POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

DD Document

Author: **Mattia Piccinato, Gabriele Puglisi, Jacopo Piazzalunga**

Academic Year: 2023-24

Contents

1 | Introduction

1.1. Purpose

The purpose of the project CodeKataBattle (CKB) is to develop a platform where Students can practice coding together. Educators set up challenges (called Battles) within Tournaments, and Students work in teams to solve them. The platform checks their code automatically, giving scores based on how well it works, how quickly they finish, and how good their code quality is. Educators may optionally give extra scores by checking the work themselves, in a so-called Consolidation Stage which starts right after the end of every Battle. Students get ranked in Tournaments based on their scores obtained in teams, and can earn Badges for some achievements, in order to make learning programming more fun and competitive.

1.1.1. Goals

- G1 Allows registered Students who enrolled according to the right modalities to participate in a Tournament of Code Kata Battles and take part in its Battles.
- G2 Allows registered Educators to manage Tournaments for which they have been granted permission.
- G3 Allows registered Students who participate in a Tournament of Code Kata Battles to be rewarded of different achievements.
- G4 Allows registered Users to visualize information for which they have granted permission.
- G5 Automates code evaluation process using GitHub Actions and some static analysis tools.

1.2. Scope

The main features which should be provided in order to achieve the aim of the project are:

- Creating Challenges: Educators can make coding challenges (Battles) that Students can join, alone or in teams (groups), within the context of a Tournament.
- Using GitHub Actions for Code Submsissions: Students are supposed to submit their code for a Battle in their GitHub repository, and the system must be informed of a new commit by a participating Student making use of GitHub Actions.
- Checking Code both Automatically and Manually: The platform assigns a score to Students' code automatically, without the intervention of any Educator. Optionally, Educators may also decide to evaluate the code themselves.
- Rankings: During each Battle, a live ranking of the involved teams is available, enabling participating Students to track their performance. Additionally, live Tournament rankings show how well each Student is performing in the Battles within a given Tournament.
- Badges for Achievements: At the end of every Tournament, Students who achieved good results may be awarded with a special Badges, which are ruled by the Educator who created the Tournament.

The main goal is to help Students practice coding, giving them feedback and comparing their results to others.

1.2.1. Definitions

- Battle: A Code Kata, that is, a challenge in which teams of players need to solve a problem in a specific coding language and submit their code to get a score according to the rules of the Battle.
- Tournament: A competition composed of many Battles in which participants' overall score is the sum of all the scores obtained in every Battle they participated in, individually or in team with other players.
- Student: The User which takes part into the Tournaments of Battles.

- Educator: The User which organizes Tournaments of Battles to which the Students can participate and who manages every aspect about them.
- Consolidation stage: The Consolidation Stage is the phase of a Battle which starts as soon as the submission deadline expires, during which the Educators who manage the Tournament can eventually assign an additional score to every team, which will be summed to the score previously assigned by the platform.
- Badge: A Badge is an achievement marker related to a certain Tournament, obtained by Students if they satisfied the specific conditions defined by the Educator during the creation process of the Tournament.

1.2.2. Acronyms

- CK: Code Kata, that is, a Battle.
- CKB: Code Kata Battle, that is, the name of the platform.
- ckbSP: Code Kata Battle Service Provider.
- DMZ: De-Militarized Zone.

1.2.3. Abbreviations

- WPn: n-th World Phenomena
- SPn: n-th Shared Phenomena
- Gn: n-th Goal
- Dn: n-th Domain Assumption
- Rn: n-th Requirement

1.3. Revision History

Revised on	Version	Description
7-Gen-2023	1.0	Initial Release of the document

1.4. Reference Documents

- Assignment document A.Y. 2023/2024
  (”Requirement Engineering and Design Project: goal, schedule and rules”)
- Software Engineering 2 A.Y. 2023/2024 Slides
  (Lecture slides provided during the course)

1.5. Document Structure

This document is composed of six sections:

- 1st Chapter: We begin by presenting the problem statement and outlining the system’s objectives. In the scope subsection, we offer insights into the various real-world and shared phenomena that the system addresses. Finally, we provide essential resources for readers, including definitions and abbreviations, to facilitate a comprehensive understanding of this document.
- 2nd Chapter: We offer a panoramic view of the system’s architecture, starting with a comprehensive overview that articulates the high-level components and their interconnections. Subsequently, it delves into the component view, the deployment view and the runtime view. Furthermore, it details component interfaces, elucidates chosen architectural styles and patterns, and encapsulates additional design decisions made during the system’s architectural conception.
- 3rd Chapter: We provide an overview of the user interface, as it was presented already in the RASD document.
- 4th Chapter: We elucidate the correlation between the delineated requirements in the

RASD document and the corresponding design elements articulated in this document, mapping how each requirement aligns with the design components.

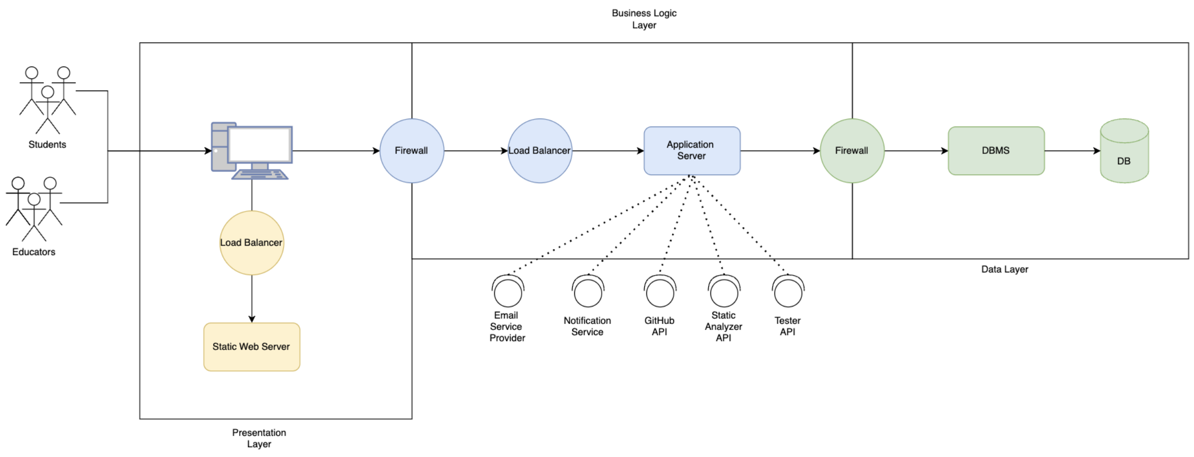
- 5th Chapter: We delineate the procedural steps and order of operations necessary to bring the design to fruition while ensuring functionality and coherence.
- 6th Chapter: We provide an estimate of the effort spent by each group member.
- 7th Chapter: We provide a list of the references used in this document.

2 | Overall Description

The purpose of this section was to present and analyze the architecture of the S2B system in a top- down manner. We first introduced the overall architecture and then provided a diagram of the system's components, focusing on the ckbSP subcomponent. Next, we used an ER diagram to describe the system's logical data and presented the system's deployment view, including the layers and tiers involved. We also used sequence diagrams to depict important runtime views and class diagrams to analyze the component interfaces. Finally, we discussed the architectural design choices and the reasons behind them.

2.1. Overview: High-level Components and Interaction

The figure shown below represents a high-level description of the components which make up the System. In this document the presentation layer and the Client (e.g. the Browser) will be referred to as the Frontend, while the Application Layer and the Data Layer will be referred to as the Backend.



A web interface will be used to access the service. A single page application (SPA) will be developed for educators and students to use the system. An SPA is a good choice for this type of application because it allows for a lot of interaction without the need for frequent page reloads, which can provide a faster and more seamless user experience. The overall architecture of the system is divided into different layers, with the application servers interacting with a database management system and using APIs to retrieve and store data. The application servers are designed to be stateless according to REST standards, and the system includes firewalls to enhance security.

2.2. Component View

In this section we show the components of the S2B and their relationships. The following sections will explain the interaction between interfaces and details on each method of interfaces with REST endpoints, if any.

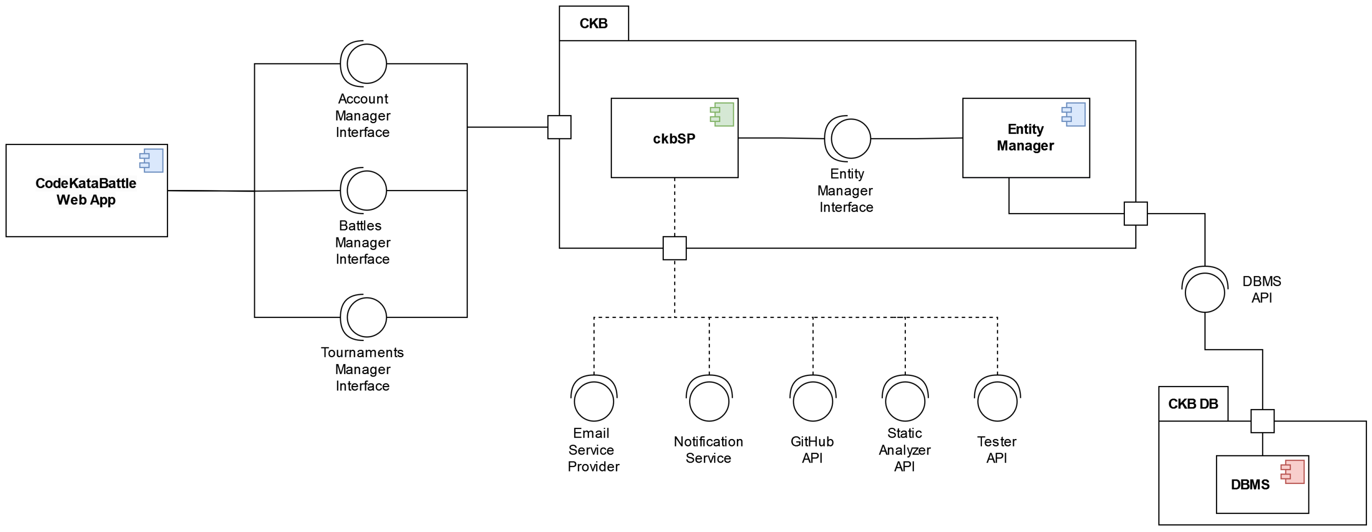


Figure 2.1: Component Diagram of the CodeKataBattle System

2.2.1. Entity Manager

This component is responsible for communication with a Database Management System (DBMS). It follows the Adapter design pattern, allowing other components to interact with the DBMS without needing to write any SQL code themselves.

