POLITECNICO
MILANO 1863

Design Document

Software Engineering 2
CodeKataBattle

Authors
Christian Rossi - 10736464
Kirolos Sharoubim - 10719510

Contents

Chapter 1

Introduction

1.1 Purpose

This document aims to provide a comprehensive insight into the CodeKataBattle system outlined in the RASD. It delves into the system's architecture, elucidating on its components, their interactions, processes, and algorithms designed to meet RASD requirements. Furthermore, it offers explicit instructions pertaining to the implementation, integration, and testing plan. Geared towards developers, testers, and project managers, this document serves as a valuable reference for the system's implementation phase.

1.2 Scope

CodeKataBattle serves as a platform utilized by educators to engage students in coding Katas - challenges designed to be addressed using a programming language specified by the challenge organizers. Educators possess the capability to establish a code Kata within a particular tournament, defining essential parameters such as the problem statement (inclusive of test cases), the minimum and maximum number of students allowed per group, and the deadlines for both code Kata registration and solution submission. Following the tournament creation, students are empowered to create groups and commence their collaborative efforts on the solution, following a test-first approach. As the ultimate deadline approaches, the system autonomously computes the final rankings, ultimately revealing the victorious participants. For a more comprehensive overview of the features accessible to end users, please consult the RASD. The architecture of the S2B is structured into three physically separated layers, each installed on distinct tiers. These layers are:

1. *Presentation layer*: responsible for overseeing the presentation logic and handling all interactions with end users.

2. *Business logic layer*: manages the application functions provided, ensuring seamless operation and functionality.

3. *Data layer*: handles secure storage and facilitates access to data, ensuring the integrity and reliability of information.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

Code Kata: adaptation of the concept of karate katas, where you repetitively refine a form, to the realm of software development, fostering iterative practice and improvement.
Test-first approach: software development process relies on the transformation of software requirements into test cases before the software is completely developed, and it involves monitoring the entire software development by iteratively testing the software against all these test cases.

1.3.2 Acronyms

CKB: CodeKataBattle
CK: Code Kata

1.4 Revision history

Version 1.0 - Release - TBD

1.5 Reference documents

Document 1 - Presentation about DD structure
Website 1 - http://codekata.com
Website 2 - https://en.wikipedia.org

1.6 Document structure

This document comprises seven sections:

1. *Introduction*: this section furnishes an overview of the Design Document (DD), encompassing the project's scope, key term definitions, references to pertinent documents, and a brief outline of the design.

2. *User interface design*: outlining the design of the user interface (UI) along with user experience (UX) flowcharts, this section provides a detailed perspective on how users will interact with the system.

3. *Architectural design*: describing the high-level components and interactions within the system, this section includes a component view, deployment view, runtime view, and insights into selected architectural styles and patterns.

4. *Requirements traceability*: establishing a clear connection between the requirements specified in the Requirements and Specification Document (RASD) and the components outlined in the DD, this section ensures traceability throughout the design process.

5. *Implementation, integration, and test plan*: this section details the plan for implementing, integrating, and testing the system. It includes the sequence in which subsystems and components will be implemented, providing a roadmap for the development process.

6. *Effort spent*: offering insights into the effort invested in the design process, this section provides information on the resources and time dedicated to the various aspects of the design.

7. *References*: this section encompasses a list of references cited within the Design Document, providing a foundation for further exploration and understanding.

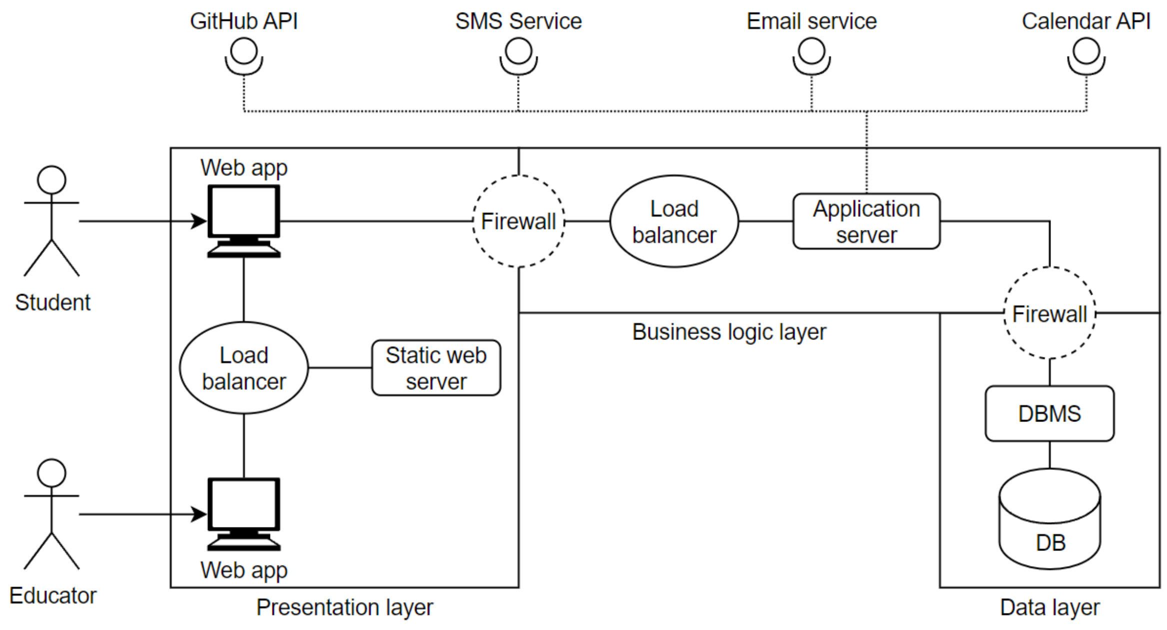
Chapter 2

Architectural design

The purpose of this section is to systematically present and assess the architecture of the S2B system in a top-down approach. We first introduce the comprehensive architecture and then provide a diagram illustrating the system's components, with a specific emphasis on the Educators and students subcomponents. Subsequently, we employ an Entity-Relationship (ER) diagram to articulate the system's logical data and present the system's deployment view, encompassing layers and tiers. We also use sequence diagrams to illustrate crucial runtime perspectives, while class diagrams are employed to scrutinize the interfaces of the components. Ultimately, we engage in a discussion on the architectural design choices and the rationale behind them.

2.1 Overview

The diagram below provides a high-level depiction of the components that form the System. In this document, we will use the term "Frontend" to encompass both the presentation layer and the Client (e.g., the Browser), while the term "Backend" will encompass both the Application Layer and the Data Layer.



The service will be accessed through a web interface, utilizing a single page application

(SPA) that proves advantageous for this application type, enabling extensive interaction without frequent page reloads, thereby ensuring a faster and smoother user experience. The system's architecture is structured three into distinct layers, with application servers interacting with a database management system and utilizing APIs for data retrieval and storage. Adhering to REST standards, the application servers are designed to be stateless, and the system incorporates firewalls to bolster security.

2.2 Component view

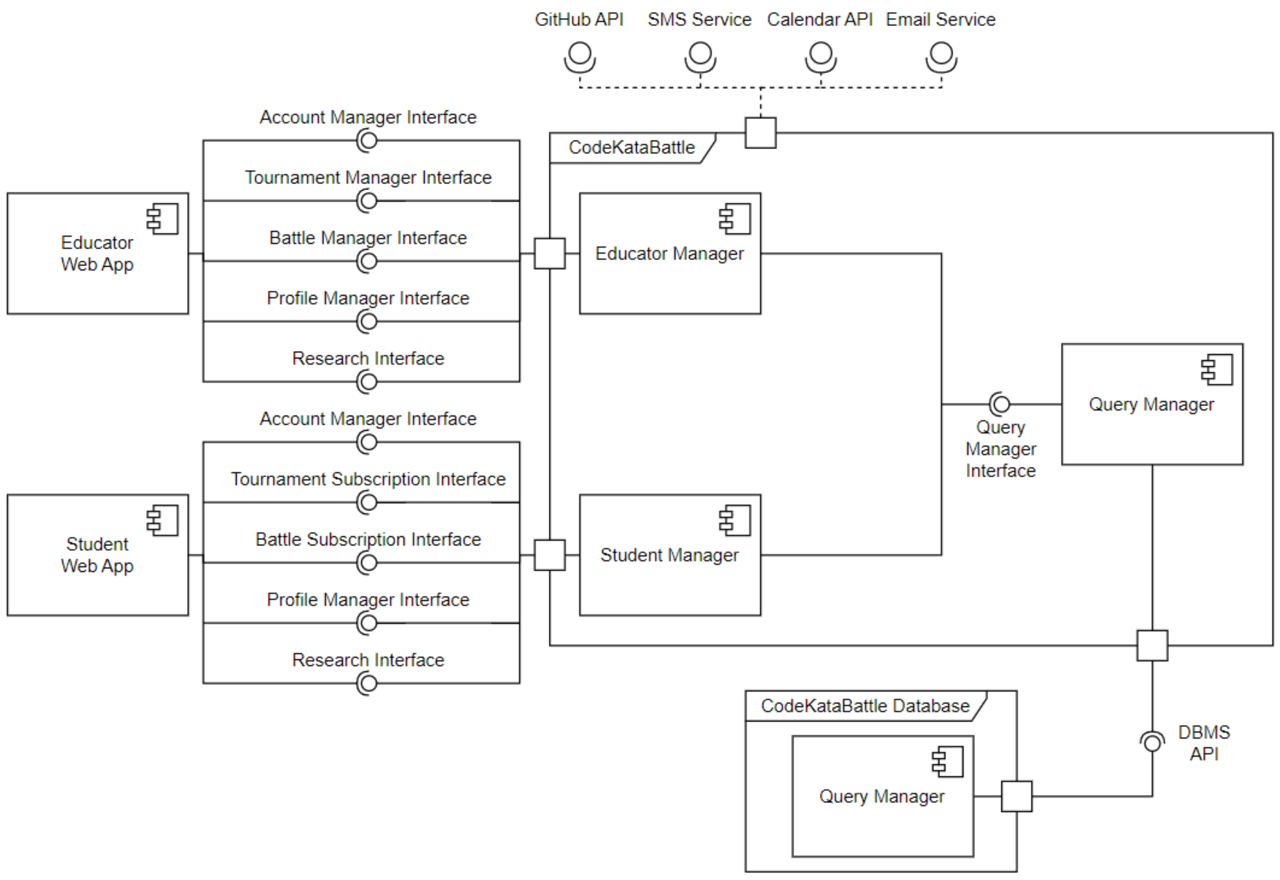


Figure 2.1: Component diagram of the CodeKataBattle system

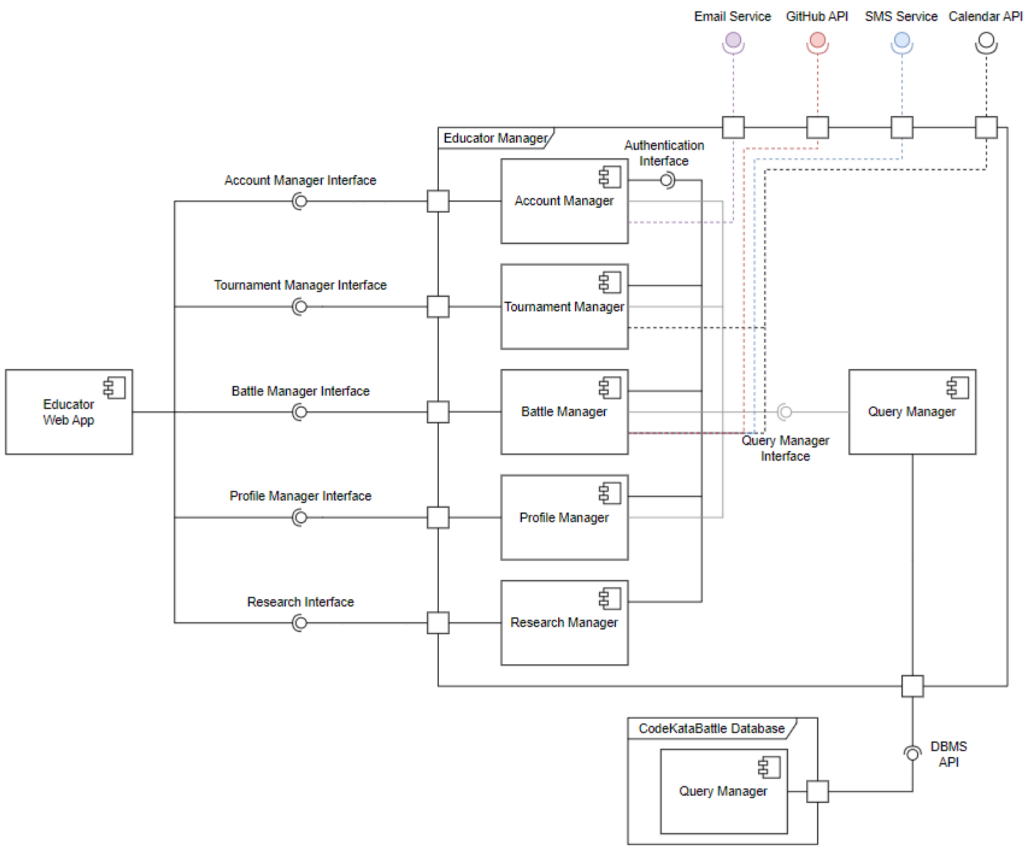
The component diagram contains all the interfaces used by the educators and the students. These components will be better explained and analyzed in the following pages. Both components have an interface with the database, that deals with the data of all the system. The system has also access to some external API that are inserted at the top of the diagram.

