POLITECNICO
MILANO 1863

CodeKataBattle

Design Document

Alessandro Griffanti
Luca Masiero

—

7 January 2024

Table of contents

1 | Introduction

1.1 Purpose

As the years go by, programming takes on an increasingly central position in society and this leads more and more students to enroll in university faculties inherent to this world. In this context, educational platforms like CodeKataBattle play a crucial role in helping future professional developers to hone their knowledge and skills.

This platform aims at doing so by offering to university students several coding challenges dealing with different programming concepts, like linked lists, trees, graphs and many more.

Educators will have the possibility to create tournaments and battles which will challenge every student who wants to grapple with them in a fun and collaborative way. In fact, the platform will not only enable students to develop their programming skills, but also allow them to do so with peers, promoting both the development of programming skills and the communication and teamwork abilities.

1.2 Scope

This document is intended to describe the structure and implementation choices for the CodeKataBattle platform. It presents the main components and how they interact with each other.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

CRUD	Create, Read, Update, Delete. They represent the main type of requests incoming from a client to a RESTful service
ASSISTANT	An educator who has been granted permissions to create battles inside a tournament of which he/she is not the owner
JSON	Javascript Object Notation. It represents a lightweight data interchange format

Table 1.1: Definitions

1.3.2 Acronyms

CKB	CodeKataBattle
API	Application Programming Interface
UI	User interface
DBMS	Database Management System
DD	Design Document
RASD	Requirements Analysis And Specification Document
REST	Representational State Transfer

Table 1.2: Acronyms

1.3.3 Abbreviations

Gn	N-th goal
Rn	N-th functional requirement

Table 1.3: Abbreviations

1.4 Revision History

Date	**Description**
7 January 2024	First Version

Table 1.4: Revision History

1.5 Reference Documents

- Specification document: “Assignment RDD AY 2023-2024”
- Lecture slides from the Software Engineering 2 course

1.6 Document Structure

The document is structured into the following main seven chapters:

- **Chapter 1: Introduction.** This chapter describes the purpose of the system to be as already presented in the RASD (*purpose*), a summary of the main architectural styles and choices used (*scope*). It also includes a section where all the definitions, acronyms and abbreviations used in the document are described (*definitions, acronyms, abbreviations*). Lastly, there is a revision history and a reference list.

- **Chapter 2: Architectural design.** This chapter contains the architectural design choices, at first in an informal way (*overview*), and later in a much more in depth manner. All the components (*component view*) and the dynamics of their interactions (*runtime view*) have been described, as well as the interfaces (*component interfaces*), the infrastructure (*deployment view*) and the design patterns used (*selected architectural style and pattern*).

- **Chapter 3: User interface design.** This chapter shows how the UI has to look like. It completes and deepens to the User Interfaces (3.1.1) section in the RASD.

- **Chapter 4: Requirements traceability.** This chapter explains how the requirements defined in the RASD map to the design elements defined in this document.

- **Chapter 5: Implementation, integration and test plan.** This chapter provides a description of the order to be followed in the implementation of the components and the order in which it is planned to integrate such components and test their integration.

- **Chapter 6: Effort spent.** This chapter reports approximately the amount of hours each member of the group worked to realise the document.

- **Chapter 7: References.** This section simply lists references and other resources used to redact this document.

2 | Architectural Design

2.1 Overview

In this section, they are briefly presented the main design choices of the CKB system:

- **Microservices.** This architectural style allows us to divide the functionalities of the system into several smaller services, which reduces dependencies between modules and increases scalability, availability and maintainability of the system. A detailed description of such microservices, and of their building components, will be provided in the following sections.

- **Rest API.** It represents the other main architectural style feature of the CKB system enabling the communication between the different components and, therefore, between the client-side and the server-side and between different microservices. Each of them will indeed expose its core operations through Rest APIs.

The following image shows a high-level view of the system architecture.

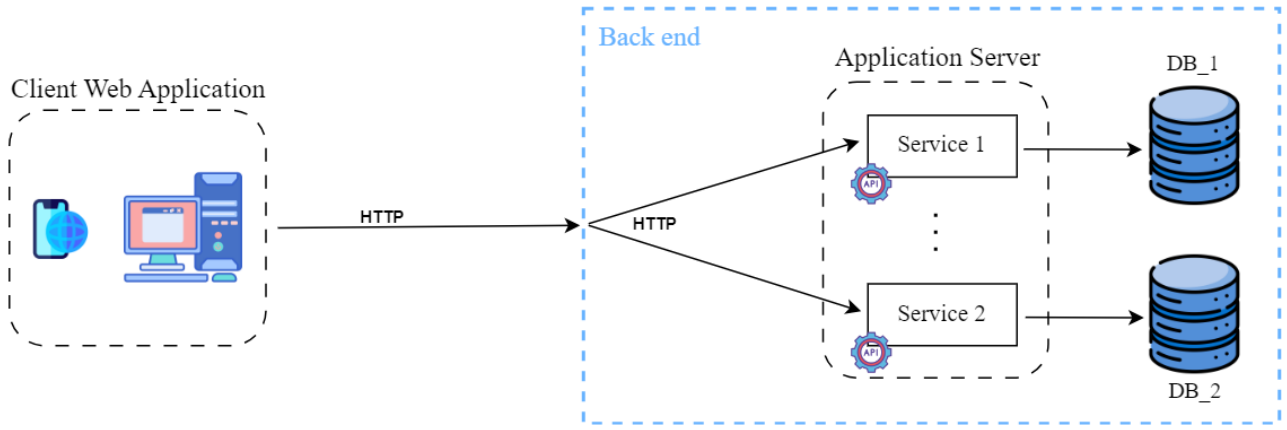

Figure 2.1: Overall Architecture

2.2 Component View

The following diagram shows the main components of the system and the interfaces through which they interact. In addition, outside the system they are shown the external services interacting with it. As for the system there are two side:

- The **Client-side** is composed of two components, the Educator and the Student component

- The **Server-side** is composed of several components that offer different functionalities. In the following diagram, each server-side component is represented as stand alone, while in the following section (2.3) it is shown how they are organized into several microservices.

After the diagram, it will follow a brief description of each identified component.

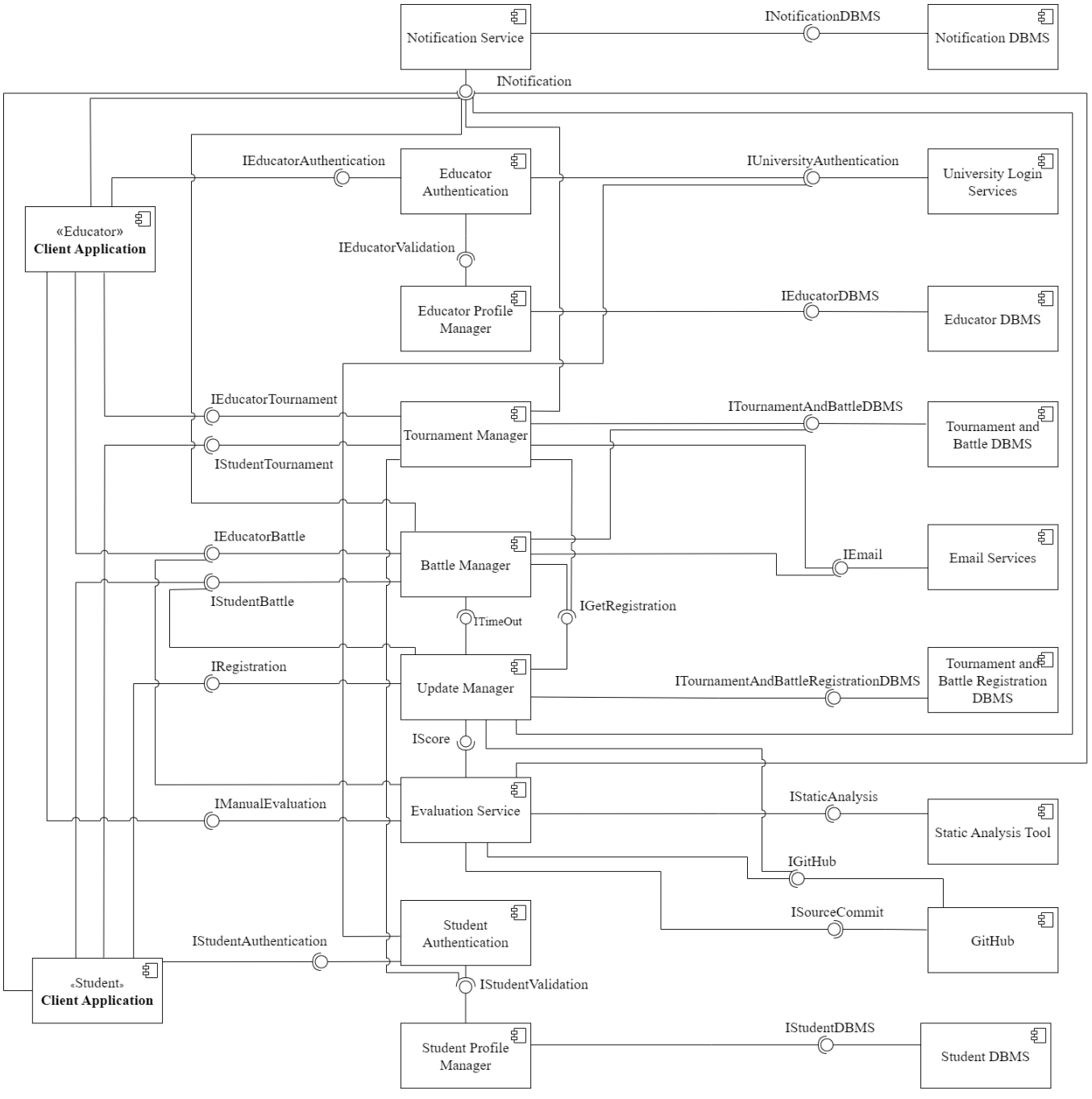

Figure 2.2: Component diagram

The components are:

- **Client Application.** Client Application (<<Student>>, <<Educator>>) represents the client system used to connect and to communicate with the CKB system.

- **Educator Authentication & Student Authentication.** This components handle the authentication process (registration & login) made by respectively educator and students.

- **Educator Profile Manager & Student Profile Manager.** These components cooperate with the Educator and Student Authentication components in the authentication process. In particular, they directly interact with the two DBMS used to store the educator and student information, which is fundamental to check if an educator's (or student) profile already exists. Moreover, the Student Profile Manager is also important to retrieve (through the Student DBMS) all the students that must be notified, for example, when a new tournament is created.

- **University Login Services.** This external component is responsible for checking the existence of an educator (and student) in the university DBMS during the authentication process.
  It helps the system verify the type of user: student or educator.

- **Educator DBMS & Student DBMS.** These DBMS, as already stated in the description of the previous components, represent the systems used to save the information regarding the educators and the students.

- **Tournament Manager & Battle Manager.** These components handle the requests, from both students and educators, dealing with all the tournaments and battles available in the platform.

- **Update Manager.** This component, unlike the Tournament Manager and Battle Manager, handles all the requests regarding all the tournaments and battles that are currently ongoing. This means that it deals with all the requests concerning the student registration for tournament and battles, as well as the update of tournament and battle scores and the retrieving, for example, of tournaments and battles a student is registered for.

- **Email Services.** This external component is responsible for notifying, through an email, all the students when it's required, for example when the registration deadline for a battle expires and the battle begins.

- **Tournament and Battle DBMS.** This DBMS represents the system used to save information regarding all the tournaments and battles on the platform.
  They store general information like deadlines, creator, assistants of a tournament and state.

- **Tournament and Battle Registration DBMS.** This DBMS represents the system used to save information regarding all the ongoing tournaments and battles on the platform. Among other information there are those regarding all registrations to a tournament or a battle, and the scores of each student enrolled in a tournament and each team enrolled in a battle.
  This DBMS is in fact crucial for providing the ranking of battles and tournaments.

- **Notification Service.** This component is responsible for handling the platform notifications both for students and for educators, such as when a student is invited to join a battle in a team or when an educator is granted the permissions to create battles in a tournament.

- **Notification DBMS.** This DBMS represents the system used to save information regarding all the platform notifications that every user (both students and educators) shall see after logging into the platform.

- **Evaluation Service.** This component is responsible for all the score updating of battles and tournaments. It indeed interacts, through proper api calls, with GitHub to retrieve all the sources, evaluate them personally and through an external static analysis tool, and with the educator during the consolidation stage when a manual evaluation is required.

- **GitHub.** This external component hosts the repositories of the battles that are forked by the students to work.

- **Static Analysis Tool.** This external component is responsible for evaluating the students' work.

2.3 Deployment View

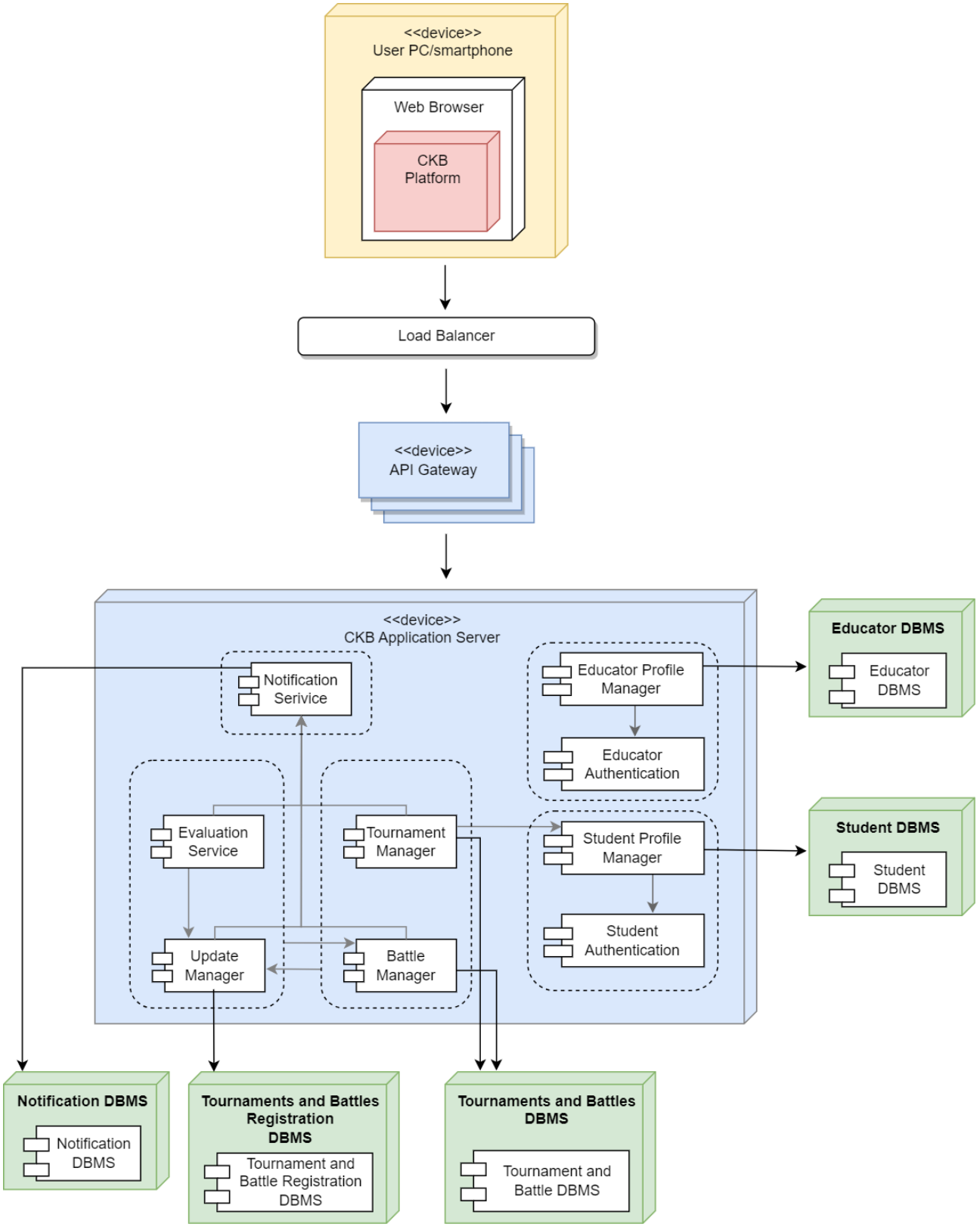

Figure 2.3: Deployment diagram

Connection to the Server

Students and educators can access the CKB system from smartphone or PC, in both cases they need to use a browser in order to connect to the web application.
Before coming to the application server a request must go through a load balancer and an API gateway in this order.
First the load balancer redirects the request to one of the instances of the API gateway; these two components improve the availability and performance of the system.
Going into more detail on the API gateway, its main functionalities are:

- **Routing**. Each client request before reaching the server is sorted by the API gateway based on its destination address and sent to the right service

- **Security**. The API gateway performs rate limiting which prevents Denial Of Service attacks against the system

Microservices

Figure 2.3 shows how the system was divided into five different services.
Each of them has its own database and no other service accesses the same one, this may cause a higher coupling between the services since they need to interact with one another, but at the same time we end up having a single entity able to modify each DBMS.
In this way if a DBMS, in particular its interface, undergoes some changes the number of services to be modified is smaller.

The microservice choice allows to improve the scalability and ease of deployment and maintenance.

2.4 Runtime View

In the following sequence diagrams, we are going to present and explain the interactions that happen between the main components of the CKB system. The description of these interactions is meant to deepen the description presented in the sequence diagram section (3.2.3) of the RASD.