2.2.2. ckbSP

The ckbSP subsystem implements CKB's logic. It provides all the system's features, interacting with the other components.

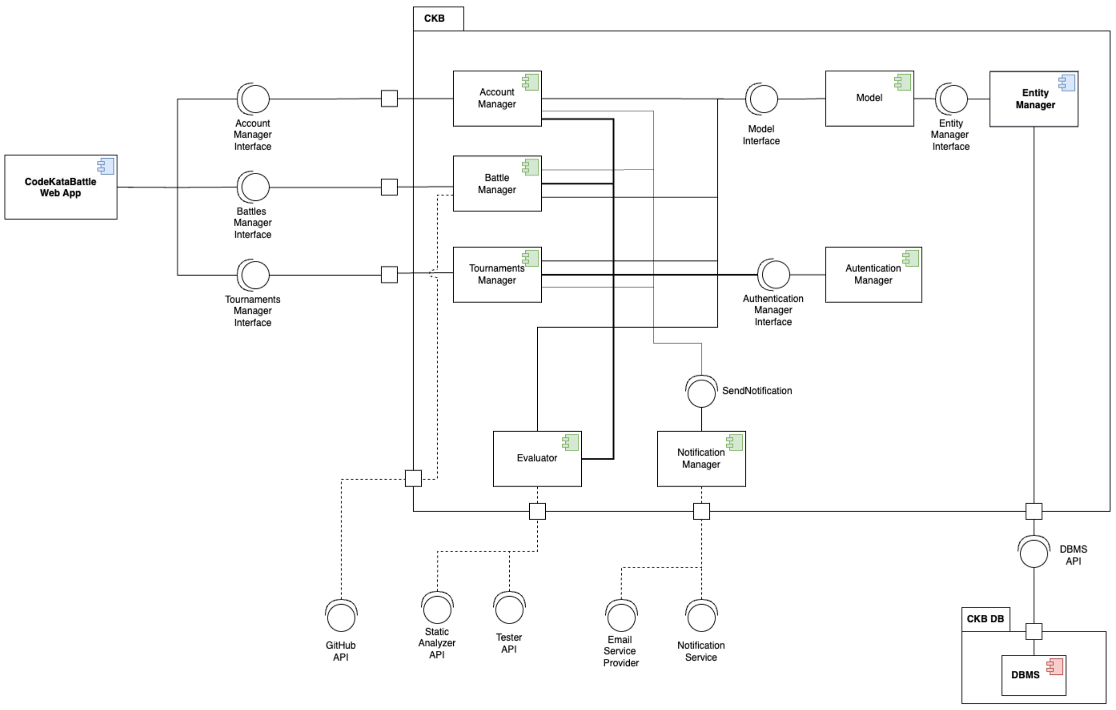


Figure 2.2: Component Diagram of the ckbSP subsystem

1. Account Manager This component handles the operations needed to manage each User's account, and to view the Students' profiles. It allows Users to login, using the Authentication Manager's interface.

2. Battle Manager This component handles the operations needed to manage a Battle, to view any information about it and to subscribe.

3. Tournament Manager This component handles the operations needed to manage a Tournament, to view any information about it and to subscribe.

4. Authentication Manager This component handles signup, login and logout operations.

5. Evaluator This component is responsible for automatically computing the score of Students' code when a commit is performed.

6. Notification Manager This component is responsible for the notification dispatch.
7. Model This component facilitates the interaction with and representation of CKBsp data.

2.2.3. Logical Description of Data

In this section the ER diagram of CKB's data is shown. An ER diagram is a graphical representation of the structure of a database, showing the relationships between entities and their attributes.

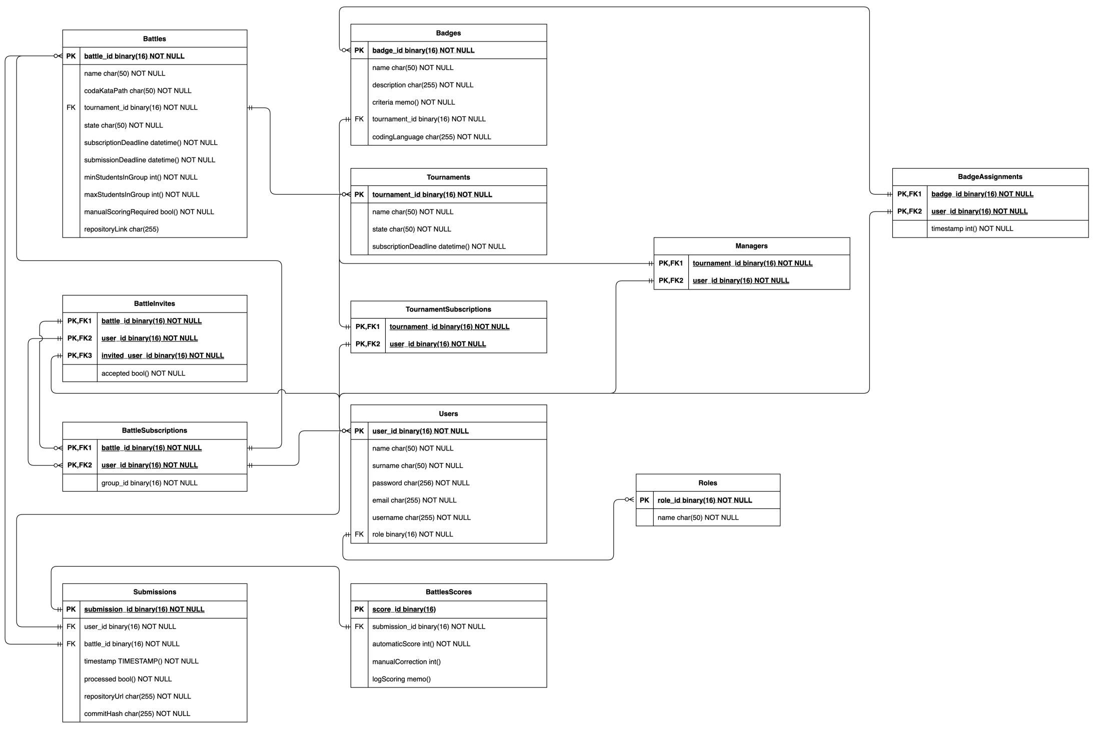

Battles
PK battle_id binary(16) NOT NULL
name char(50) NOT NULL
codaKataPath char(50) NOT NULL
FK tournament_id binary(16) NOT NULL
state char(50) NOT NULL
subscriptionDeadline datetime() NOT NULL
submissionDeadline datetime() NOT NULL
minStudentsInGroup int() NOT NULL
maxStudentsInGroup int() NOT NULL
manualScoringRequired bool() NOT NULL
repositoryLink char(255)
Badges
PK badge_id binary(16) NOT NULL
name char(50) NOT NULL
description char(255) NOT NULL
criteria memo() NOT NULL
FK tournament_id binary(16) NOT NULL
codingLanguage char(255) NOT NULL
Tournaments
PK tournament_id binary(16) NOT NULL
name char(50) NOT NULL
state char(50) NOT NULL
subscriptionDeadline datetime() NOT NULL
BadgeAssignments
PK,FK1 badge_id binary(16) NOT NULL
PK,FK2 user_id binary(16) NOT NULL
timestamp int() NOT NULL
Managers
PK,FK1 tournament_id binary(16) NOT NULL
PK,FK2 user_id binary(16) NOT NULL
BattleInvites
PK,FK1 battle_id binary(16) NOT NULL
PK,FK2 user_id binary(16) NOT NULL
PK,FK3 invited_user_id binary(16) NOT NULL
accepted bool() NOT NULL
TournamentSubscriptions
PK,FK1 tournament_id binary(16) NOT NULL
PK,FK2 user_id binary(16) NOT NULL
BattleSubscriptions
PK,FK1 battle_id binary(16) NOT NULL
PK,FK2 user_id binary(16) NOT NULL
group_id binary(16) NOT NULL
Users
PK user_id binary(16) NOT NULL
name char(50) NOT NULL
surname char(50) NOT NULL
password char(256) NOT NULL
email char(255) NOT NULL
username char(255) NOT NULL
FK role binary(16) NOT NULL
Roles
PK role_id binary(16) NOT NULL
name char(50) NOT NULL
Submissions
PK submission_id binary(16) NOT NULL
FK user_id binary(16) NOT NULL
FK battle_id binary(16) NOT NULL
timestamp TIMESTAMP() NOT NULL
processed bool() NOT NULL
repositoryUrl char(255) NOT NULL
commitHash char(255) NOT NULL
BattlesScores
PK score_id binary(16)
FK submission_id binary(16) NOT NULL
automaticScore int() NOT NULL
manualCorrection int()
logScoring memo()

Figure 2.3: ER Diagram of the ckbSP subsystem

Additional constraints and triggers must be established in the database management system to ensure data coherence and consistency.

When a battle deadline is reached and its status changes, a trigger should be implemented. This trigger will verify that each group meets the minimum required number of Students. Groups failing to meet this requirement will be automatically unsubscribed from the Battle.

Furthermore, constraints are necessary to maintain role-specific restrictions. Educators must not be included in Battle subscriptions and invitations, and students should not be allowed to manage Tournaments.

Another important constraint is to prevent Users from being enrolled in the same Battle under different groups. This combination of triggers and constraints is crucial for upholding data integrity and consistency in the system.

2.3. Deployment View

The system adopts a robust 3-tier architecture hosted on cloud infrastructure, meticulously designed to optimize performance, scalability, and security.

The entire architecture is hosted on a cloud provider. This offers several advantages compared to traditional in-house hosting, such as:

1. Scalability and Flexibility - the ability to add or remove resources such as virtual machines, performance cores, or memory as needed, and the use of load balancing services, allows the servers to adapt to changes in traffic or workload.
2. Security - services like live monitoring and firewalls help to protect the application server against data breaches and other security threats.
3. Cost-efficiency - the pay-as-you-go model of a cloud provider allows to only pay for the resources that are actually used, which can help to lower the overall costs.

These features make a cloud provider an ideal choice for hosting large, high-traffic applications. The chosen cloud provider will need to offer all of these features in order to meet our needs.

The web server resides in the DMZ and functions as the primary interface for user interaction, managing HTTP requests and delivering web pages. The application server executes the core application logic, processing user requests and handling business operations. Load balancers are implemented to evenly distribute incoming traffic across multiple instances of Web and Application Servers.