2.2.1 Query manager

This element is tasked with interfacing with a Database Management System (DBMS). It adheres to the Adapter design pattern, enabling seamless interaction between other components and the DBMS without necessitating the manual writing of SQL code.

2.2.2 Educator manager

The educator manager component is responsible for interacting with educators. It allows the educator to log in, register, manage the profile, search for a tournament, create a tournament, create a battle, check the leaderboards, manage a tournament and give points to the students.



The educator manager component is expandend in five component as follows:

- *Account Manager*: this component allows the educators to log in and register to the system. It has an authentication interfaces that allows all other components to check if the educator is logged in and is allowd to do certain actions. The account manager is also linked to the database through the query manager interface, that allows it to manage data in the tables dedicated to the educator. The only external API used by it is the email service that is utilized to verify the email whenever a new eduator tries to register.

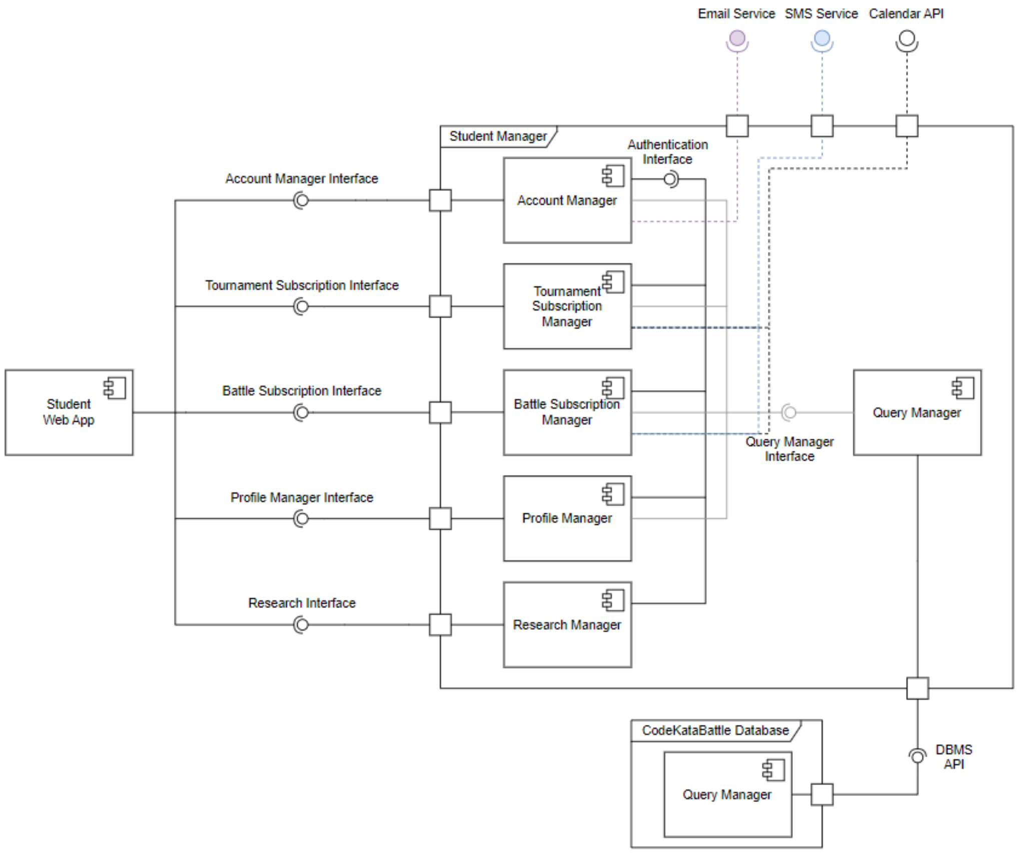
- *Tournament Manager*: this component allows the educators to manage a tournament with the following actions: create a tournament, add a collaborator to a tournament, and close a tournament. It has a connection to the authentication manager since it needs to verify the user that is tring to modify the tournament. The tournament manager is also linked to the database through the query manager interface, that allows it to gather and add data to the tables dedicated to the tournament. The only external API used by it is the calendar API that is utilized to check the current date and the closure date of the tournament.

- *Battle Manager*: this component allows the educators to manage a battle with the following actions: create a battle in a managed tournament, give personal evaluation (that modifies the gruop point), and close a battle. It has a connection to the authentication manager since it needs to verify the user that is tring to modify the battle. The battle manager is also linked to the database through the query manager interface, that allows it to gather and add data to the tables dedicated to the battle. The external APIs used by this component are: GitHub API to create the repository linked to the battle, SMS service to notify the user on the event of the newly created battle, and calendar API to check the main deadlines and notify the educators that manage the battle.

- *Profile Manager*: this component allows the educators to check the personal profile with personal information and tournaments information. It has an authentication interfaces that allows all other components to check if the educator is logged in and is allowed to check the desired profile. The profile manager is also linked to the database through the query manager interface, that allows it to manage data in the tables dedicated to the educator's profile.
- *Research Manager*: this component allows the educators to search all the active tournament in the system and to check all the leaderboards and rankings. It has an authentication interfaces that allows all other components to check if the educator is logged in. The research manager is also linked to the database through the query manager interface, that allows it to manage data in the tables dedicated to the tournaments.

2.2.3 Student manager

The student manager component is responsible for interacting with students. It allows the student to log in, register, manage the profile, search for a tournament, join a tournament, join a battle, and check the leaderboards.



The student manager component is expandend in five component as follows:

- *Account Manager*: this component allows the students to log in and register to the system. It has an authentication interfaces that allows all other components to check if the student is logged in and is allowed to do certain actions. The account manager is also linked to the database through the query manager interface, that allows it to manage data in the tables dedicated to the student. The only external API used by it is the email service that is utilized to verify the email whenever a new student tries to register.
- *Tournament Subscription Manager*: this component allows the students to enroll in a tournament with the following actions. It has a connection to the authentication manager since it needs to verify the user that is tring to enroill in a tournament. The tournament subscription manager is also linked to the database through the query manager interface, that allows to register the student to the tournament. The external APIs used by this component are: SMS service to notify the user on the event of the tournament, and calendar API to check the main deadlines and notify the students that are enrolled in the tournament.
- *Battle Subscription Manager*: this component allows the students to enroll in a battle within a tournament where they are already subscribed. It has a connection to the authentication manager since it needs to verify the user that is tring to enroll in the battle. The battle subscription manager is also linked to the database through the query manager interface, that allows it to gather and add data to the tables dedicated to the battle. The external APIs used by this component are the SMS service to notify the user on the event of the newly created battle, and the calendar API to check the main deadlines and notify the students that are enrolled in the battle.
- *Profile Manager*: this component allows the students to check the personal profile with personal information and enrolled tournaments information. It has an authentication interfaces that allows all other components to check if the student is logged in and is allowed to check the desired profile. The profile manager is also linked to the database through the query manager interface, that allows it to manage data in the tables dedicated to the student's profile.
- *Research Manager*: this component allows the students to search all the active tournament in the system and to check all the leaderboards and rankings. It has an authentication interfaces that allows all other components to check if the student is logged in. The research manager is also linked to the database through the query manager interface, that allows it to manage data in the tables dedicated to the tournaments.

2.2.4 Database schema

In this section there is the E-R schema used for the database of the system.

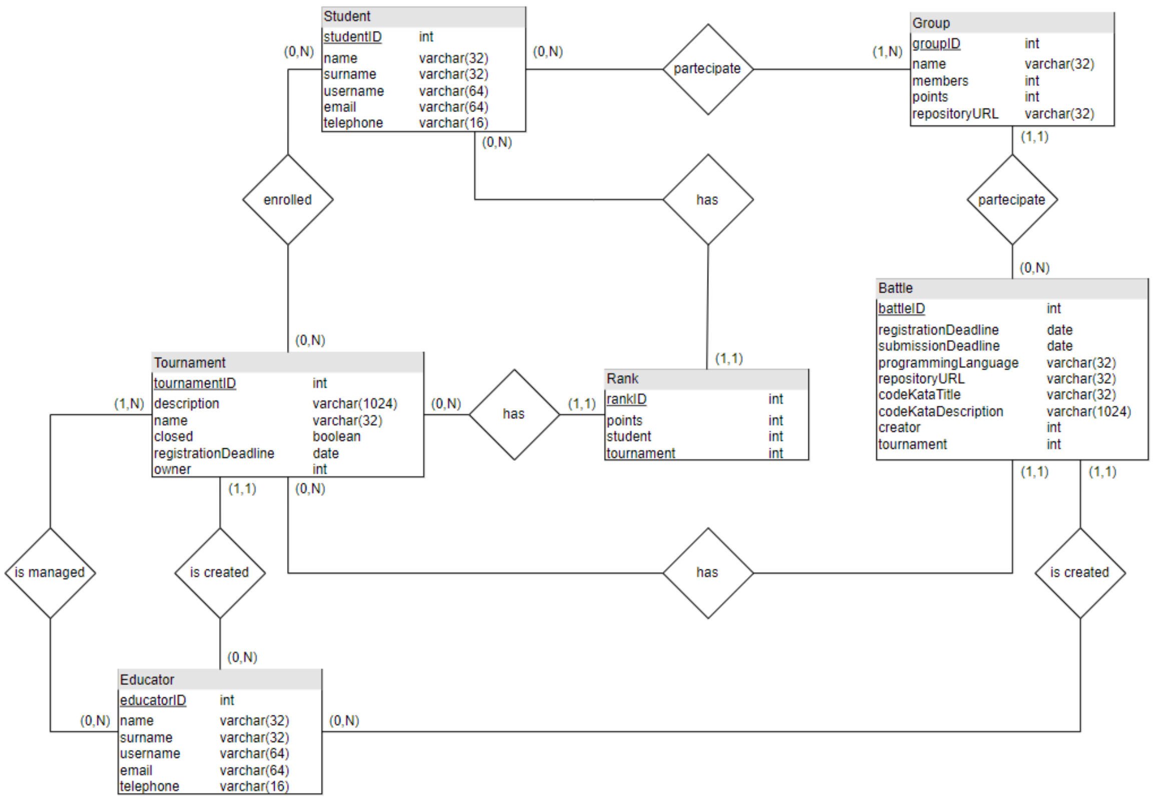


Figure 2.2: Detailed component diagram for the Student

Students and educators are stored in different tables to have better performance when searching the user in the database. The attributes of both tables are the same, and comprises: name, surname, username, email, and telephone. Each student and educator has a unique numerical ID (it is possible to use also the username since we have the uniqueness constraint on it).