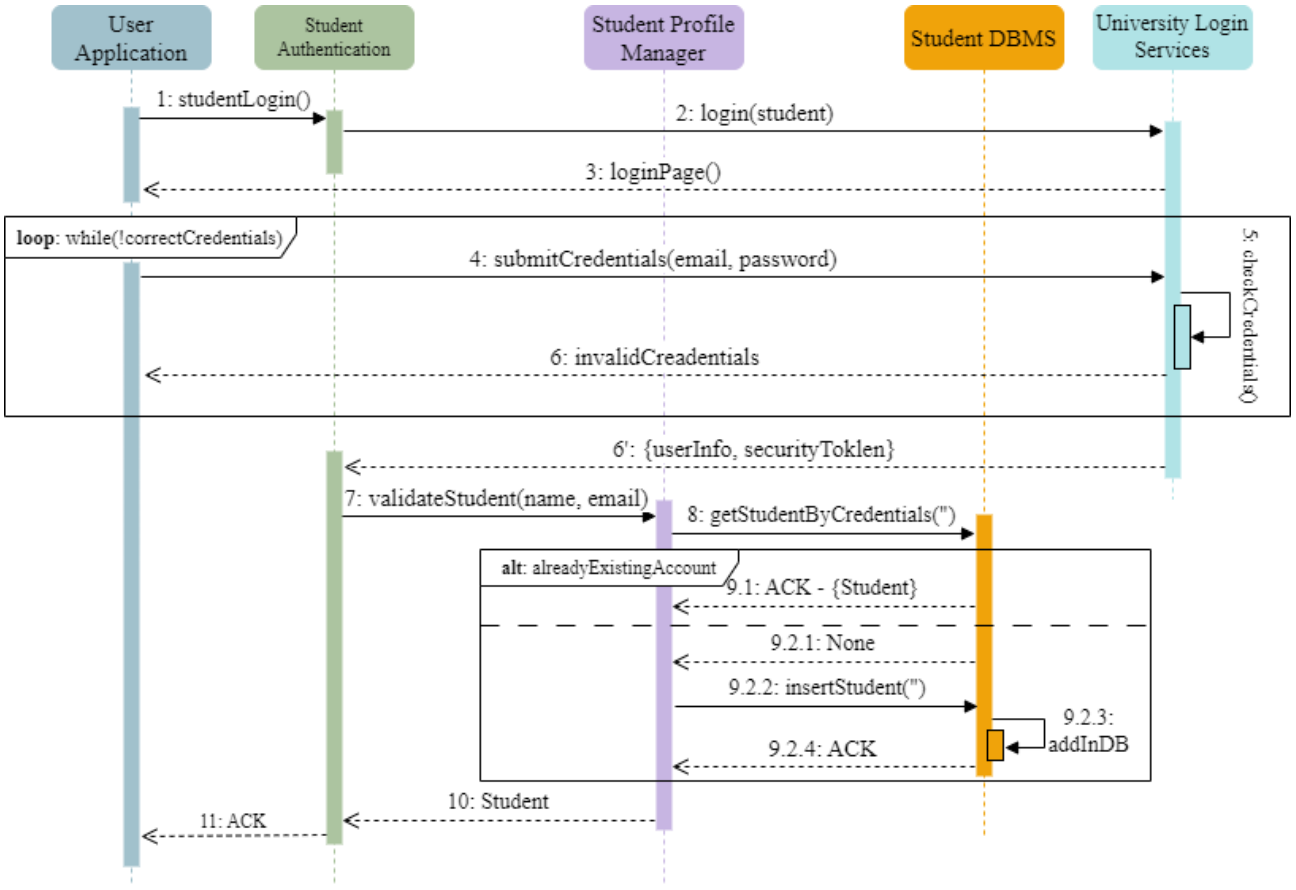

Figure 2.4: Student registration and login sequence diagram

The login approach of the system is inspired by the SSO (Single Sign On) technique, in fact the external component “University Login Services” deals with the authentication and authorization of the user.
When a student wants to log into the CKB platform, the client application calls the “studentLogin” method of the “student Authentication” component. This component then contacts the login service component in the University system, which requires the user (a student in this case) to provide his/her credentials. Once the user is verified by the university login services the CKB system checks for an existing account and if necessary creates a new one, adding it to the “StudentDBMS”.

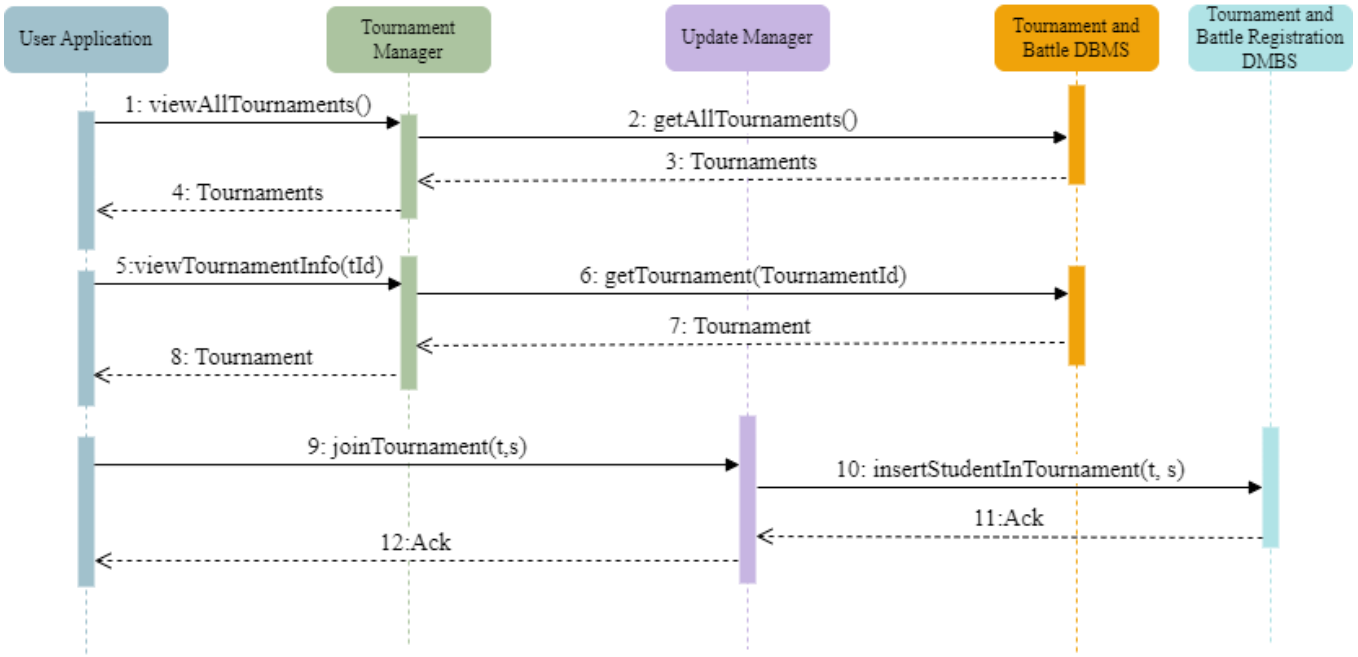

Figure 2.5: Student tournament registration sequence diagram

When a student has logged in, he can decide to register for a tournament available on the platform. To achieve so, he/she first retrieves all the available tournaments through the "viewAllTournaments" method offered by the "Tournament Manager" component. After that, the student selects the tournament he wants to register for and views all its information, such as the programming language, the tournament creator and the registration deadline, through the "viewTournamentInfo" method offered again by the "Tournament Manager" component. In the end, the student makes the request to join the tournament through the "joinTournament" method offered by the "Update Manager", which updates the "Tournament and Battle Registration DBMS" inserting the new student.

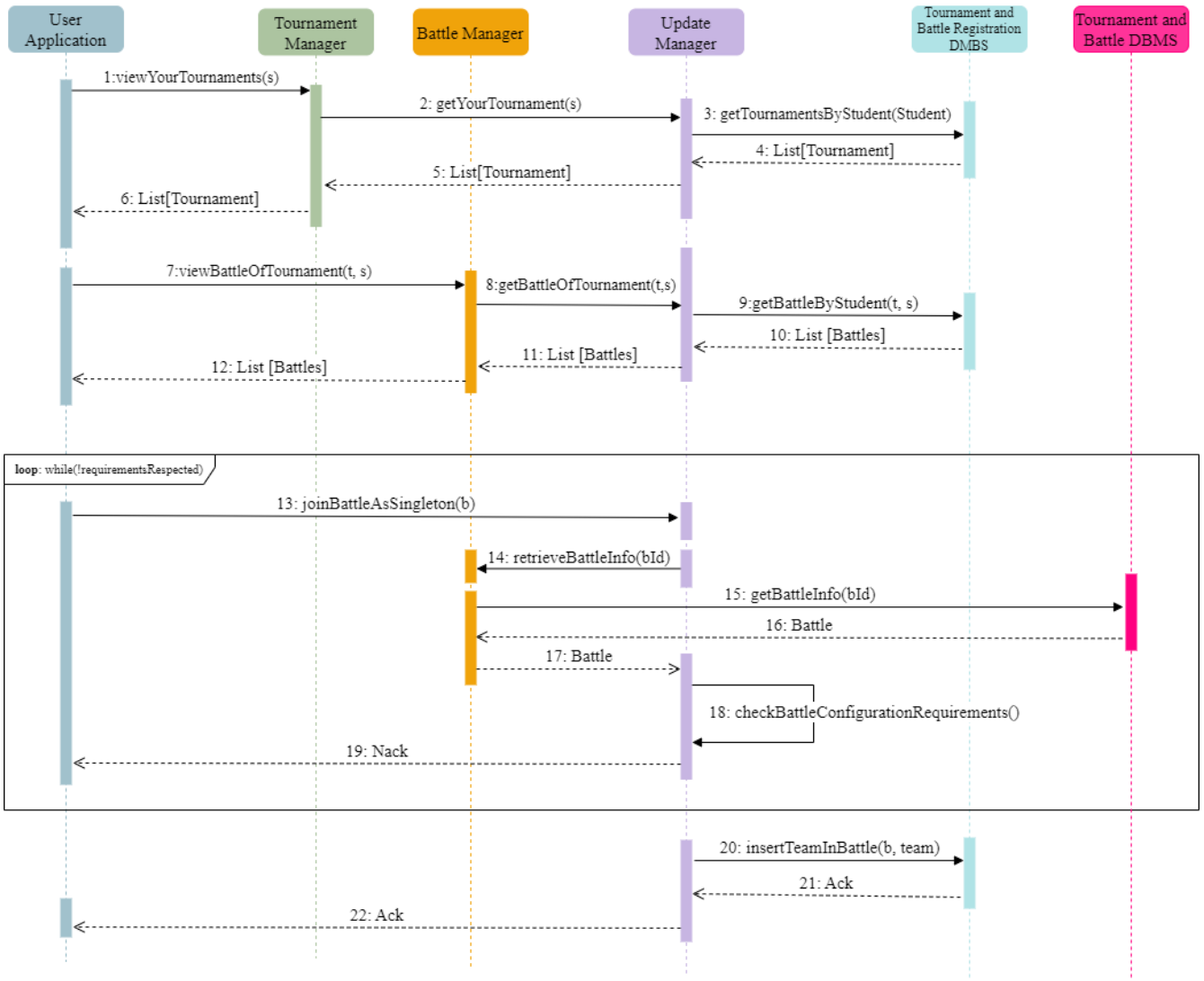

Figure 2.6: Student battle registration as a singleton sequence diagram

After joining a tournament, a student can decide to register for one of its battles. To achieve so, he/she first retrieves all the tournaments he is enrolled in through the "viewYourTournaments" method offered by "Tournament Manager", which forwards the request, thanks to the "getYourTournament" method, to the "Update Manager", which then retrieves the tournaments from the "Tournament and Battle Registration DBMS". Then, the student opens one of the tournaments he/she is enrolled in, which means he can see all the battles it contains. This is done in a similar way to what just described, through the "Battle Manager" and the "Update Manager" which retrieves all the battles the student is registered for in a given tournament. After that, the student makes the request to join as a singleton through the "joinBattleAsSingleton" method offered by the "Update Manager" which is responsible for retrieving all the information regarding that same battle through the "Battle Manager" and the method "retrieveBattleInfo". The information of the battle is retrieved from the "Tournament and Battle DBMS" and propagated to the "Update Manager", which checks if the battle configuration requirements are satisfied: for example, the creator of the battle could have set as '2' the minimum number of students per team for that battle. If so, the student can't join as a singleton and the system notifies so. On the other hand, if the

requirements are fulfilled, the student is inserted, as a team, in the “Tournament and Battle Registration DBMS”.

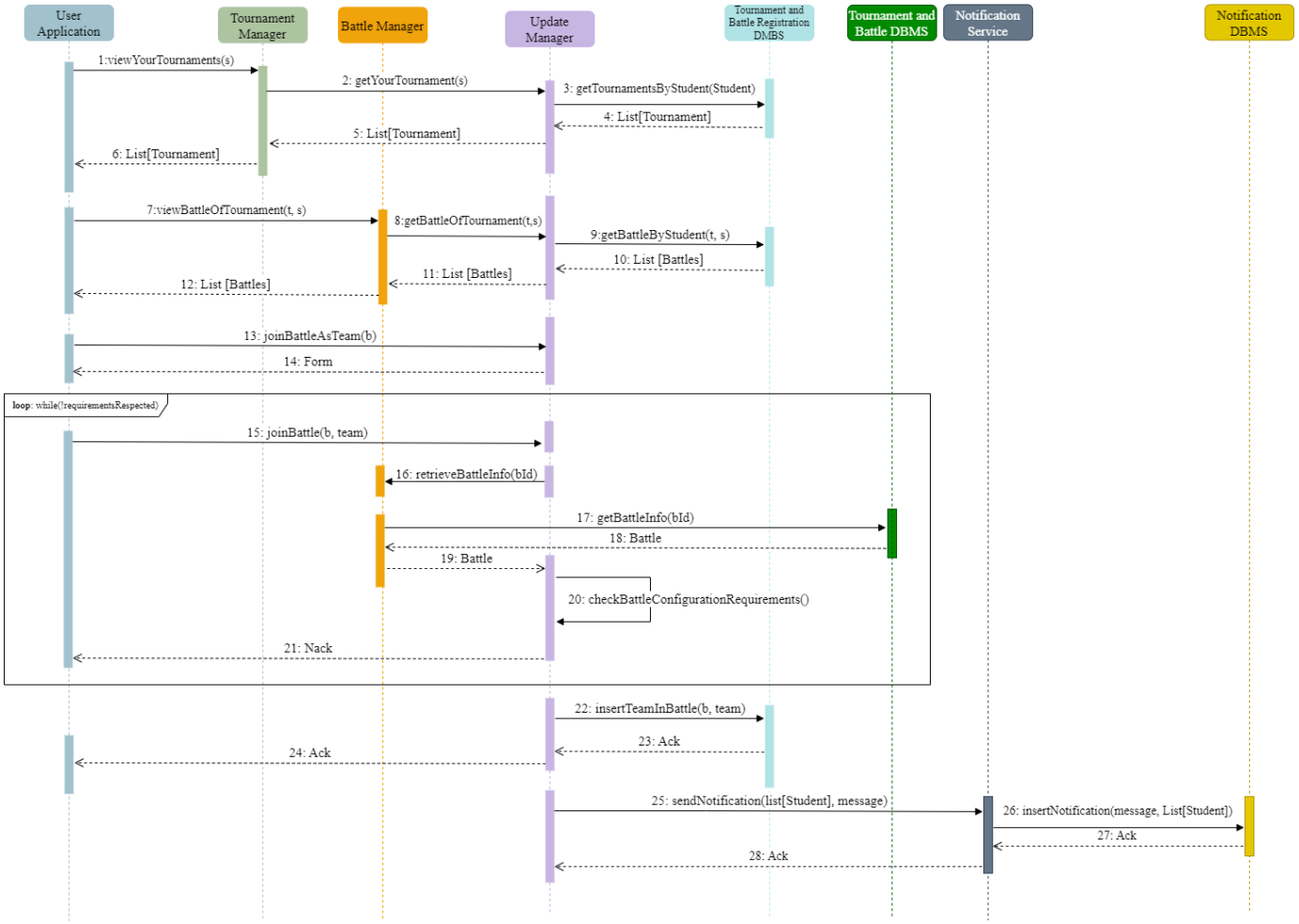

Figure 2.7: Student battle registration as a team sequence diagram [1]

As described in the previous sequence diagram, a student opens a battle of a tournament he/she is enrolled in but now he/she decides not to join as a singleton but in a team. To achieve so, he makes the request “joinBattleAsTeam” to which the system (the “Update Manager” in particular) responds with the form regarding the peers the student wants to join with. After filling out the form, the student makes the request to join the battle through the “joinBattle” method offered by the “Update Manager”. Then, in the exact same way as described in the previous sequence diagram, the system checks that the battle requirements are satisfied. If so, the whole team, peers included, is inserted in the “Tournament and Battle Registration DBMS” and an invite notification is sent to the peers through the “sendNotification” method offered by the “Notification Service” component.