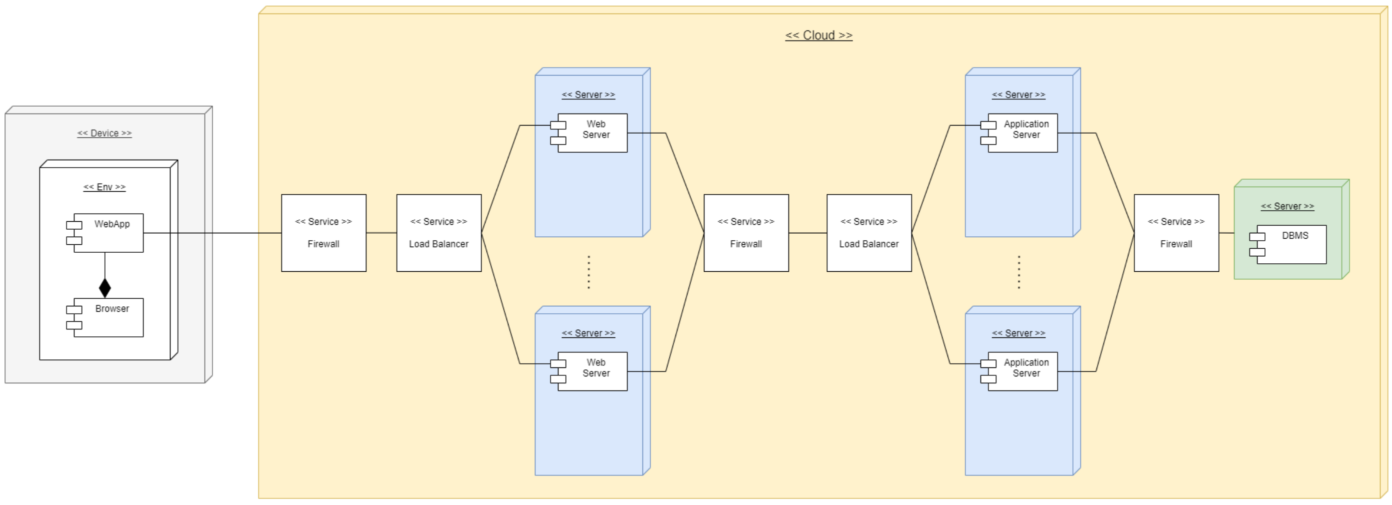


Figure 2.4: Component Diagram of the ckbSP subsystem

2.4. Runtime View

User Registers

The diagram represented down below shows the process of a User creating an account to the CKB website. First the User compiles the sign up form with all the requested information, then, as the request is submitted through the proper API call, the AccountManager component handles the request, and, if the parameters of the request were valid, the AccountManager triggers the verification email through the EmailServiceProvider.

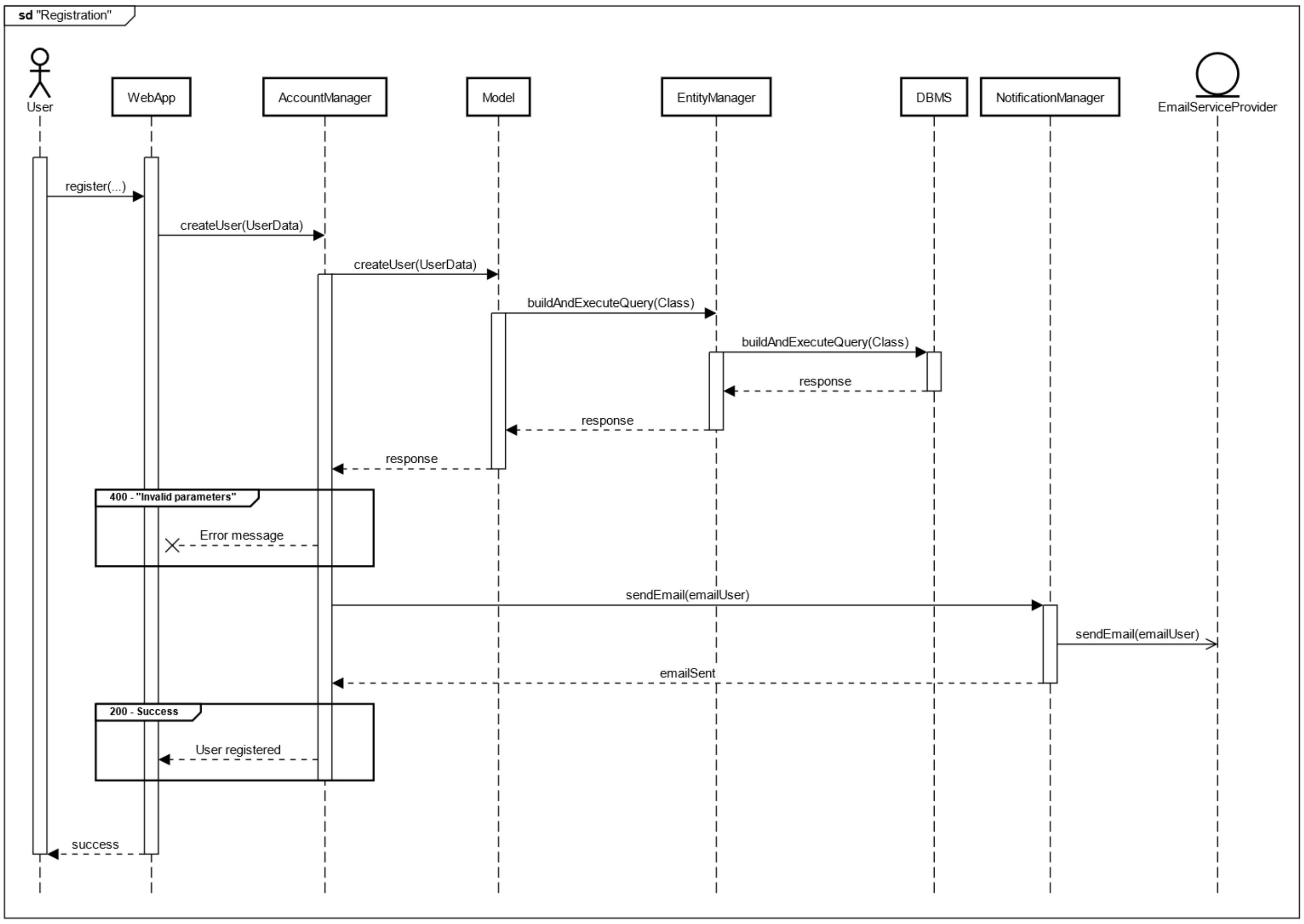


Figure 2.5: User registers to CKB

User Logs in

The diagram represented down below shows the process of a User logging into their CKB account. First the User compiles the log in form with their credentials, then, as the request is submitted through the proper API call, the AccountManager component handles the request, and, if the credentials were valid, the user is provided with the authentication token generated by the AuthenticationManager through the AccountManager.

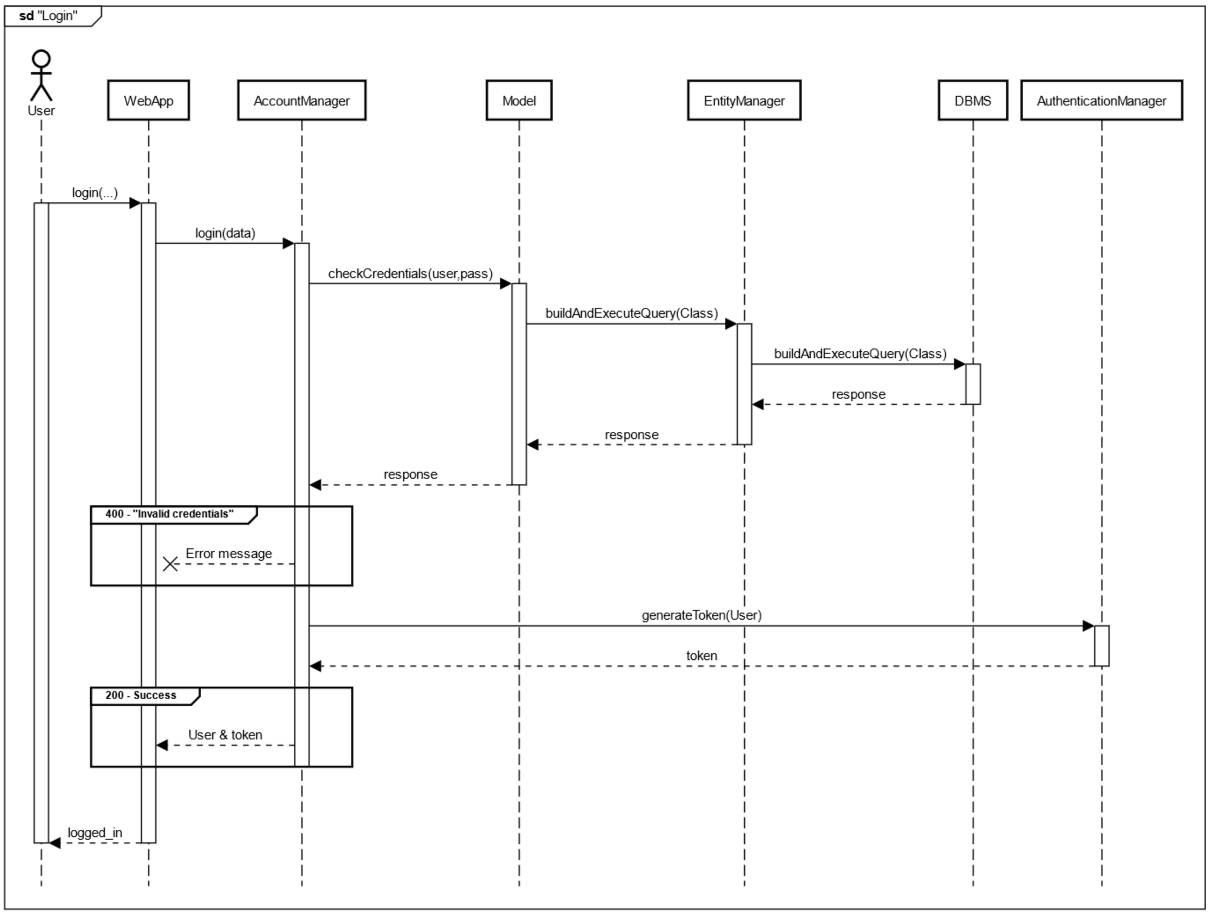


Figure 2.6: User log in

User sees tournaments list

The diagram represented down below shows the process of a User visualizing the list of Tournaments. It comprehends the entire history of Tournaments ever created in the system. Upon successful validation, the TournamentManager interacts with the Model to retrieve Tournament data using the EntityManager, querying the DBMS for information.