Group is a table that contains all the groups linked to all the existing battles. It has the following attributes: name, members (the number of memmbers in the considered group), points (of the group in a battle), and repositoryURL (a string with the URL of the group repository for a battle). The ID is unique over all possible groups in all possible battles.

Battle represents a single battle with the description of the problem associated with it. The attributes are: registrationDeadline, submissionDeadline, programmingLanguage, repositoryURL (of the general repository to be forked), codeKataTitle, CodeKataDescription, creator (that is the ID of the educator that created the battle), and tournament (that is the ID of the tournament where the battle were created). The ID is unique in the whole system.

Tournament represents the list of tournament that have ended (closed set to one) or are running (closed set to zero). Other attributes are: description, name, registrationDeadline, and owner (that is the ID of the educator that created the tournament initially). Also in this case we have a unique ID as key.

Rank contains the point associated to a student (ID of the student) for a certain tournament where he is enrolled (tournament is the ID of the linked tournament). Each rank has a single ID.

2.3 Deployment view

Our setup comprises two essential components: a static web server serving as the entry point for clients to access the Single Page Application (SPA), and an application server providing the requisite APIs for the SPA's functionality. To optimize performance and leverage distinct advantages, we've adopted distinct solutions for each part. The static web server is deployed on a Content Delivery Network (CDN) to ensure rapid response times. This is achieved through the utilization of edge location caches and reverse proxies. On the other hand, the application server, encompassing both a business logic layer and a data tier, is hosted on a cloud provider. This choice offers several benefits over conventional in-house hosting:

- *Scalability and flexibility*: the cloud infrastructure allows us to dynamically adjust resources such as virtual machines, performance cores, or memory based on demand. Incorporating load balancing services enables the application server to seamlessly adapt to fluctuations in traffic or workload.
- *Security*: leveraging services like live monitoring and firewalls enhances the security posture of the application server, safeguarding it against data breaches, cyberattacks, and other security threats.
- *Cost-efficiency*: the cloud provider's pay-as-you-go model ensures cost optimization by billing only for the actual resources consumed. This approach proves economical and contributes to lowering overall operational expenses.

These features collectively position a cloud provider as an optimal choice for hosting large, high-traffic applications. It is imperative that the chosen cloud provider offers comprehensive support for scalability, security, and cost-efficiency to align with our specific requirements.

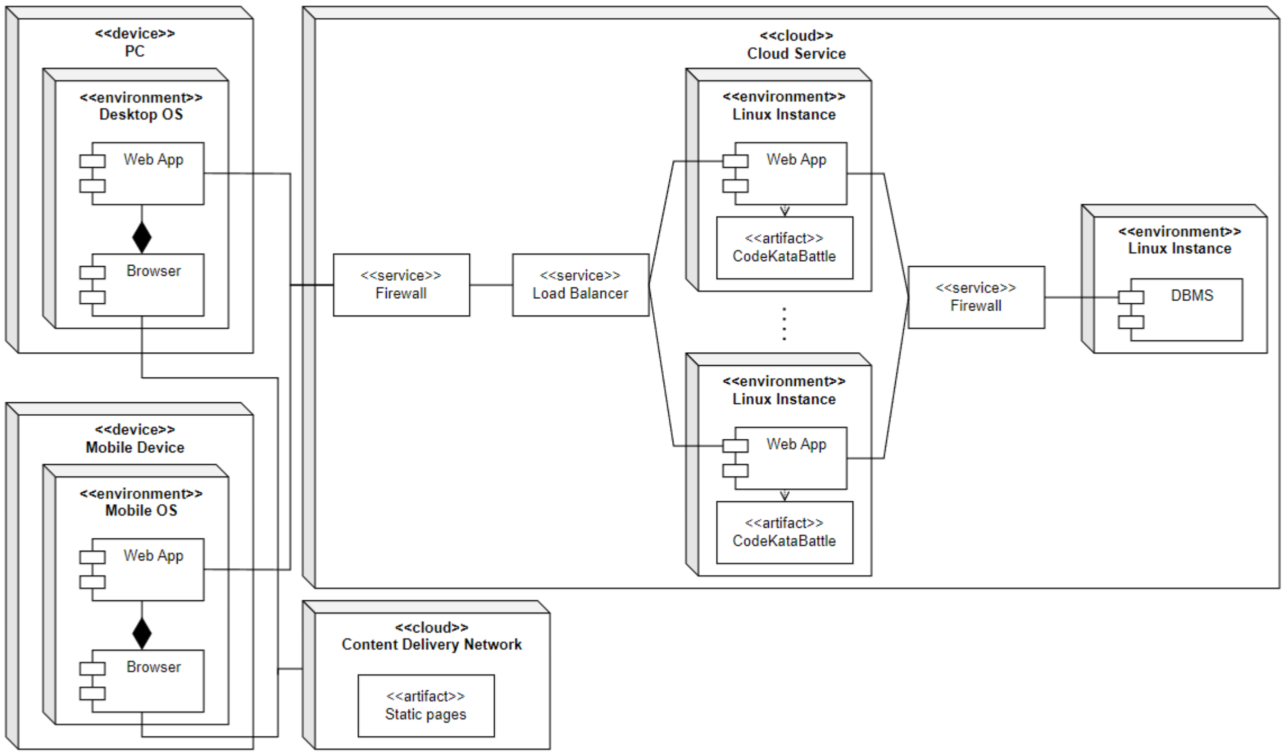


Figure 2.3: Deployment views

The devices called PC and Mobile are any device with a moder browser capable of running JavaScript.

The static pages are handled by the Content Delivery Network to avoid affecting the performance of the main application. This part handles the welcome page and login/register pages. These pages are static in the sense that they have no need to have code that runs on server side.

Finally, the cloud service will host all the business and data logic. This service contains the following elements:

- *Firewall services*: employed to filter incoming connections to both the business and data layers of a system, firewall services utilize a predetermined set of rules selected based on specific criteria. This component serves as a crucial defense mechanism against malicious attacks, offering generic filtering capabilities.
- *Load balancer*: tasked with distributing incoming traffic among multiple instances of an application, the load balancer serves to optimize resource utilization, enhance performance, and ensure high availability. Its role is critical in preventing application overload or downtime, and it contributes to a stable and reliable user experience by redirecting traffic to the least busy application instance.
- *Multiple application instances*: operating in parallel and independently, multiple copies of the application can be dynamically created or deleted to meet demand. This approach enables the application to efficiently handle a high volume of requests without encountering performance issues. Additionally, it provides fault tolerance by redirecting traffic to alternative instances in case one becomes unavailable.
- *Data instance*: this component involves a data-optimized virtual machine housing the Database Management System (DBMS) and the associated database. It plays a pivotal role in managing and organizing data efficiently within the system.