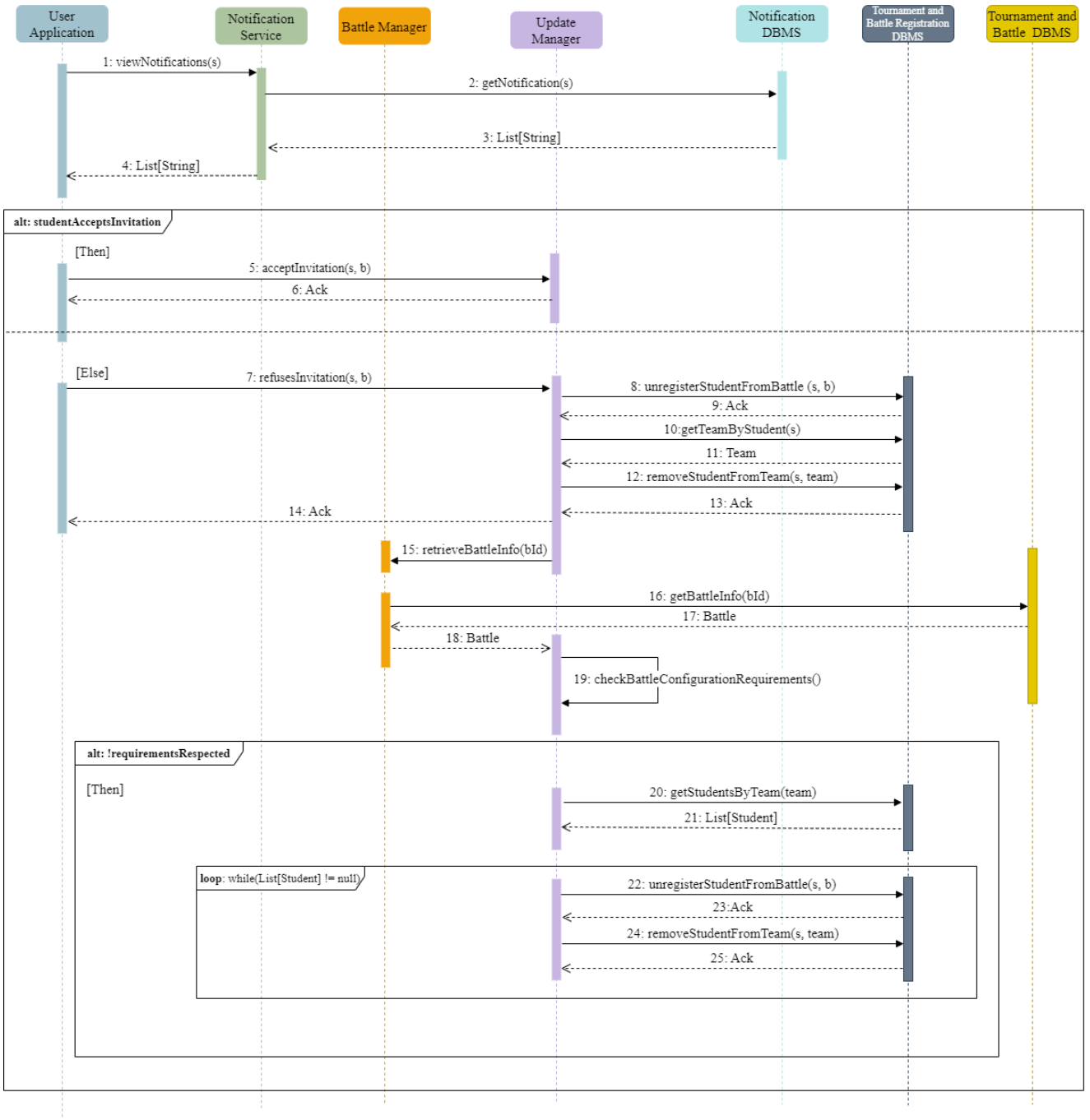

Figure 2.8: Student battle registration as a team sequence diagram [2] - student accepts an invitation

A peer who received an invitation to join a battle can accept or refuse it. First of all, the student retrieves all its notifications through the “Notification Service” which recovers them from the “Notification DBMS”. If the student accepts the invite, nothing really happens, since he is already part of the team who registered for that battle (see Figure 2.6). On the other hand, if he/she refuses the invite, he is unregistered from the battle and from the team he/she was part of through the “unregisterStudentFromBattle” and the “removeStudentFromTeam” methods. After that, in the exact same way as described in the previous sequence diagrams, the system checks that the battle requirements are satisfied. In fact, it could happen that, after a student’s refusal, the team no longer satisfies the

requirements for that battle (e.g. the minimum number of students per team for that battle is 3 and, in a team of 3 peers, one of them refuses the invitation, which means that the team is now composed of 2 peers). If the requirements are no longer satisfied, the system unregisters all the students of that team from the battle and, obviously, from the team.

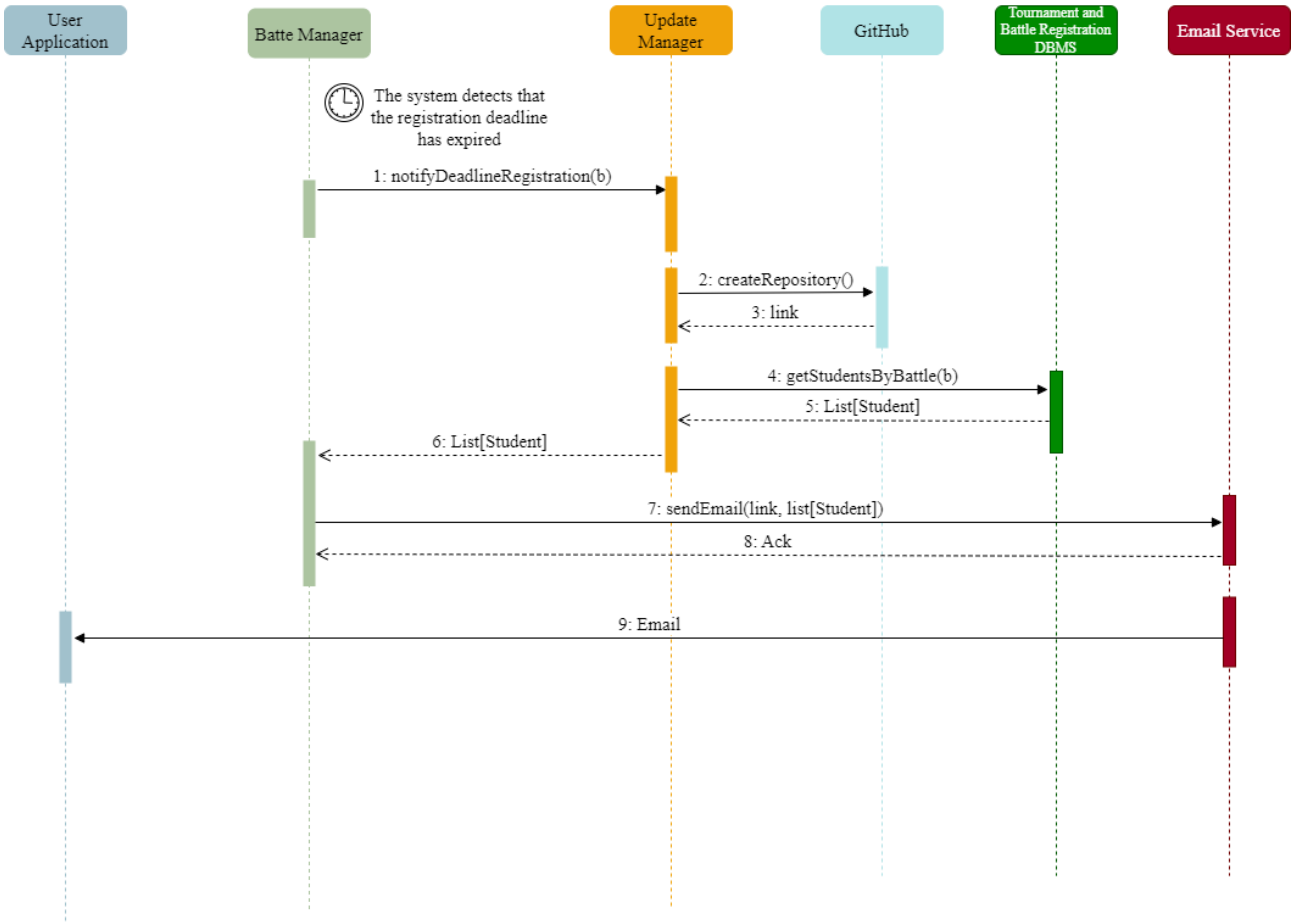

Figure 2.9: Battle begins sequence diagram

When the registration deadline of a battle expires, the “Battle Manager” component notifies the “Update Manager” which through the API offered by GitHub creates a repository, whose link is then sent to all the students enrolled in the battle. The “Update Manager” indeed retrieves all the students enrolled in the battle through the “getStudentsByBattle” method and then an email is sent through the external Email Services.

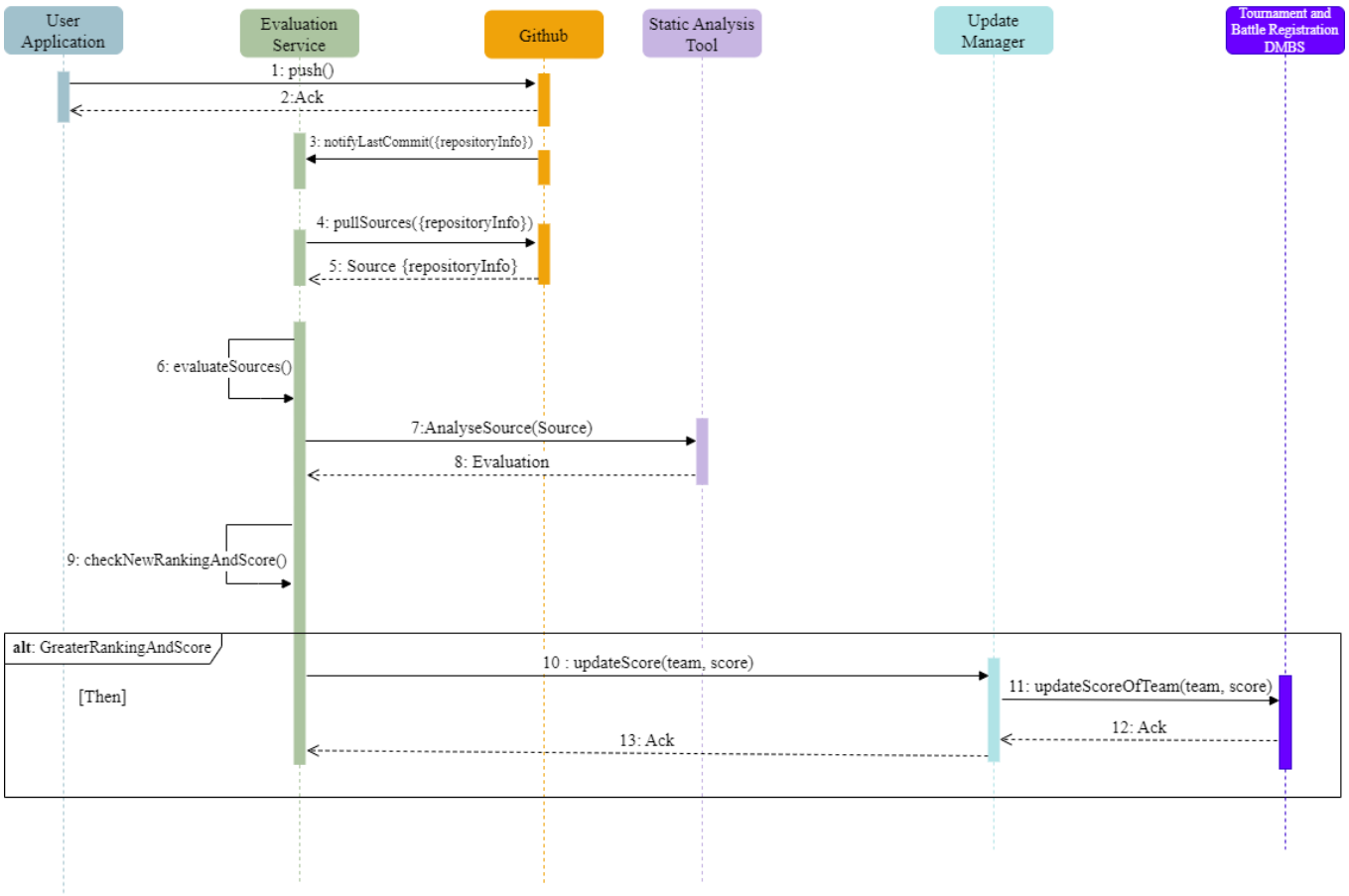

Figure 2.10: Battle ranking and score are updated sequence diagram

When a student performs a push on the forked repository, a new GitHub action workflow is performed. First, the system, in particular the "Evaluation Service", is notified of the new push and then it retrieves the sources from the repository. All the repository information, such as the creator, the date of the last commit and the code source, is passed with JSON. After that, the system evaluates the sources both personally and through an external static analysis tool. In the end, if the score of the team is greater than the one currently registered, the system updates it.

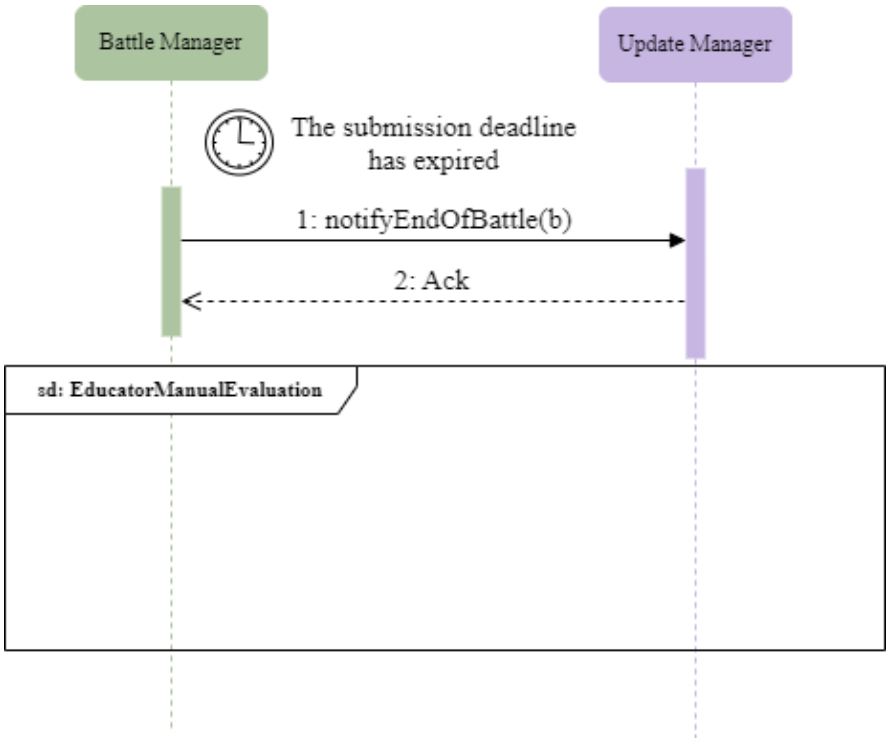

Figure 2.11: Final battle ranking and score are updated sequence diagram

When the submission deadline of a battle expires, the “Battle Manager” component notifies the “Update Manager” and, after that, the manual evaluation of an educator is required (see Figure 2.15).

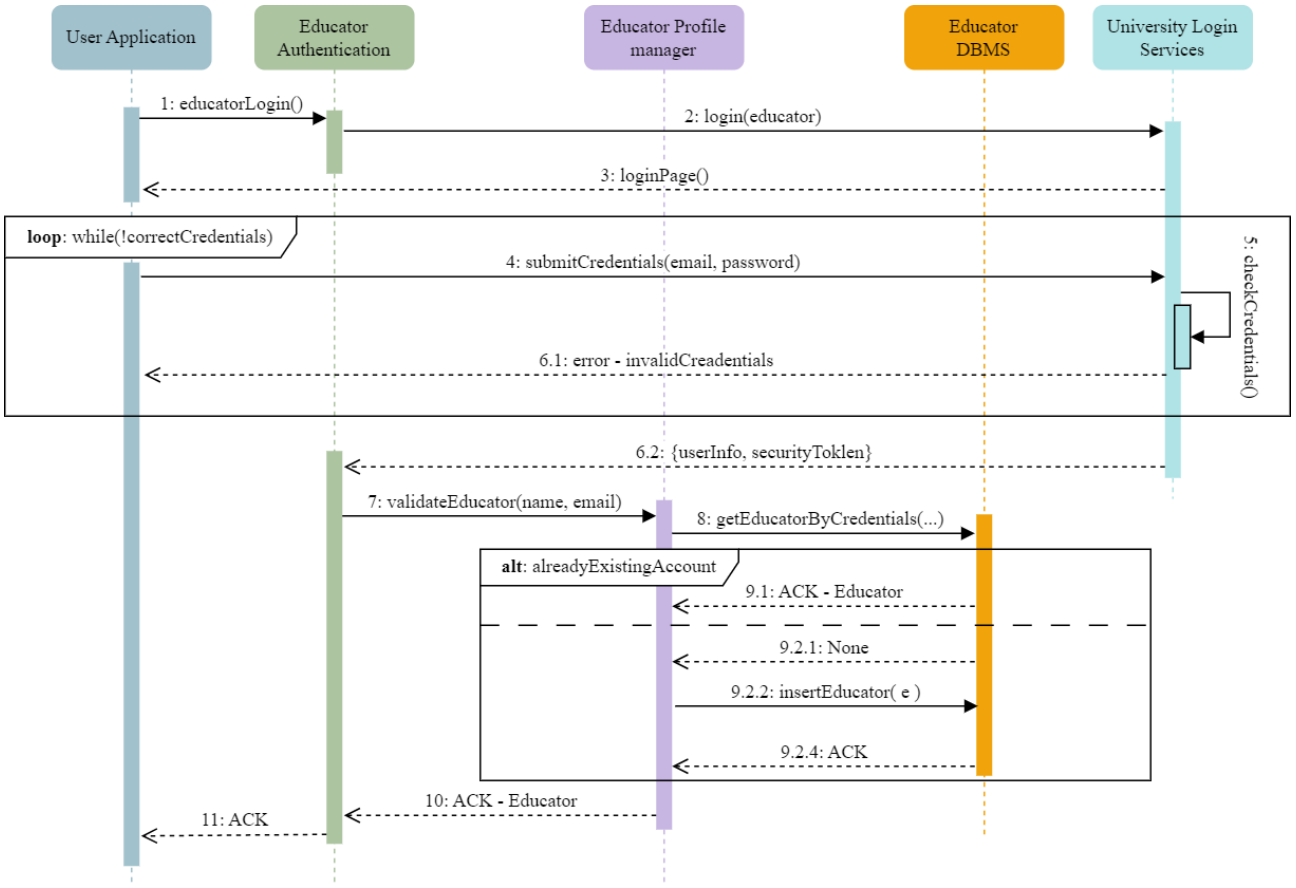

Figure 2.12: Educator logs in sequence diagram

The login approach of the system is inspired by the SSO (Single Sign On) technique, in fact the external component “University Login Services” deals with the authentication and authorization of the user.
When an educator wants to log into the CKB platform, the client application calls the “educatorLogin” method of the “Educator Authentication” component. This component then contacts the login service component in the University system, which requires the user (an educator in this case) to provide his/her credentials. Once the user is verified by the university login services the CKB system checks for an existing account and if necessary creates a new one, adding it to the “EducatorDBMS”.