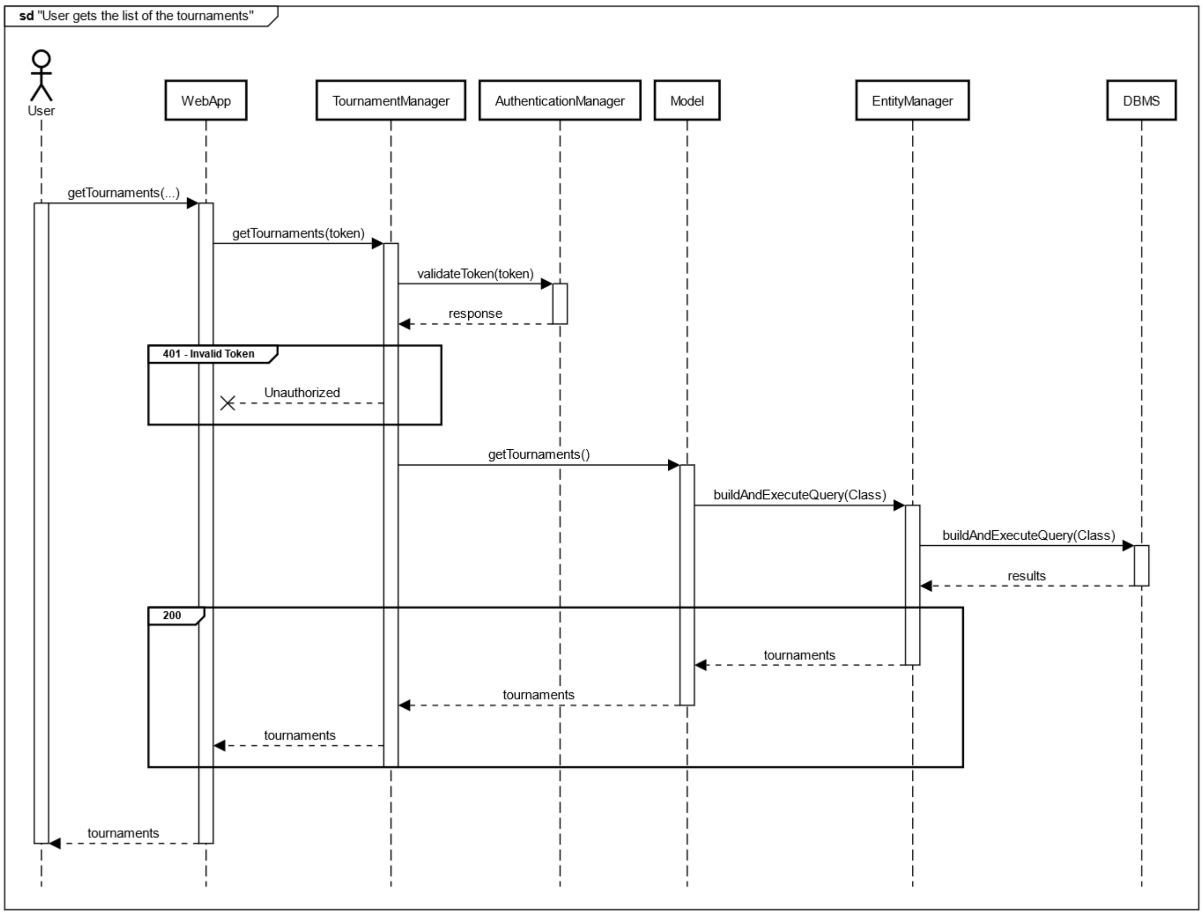


Figure 2.7: A user sees tournaments list

User sees the details of a tournament

The diagram represented down below shows the process of a User visualizing the details of a Tournament. Once authenticated, the TournamentManager communicates with the Model to fetch the Tournament data using the provided Tournament ID. This involves the EntityManager executing a query on the DBMS. If the Tournament ID is invalid, the EntityManager returns a null response, subsequently transmitted to the TournamentManager and then to the WebApp.

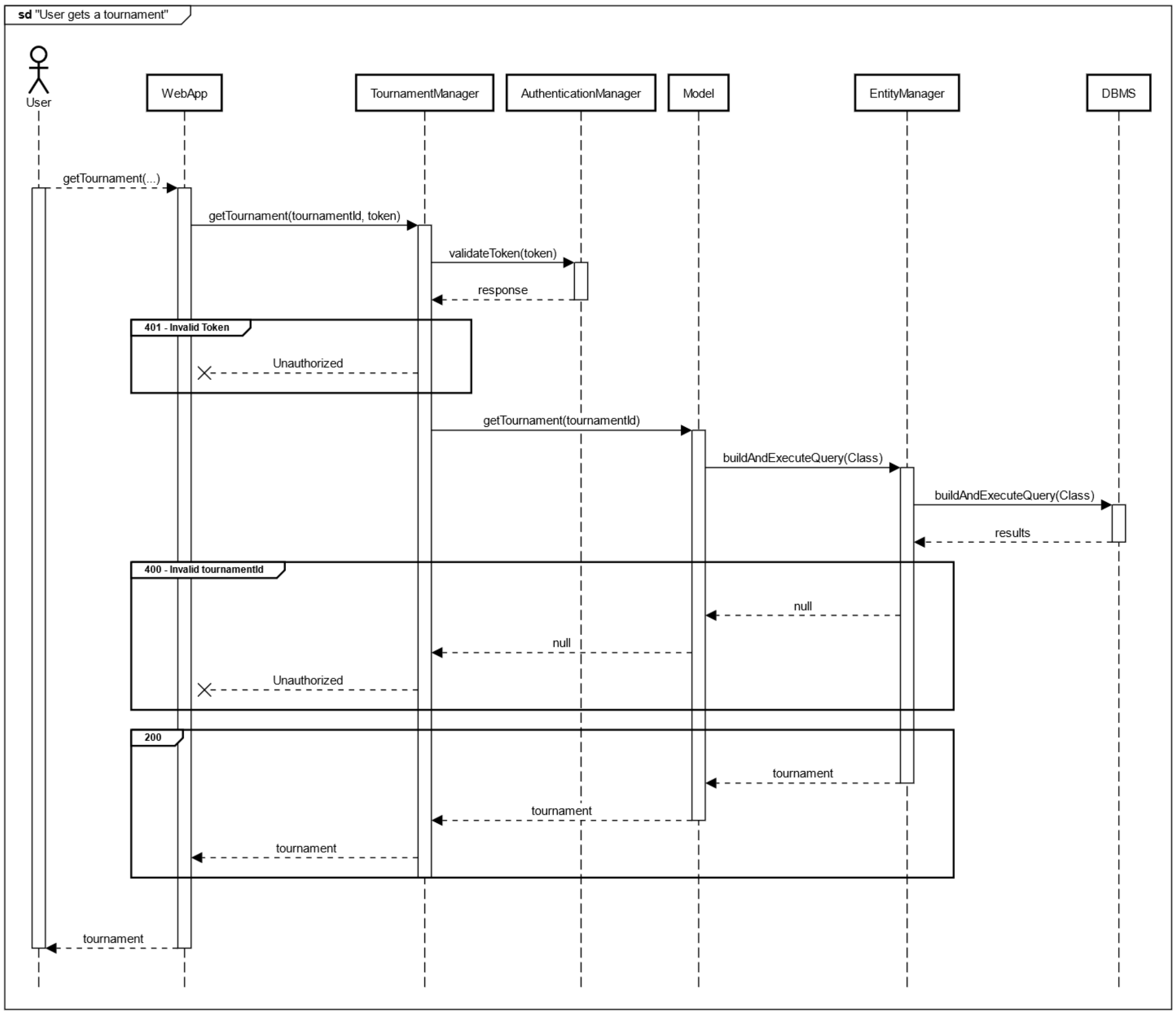


Figure 2.8: A User sees the details of a tournament

User sees the ranking of a tournament

The diagram represented down below shows the process of a User visualizing the ranking of a Tournament. Upon successful authentication, the TournamentManager communicates with the Model to retrieve the Tournament's ranking using the provided Tournament ID. This involves the EntityManager executing a query on the DBMS. If the Tournament ID is invalid, the EntityManager returns a null response, subsequently transmitted to the TournamentManager and then to the WebApp.