2.4 Runtime view

2.4.1 Student's runtime view

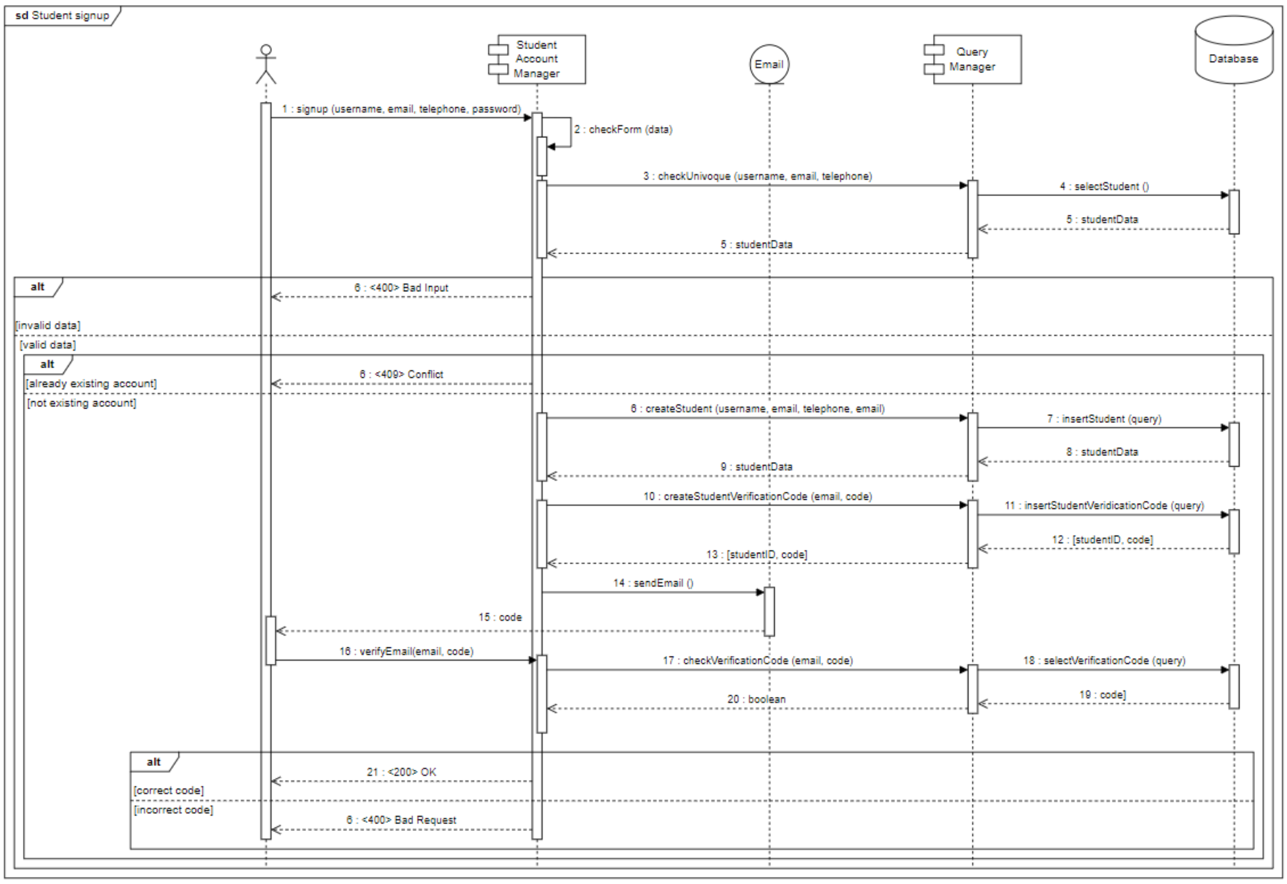


Figure 2.4: Student signup's runtime views

Student signup

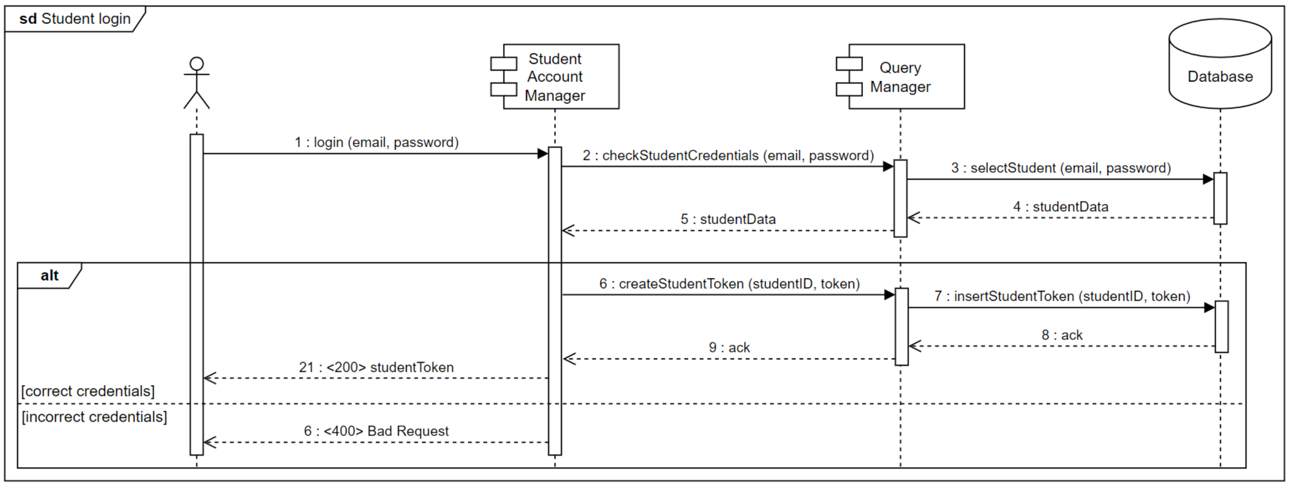


Figure 2.5: Student log in's runtime views

Student log in

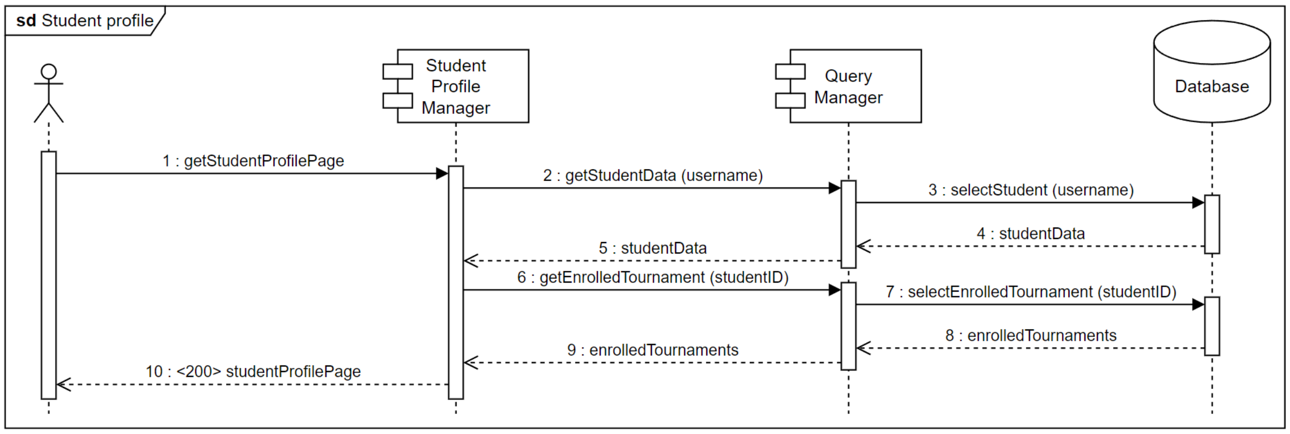

sd Student profile
Student Profile Manager
Query Manager
Database
1 : getStudentProfilePage
2 : getStudentData (username)
3 : selectStudent (username)
4 : studentData
5 : studentData
6 : getEnrolledTournament (studentID)
7 : selectEnrolledTournament (studentID)
8 : enrolledTournaments
9 : enrolledTournaments
10 : <200> studentProfilePage

Figure 2.6: Student profile's runtime views

Student profile

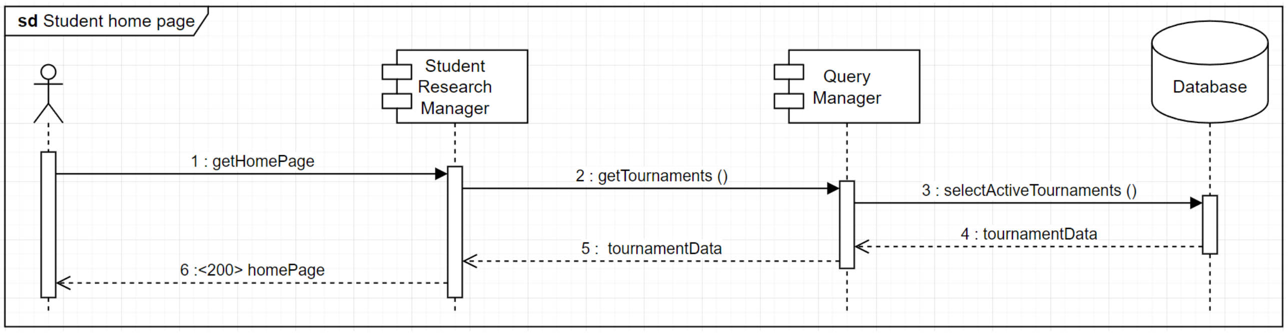


Figure 2.7: Student home page's runtime views

Student home page

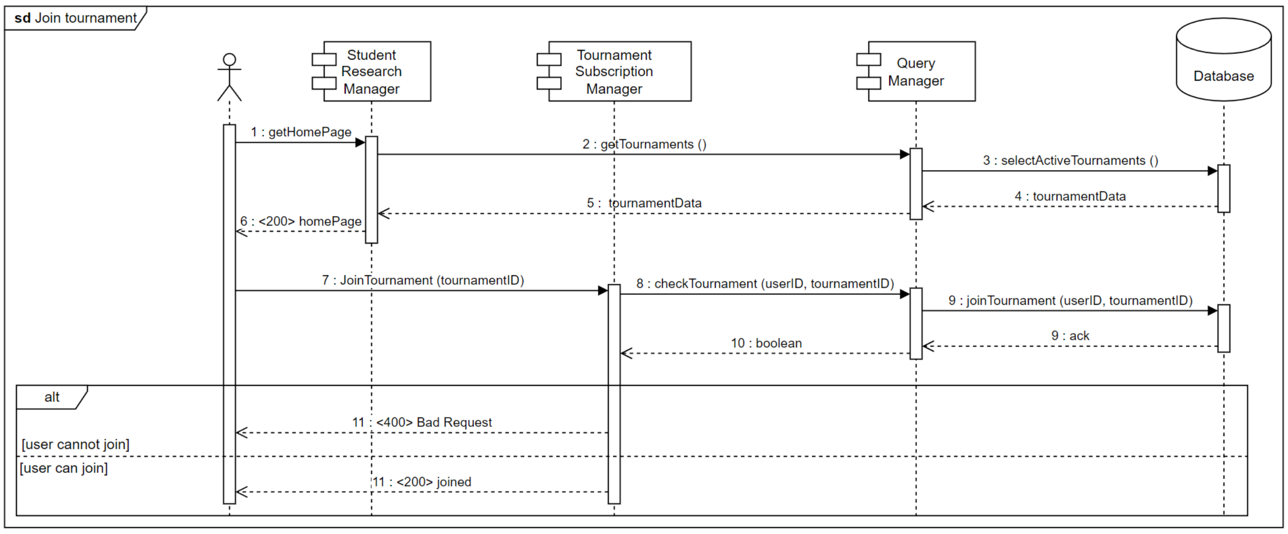


Figure 2.8: Student join tournament's runtime views

Student join tournament

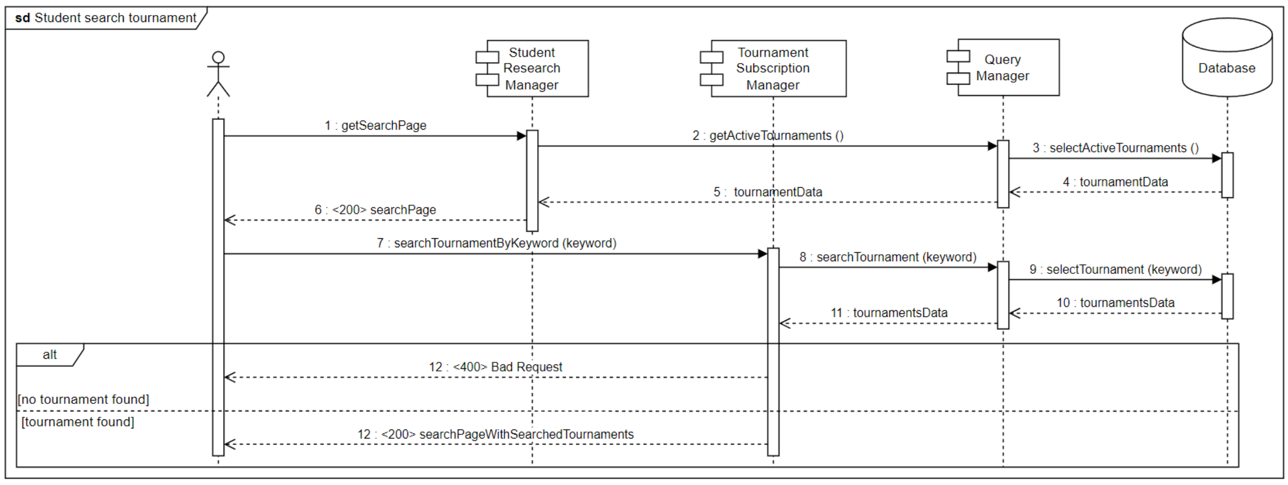

sd Student search tournament
Student Research Manager
Tournament Subscription Manager
Query Manager
Database
1 : getSearchPage
2 : getActiveTournaments ()
3 : selectActiveTournaments ()
4 : tournamentData
5 : tournamentData
6 : <200> searchPage
7 : searchTournamentByKeyword (keyword)
8 : searchTournament (keyword)
9 : selectTournament (keyword)
10 : tournamentsData
11 : tournamentsData
alt
12 : <400> Bad Request
[no tournament found]
[tournament found]
12 : <200> searchPageWithSearchedTournaments

Figure 2.9: Student search tournament's runtime views

Student search tournament

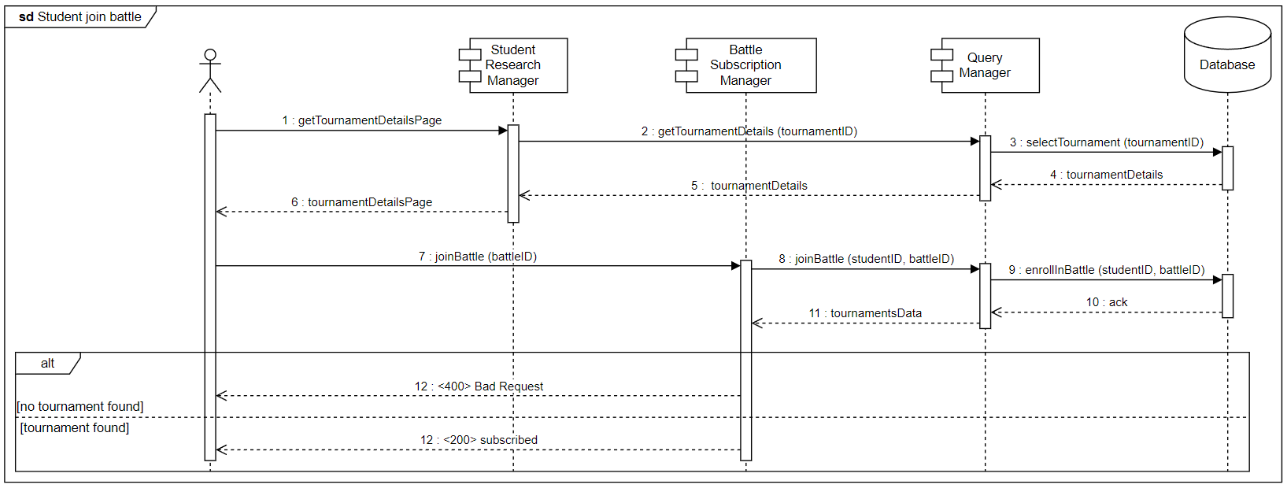

sd Student join battle
Student Research Manager
Battle Subscription Manager
Query Manager
Database
1 : getTournamentDetailsPage
2 : getTournamentDetails (tournamentID)
3 : selectTournament (tournamentID)
4 : tournamentDetails
5 : tournamentDetails
6 : tournamentDetailsPage
7 : joinBattle (battleID)
8 : joinBattle (studentID, battleID)
9 : enrollInBattle (studentID, battleID)
10 : ack
11 : tournamentsData
alt
12 : <400> Bad Request
[no tournament found]
[tournament found]
12 : <200> subscribed

