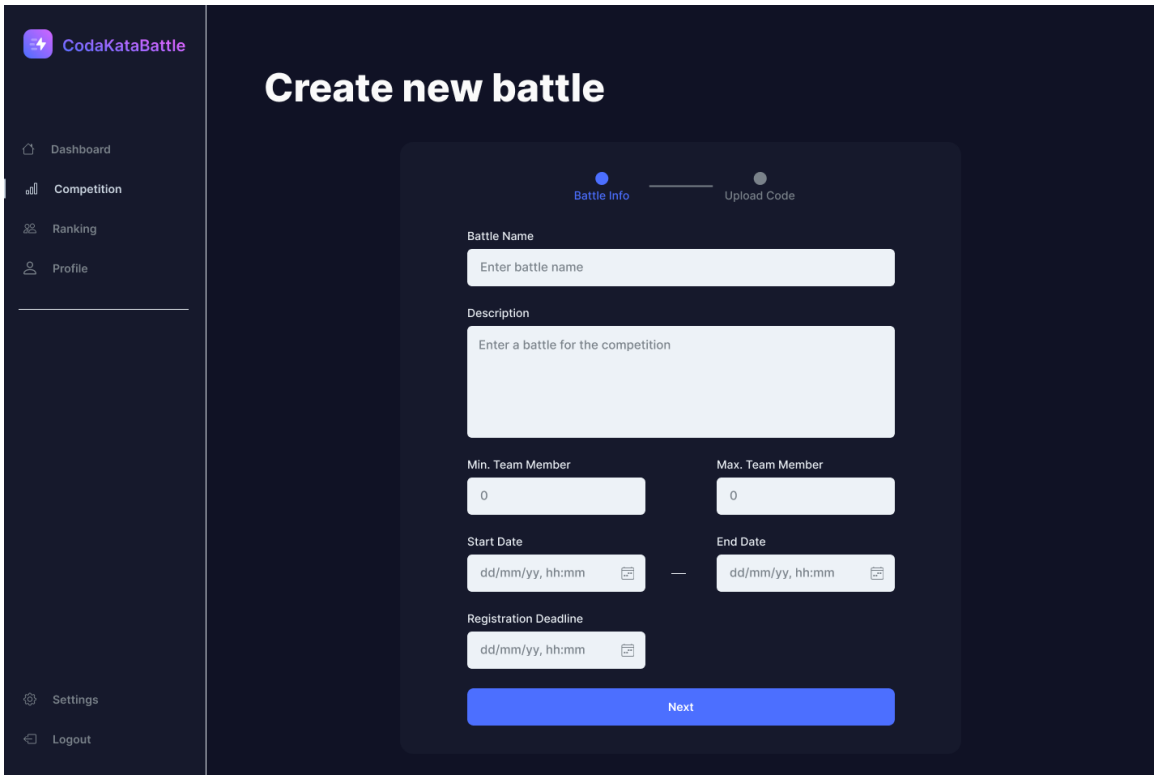

Figure 3.12: CKB create battle page - step 1

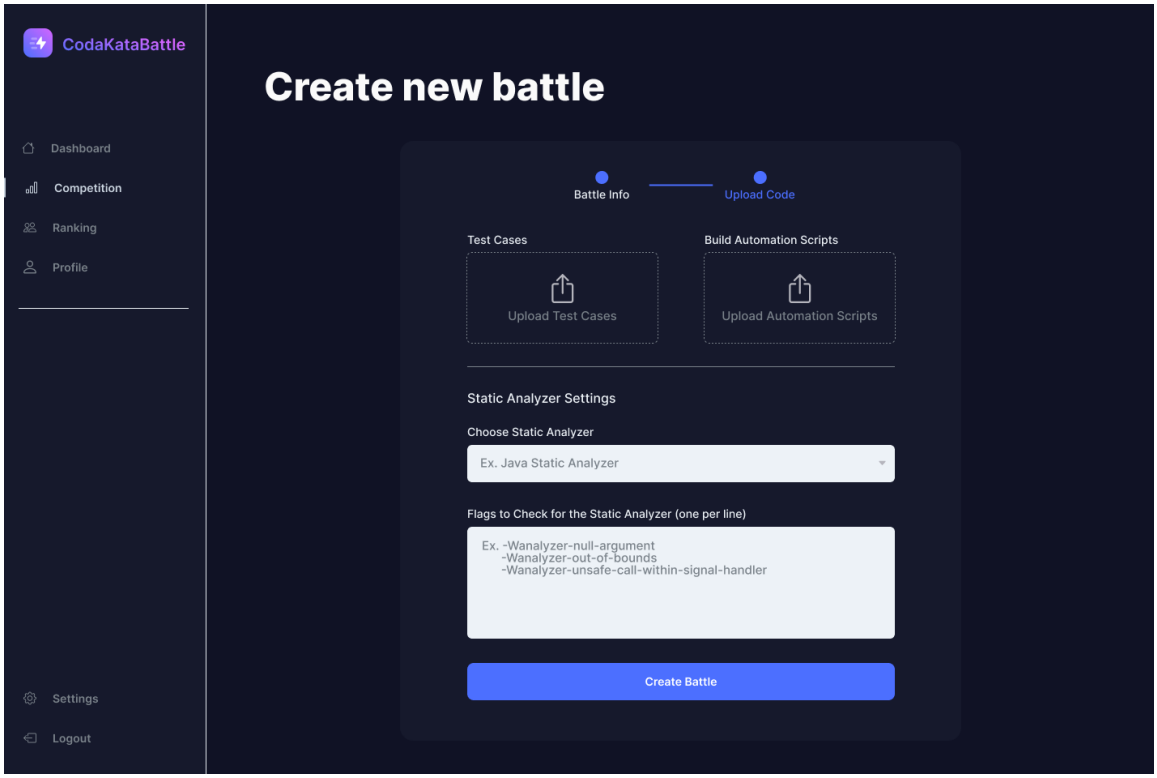

Figure 3.13: CKB create battle page - step 2

Create Badge Pages

Similarly to the last function, the creation of a badge is divided in two steps. In the first step the ED can insert all the information of the badge, that include the name of the badge, a description and a picture. In the second step the ED can choose the criteria that the ST has to satisfy to earn the badge, this is done by a set of pre-defined criteria that the ED can choose from.