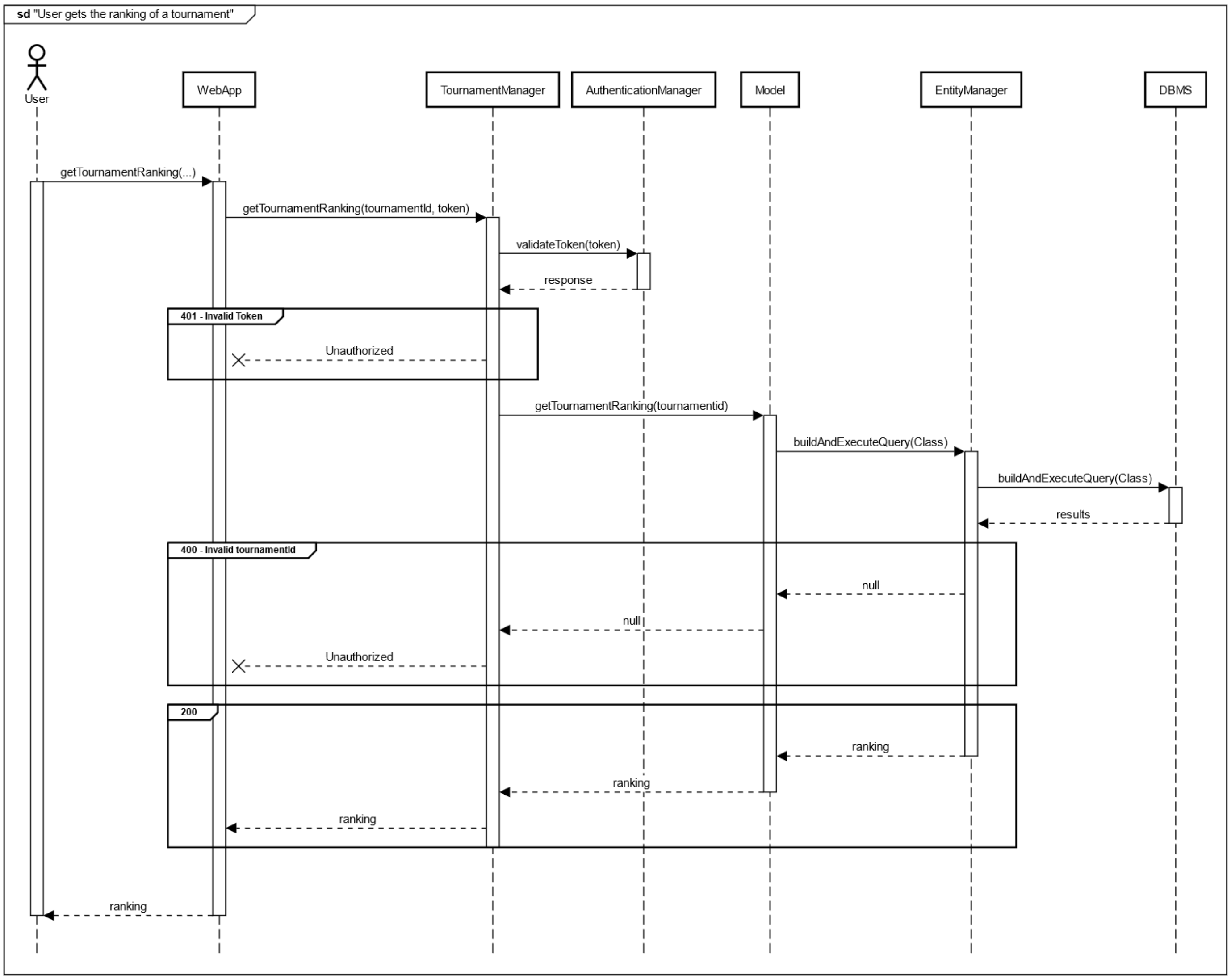

sd "User gets the ranking of a tournament"
User
WebApp
TournamentManager
AuthenticationManager
Model
EntityManager
DBMS
getTournamentRanking(...)
getTournamentRanking(tournamentId, token)
validateToken(token)
response
401 - Invalid Token
Unauthorized
getTournamentRanking(tournamentid)
buildAndExecuteQuery(Class)
buildAndExecuteQuery(Class)
results
400 - Invalid tournamentId
null
null
Unauthorized
200
ranking
ranking
ranking
ranking

Figure 2.9: A User sees the ranking of a tournament

User searches a tournament

The diagram represented down below shows the process of a User searching for a Tournament through keyword search. Following successful authentication, the TournamentManager communicates with the Model to search for Tournaments based on the provided keyword. This interaction triggers the EntityManager to execute a query on the DBMS. The Tournaments flow back through the components, starting from the EntityManager to the TournamentManager, concluding with the TournamentManager transmitting the list of Tournaments to the WebApp.

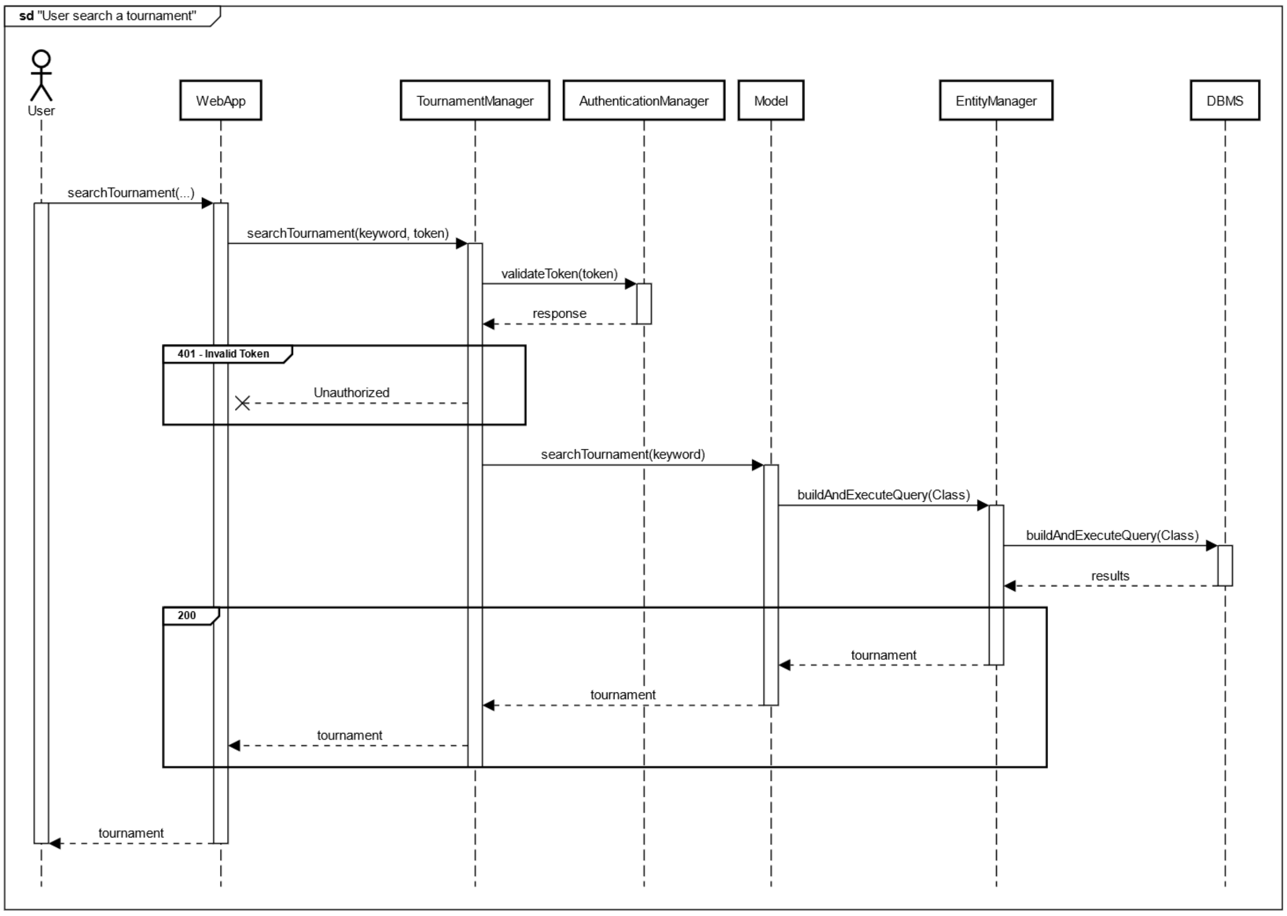


Figure 2.10: A user searches a tournaments

Educator creates a tournament

The diagram represented down below shows the process of an Educator creating a Tournament. First the Educator compiles the creation form with all the requested information, then, as the request is submitted through the proper API call, the TournamentManager handles the request. Upon successful token validation and role authorization, the TournamentManager communicates with the Model to create the Tournament using the provided data and the user's ID. This interaction triggers the EntityManager to execute a query on the DBMS. If the Tournament data is invalid, the EntityManager returns a null response, which is transmitted to the TournamentManager and subsequently to the WebApp. In the event of a successful Tournament creation, the success indication flows back through the components, concluding with the TournamentManager transmitting the success signal back to the WebApp.

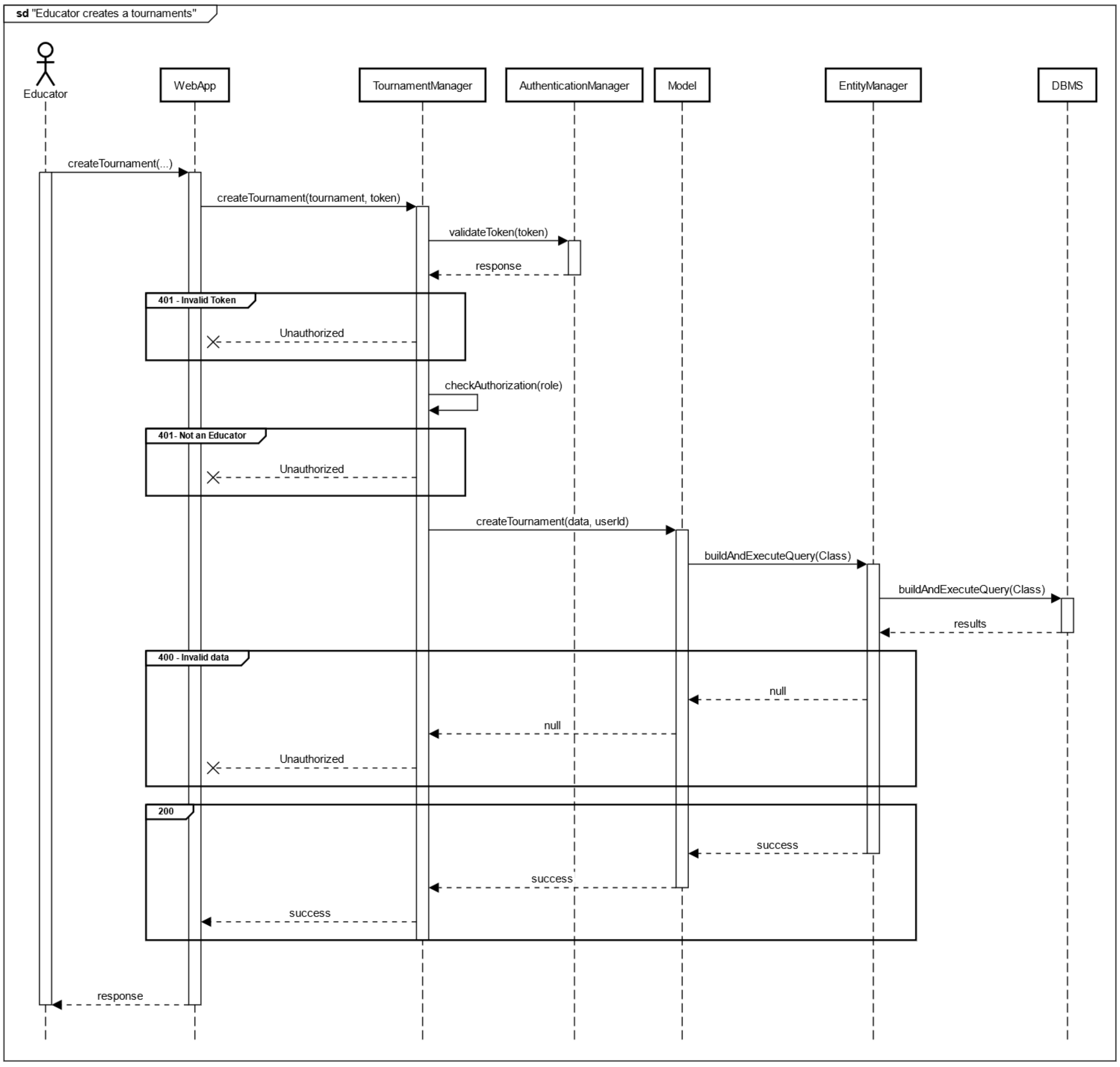


Figure 2.11: An Educator creates a tournament

Educator closes a tournament

The diagram represented down below shows the process of an Educator closing a Tournament. The WebApp triggers the closure request by sending the Tournament ID and a token to the TournamentManager. Upon successful token validation and role authorization, error handling comes into play if the provided Tournament ID is invalid, that is, if the Tournament cannot be closed due to the presence of Battles which did not end. In the event of a successful Tournament closure, the success indication travels back through the components, culminating with the TournamentManager transmitting the success signal back to the WebApp.