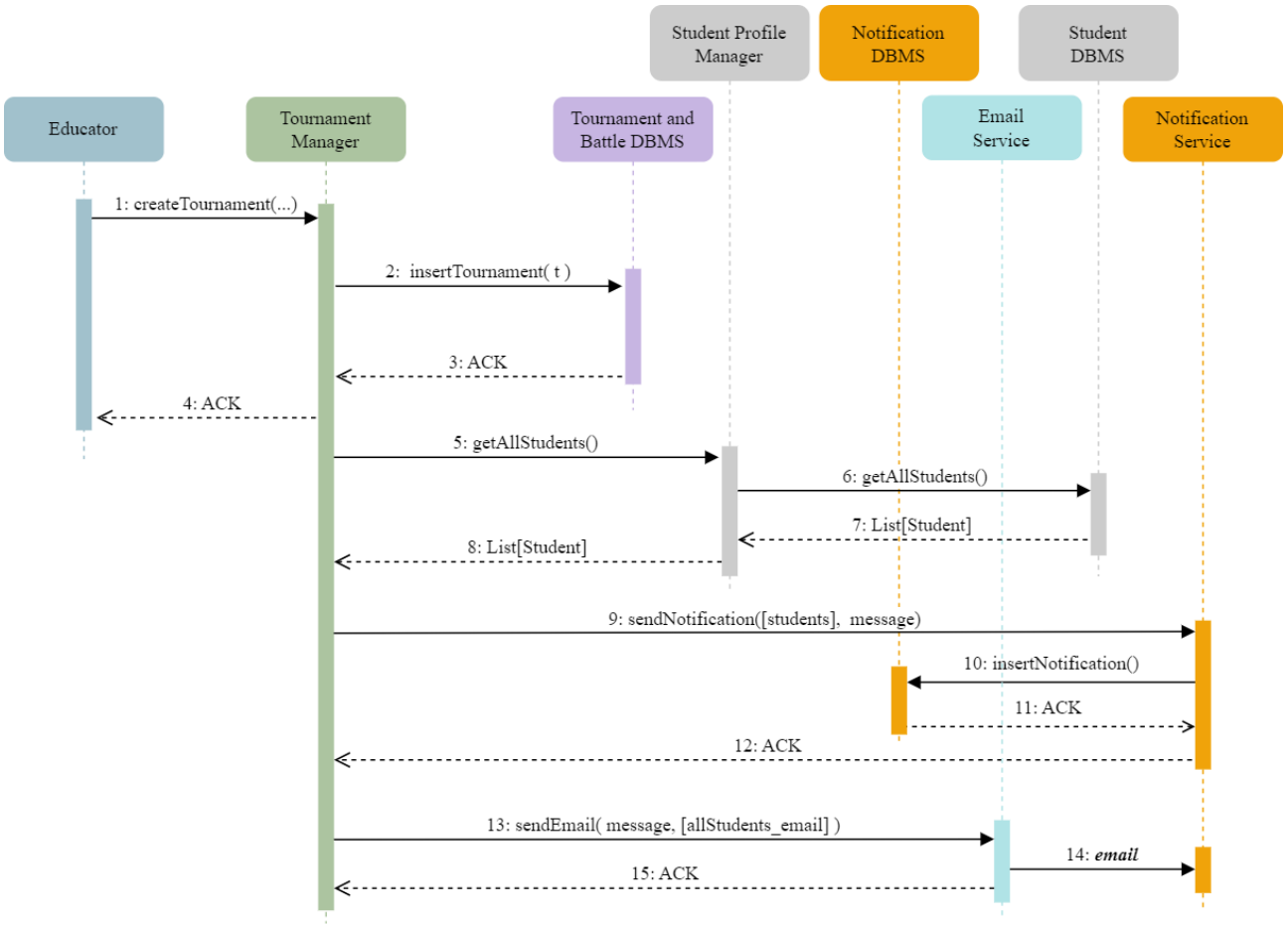

Figure 2.13: Educator creates tournament sequence diagram

When an educator is on his/her personal page on the CKB platform, he/she can choose to create a tournament.
The client application requires the educator to enter all the compulsory information and then initiates the tournament creation by calling the "createTournament" method of the "Tournament Manager" component.
The provided parameters are: title, registration deadline, programming language and a list of educators with permissions to create battles in the tournament.
The "Tournament Manager" in turn takes care of adding a new entry into the database and notifying all the students registered on the platform by means of an email and a platform message.
The components used to manage the notifications are the "Notification Service" and the "Notification DBMS"; with them the system keeps track of the notifications addressed to each user.

Pay attention to the fact that in order to choose the educators to whom grant permissions to create battles inside the tournament, the system must check for the existence of their accounts; to have an idea of how this work make reference to the following sequence diagram ("Educator grants permissions").

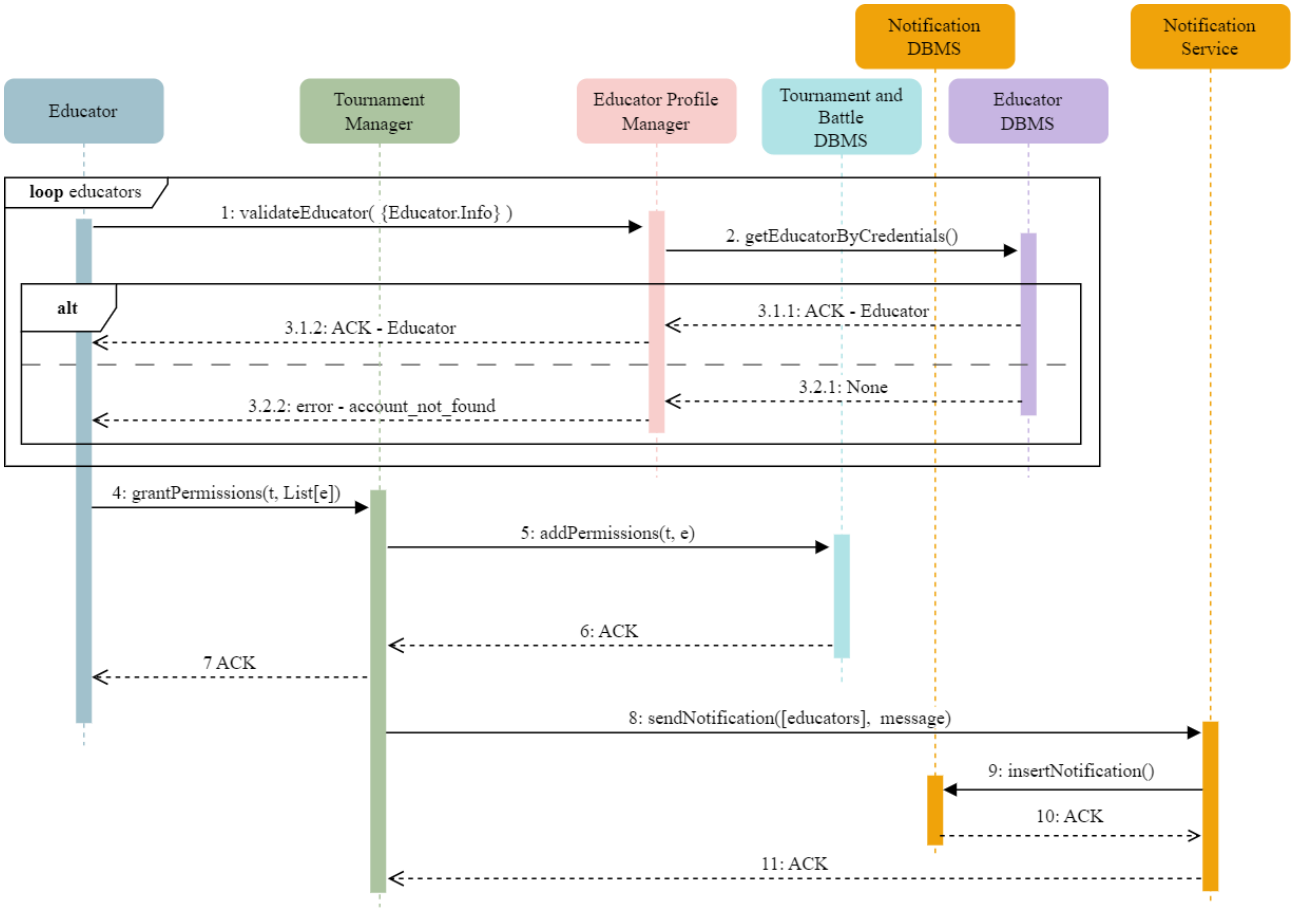

Figure 2.14: Educator grants permissions sequence diagram

The creator of a tournament can choose other educators as assistants; once granted permission the assistants can create battles inside that tournament.
The process starts with the educator selecting the assistants, and for every assistant the client application calls the “validateEducator " method of the “Educator Profile Manager” component. It then checks whether an account exists for that educator by querying the “Educator DBMS”.
When the educator confirms his/her choices through the “grantPermissions” method, the “Tournament Manager” component adds the information to the “Tournament and Battle DBMS” and finally notifies the involved educators through an application message.

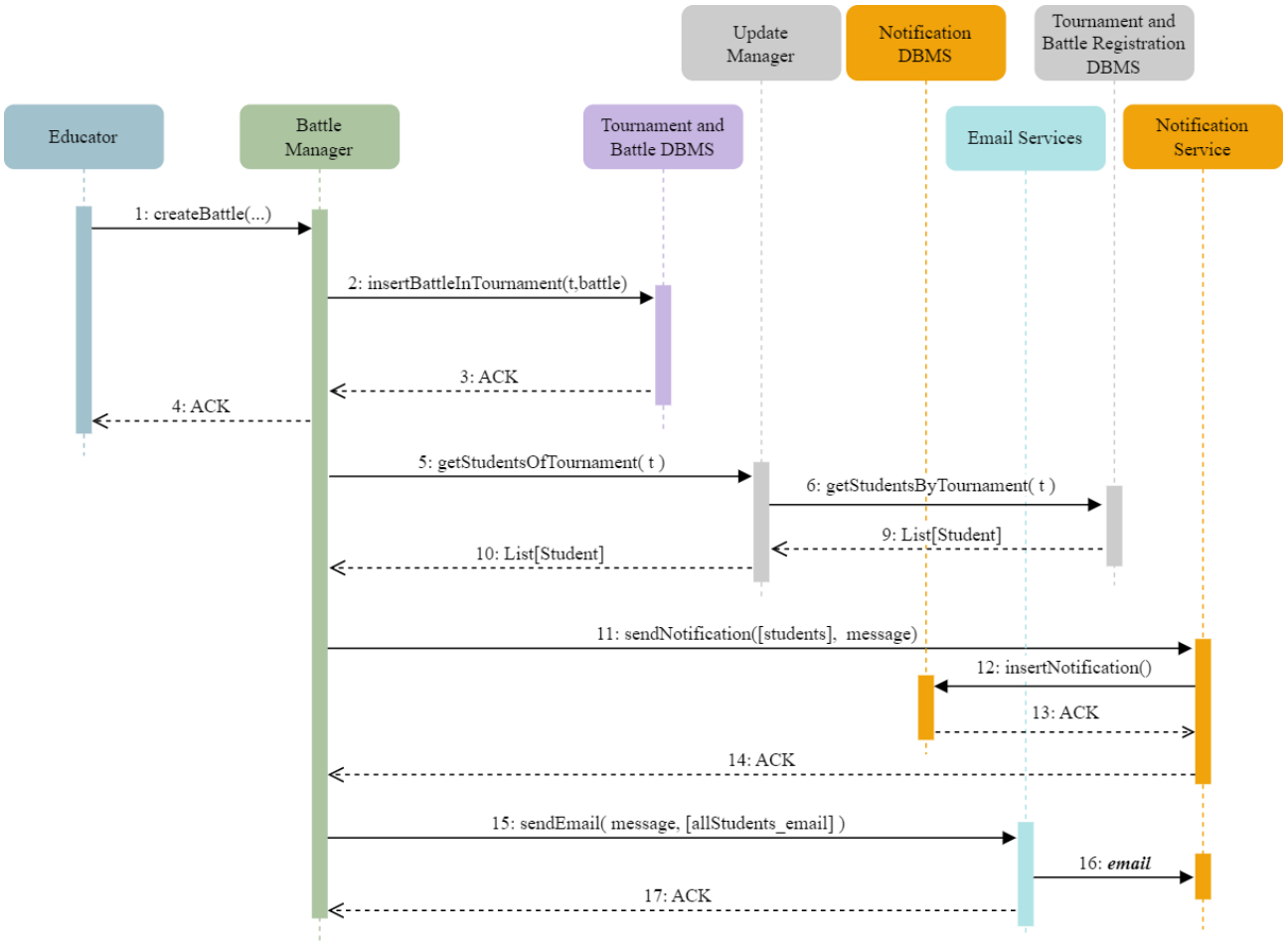

Figure 2.15: Educator creates battle sequence diagram

An educator can also create a battle in the CKB platform. Firstly the educator provides the information about the new battle and then the client application calls the “createBattle” method of the “Battle Manager” component.
At this point the process is similar to the “Educator creates tournament” sequence diagram; in this case is the “Battle Manager” component that deals with the “Tournament and Battle DBMS” inserting the new entry and notifying, through an email and a platform message, all the students who are enrolled in the tournament to which the battle belongs.

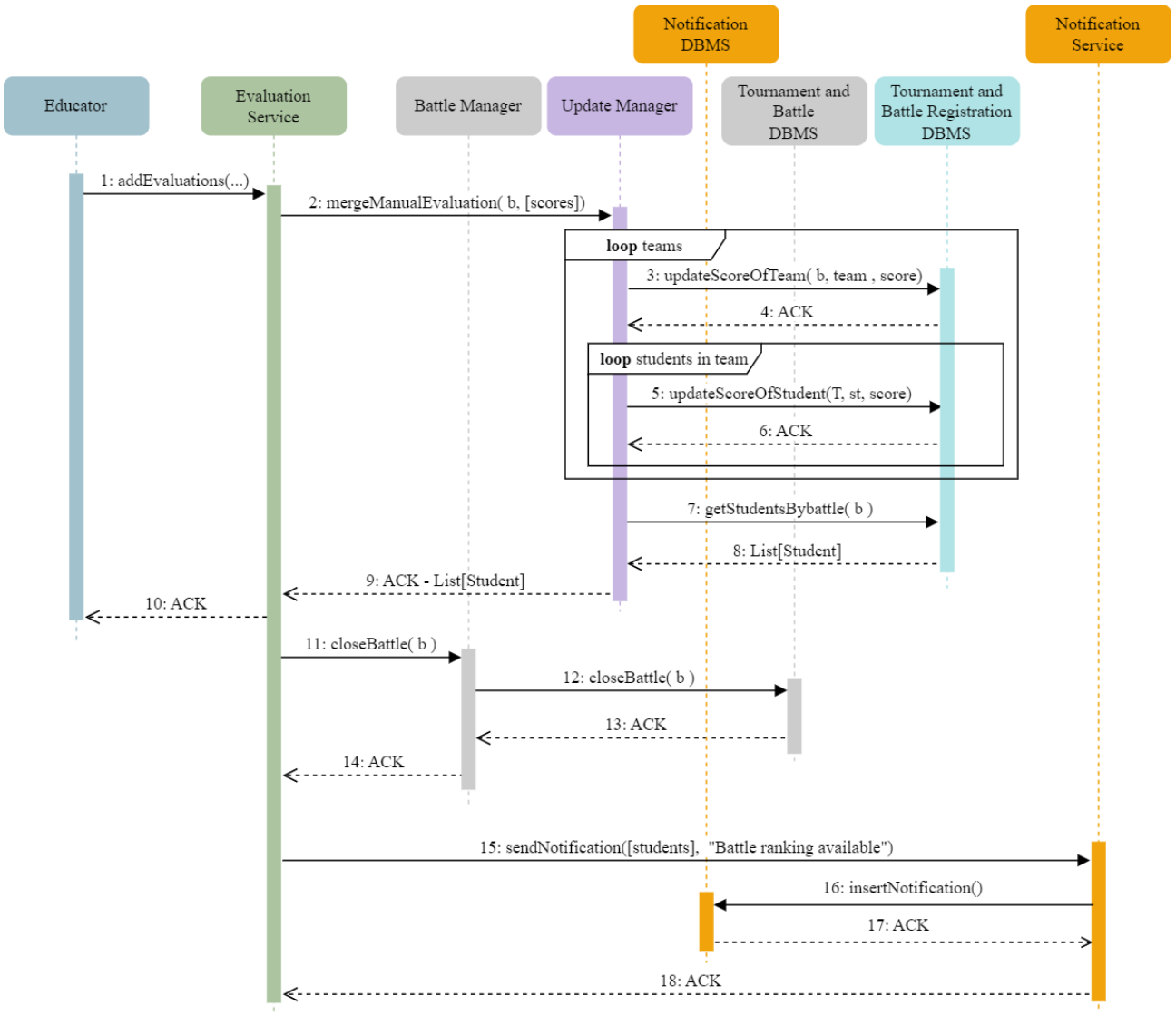

Figure 2.16: Manual evaluation of a battle sequence diagram

At the end of a battle the educator can add his/her personal evaluation to the source submitted by each team.
After the educator specifies the score awarded to each team the client application calls the "addEvaluation" method of the "Evaluation Service" component passing the list of scores and the battle they refer to.
The "Evaluation service" uses the "Update Manager" component as an intermediary to update the score of the battle for each team participating in it. At this point the result is definitive, so the "Update Manager" also updates the score of each student taking part in the tournament.
The "Evaluation service" then set the battle as closed modifying the "Tournament and Battle DBMS".
The last set of messages is for notifying the students that the battle final ranking is available; again the "Notification Service" and the "Notification DBMS" are used to keep track of the notifications of each user.