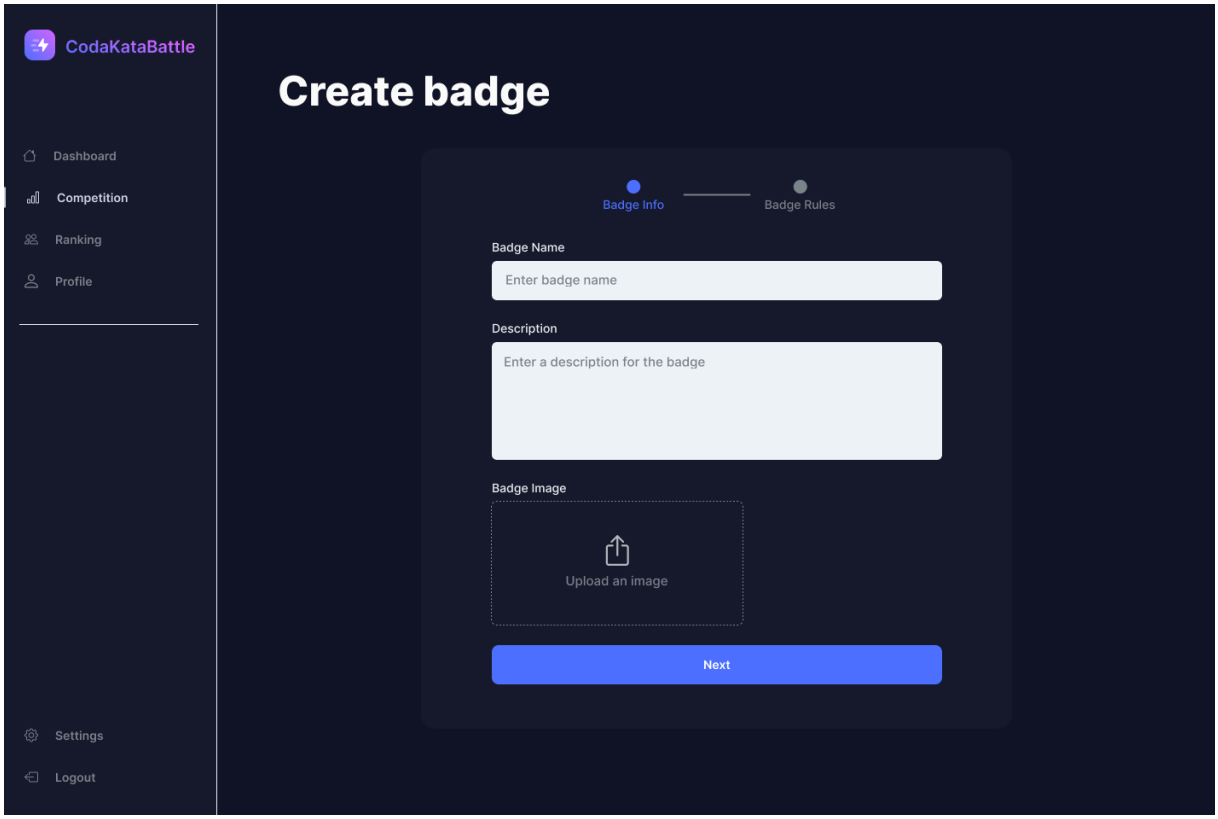

Figure 3.14: CKB create badge page - step 1

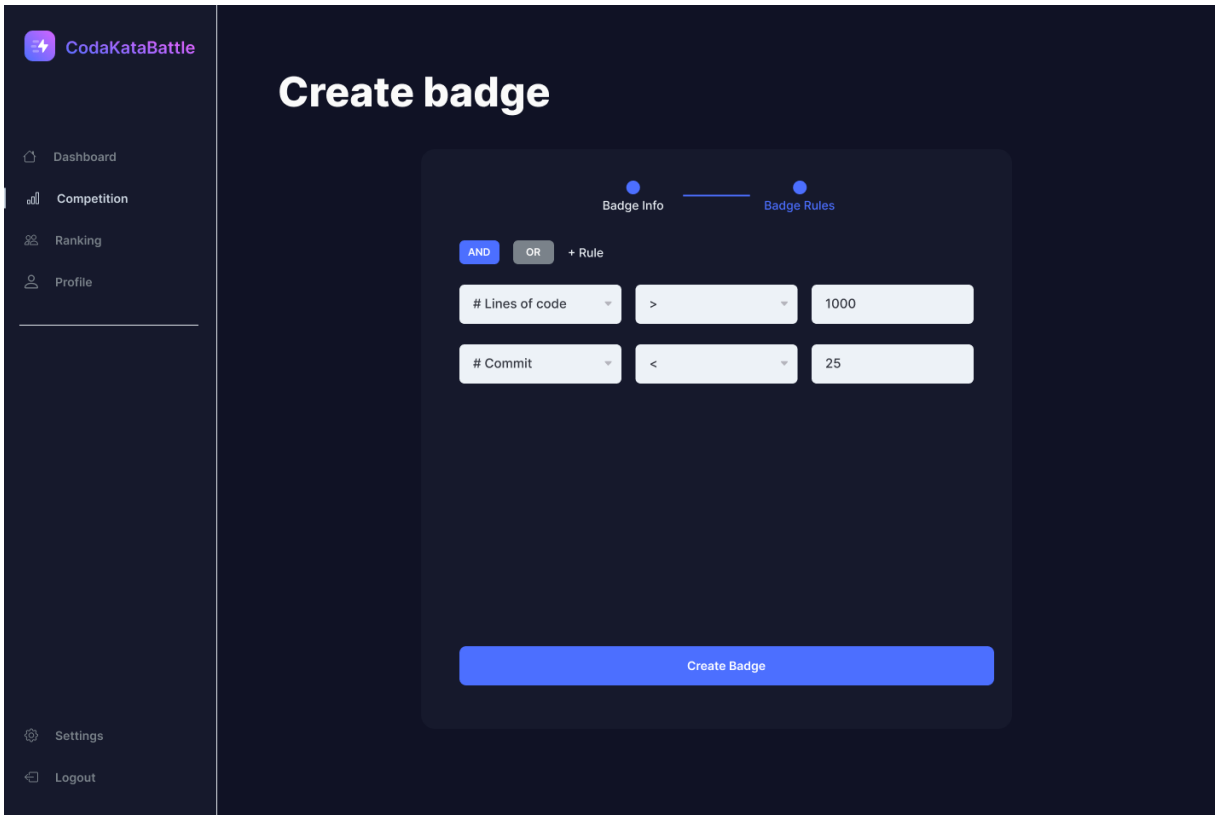

Figure 3.15: CKB create badge page - step 2

Manual Evaluation Page

This is the page where the ED can manually evaluate the code of a T. In particular, the ED can see some information about the latest submission of a T and can visit GitHub to see the code of the T. Then the ED can evaluate the latest submission of the T, assigning a score.