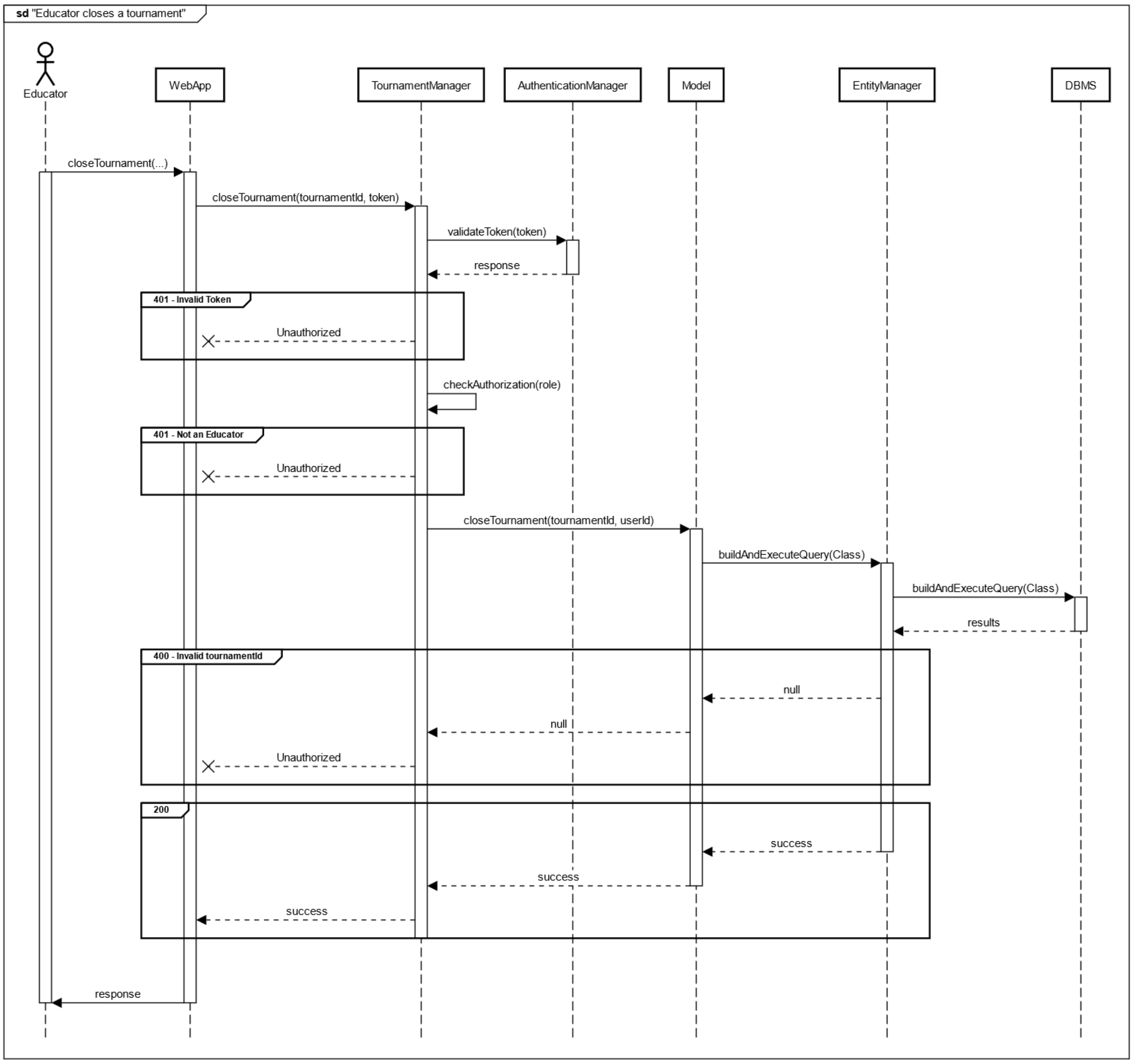


Figure 2.12: An Educator closes a tournament

Student subscribes to a tournament

The diagram represented down below shows the process of a Student enrolling into a Tournament. The WebApp triggers the request by sending the Tournament ID and a token to the TournamentManager. Upon successful token validation and role authorization, in the event of a successful Tournament enrollment, the success indication travels back through the components, culminating with the TournamentManager transmitting the success signal back to the WebApp.

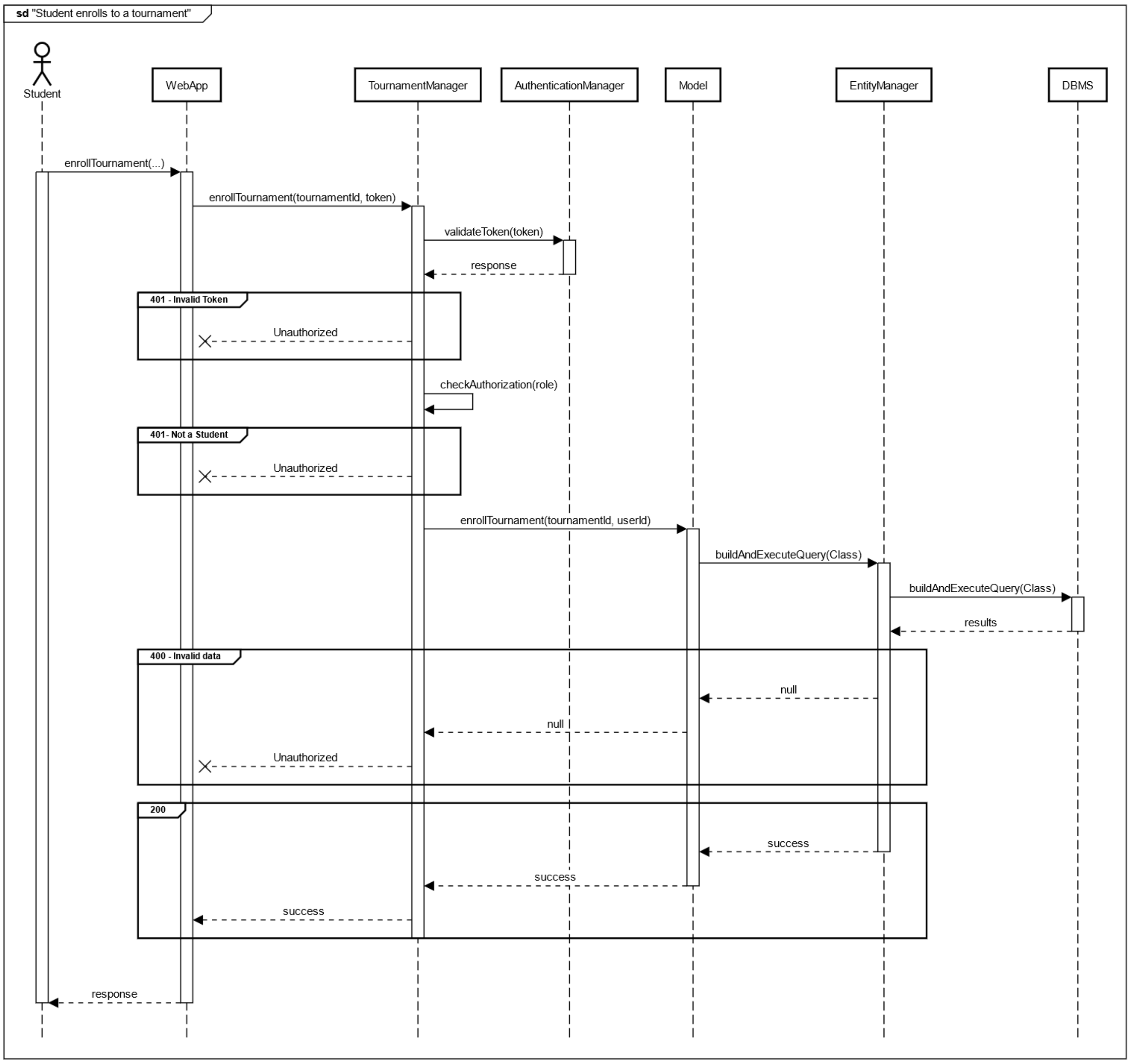


Figure 2.13: A student enrolls in a tournament

Student sees the tournaments he is enrolled in

The diagram represented down below shows the process of a Student visualizing Tournaments they are enrolled in. Upon successful authentication, the TournamentManager communicates with the Model to retrieve the Tournaments the Student is enrolled in.