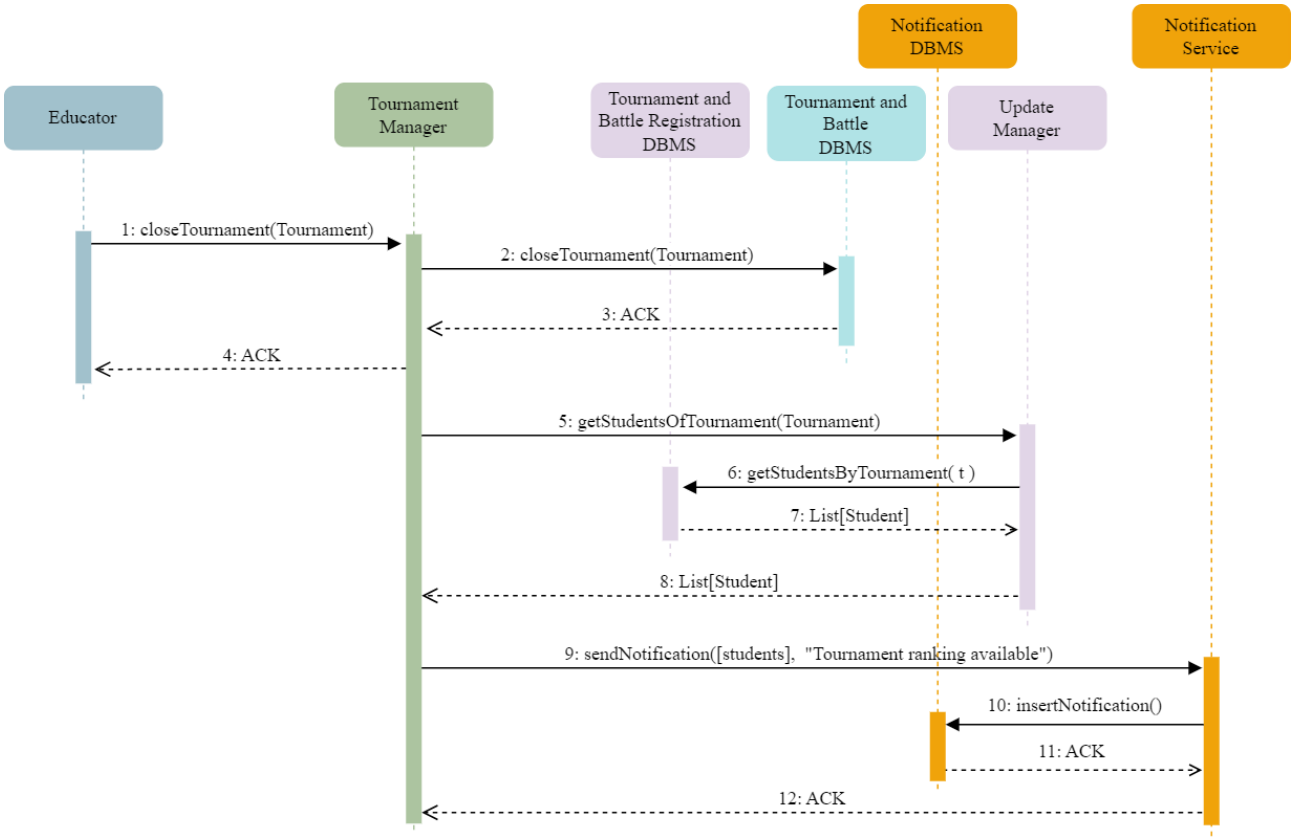

Figure 2.17: End of tournament sequence diagram

First of all the educator closes the tournament by calling the “closeTournament " method which, through the “Tournament Manager” component, ends up modifying the “Tournament and Battle DBMS”.
There is no need to modify the scores of the students because when a battle ends they are automatically updated.
Since the final scores are already available there is nothing else to do but retrieve the students who took part in the tournament from the “Tournament and Battle Registration DBMS” through the “Update Manager” component and notify them using the same process as the previous sequence diagrams.

2.5 Component Interfaces

In this section the interfaces of the components are presented and analysed, defining the methods to access them.

Educator Authentication

IEducatorAuthentication

- `educatorLogin(): void`

Student Authentication

IStudentAuthentication

- `studentLogin(): void`

University Login Services

IUniversityAuthentication

- `login(userType: String): {name:String, email:String}`
- `submitCredentials(email:String, password:String): boolean`

Educator Profile Manager

IEducatorValidation

- `validateEducator(name:String, email:String): Educator`

Student Profile Manager

IStudentValidation

- `validateStudent(name:String, email:String): boolean`
- `getAllStudents(): List[Student]`

Tournament Manager

IEducatorTournament

- `createTournament(title:String, registrationDeadline:Date, assistants:List[Educator], language:String): boolean`
- `grantPermissions(t:Tournament, assistants:List[Educator]: boolean`
- `closeTournament(t:Tournament): boolean`
- `viewAllTournaments(): List[Tournament]`
- `viewTournamentInfo(tId:int): Tournament`
- `viewYourTournaments(e:Educator): List[Tournament]`

IStudentTournament

- `viewAllTournaments(): List[Tournament]`
- `viewTournamentInfo(tId:int): Tournament`
- `viewYourTournaments(s:Student): List[Tournament]`

Battle Manager

IEducatorBattle

- `createBattle({battleInfo}): boolean`
- `viewBattleInfo(bId:int): Battle`
- `closeBattle(b: Battle): boolean`

IStudentBattle

- `viewBattlesOfTournament(t:Tournament, s:Student): List[Battle]`
- `retrieveBattleInfo(bId: int): Battle`

Notification Service

INotification

- `viewNotifications(u: User): List[Notifications]`
- `sendNotification(e: List[Educator], message:String): boolean`
- `sendNotification(s: List[Student], message:String): boolean`

Email Services

IEmail

- `sendEmail(message:String, to:List[String]): boolean`

Update Manager

IRegistration

- `joinTournament(t:Tournament, s:Student): boolean`
- `joinBattleAsSingleton(b:Battle, s:Student): boolean`
- `joinBattleAsTeam(b:Battle): Team_Creation_Form`
- `joinBattle(b:Battle, t:Team): boolean`
- `acceptInvitation(s: Student, b: Battle): boolean`
- `refusesInvitation(s: Student, b: Battle): boolean`

IScore

- `mergeManualEvaluation(b:Battle, evaluation:List[int]): List[Students]`
- `updateScore(t:Team, score:int): boolean`

ITimeOut

- `notifyEndOfBattle(b:Battle): boolean`
- `notifyDeadlineRegistration(b:Battle): List[Student]`
- `notifyDeadlineRegistration(t:Tournament): List[Student]`

IGetRegistration

- `getYourTournament(s:Student): List[Tournament]`
- `getBattleOfTournament(t:Tournament, s:Student): List[Battle]`
- `getStudentsOfTournament(t:Tournament): List[Student]`

Static Analysis Tool

IStaticAnalysis

- `analyseSource(source:String): int`

Evaluation Service

IManualEvaluation

- `addEvaluations(b:Battle, evaluation:List[int]): boolean`

ISourceCommit

- `notidyLastCommit(source:String): boolean`

GitHub

IGitHub

- `Push({source}): boolean`
- `pullSources({repositoryInfo}): {source, repositoryInfo}`
- `createRepository(): {link_to_repository}`

Educator DBMS

IEducatorsDBMS

- `getEducatorByCredentials(name:String, email:String): Educator`
- `insertEducator(e:Educator): boolean`

Student DBMS

IStudentsDBMS

- `getStudentByCredentials(name:String, email:String): Student`
- `insertStudent(s:Student): boolean`
- `getAllStudents(): List[Student]`

Tournament and Battle DBMS

ITournamentsAndBattlesDBMS

- `getAllTournaments(): List[Tournament]`
- `getTournament(tId:int): Tournament`
- `getTournamentByCreator(e:Educator): List[Tournament]`
- `getTournamentByAssistant(a:Educator): List[Tournament]`
- `addPermissions(t:Tournament, e:Educator): boolean`
- `insertTournament(t:Tournament): boolean`
- `closeTournament(t:Tournament): boolean`
- `getBattleInfo(bId: int): Battle`
- `getAllBattlesOfTournament(t:tournament): List[Battle]`
- `getBattleById(t:Tournament, bId:int): Battle`
- `getBattlesByCreator(t:Tournament, e:Educator): List[Battle]`
- `insertBattleInTournament(t:Tournament, b:Battle): boolean`
- `closeBattle(b:Battle): boolean`

Tournament and Battle Registration DBMS

ITournamentsAndBattlesRegistrationsDBMS

- `getTournamentsByStudent(s:Student): List[Tournament]`
- `getStudentsByTournament(t:Tournament): List[Student]`
- `insertStudentInTournament(t:Tournament, s:Student): boolean`
- `getBattlesByStudent(t:Tournament, s:Student): List[Battle]`
- `getStudentsbyBattle(b:Battle): List[Student]`
- `getStudentsByTeam(t: Team) : List[Student]`
- `getTeamByStudent[s:Student]: Team`
- `insertTeamInBattle(b:Battle, t:Team): boolean`
- `removeStudentFromTeam(s:Student, t:Team): boolean`
- `unregisterStudentFromBattle(s:Student, b:Battle): boolean`
- `updateScoreOfStudent(t:Tournament, s:Student, score:int): boolean`
- `updateScoreOfTeam(b:Battle, t:Team, score:int): boolean`
- `getRankingOfTournament(t:Tournament): List[Student]`
- `getRankingOfBattle(b:Battle): List[Team]`

Notification DBMS

INotificationDBMS

- `insertNotification(message:String, s:List[Student]): boolean`
- `insertNotification(message:String, e:List[Educator]): boolean`
- `getNotifications(s: Student): List[String]`
- `getNotifications(e: Educator): List[String]`

2.6 Selected Architectural Styles and Patterns

2.6.1 Architectural Styles

In the overview section of this document (section 2.1) there is a description of how a microservices architecture has been used to design the system, which not only increases scalability through selected replication, resilience, availability and maintainability of the system, but also offers the possibility to several different teams to work on the different microservices in parallel.

In the overview section was also presented how the REST APIs architecture is used to enable the communication between the client-side and the server-side and between the different server-side components (and therefore microservices). It naturally follows that HTTPS will be employed and the standard http methods will be used to perform CRUD operations (get, post, put, delete).

2.6.2 Security Pattern

As already presented in the deployment view (section 2.3), an important design decision lies in the employment of an API Gateway which is a key aspect for routing and security reasons. The API gateway acts as a mediator and sits between a service client and a service being invoked, which means that the service clients talk only to the gateway which is responsible for mapping the request to the right microservice. One problem with this solution is that the API gateway represents a single point of failure, but this issue can be solved through the replication of the API gateway itself and the employment of a load balancer handing out the incoming requests.

3 | User Interface Design

In this section, which is meant to deepen the "User Interfaces" section of the RASD (section 3.1.1), we are going to provide a complete overview of how the UI the user will interact with has to look like.

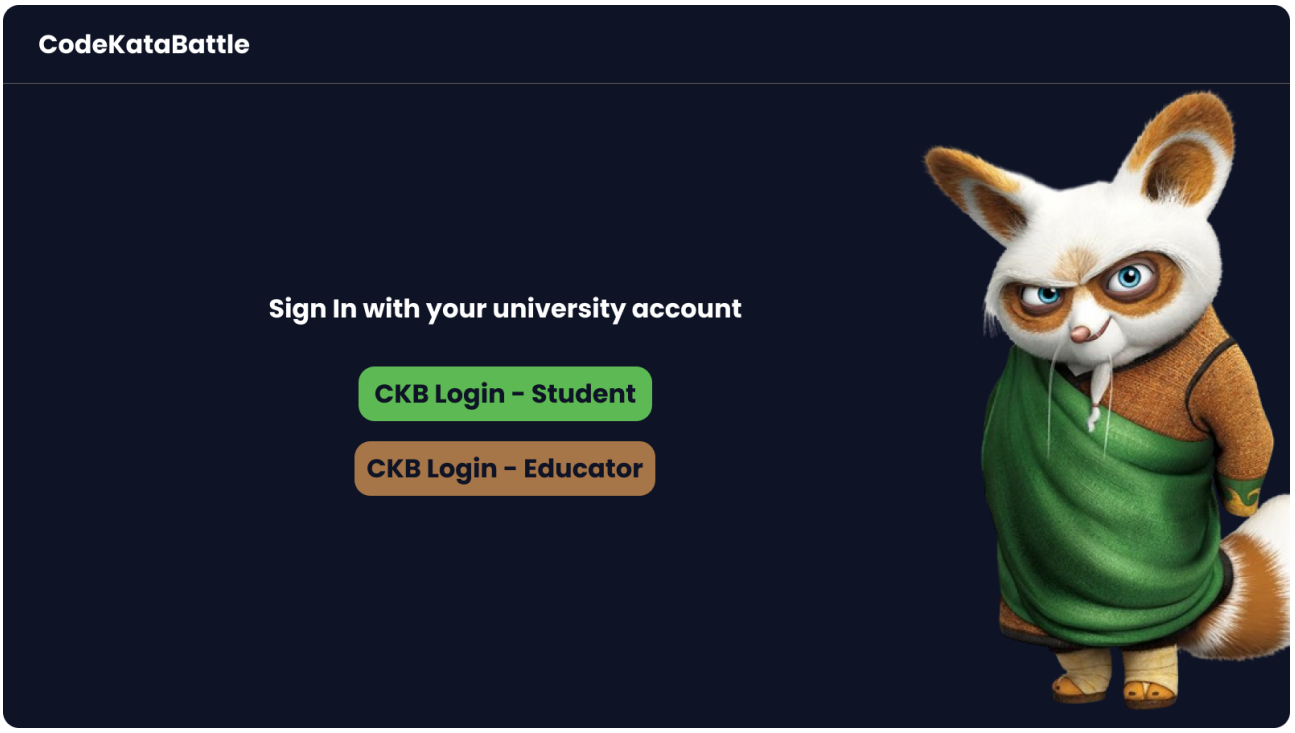

Figure 3.1: CKB login page

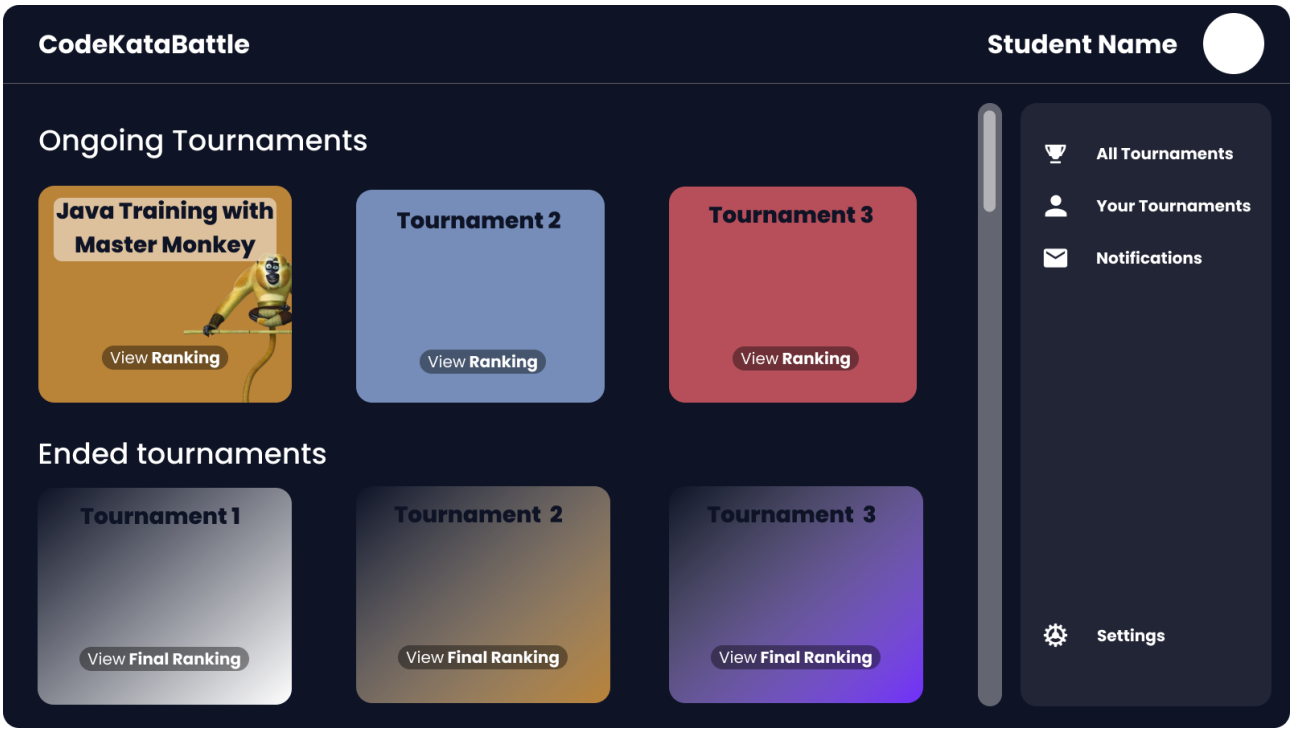

Figure 3.2: Student personal page

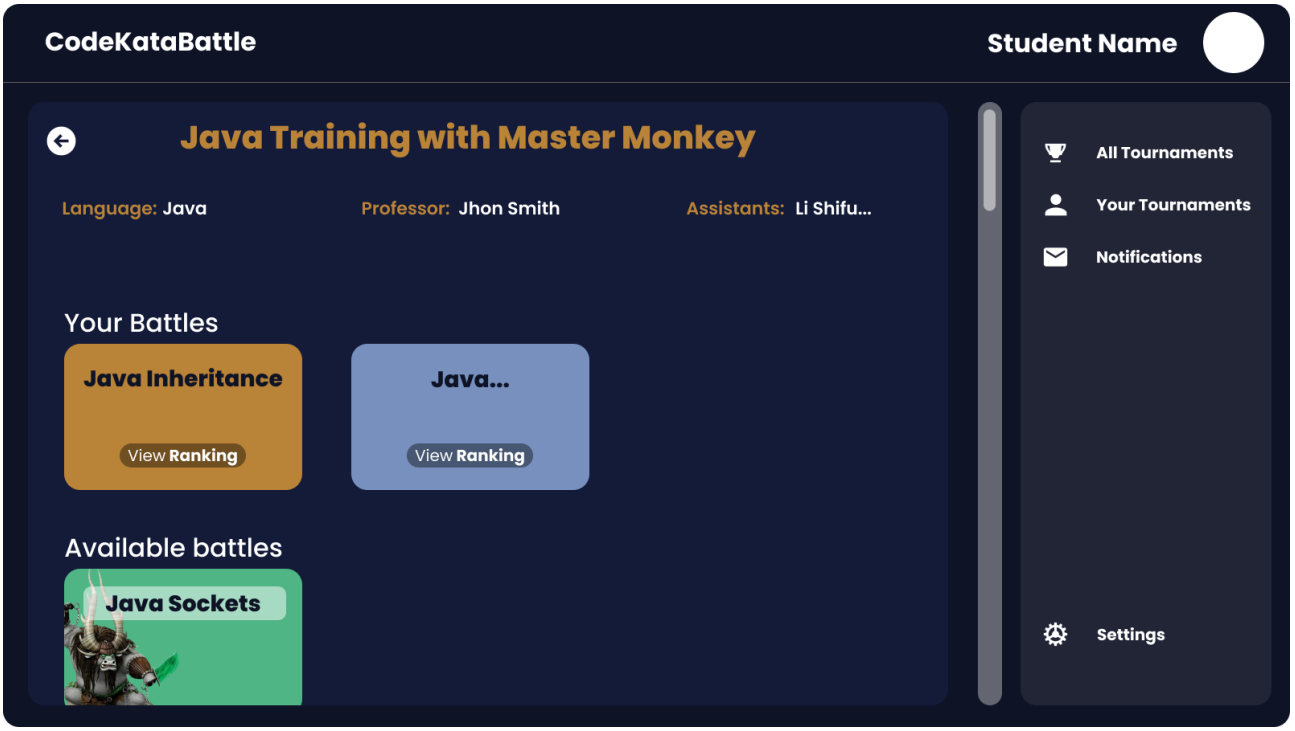

Figure 3.3: Student OnGoing tournament page[1]