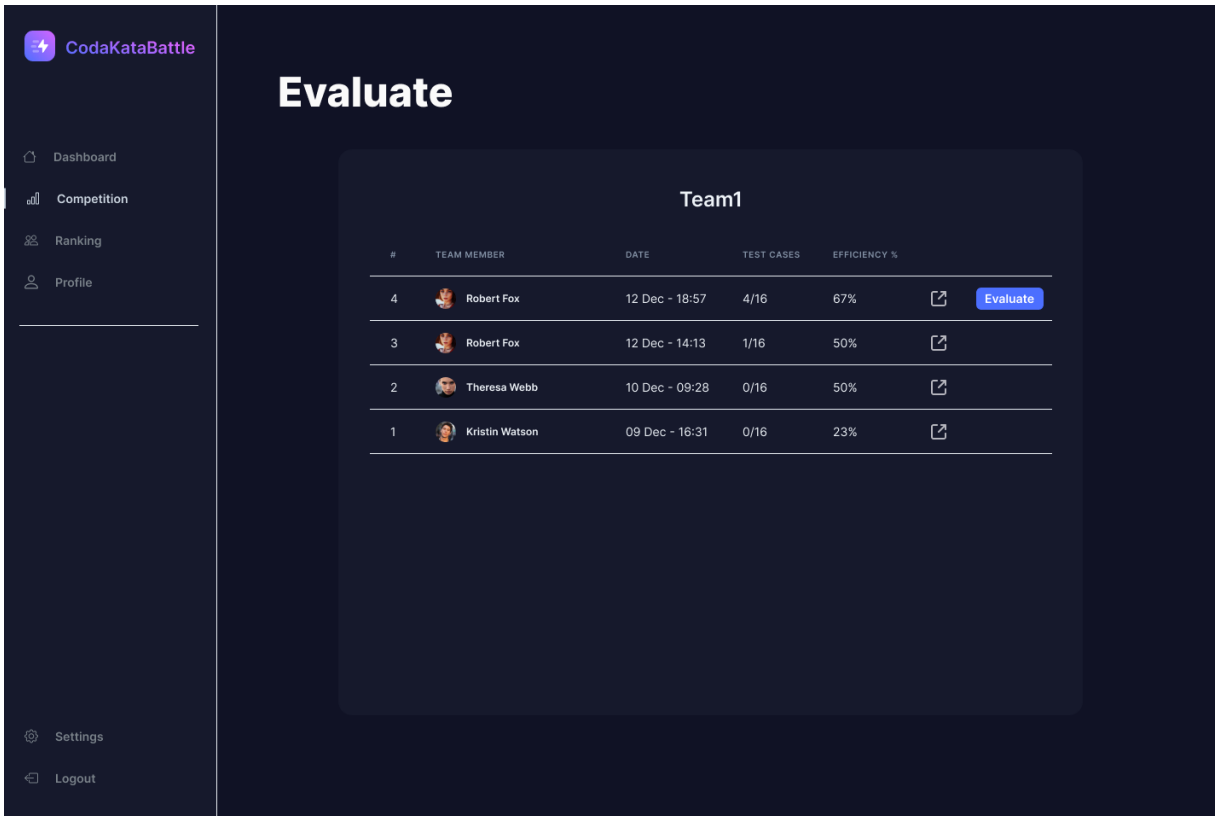

Figure 3.16: CKB manual evaluation page

4 | Requirements traceability

4.1. Requirement Traceability

	Description	Components
R1	CKB shall allow an unregistered user to create an account	Authentication Service, Data Manager
R2	CKB shall allow users to log in	Authentication Service, Data Manager
R3	CKB shall allow ED to create competition	Dashboard Manager, Competition Manager, Data Manager
R4	CKB shall allow ED to create battle within a competition that he/she manages	Dashboard Manager, Battle Manager, Data Manager
R5	CKB shall allow ED to invite other EDs to manage battles in a competition	Dashboard Manager, Competition Manager, Battle Manager, Data Manager, Notification Service
R6	CKB shall allow ED to upload the code kata	Dashboard Manager, Battle Manager, Data Manager
R7	CKB shall allow ED to set a registration deadline to the battle	Dashboard Manager, Battle Manager, Data Manager
R8	CKB shall allow ED to set a minimum number of STs per group in a battle	Dashboard Manager, Battle Manager, Data Manager
R9	CKB shall allow ED to set the maximum number of STs per group in a battle	Dashboard Manager, Battle Manager, Data Manager
R10	CKB shall allow ED to set a final submission deadline	Dashboard Manager, Battle Manager, Data Manager
R11	CKB shall allow ED to set how to perform static analysis	Dashboard Manager, Battle Manager, Data Manager
R12	CKB shall allow ST to subscribe to a competition	Dashboard Manager, Competition Manager, Data Manager

R13	CKB shall send notifications about a new competition to ST	Competition Manager, Notification Service
R14	CKB shall send notification about battle created within a competition ST are subscribed in	Battle Manager, Notification Service
R15	CKB shall allow ST to join a battle on his own	Dashboard Manager, Battle Manager, Team Manager, Data Manager
R16	CKB shall allow ST to invite other ST in a T for a battle	Dashboard Manager, Team Manager, Data Manager, Notification Service
R17	CKB shall create a GitHub repository containing the description, software project and the build automation scripts	Dashboard Manager, Battle Manager, Data Manager
R18	CKB shall send the Github repository link to ST member of a T competing in the battle	Battle Manager, Team Manager, Data Manger, Notification Service
R19	CKB shall supply API to call with Github actions	Battle Manager
R20	CKB shall be able to pull sources from GitHub	Battle Manager
R21	CKB shall be able to send the ST source code to the correct SAT	Evaluator Controller, Static Analyzer