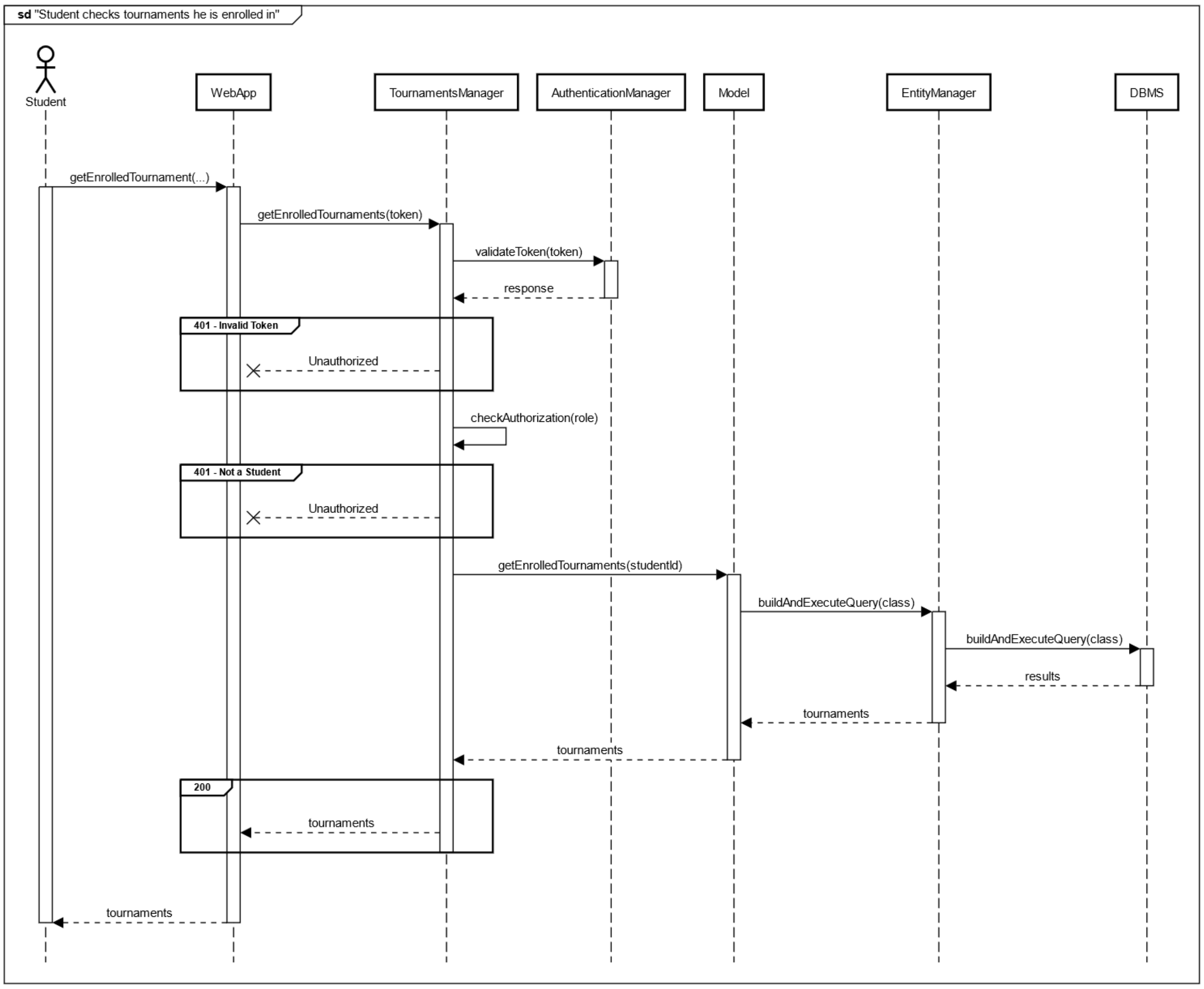


Figure 2.14: A Student sees the tournaments he is enrolled in

Educator sees the tournaments they can manage

The diagram represented down below shows the process of a Student visualizing Tournaments they can manage. The TournamentsManager communicates with the Model to retrieve created Tournaments based on the Educator's ID. This triggers a database query executed by the EntityManager. Upon receiving the results, the TournamentsManager sends the list of Tournaments back to the WebApp.

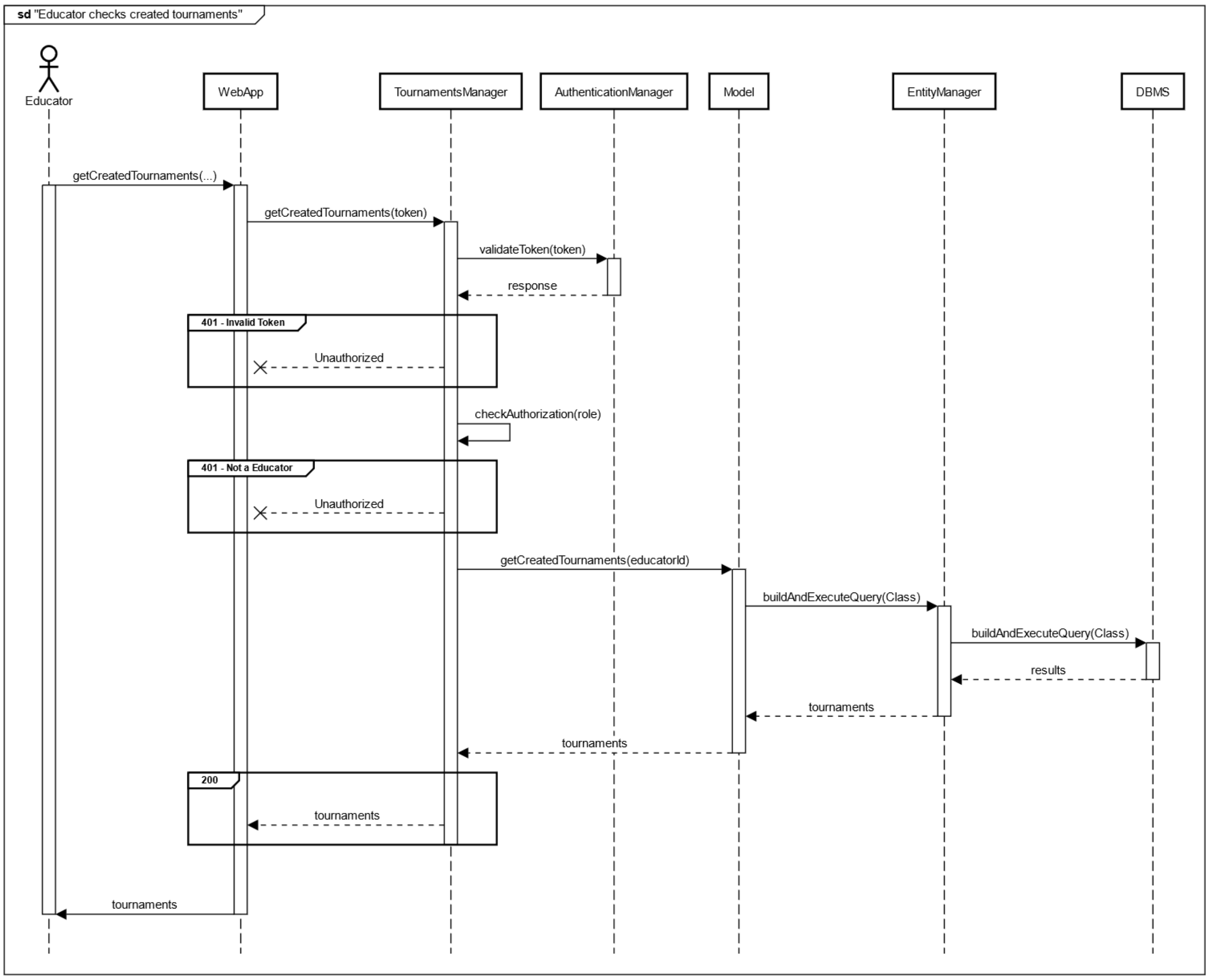


Figure 2.15: An Educator sees the tournaments he created

User sees battles in a tournament

The diagram represented down below shows the process of a User visualizing the list of all the Battles in a Tournament. It comprehends the entire history of Battles ever created in the context of a Tournament. Although, as specified in section 2.5, it is possible to retrieve only the Battles in a certain status. Upon successful token validation, the TournamentsManager communicates with the Model to retrieve the Battles linked to the specified Tournament and the User's ID. This interaction triggers the EntityManager to execute a query on the DBMS. In a successful scenario, the Battles related to the Tournament flow back through the components, concluding with the TournamentsManager transmitting the list of Battles back to the WebApp.