Figure 3.4: Student OnGoing tournament page[2]

Figure 3.5: Student battle join page

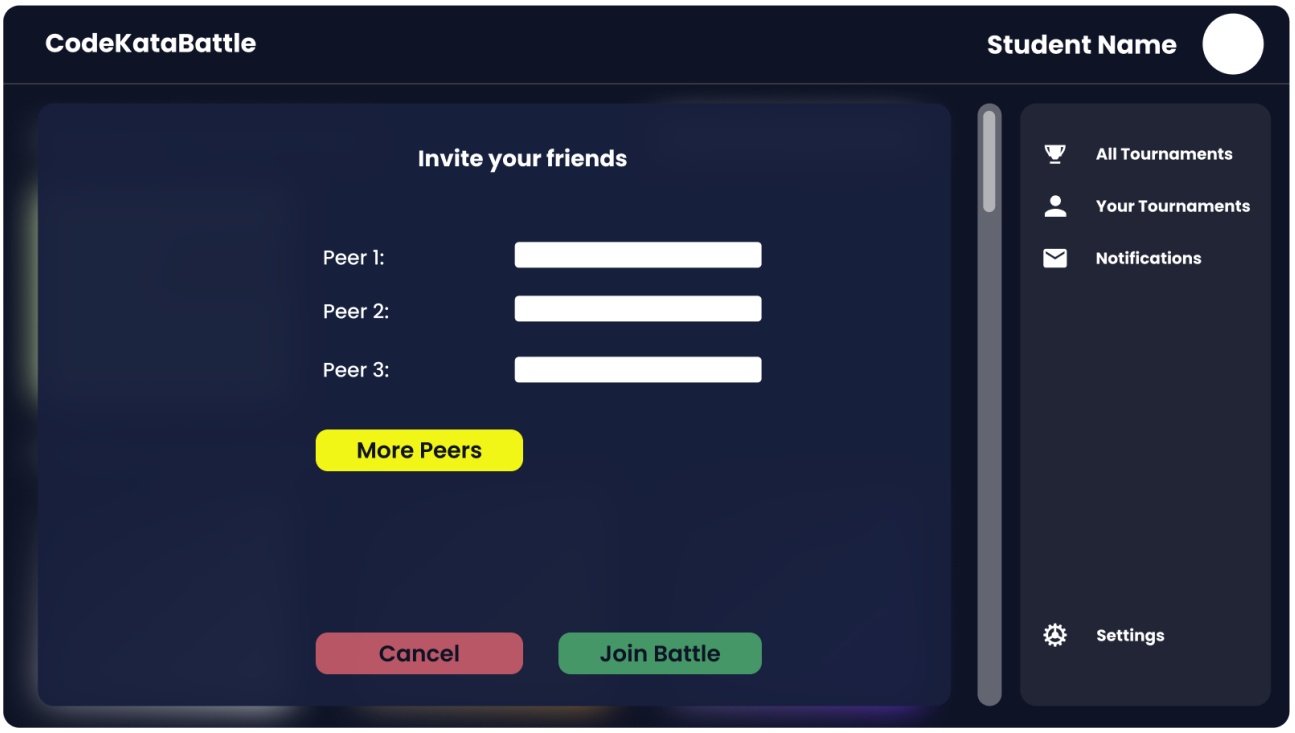

Figure 3.6: Student peer invitation page

Figure 3.7: Student battle final ranking page

Figure 3.8: Student all available tournaments page

Figure 3.9: Student tournament join page

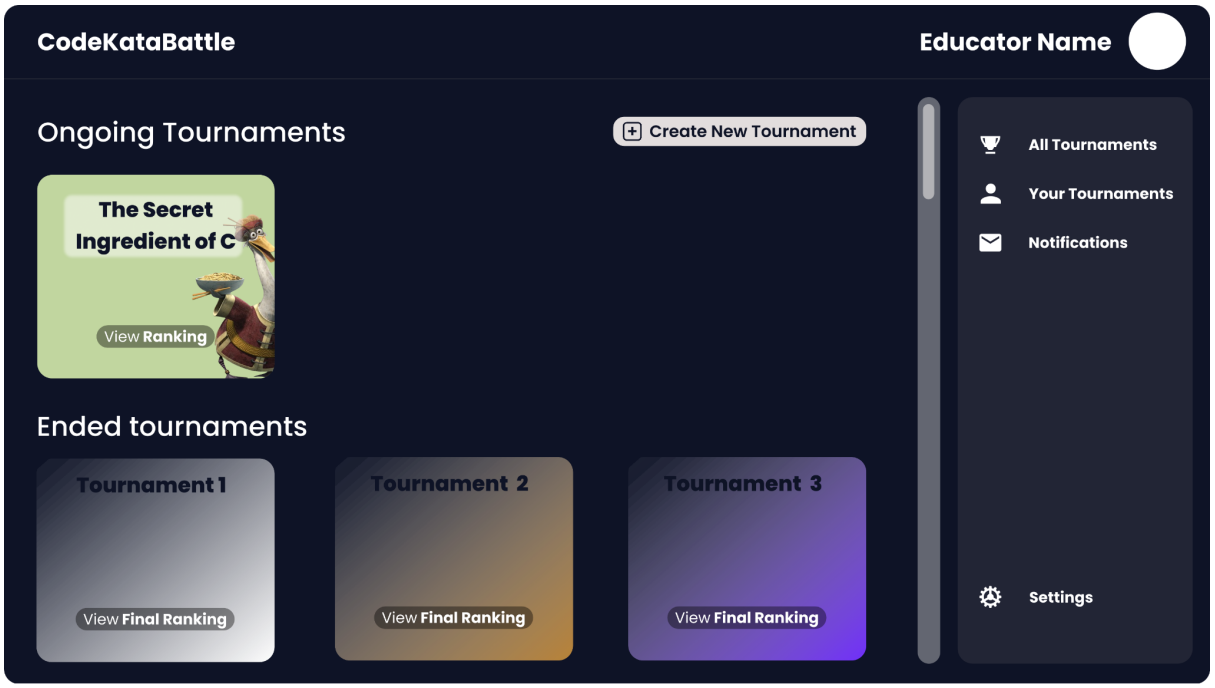

Figure 3.10: Educator personal page

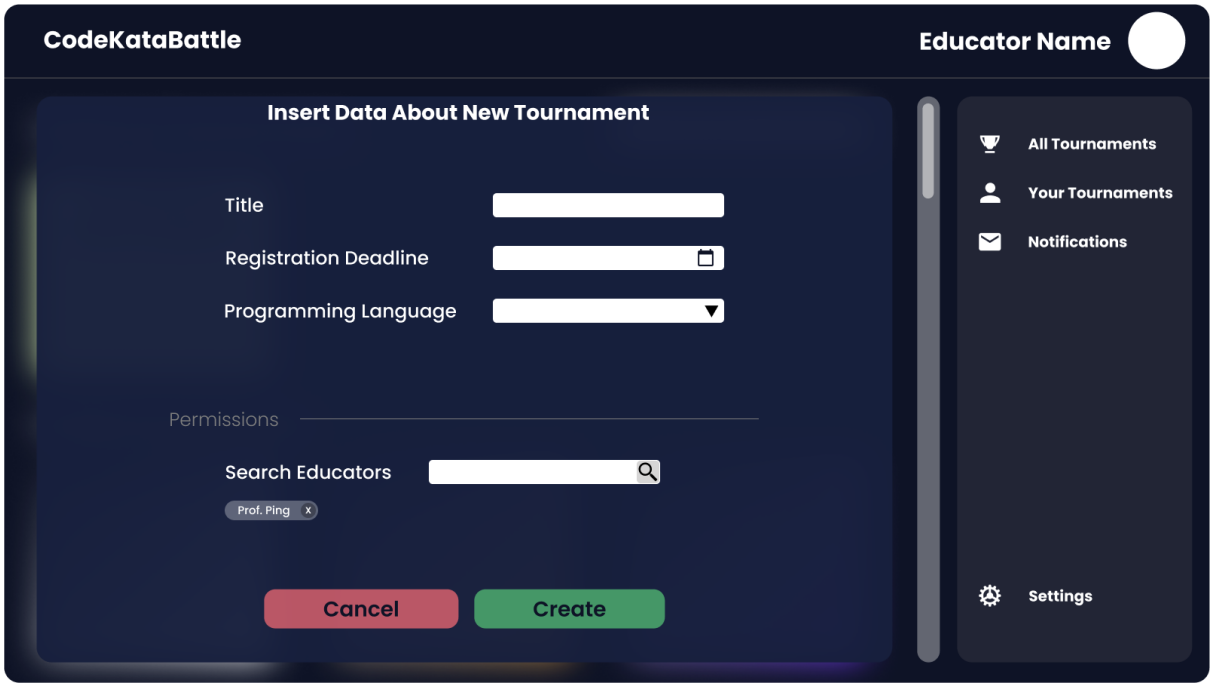

Figure 3.11: Educator create tournament page

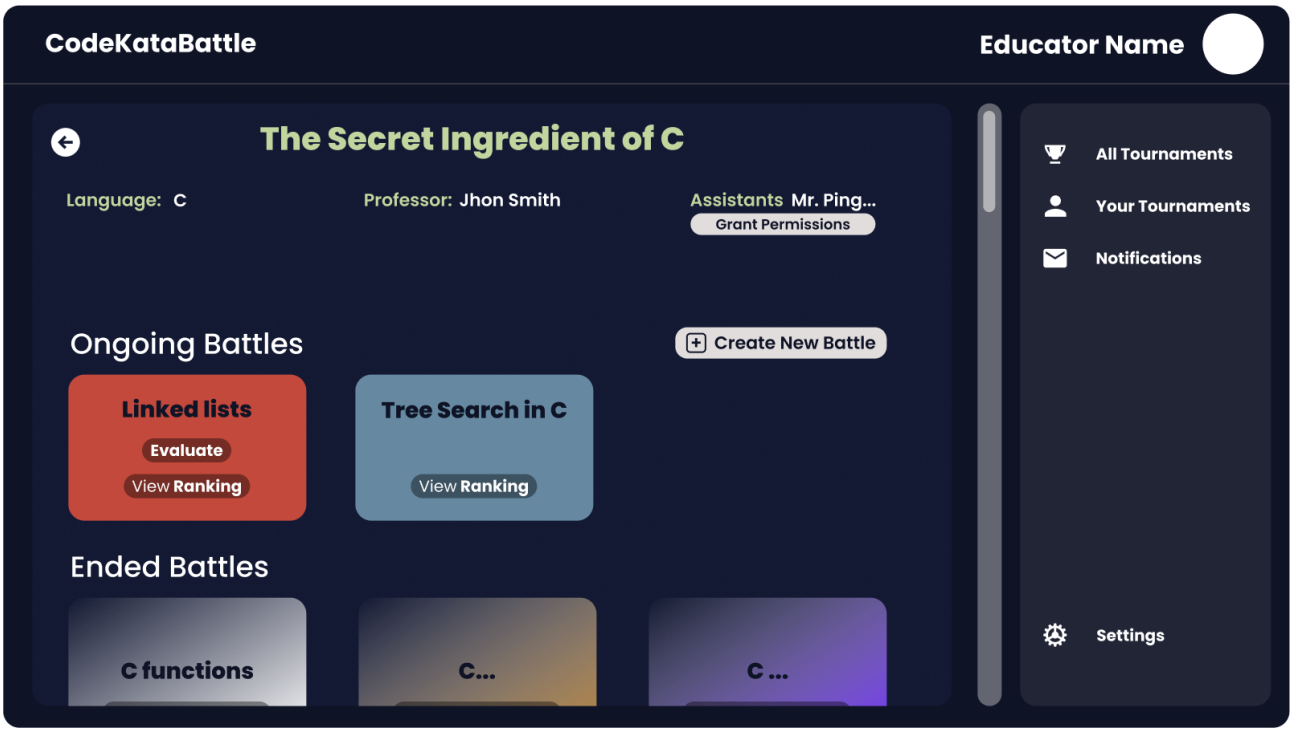

Figure 3.12: Educator OnGoing tournament page

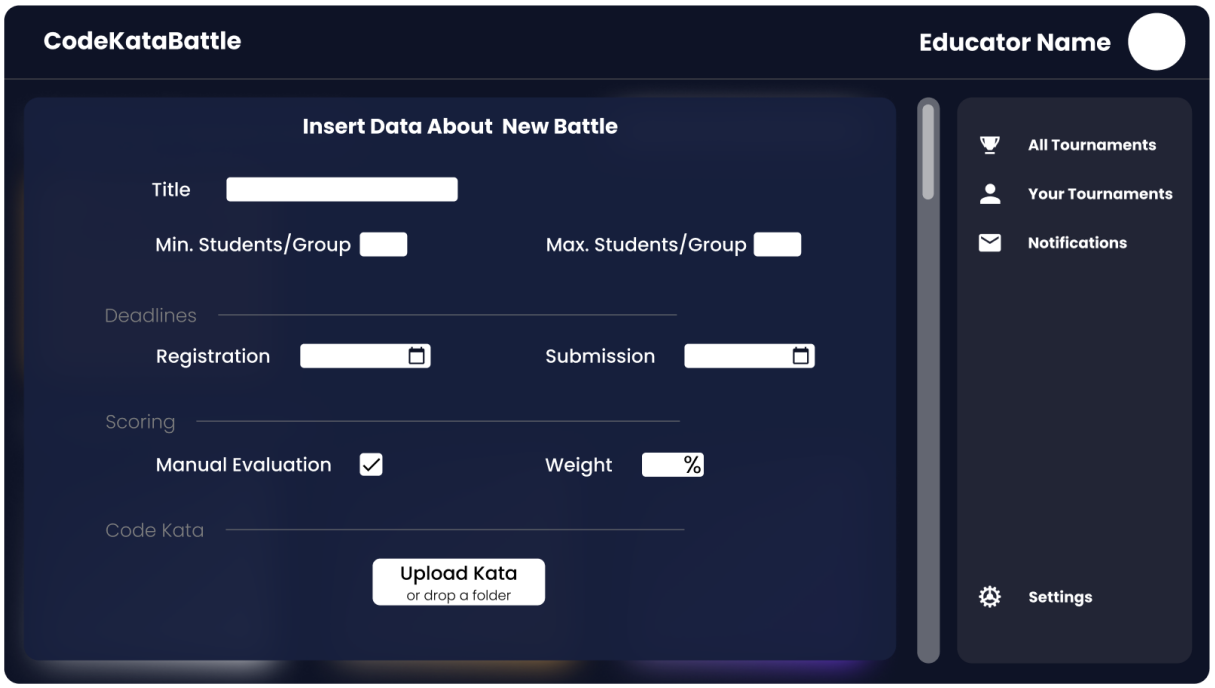

Figure 3.13: Educator create battle page[1]