R22	CKB shall be able to receive the evaluation given by SAT on a source code	Point Manager, Static Analyzer,Data Manager
R23	CKB shall be able to run tests on code	Code Evaluator, Evaluator Controller
R24	CKB shall evaluate the code in terms of test cases passed	Code Evaluator, Evaluator Controller, Point Manager
R25	CKB shall evaluate the code in terms of timeliness	Code Evaluator, Evaluator Controller, Point Manager
R26	CKB shall allow ED to assign a score to codes	Dashboard Manager, Team Manager, Data Manager
R27	CKB shall update the score of a T (as soon as new push actions are performed)	Evaluator Controller, Code Evaluator, Static Analyzer, Point Manger, Data Manger
R28	CKB shall allow ED to go through sources produced by Ts	Dashboard Manager, Battle Manager

R29	CKB shall notify ST when final battle ranks are available	Battle Manager, Notification Service
R30	CKB shall update the personal competition score of a ST at the end of each battle	Battle Manager, Competition Manager, Data Manager
R31	CKB shall create a rank with students' performances in a competition	Competition Manager, Data Manager
R32	CKB shall allow ST to see all ST's rank in battle where is enrolled	Dashboard Manager, Battle Manager, Data Manager
R33	CKB shall allow ED to see all ST's ranks in the battle that he/she manages	Dashboard Manager, Battle Manager, Data Manager
R34	CKB shall allow EDs and STs to see all ST's rank in competitions	Dashboard Manager, Competition Manager, Data Manager
R35	CKB shall allow ST to see the list of ongoing competitions	Dashboard Manager, Competition Manager, Data Manager
R36	CKB shall allow ED to close a competition	Dashboard Manager, Competition Manager, Data Manager
R37	CKB shall allow ED to define badges in the context of a competition	Dashboard Manager, Competition Manager, Badge Manager, Data Manager
R38	CKB shall assign badges to students at the end of the competition	Competition Manager, Badge Manager, Data Manager
R39	CKB shall allow ED to define new rules for badges	Dashboard Manager, Competition Manager, Badge Manager, Data Manager
R40	CKB shall allow ED to define new variables for badges	Dashboard Manager, Competition Manager, Badge Manager, Data Manager
R41	CKB shall allow users to visualize badges obtained by a ST	Dashboard Manager, Competition Manager, Badge Manager, Data Manager
R42	CKB shall allow users to visualize a ST profile	Dashboard Manager, Data Manager
R43	CKB shall allow ST to join a T for which is invited	Dashboard Manager, Team Manager, Notification Manager, Data Manager