Figure 2.10: Student join battle's runtime views

Student join battle

2.4.2 Educator's runtime view

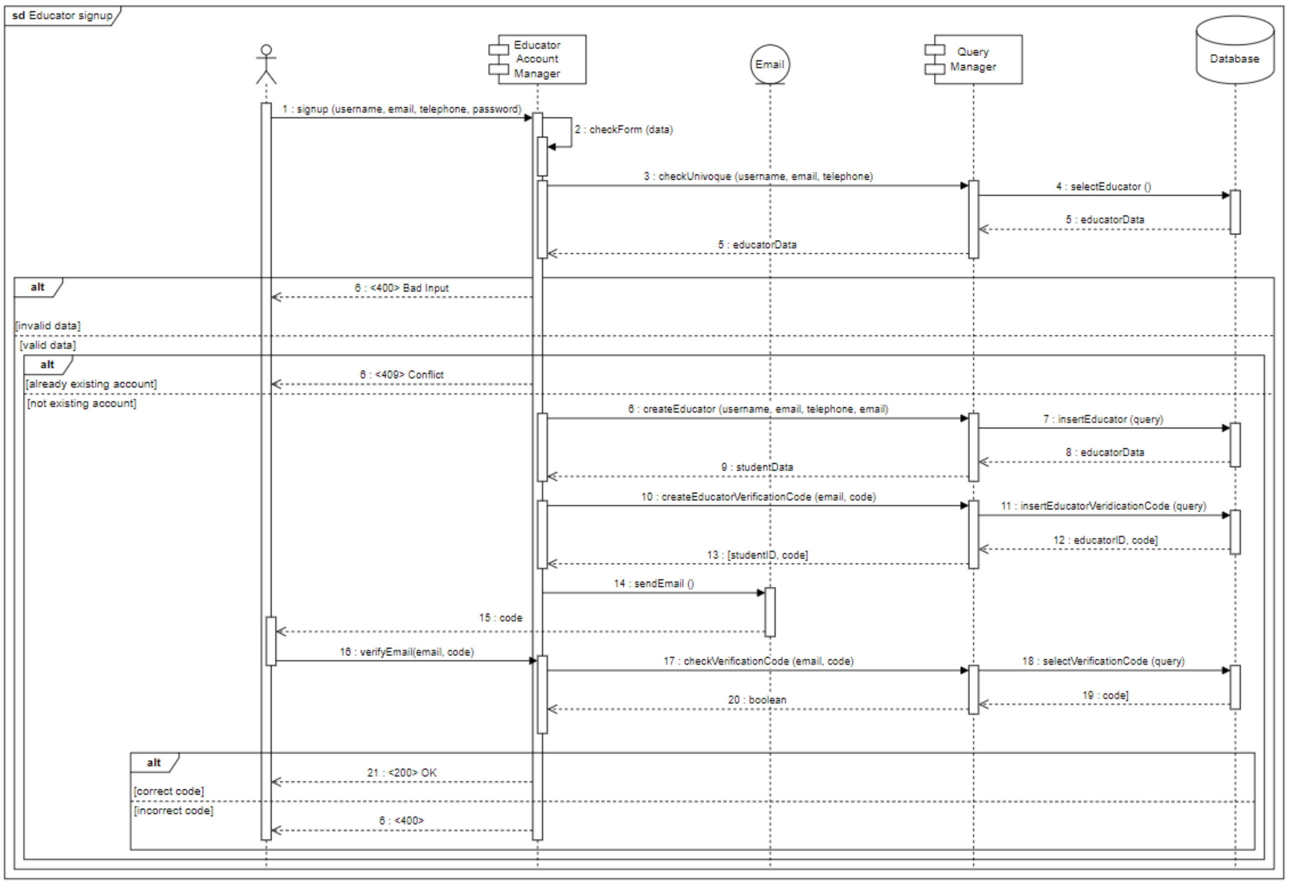


Figure 2.11: Educator signup's runtime views

Educator signup

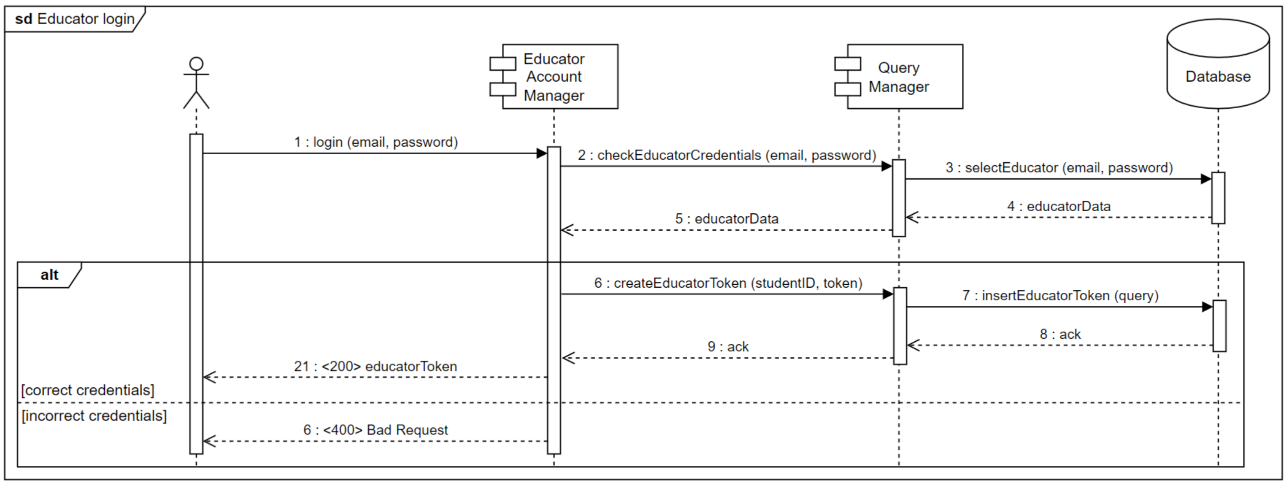


Figure 2.12: Educator log in's runtime views

Educator log in

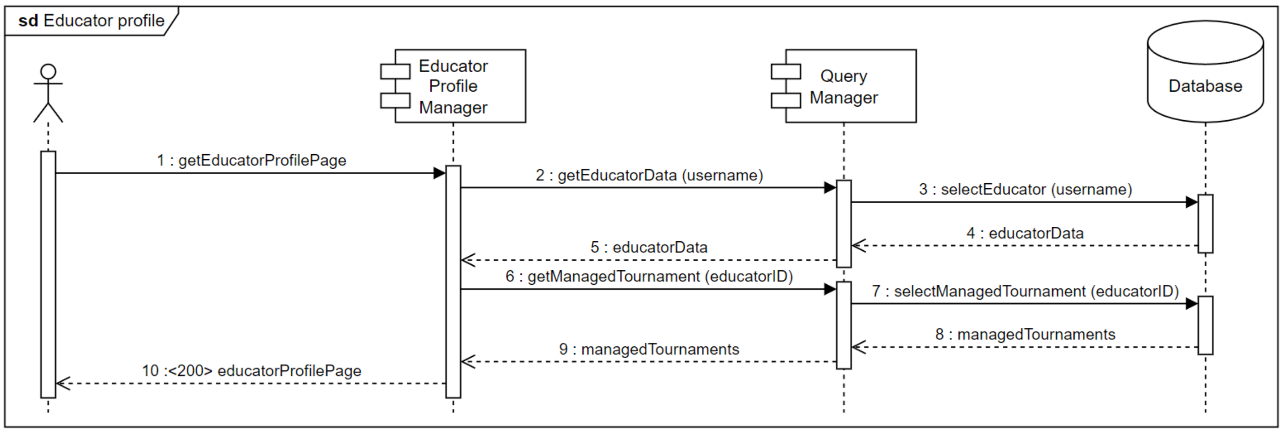

sd Educator profile
Educator Profile Manager
Query Manager
Database
1 : getEducatorProfilePage
2 : getEducatorData (username)
3 : selectEducator (username)
4 : educatorData
5 : educatorData
6 : getManagedTournament (educatorID)
7 : selectManagedTournament (educatorID)
8 : managedTournaments
9 : managedTournaments
10 :<200> educatorProfilePage

Figure 2.13: Educator profile's runtime views

Educator profile

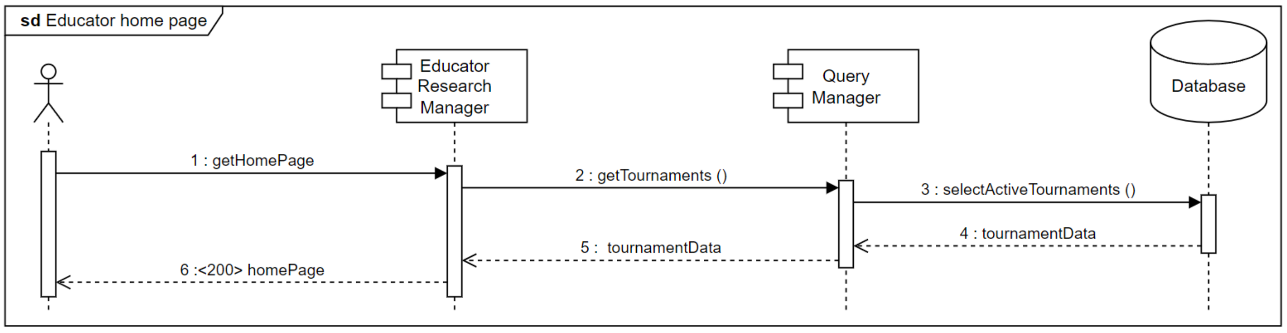


Figure 2.14: Educator profile's runtime views

Educator home page

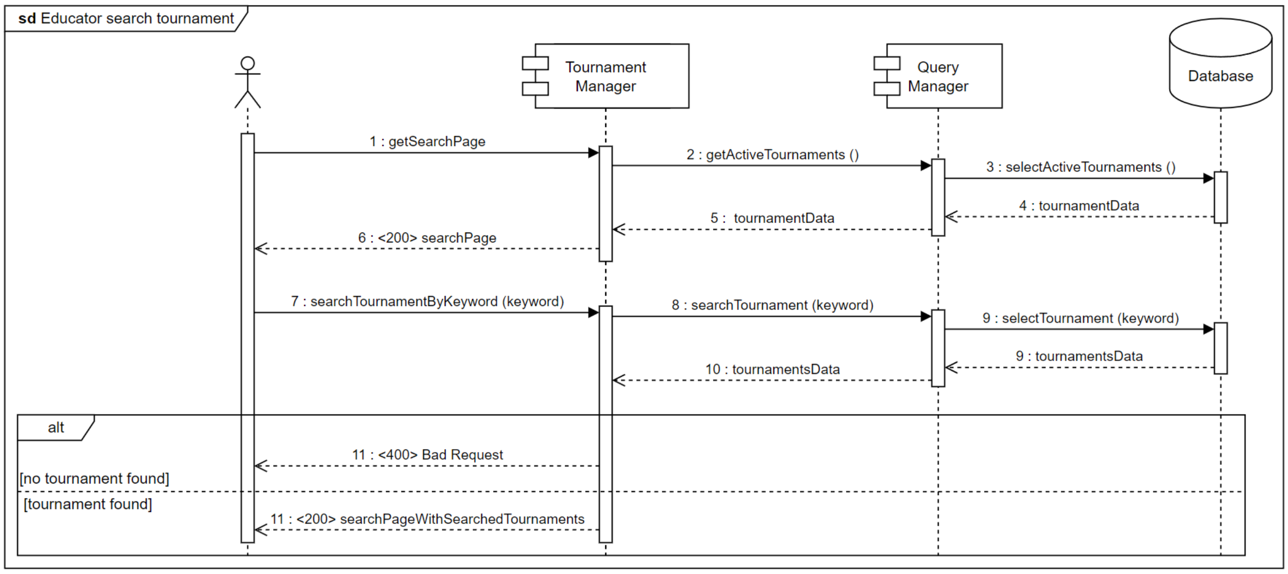

sd Educator search tournament
Tournament Manager
Query Manager
Database
1 : getSearchPage
2 : getActiveTournaments ()
3 : selectActiveTournaments ()
4 : tournamentData
5 : tournamentData
6 : <200> searchPage
7 : searchTournamentByKeyword (keyword)
8 : searchTournament (keyword)
9 : selectTournament (keyword)
9 : tournamentsData
10 : tournamentsData
alt
11 : <400> Bad Request
[no tournament found]
[tournament found]
11 : <200> searchPageWithSearchedTournaments