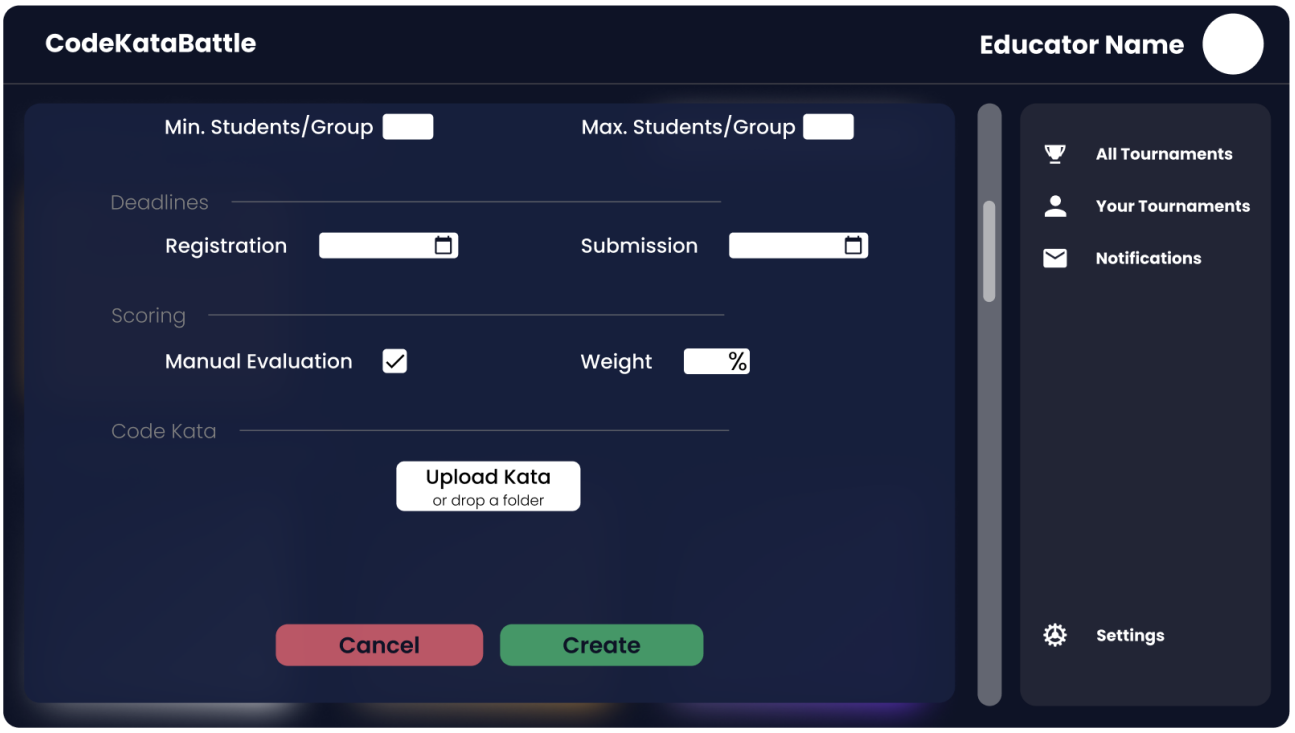

Figure 3.14: Educator create battle page[2]

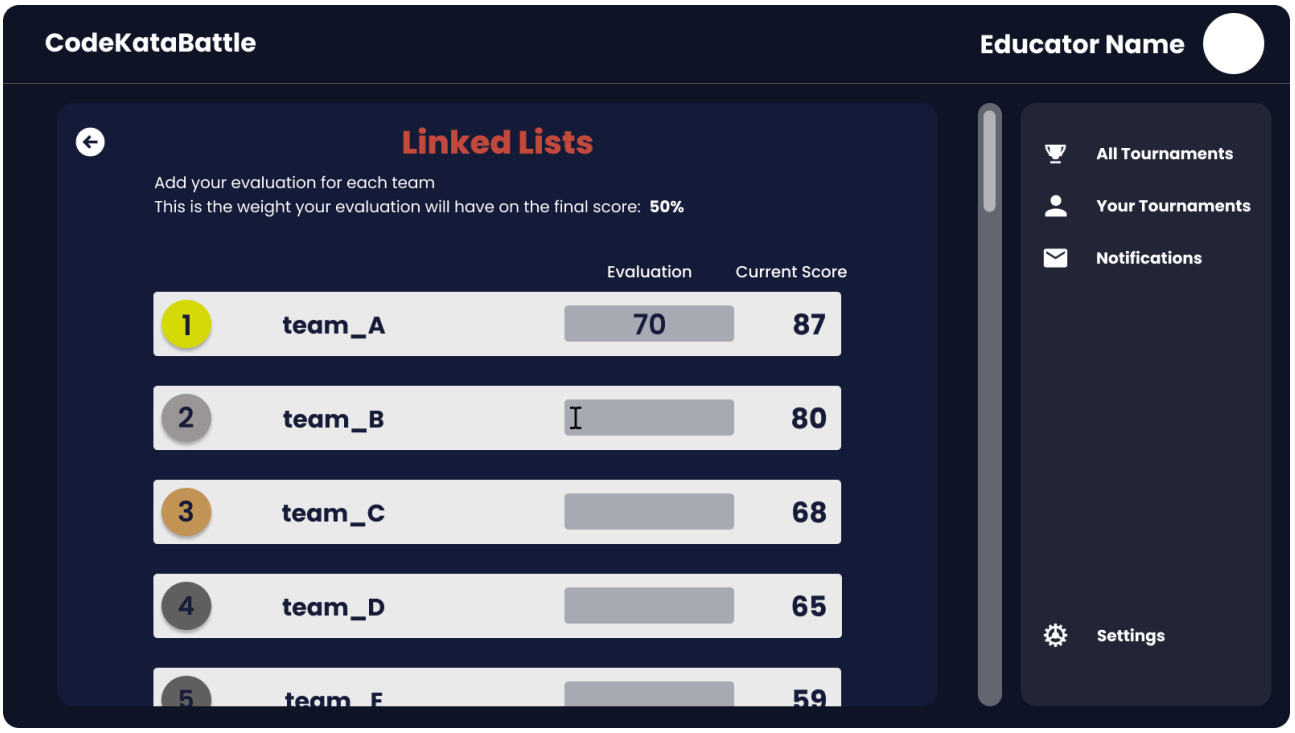

Figure 3.15: Educator manual evaluation page

4 | Requirements Traceability

This chapter explains how the requirements defined in the RASD map to the design elements (components) defined in the section 2.2 of this document focussing on the server side of the components. To enhance readability, all goals and their requirements are listed again, and then their components are mapped.

G1. All students and educators can register and then log in to the CKB platform for, respectively, participating and creating tournaments and battles.

Requirement	Components Mapped
R1: The system allows students to create and access into their personal account using their university credentials (email, password)	• Student Authentication • Student Profile Manager • University Login Services • Student DBMS
R2: The system allows educators to create and access into their personal account using their university credentials (email, password)	• Educator Authentication • Educator Profile Manager • University Login Services • Educator DBMS

Table 4.1: Requirements traceability goal 1

G2. Students can improve their software development skills by joining the coding battles available for each tournament.

Requirement	Components Mapped
R3: When a new tournament is created, the system notifies the students registered on the platform through both a notification message on the platform and an email to their institutional email address	• Tournament Manager • Tournament and Battle DBMS • Email Services
R4: The system allows every registered	• Tournament Manager

student to review information about all the available tournaments on the platform (i.e. whose registration period has not expired yet) and those he/she registered for, such as the title/name, the registration deadline, the name of the professor who created it and the other educators with permissions to create battles	• Tournament and Battle DBMS • Update Manager • Tournament and Battle Registration DBMS
R5: The system allows a student to review information about every battle inside a tournament he/she registered for, such as the title/name, the registration and submission deadlines, the minimum and maximum number of students per group and the name of the educator who created it	• Tournament Manager • Battle Manager • Update Manager • Tournament and Battle Registration DBMS
R6: The system allows every registered student to enroll in a tournament for which the registration period has not expired yet	• Tournament Manager • Update Manager • Tournament and Battle DBMS • Tournament and Battle Registration DBMS
R7: Through both a notification message on the platform and an email to their institutional address, the system notifies all the students registered for a tournament when a new battle is created within that same tournament	• Email Services • Notification Service • Notification DBMS
R8: Until the registration deadline expires, the system allows a student to join as a singleton a battle of a tournament he/she has enrolled in, if it respects the minimum number of students per group set for that battle	• Tournament Manager • Battle Manager • Update Manager • Tournament and Battle Registration DBMS
R9: Until the registration deadline expires the system allows a student to join as a team a battle of a tournament he/she has enrolled in, inviting the other peers at registration time without exceeding the minimum and maximum number of	• Tournament Manager • Battle Manager • Update Manager

students per group set for that battle	• Tournament and Battle Registration DBMS
R10: The system notifies a student, through a platform notification, when he/she has been invited by a peer to join a team for taking part in a battle	• Notification Service • Notification DBMS
R11: The system allows a student registered for a tournament to accept an invitation, in the form of a notification message on the platform, in order to join a battle within that same tournament	• Notification Service • Notification DBMS • Update Manager
R12: As soon as the registration deadline for a battle expires, the system emails each participating student the GitHub repository containing the code kata	• Battle Manager • Update Manager • GitHub • Tournament and Battle Registration DBMS • Email Services
R13: The system notifies a student when a battle's final ranking is available with a message on the platform, if he/she is registered for that battle	• Battle Manager • Update Manager
R14: The system allows a student to view the updated ranking of a battle he/she is taking or has taken part in, even if the battle has finished or the tournament including that battle has been closed	• Tournament Manager • Battle Manager • Update Manager • Tournament and Battle Registration DBMS
R15: The system allows a student to view the updated score, a natural number between 0 and 100, of a battle he/she is taking or has taken part in, even if the battle has finished or the tournament including that battle has been closed	• Tournament Manager • Battle Manager • Update Manager • Tournament and Battle Registration DBMS
R16: Through a platform notification message, the system notifies each student	• Notification Service

enrolled in a tournament when the final ranking of that same tournament is available	• Notification DBMS
R17: The system allows a student to view the updated ranking of a tournament he/she is taking or has taken part in, even if the tournament has been closed	• Tournament Manager • Update Manager • Tournament and Battle Registration DBMS
R18: At the end of each battle, the system updates the score of the tournament containing that same battle and allows a student to view the updated score	• Battle Manager • Update Manager • Evaluation Service • Tournament and Battle Registration DBMS

Table 4.2: Requirements traceability goal 2

G3. Educators can create and manage tournaments, battles within each of them and evaluate the code submitted by the students.

Requirement	Components Mapped
R19: The system allows an educator to create a new tournament, specifying a name, a registration deadline and a programming language.	• Tournament Manager • Tournament and Battle DBMS
R20: The system allows an educator to create a new battle within a tournament for which he/she has the permissions to create one, specifying: • A title/name for the tournament • *Code Kata* • Minimum and maximum number of students per group • Registration deadline • Final submission deadline • Additional configuration for scoring	• Tournament Manager • Battle Manager • Update Manager • Tournament and Battle DBMS
R21: At any point in time from the creation to the closure of a tournament, the system allows an educator to grant permissions to	• Tournament Manager • Tournament and Battle DBMS

create battles in a tournament he/she has created to some of his/her colleagues	
R22: The system notifies an educator when he/she has been granted permissions to create battles within a specific tournament, through a platform notification	• Notification Service • Notification DBMS
R23: The system allows an educator to review information about all the tournaments available on the platform (i.e. whose registration deadline has not expired yet) and those for which he/she has permissions to create battles	• Tournament Manager • Tournament and Battle DBMS • Update Manager • Tournament and Battle Registration DBMS
R24: The system allows an educator to review information about all the battles created within a tournament for which he/she has the necessary permissions (title/name, registration and submission deadlines, minimum and maximum number of students per group and name of the educator who created it)	• Tournament Manager • Tournament and Battle DBMS • Battle Manager • Update Manager • Tournament and Battle Registration DBMS
R25: The system allows an educator to view the ranking of each battle within a tournament for which he/she has the permissions to create a battle	• Tournament Manager • Tournament and Battle DBMS • Battle Manager • Update Manager • Tournament and Battle Registration DBMS
R26: The system allows an educator to evaluate each team's source code assigning it a score	• Evaluation Service • Update Manager • Tournament and Battle Registration DBMS
R27: The system allows an educator to view all tournaments' ranking he/she has created or for which has been given	• Tournament Manager • Update Manager

permissions to create battles	• Tournament and Battle Registration DBMS
R28: The system allows an educator to close a tournament he/she has previously created	• Tournament Manager • Update Manager • Tournament and Battle Registration DBMS • Tournament and Battle DBMS

Table 4.3: Requirements traceability goal 3

5 | Implementation, Integration and Test Plan

This chapter explains how to implement, integrate, and test the system designed in this document. As already stated and described in the previous sections of this document, the architectural style employed for the development of the CKB system is represented by the microservices. Among the other advantages of this design decision (described in other sections like 2.3 and 2.6), a key aspect of this choice is that it allows the system to be developed by several different teams working on different microservices in parallel. To illustrate the test plan and the integration, we will refer to the respective teams as Team A, Team B, Team C, Team D and Team E. Following the just mentioned details, it can be inferred that the employed approach is a parallel bottom-up approach, in which each team focuses on its own microservices and, after their implementation, their integration is tested.

Team A

Team A is assigned the development of the tournament and battle microservice. Consequently, the initial step involves configuring and testing the Tournament and Battle DBMS. After that, the development of the Tournament Manager and the Battle Manager components follows.

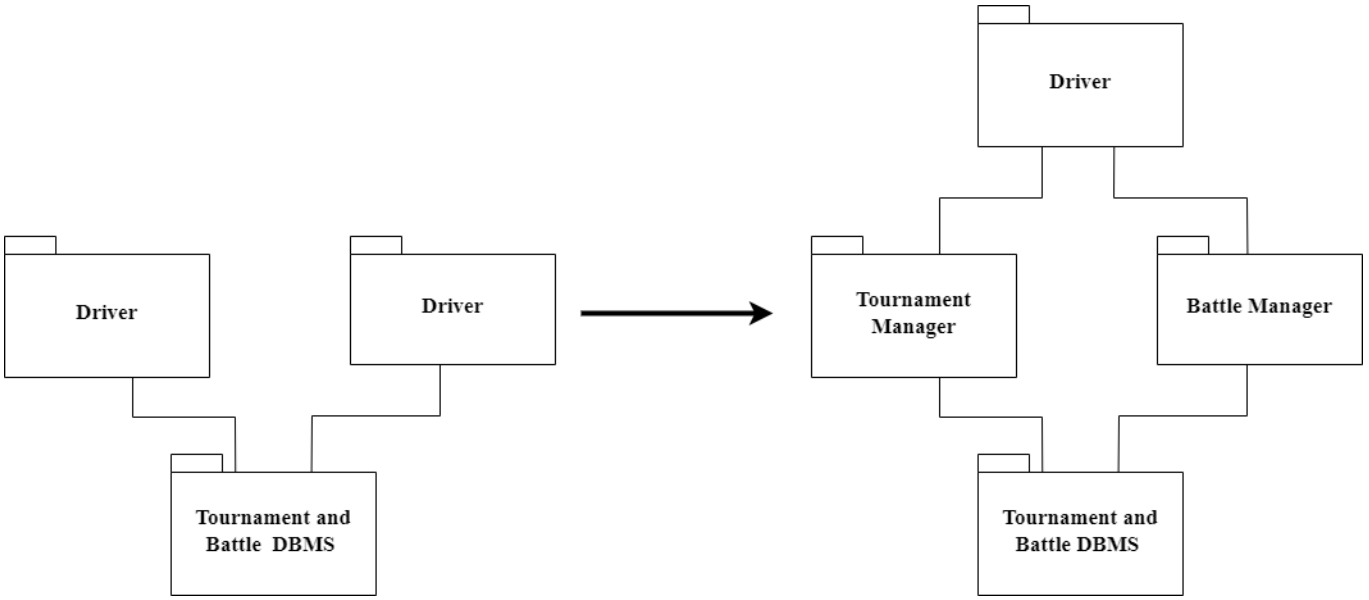

Figure 5.1: Tournament and Battle Microservice Test Plan

At this point unit tests regarding the developed components, such as the view of all tournaments available and the creation of new tournaments and battles, should be produced and it's shown how a driver, representing the client-side component needed, is required to perform them.

Team B

Team B is assigned the development of the tournament and battle registration microservice dealing with all the ongoing tournaments and battles. Consequently, the initial step involves configuring and testing the Tournament and Battle Registration DBMS. After that, the development of the Update Manager and the Evaluation Service follows.

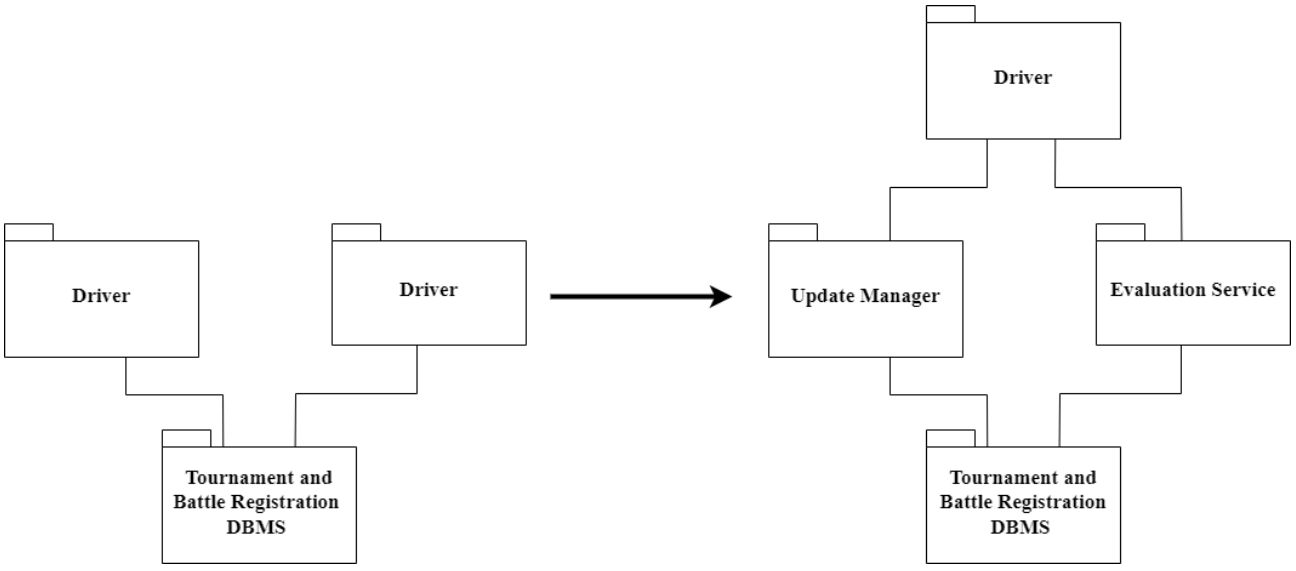

Figure 5.2: Tournament and Battle Registration Microservice Test Plan

At this point unit tests regarding the developed components, such as a simple registration to tournament and battles, should be produced.