R44	CKB shall allow ST to join a public T	Dashboard Manager, Team Manager, Data Manager
R45	CKB shall allow ST to create a T	Dashboard Manager, Battle Manager,Team Manager, Data Manager
R46	CKB shall allow ST to set a T to public or private	Dashboard Manager, Battle Manager,Team Manager, Data Manager
R47	CKB can distinguish between an ED user and a ST user	Authentication Service, Data Manager
R48	CKB shall not allow ST/ED to see the rankings of battles they are not involved in	Battle Manager, Data Manager
R49	CKB shall have the environments for all the programming language it supports	Static Analyzer, Evaluation Controller, Code evaluator
R50	CKB shall allow ED to close a battle they manage	Dashboard Manager, Battle Manager, Data Manager

5 | Implementation, integration and test plan

The implementation, integration and test plan will follow a **bottom-up approach**, starting from the components with no dependencies and then integrating them together.

5.1. Plan Definition

Since the application is mostly server-side, we will only describe the implementation of the server components. The client-side, which is actually a presentation layer, will be implemented and tested in parallel with the server-side.

Since our application is developed on two different servers, we will describe separately the implementation plan for the two servers: the *CKB Server* and the *Evaluation Server*.

CKB Server

In the first step, the *Model* (under the specified assumption of *MVC pattern*) and the *Data Manager*, will be implemented and unit tested with a *Driver* which will substitute components which are not already implemented.

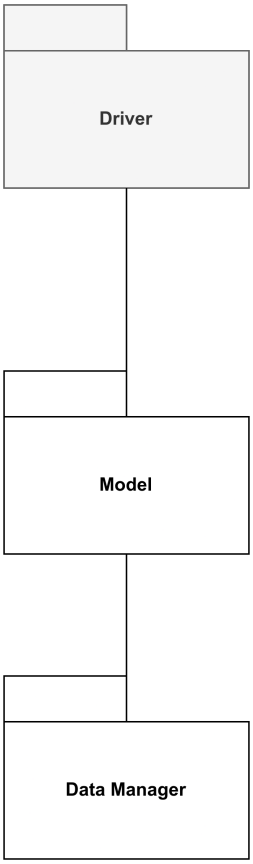

Figure 5.1: Step 1

In the second step, the Notification Manager will be implemented and tested with a *Driver* which will substitute: the *Competition Manager*,the *Battle Manager* and the *Authentication Service*. It will also use a *Stub* to simulate the *Mail Server*.

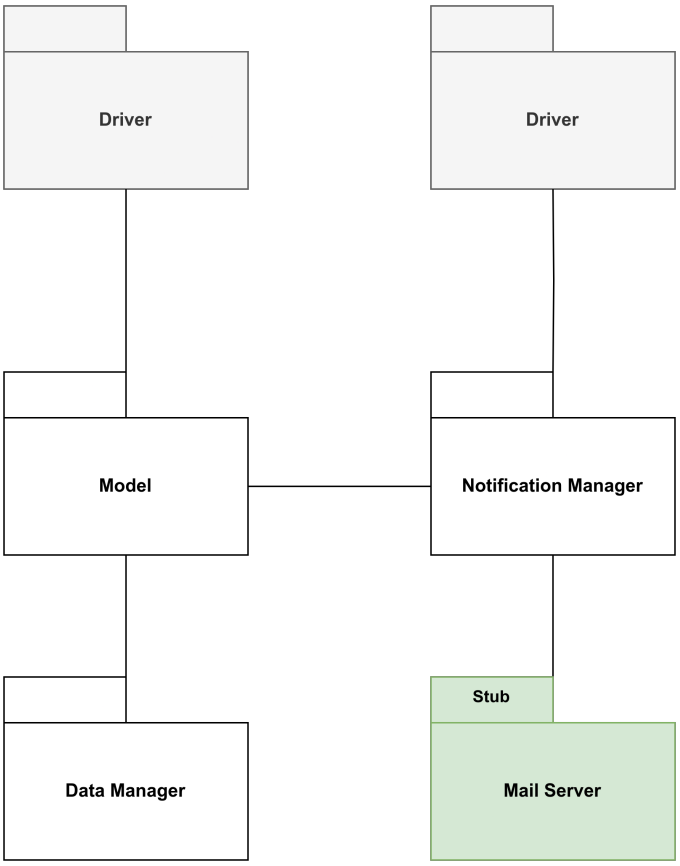

Figure 5.2: Step 2 - CKB Server

In the third step, the Authentication Service will be implemented and tested, a *Driver* will substitute the *Dashboard Manager*.