Figure 2.15: Educator search tournament's runtime views

Educator search tournament

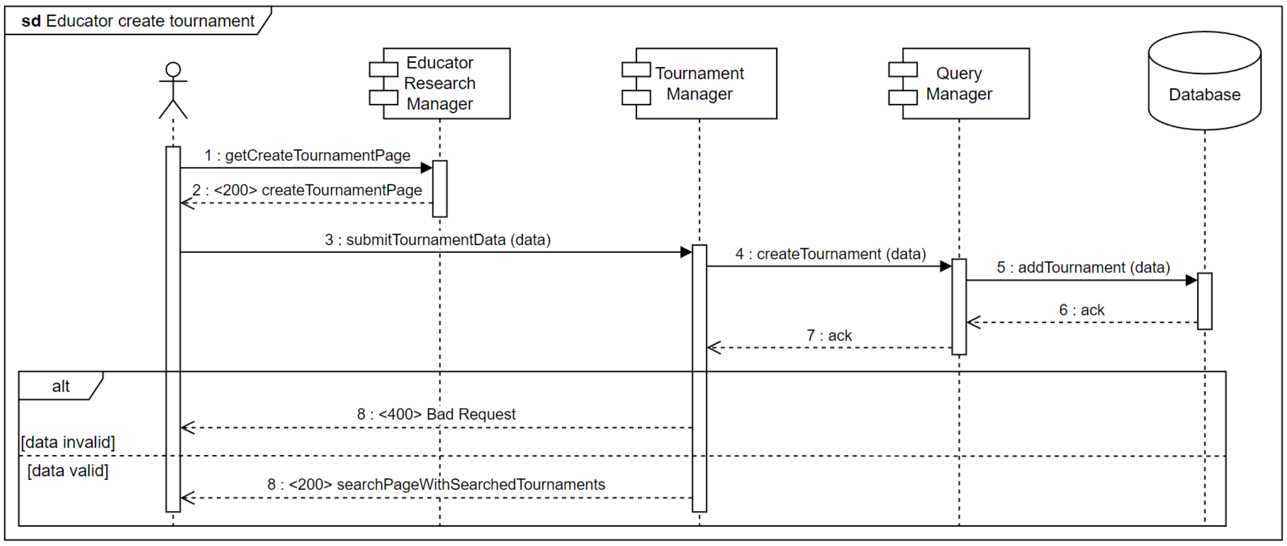

sd Educator create tournament
Educator Research Manager
Tournament Manager
Query Manager
Database
1 : getCreateTournamentPage
2 : <200> createTournamentPage
3 : submitTournamentData (data)
4 : createTournament (data)
5 : addTournament (data)
6 : ack
7 : ack
alt
8 : <400> Bad Request
[data invalid]
[data valid]
8 : <200> searchPageWithSearchedTournaments

Figure 2.16: Educator create tournament's runtime views

Educator create tournament

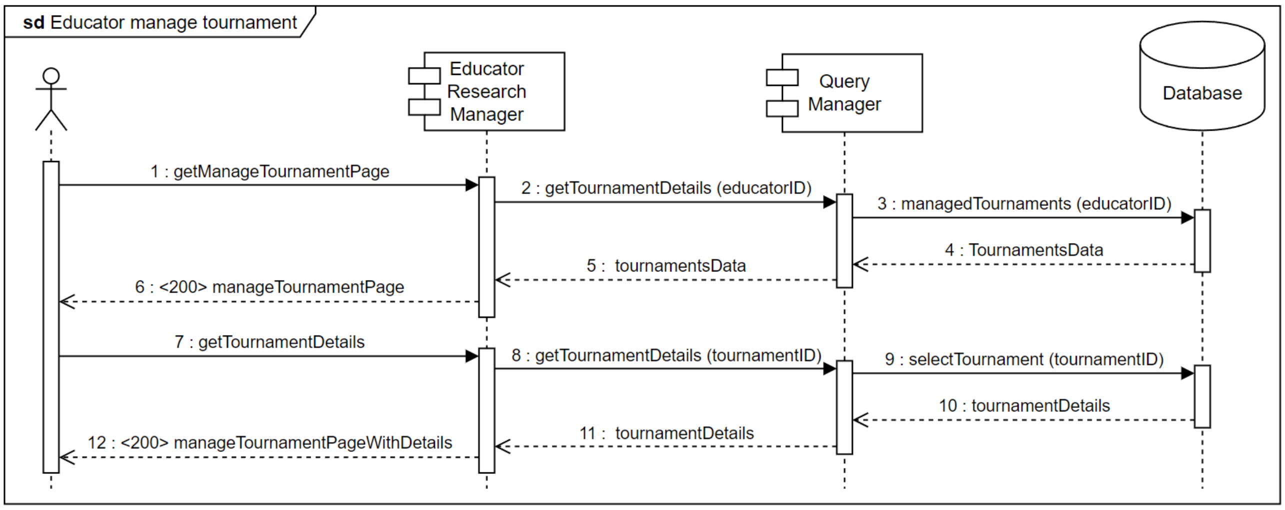

sd Educator manage tournament
Educator Research Manager
Query Manager
Database
1 : getManageTournamentPage
2 : getTournamentDetails (educatorID)
3 : managedTournaments (educatorID)
4 : TournamentsData
5 : tournamentsData
6 : <200> manageTournamentPage
7 : getTournamentDetails
8 : getTournamentDetails (tournamentID)
9 : selectTournament (tournamentID)
10 : tournamentDetails
11 : tournamentDetails
12 : <200> manageTournamentPageWithDetails

Figure 2.17: Educator manage tournament's runtime views

Educator manage tournament

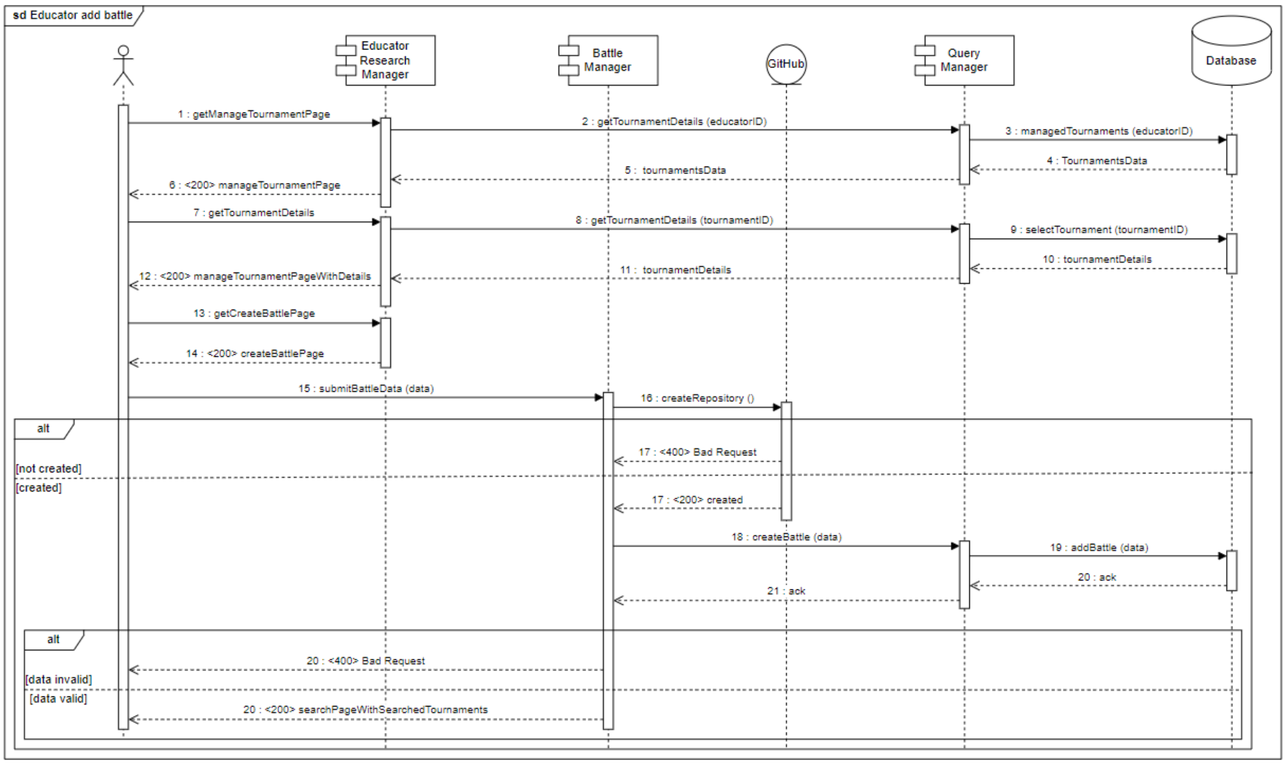


Figure 2.18: Educator add battle's runtime views

Educator add battle

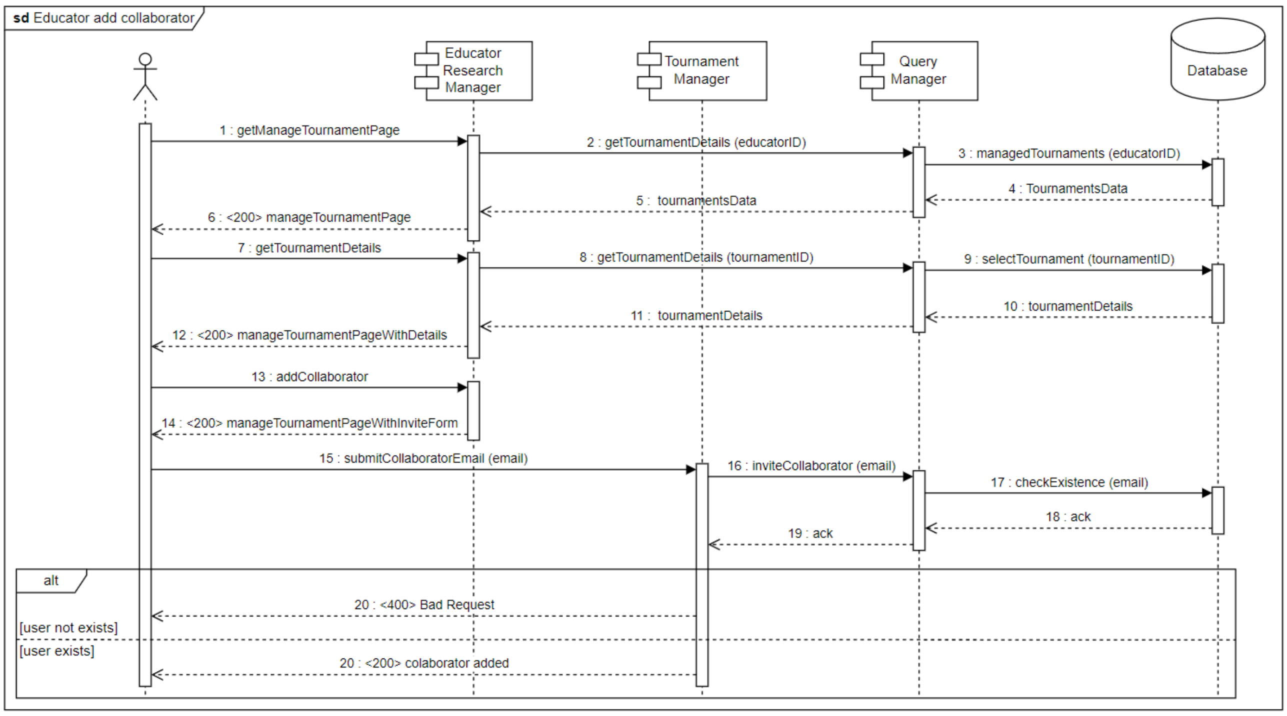

sd Educator add collaborator
Educator Research Manager
Tournament Manager
Query Manager
Database
1 : getManageTournamentPage
2 : getTournamentDetails (educatorID)
3 : managedTournaments (educatorID)
4 : TournamentsData
5 : tournamentsData
6 : <200> manageTournamentPage
7 : getTournamentDetails
8 : getTournamentDetails (tournamentID)
9 : selectTournament (tournamentID)
10 : tournamentDetails
11 : tournamentDetails
12 : <200> manageTournamentPageWithDetails
13 : addCollaborator
14 : <200> manageTournamentPageWithInviteForm
15 : submitCollaboratorEmail (email)
16 : inviteCollaborator (email)
17 : checkExistence (email)
18 : ack
19 : ack
alt
20 : <400> Bad Request
[user not exists]
[user exists]
20 : <200> colaborator added