Team C

Team C is assigned the development of the notification microservices. Consequently, the initial step involves configuring and testing the Notification DBMS. After that the development of the Notification Service component follows

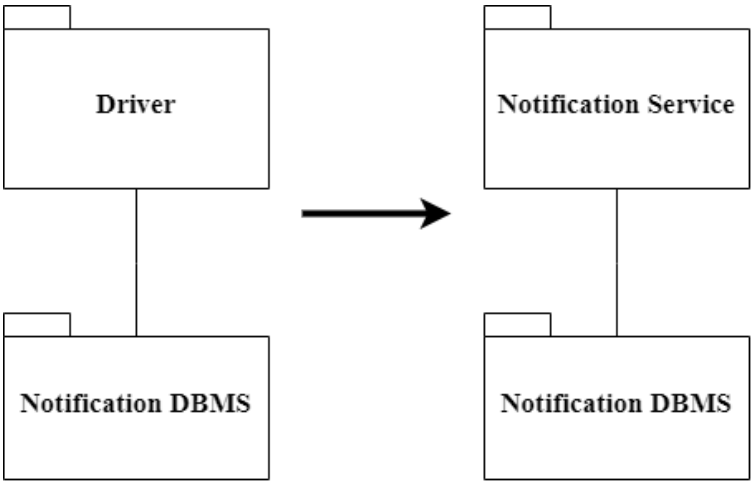

Figure 5.3: Notification Microservice Test Plan

At this point unit tests regarding the developed components should be produced.

Team D, Team E

Team D and Team E are assigned the development of the authentication microservices. Consequently, the initial step involves configuring and testing the Educator DBMS and the Student DBMS. After that the development of the Student Profile Manager and Educator Profile Manager component follows.

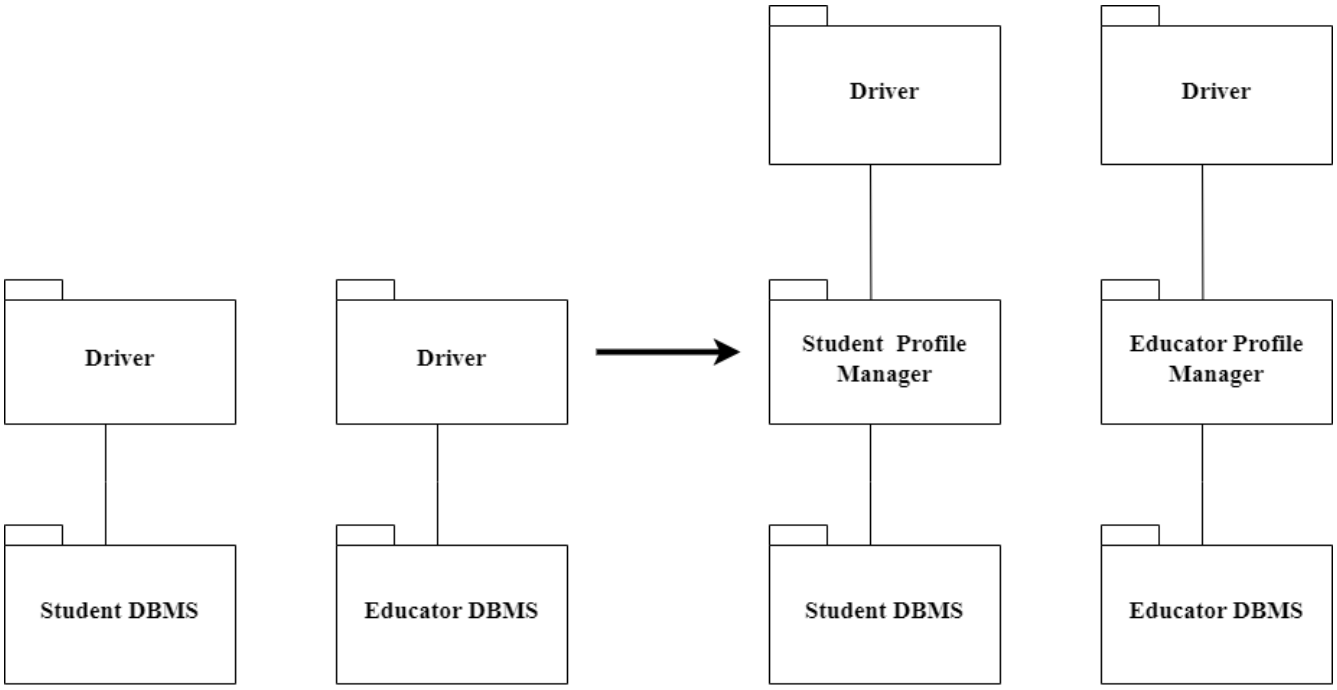

Figure 5.4: Authentication Microservice Test Plan [1]

At this point unit tests regarding the developed components should be produced. To achieve so, two drivers are required, representing the “Student Authentication” and the “Educator Authentication” components which use the exposed interfaces of the just developed components. After that, what are now the drivers can be implemented and integrated as follows.

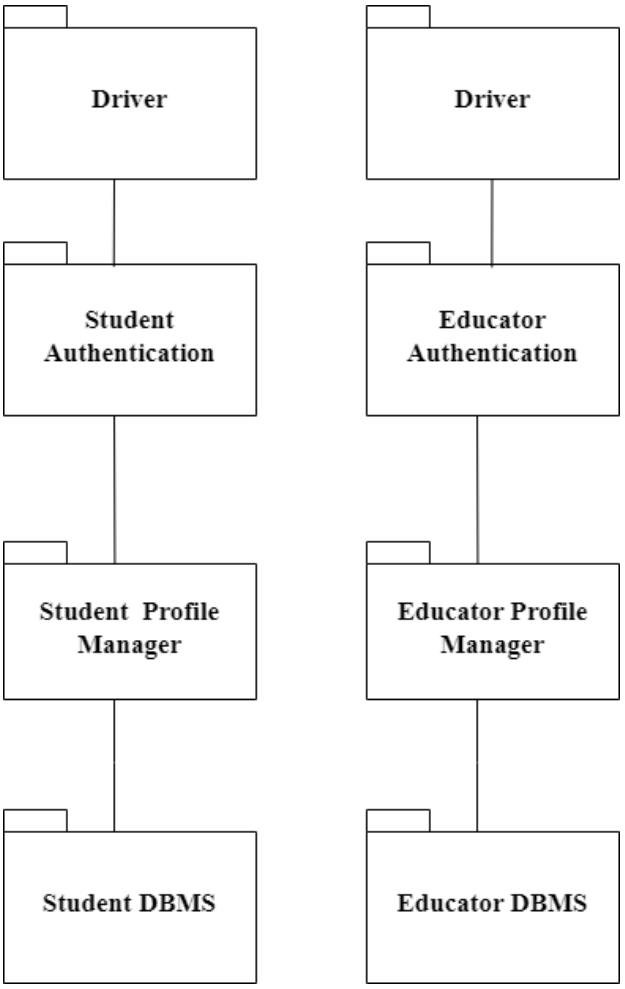

Figure 5.5: Authentication Microservice Test Plan [2]

At this point, after the integration of the newly developed components, the full authentication process can be tested, even relying on the university login services (external reliable component).

Once all the single microservices have been developed and tested, their integration shall follow.

Team C, Team D, Team E

The authentication microservice will be integrated with the notification microservice as follows

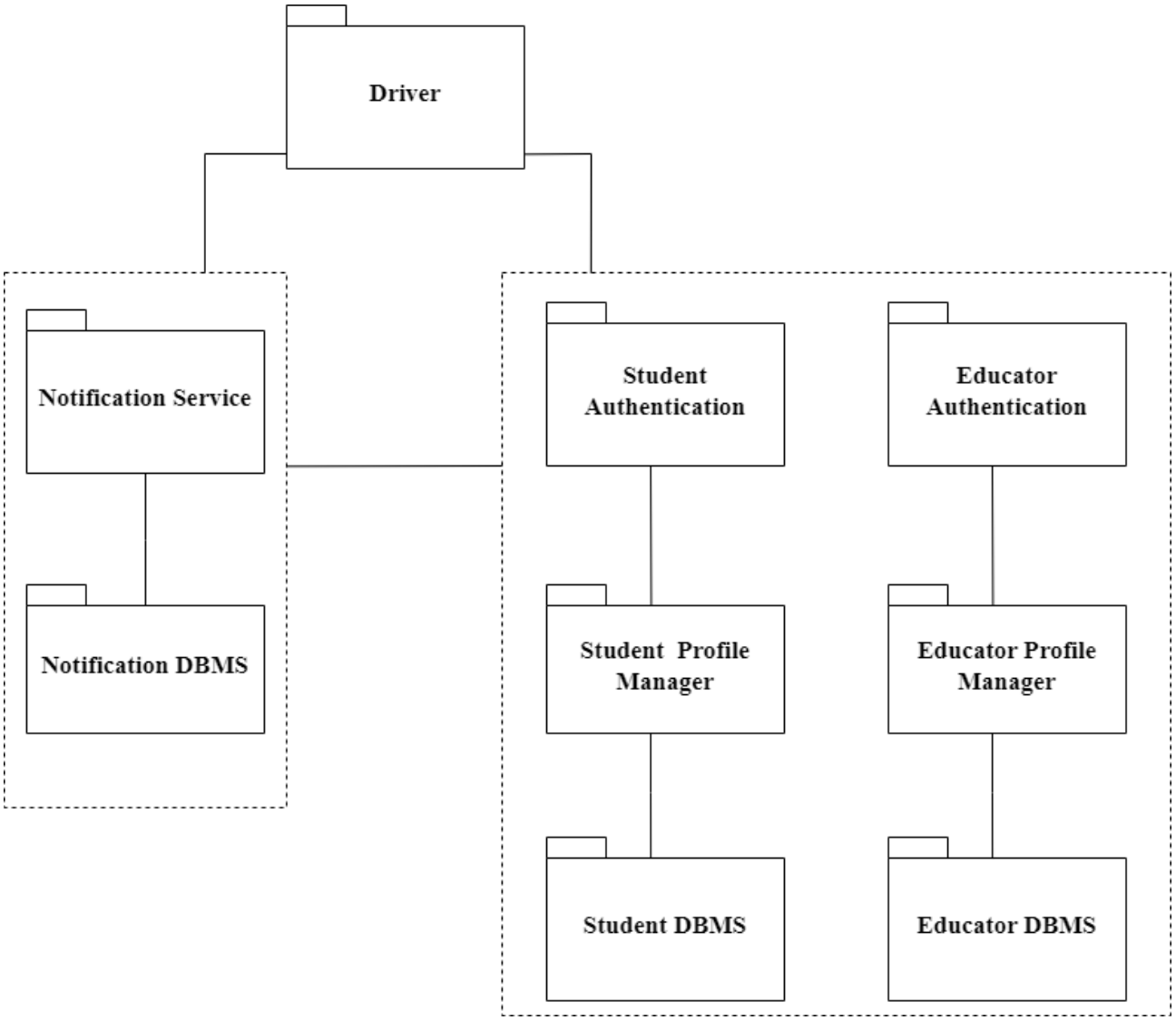

Figure 5.6: Authentication and Notification Microservices Integration Plan

At this point, the whole subsystem should be tested: the entire authentication process and notification retrieval can be tested indeed.

Team A, Team B

The tournament and battle microservice will be integrated with the tournament and battle registration microservice as follows

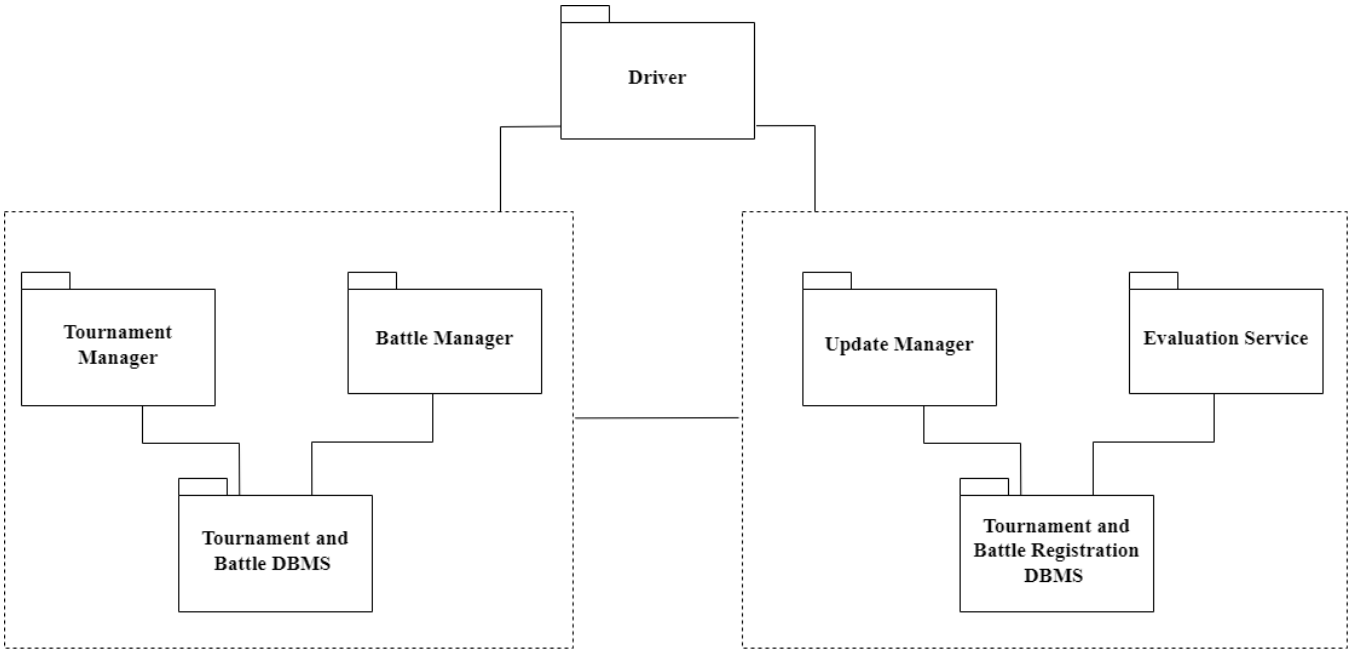

Figure 5.7: Tournament and Battle and Tournament and Battle Registration microservices Integration Plan

At this point the whole subsystem should be tested: the complete processes of a student registering for a tournament and/or a battle can be tested, along with the creation of tournaments and battles by educators and the beginning of a battle, which relies on GitHub (external reliable component).

At this point the two subsystem can be integrated and the CKB system can be tested as a whole

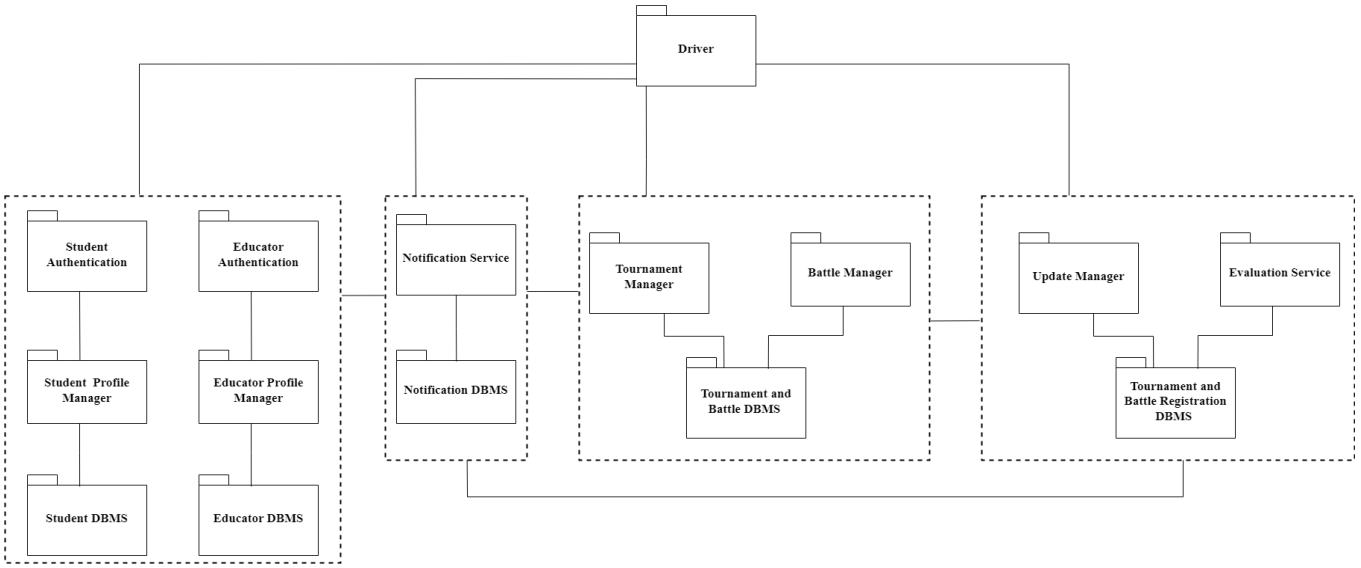

Figure 5.8: All System Integration Plan

6 | Effort Spent

Alessandro Griffanti	Introduction Architectural Design User Interface Design Requirements Traceability Implementation, Integration and Test Plan Reasoning	3*h* 21*h* 1*h* 2*h* 5*h* 9*h*
Luca Masiero	Introduction Architectural Design User Interface Design Requirements Traceability Implementation, Integration and Test Plan Reasoning	1*h* 24*h* 1*h* 1*h* 1*h* 9*h*

Table 6.1: Effort spent

7 | References

7.1 Paper References

- Specification document: “Assignment RDD AY 2023-2024”
- Lecture slides from the Software Engineering 2 course

7.2 Used Tools

- Writing and drafting the document: Google Docs
- All diagrams made with: Draw.io
- Mockups made with: Figma
- Versioning: GitHub
- Reasoning and notes: Notion

List of Figures

List of Tables