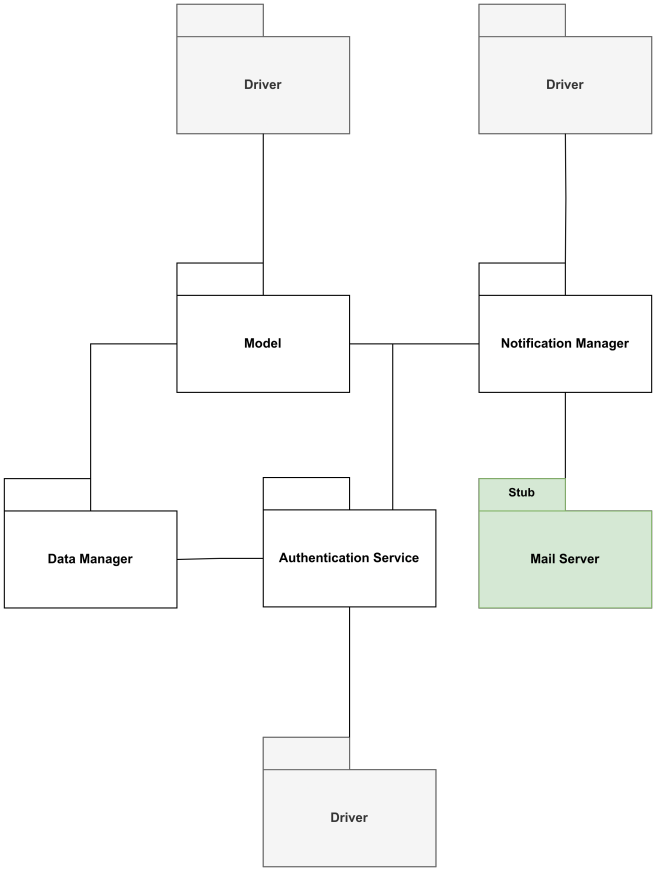

Figure 5.3: Step 3 - CKB Server

In the fourth step, the *Competition Manager*, the *Battle Manager*, the *Team Manager* and the *Badge Manager* and the will be implemented and tested in parallel, a *Driver* will substitute the *Dashboard Manager*.