Figure 2.19: Educator add collaborator's runtime views

Educator add collaborator

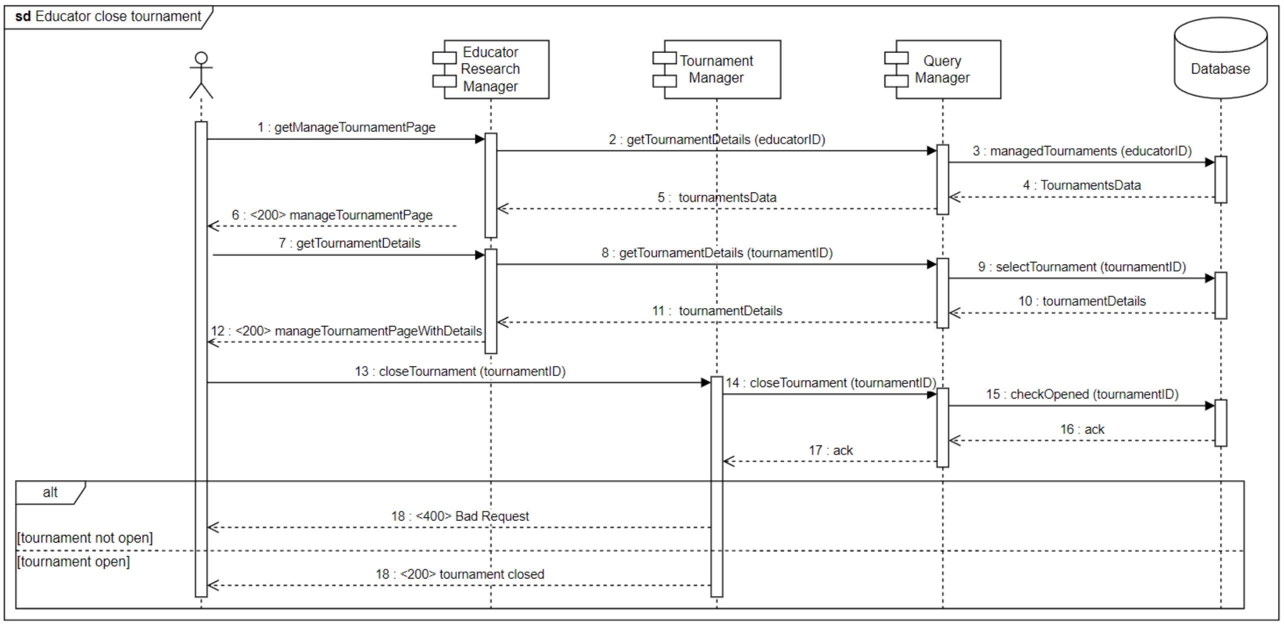

sd Educator close tournament
Educator Research Manager
Tournament Manager
Query Manager
Database
1 : getManageTournamentPage
2 : getTournamentDetails (educatorID)
3 : managedTournaments (educatorID)
4 : TournamentsData
5 : tournamentsData
6 : <200> manageTournamentPage
7 : getTournamentDetails
8 : getTournamentDetails (tournamentID)
9 : selectTournament (tournamentID)
10 : tournamentDetails
11 : tournamentDetails
12 : <200> manageTournamentPageWithDetails
13 : closeTournament (tournamentID)
14 : closeTournament (tournamentID)
15 : checkOpened (tournamentID)
16 : ack
17 : ack
alt
18 : <400> Bad Request
[tournament not open]
[tournament open]
18 : <200> tournament closed

Figure 2.20: Educator close tournament's runtime views

Educator close tournament

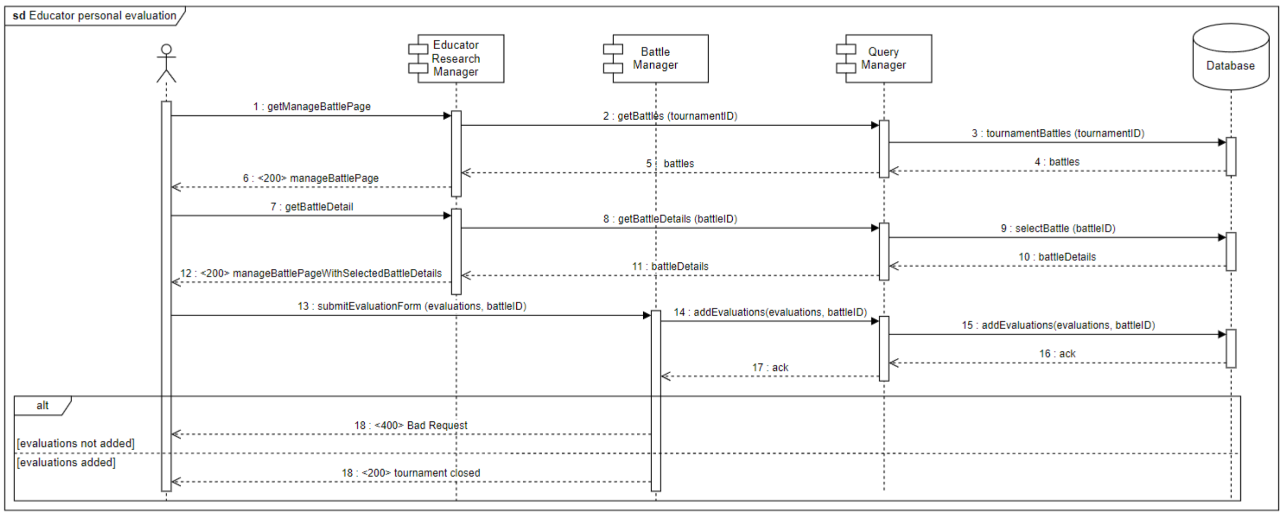

sd Educator personal evaluation
Educator Research Manager
Battle Manager
Query Manager
Database
1 : getManageBattlePage
2 : getBattles (tournamentID)
3 : tournamentBattles (tournamentID)
4 : battles
5 : battles
6 : <200> manageBattlePage
7 : getBattleDetail
8 : getBattleDetails (battleID)
9 : selectBattle (battleID)
10 : battleDetails
11 : battleDetails
12 : <200> manageBattlePageWithSelectedBattleDetails
13 : submitEvaluationForm (evaluations, battleID)
14 : addEvaluations(evaluations, battleID)
15 : addEvaluations(evaluations, battleID)
16 : ack
17 : ack
alt
18 : <400> Bad Request
[evaluations not added]
[evaluations added]
18 : <200> tournament closed

Figure 2.21: Educator personal evaluation's runtime views

Educator personal evaluation

2.4.3 Both users runtime view

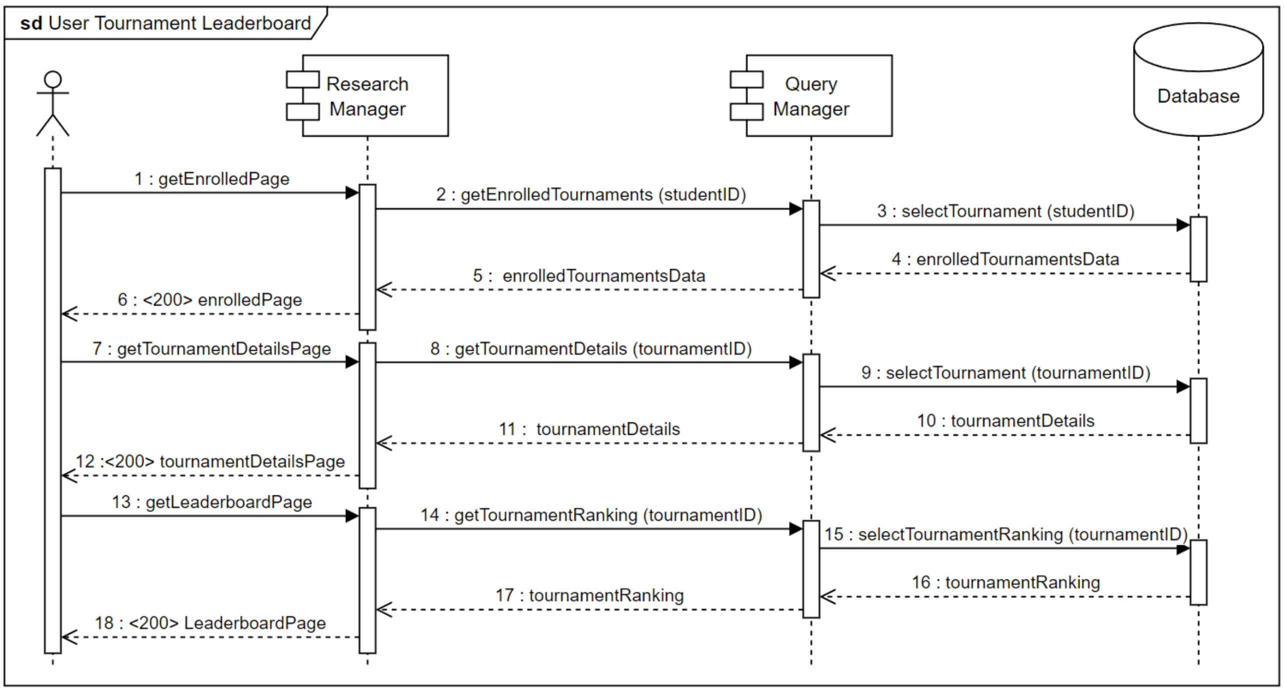

sd User Tournament Leaderboard
Research Manager
Query Manager
Database
1 : getEnrolledPage
2 : getEnrolledTournaments (studentID)
3 : selectTournament (studentID)
4 : enrolledTournamentsData
5 : enrolledTournamentsData
6 : <200> enrolledPage
7 : getTournamentDetailsPage
8 : getTournamentDetails (tournamentID)
9 : selectTournament (tournamentID)
10 : tournamentDetails
11 : tournamentDetails
12 :<200> tournamentDetailsPage
13 : getLeaderboardPage
14 : getTournamentRanking (tournamentID)
15 : selectTournamentRanking (tournamentID)
16 : tournamentRanking
17 : tournamentRanking
18 : <200> LeaderboardPage