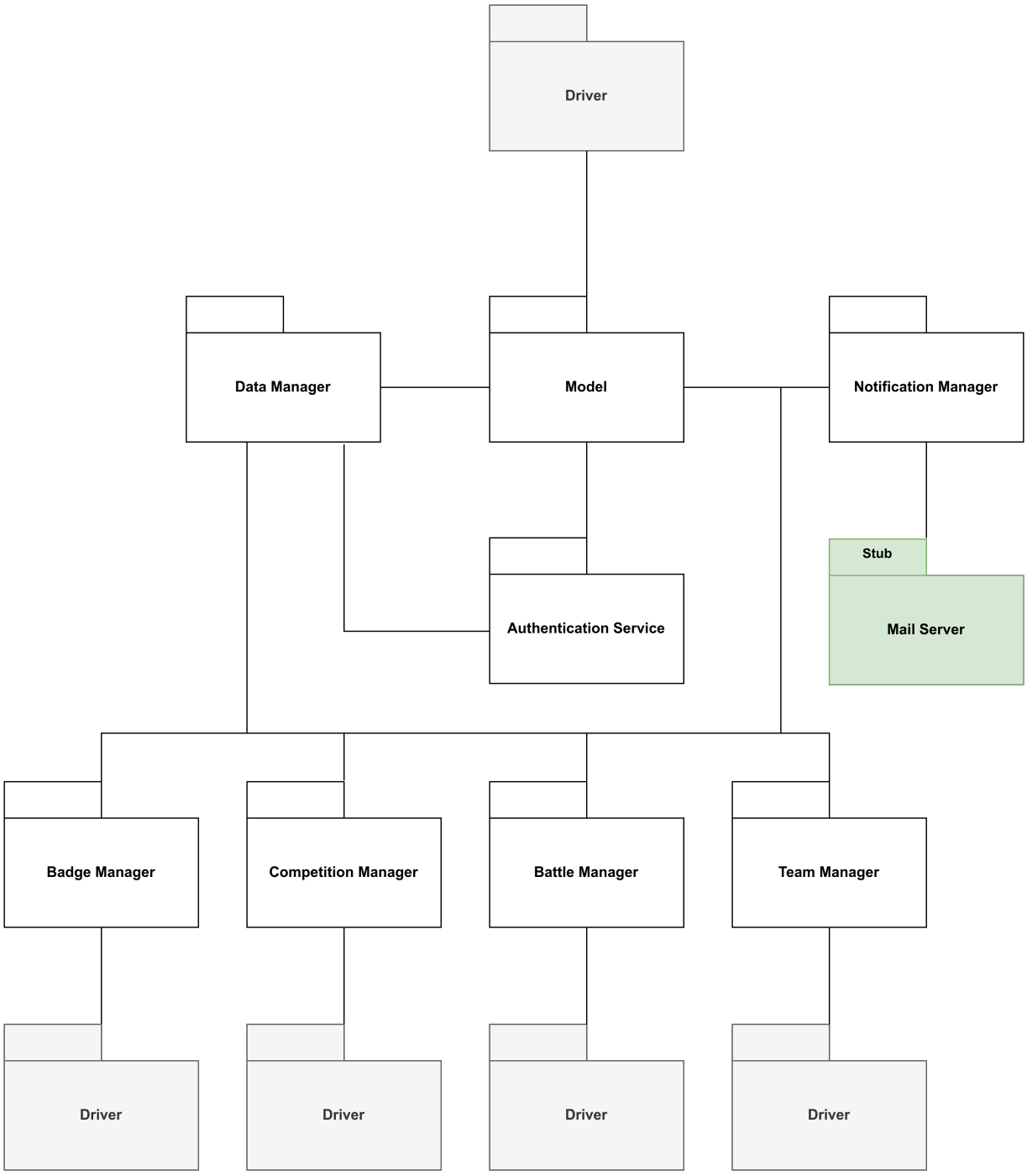

Figure 5.4: Step 4 - CKB Server

Last but not least, the *Dashboard Manager* will be implemented and tested, a *Driver* will be used to simulate the behavior of the *WebAppUI*.

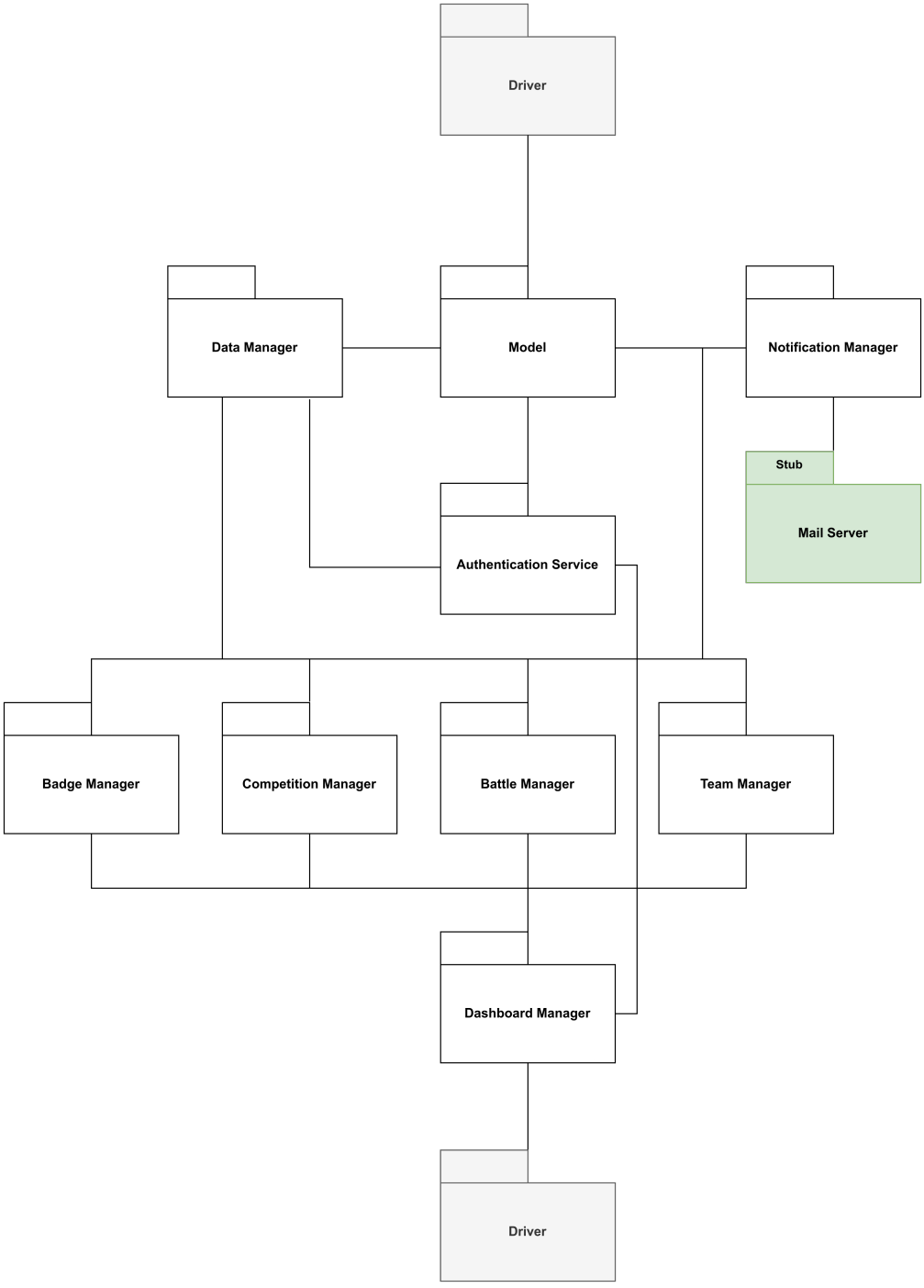

Figure 5.5: Step 5 - CKB Server

Evaluation Server

For the implementation of the *Evaluation Server*, that can be implement and tested in parallel with the *CKB Server*, we will start with the *Point Manager*, we will use a *stub* that simulate the usage of the *Data Manager* and with a *driver* that simulate both *Code Evaluator*, *Static Analyzer*.

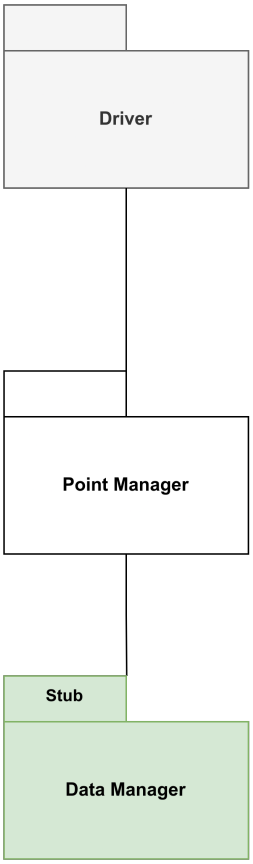

Figure 5.6: Step 1 - Evaluation Server

Once finished with the *Evaluation Server*, the implementation and testing will focus on both the *Code Evaluator* and the *Static Analyzer*, a *driver* will be used to simulate the *Evaluation Controller*

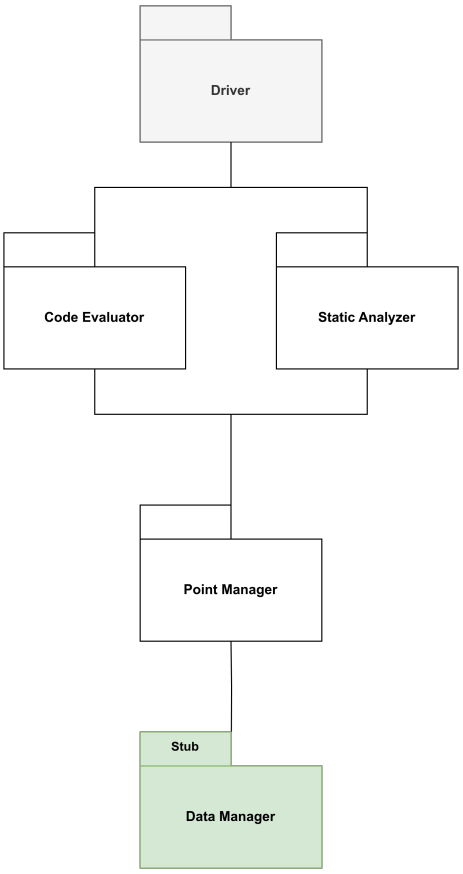

Figure 5.7: Step 2 - Evaluation Server

For the last step, the *Evaluation Contoller* will be tested and implemented, a *driver* will simulate the *github actions*

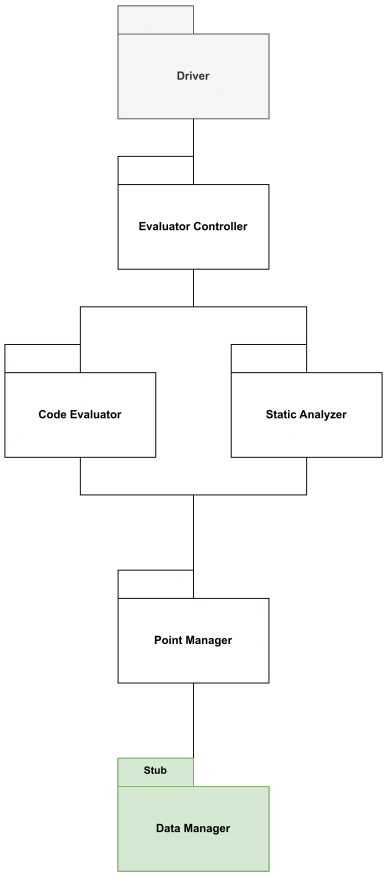

Figure 5.8: Step 3 - Evaluation Server

6 Effort Spent

Members of group	Effort spent (hours)	
Filippo Balzarini	Introduction	$0h$
	Architectural design	$11h$
	User interface design	$0h$
	Requirements traceability	$2h$
	Implementation, integration and test plan	$4h$
	Reasoning	$3h$
Christian Biffi	Introduction	$1h$
	Architectural design	$7h$
	User interface design	$8h$
	Requirements traceability	$1h$
	Implementation, integration and test plan	$0h$
	Reasoning	$4h$
Michele Cavicchioli	Introduction	$1h$
	Architectural design	$9h$
	User interface design	$0h$
	Requirements traceability	$1h$
	Implementation, integration and test plan	$2h$
	Reasoning	$5h$

Table 6.1: Effort spent by each member of the group

References

Tool Used

- **TeXstudio** to compile and format this document.
- **draw.io** to draw the component diagram and the deployment view.
- **sequencediagram.org** to draw the sequence diagrams.
- **GitHub** to share and collaborate on the project.
- **Visual Studio Code** and **Kile** to write this document.
- **Trello** to organize the work.
- **Google Docs** to writes notes for writing this document.
- **Figma** to draw the mockups.