Figure 2.22: User checks tournament leaderboard's runtime views

User checks tournament leaderboard

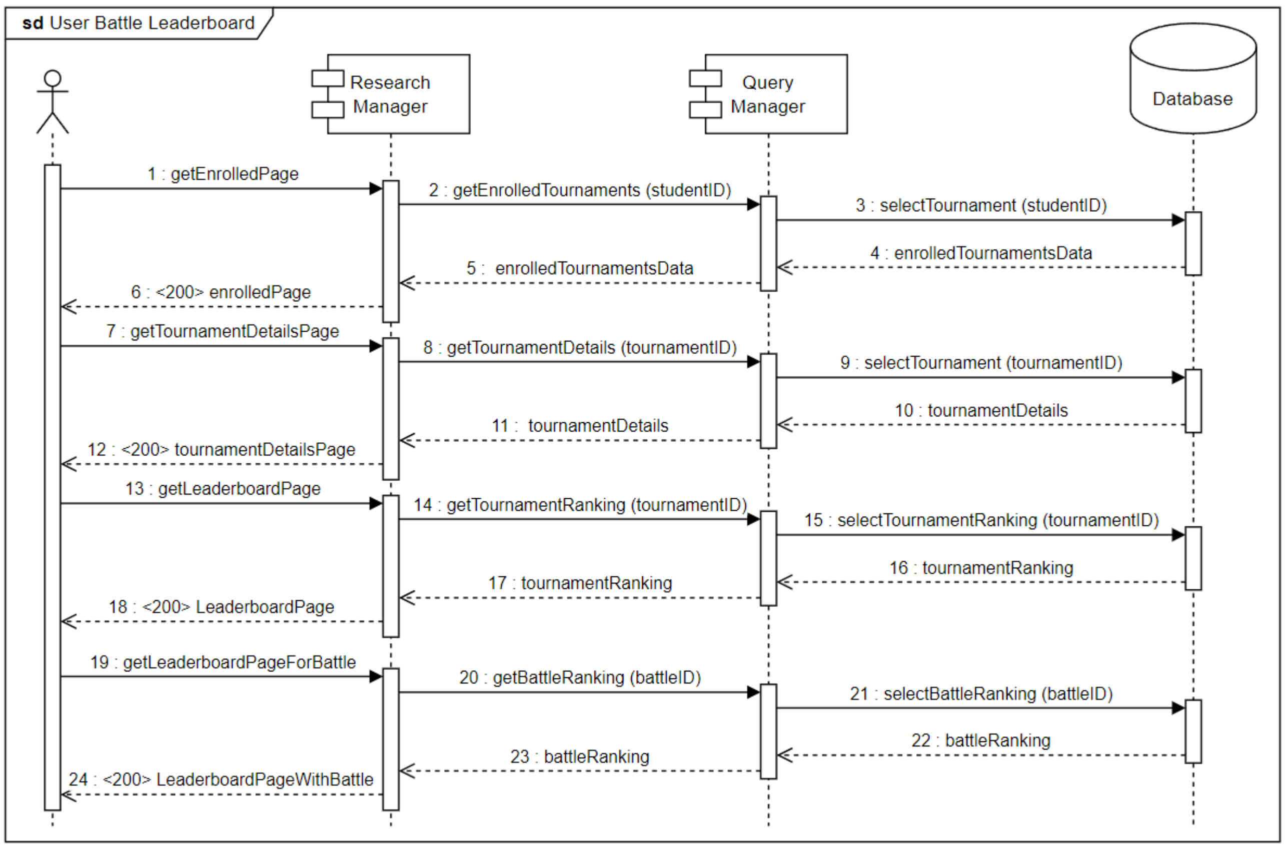

sd User Battle Leaderboard
Research Manager
Query Manager
Database
1 : getEnrolledPage
2 : getEnrolledTournaments (studentID)
3 : selectTournament (studentID)
4 : enrolledTournamentsData
5 : enrolledTournamentsData
6 : <200> enrolledPage
7 : getTournamentDetailsPage
8 : getTournamentDetails (tournamentID)
9 : selectTournament (tournamentID)
10 : tournamentDetails
11 : tournamentDetails
12 : <200> tournamentDetailsPage
13 : getLeaderboardPage
14 : getTournamentRanking (tournamentID)
15 : selectTournamentRanking (tournamentID)
16 : tournamentRanking
17 : tournamentRanking
18 : <200> LeaderboardPage
19 : getLeaderboardPageForBattle
20 : getBattleRanking (battleID)
21 : selectBattleRanking (battleID)
22 : battleRanking
23 : battleRanking
24 : <200> LeaderboardPageWithBattle

Figure 2.23: User checks battle leaderboard's runtime views

User checks battle leaderboard

2.4.4 API endpoints

2.5 Component interfaces

2.6 Selected architectural styles and patterns

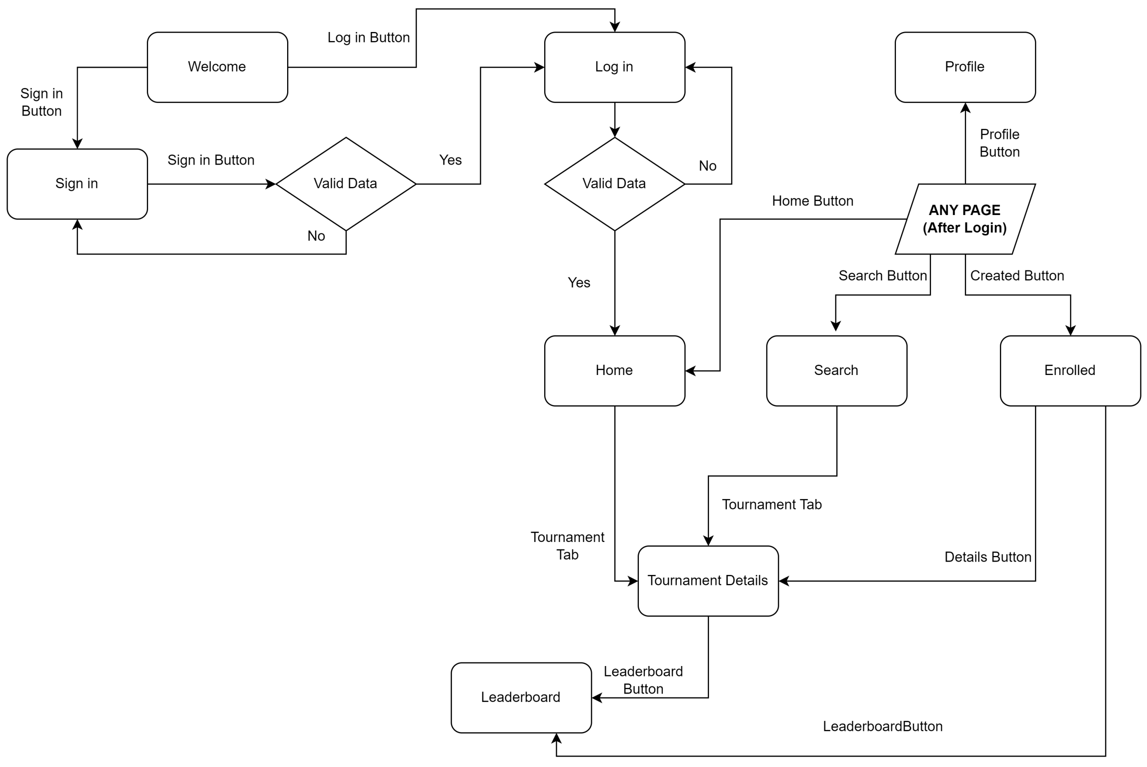
2.7 Other design decisions

Chapter 3

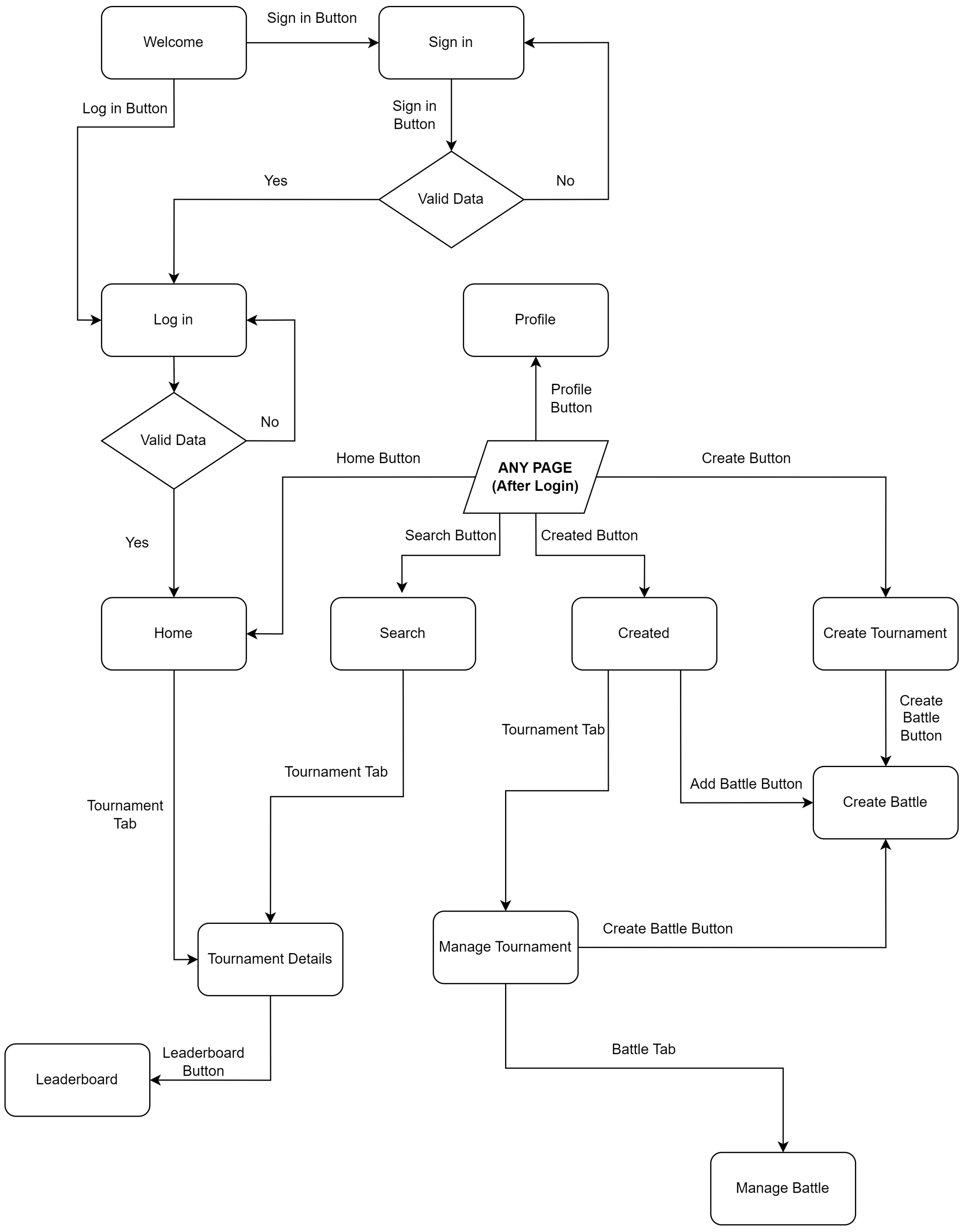
User interface design

The user interface is the visual representation of how the application appears to the end user. It should be intuitive and straightforward, ensuring easy access to all features. Mockups were introduced in the RASD document, and now flowcharts for both students and educators will be presented. Flowcharts offer a concise and clear overview of the interactions and steps involved in utilizing the web app.

3.1 Flowchart for students UI



3.2 Flowchart for educators UI



Chapter 4

Requirements traceability

Chapter 5

Implementation, integration and test plan

Chapter 6

Effort spent

Chapter 7

References