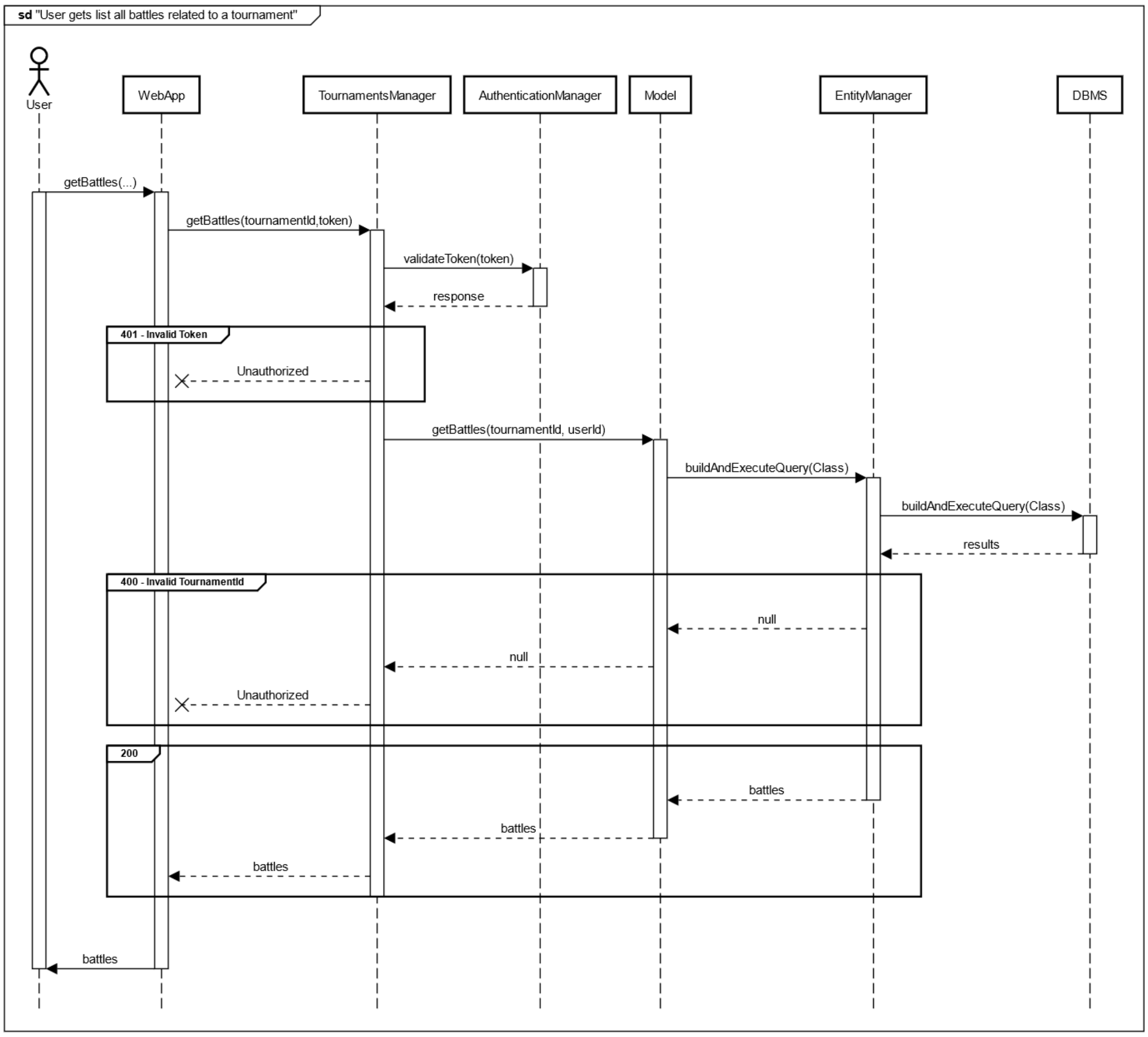


Figure 2.16: User sees battles list

User sees the details of a battle

The diagram represented down below shows the process of a User visualizing the details of a Battle. Upon successful token validation, error handling comes into play if the provided Battle ID is invalid. If such an issue arises, the EntityManager returns a null response. In a successful scenario, the details of the specific Battle are retrieved and flow back through the components, concluding with the BattlesManager transmitting the Battle details back to the WebApp.

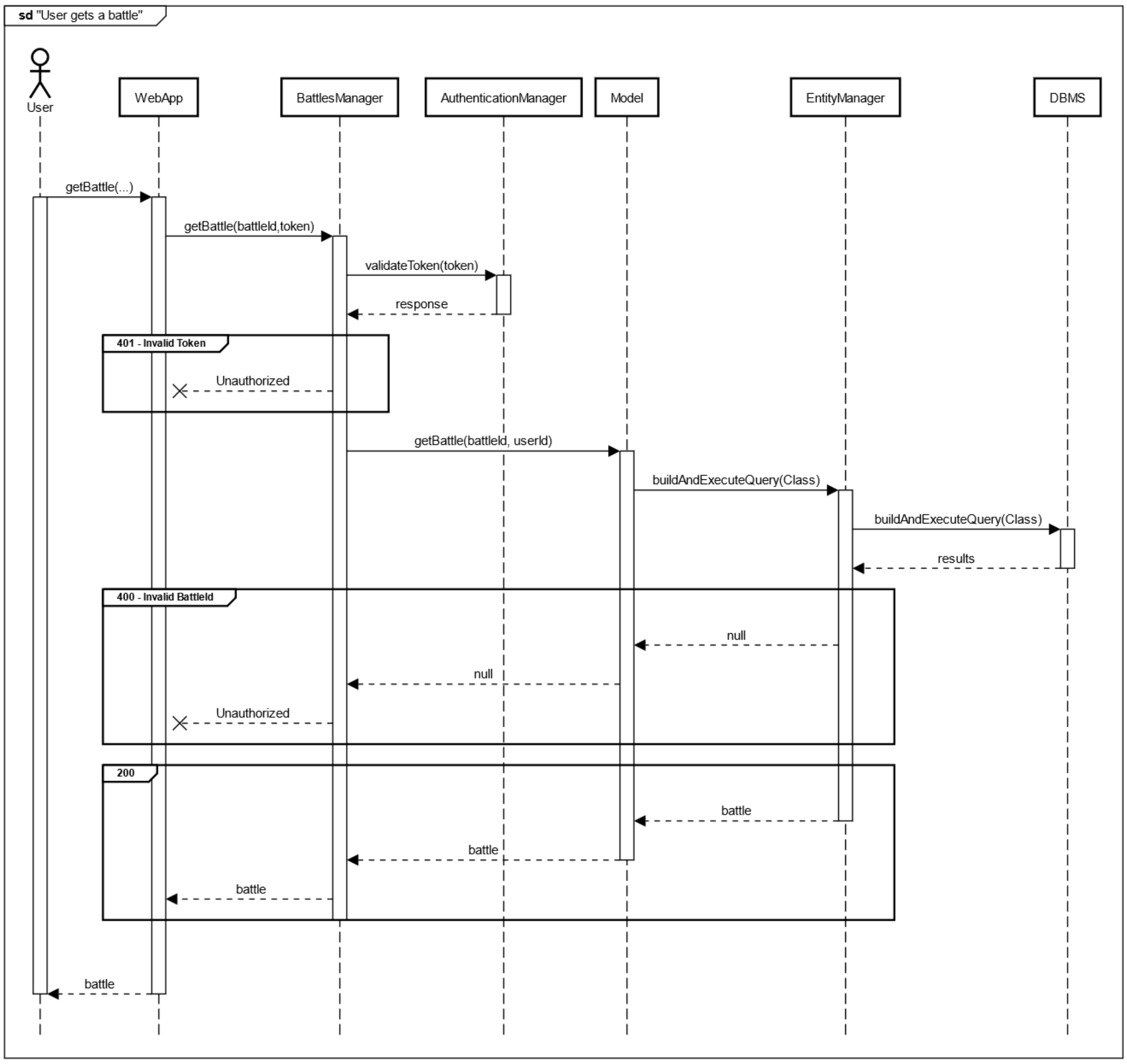


Figure 2.17: A User sees the details of a battle

Educator creates a battle

The diagram represented down below shows the process of an Educator creating a Battle in a Tournament. Upon successful token validation and role authorization, the BattleManager communicates with the Model to create the Battle using the provided data and the Educator's ID. Any invalid data triggers error handling, resulting in a null response sent back to the WebApp. In a successful scenario, the Battle creation process proceeds smoothly, signaling success as the Battle details flow back through the components, concluding with the BattlesManager transmitting the Battle details back to the WebApp.

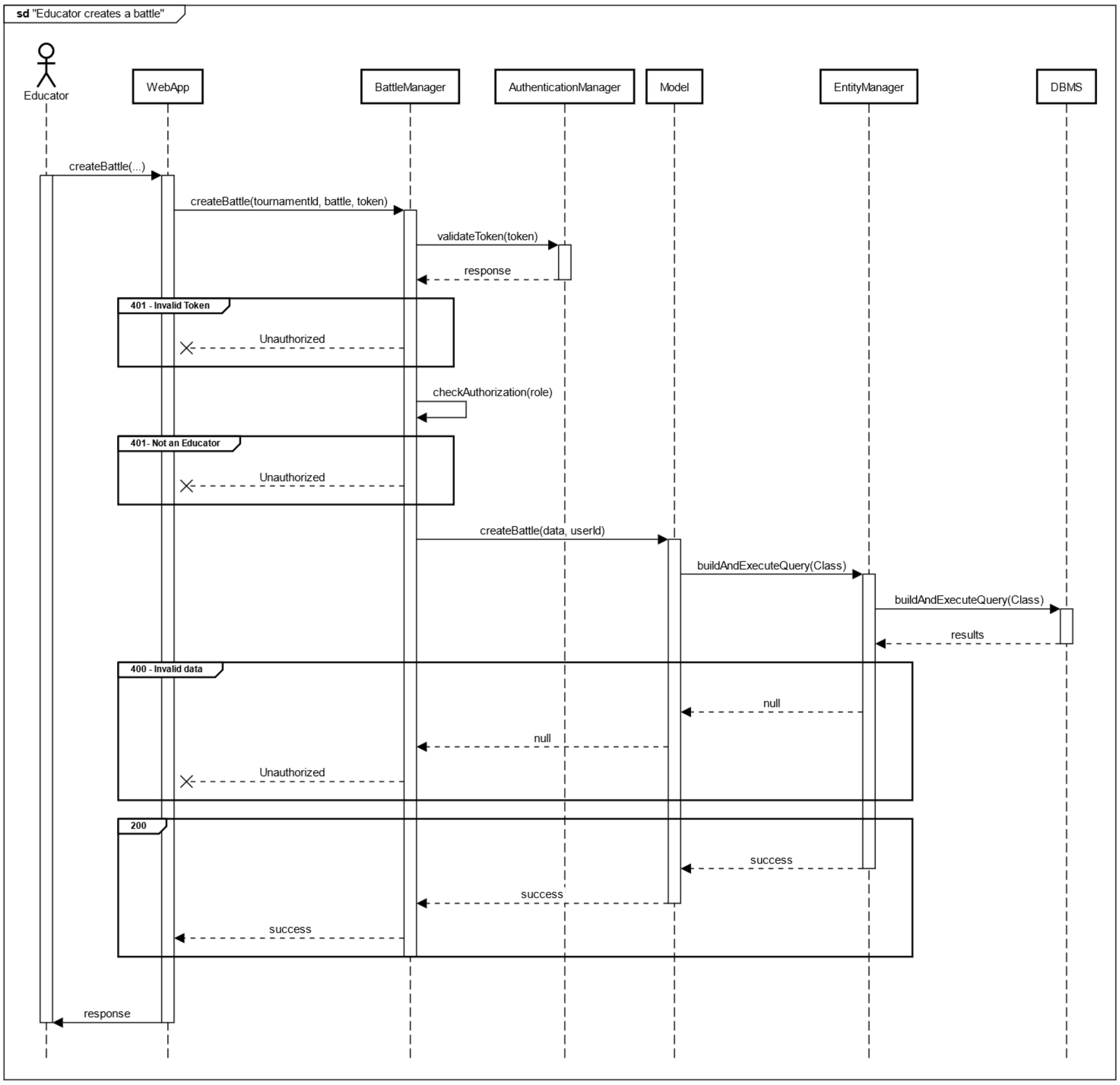


Figure 2.18: An Educator creates a battle

Educator manually corrects score of a battle

The diagram represented down below shows the process of an Educator manually assigning score to a group participating to a Battle. Notice that the score of a team for a Battle is the score obtained through the last submission. Thus, in order to modify the score of a team, the request is sent with the submission ID as a parameter. Upon successful token validation and role authorization, the BattleManager interacts with the Model to correct the score based on the provided submission ID, correction data and Educator's ID. Any invalid data triggers error handling, resulting in a null response. In a successful scenario, the score correction proceeds smoothly, and success indications flow from the EntityManager to the Model, concluding with the BattleManager transmitting the success response back to the WebApp.

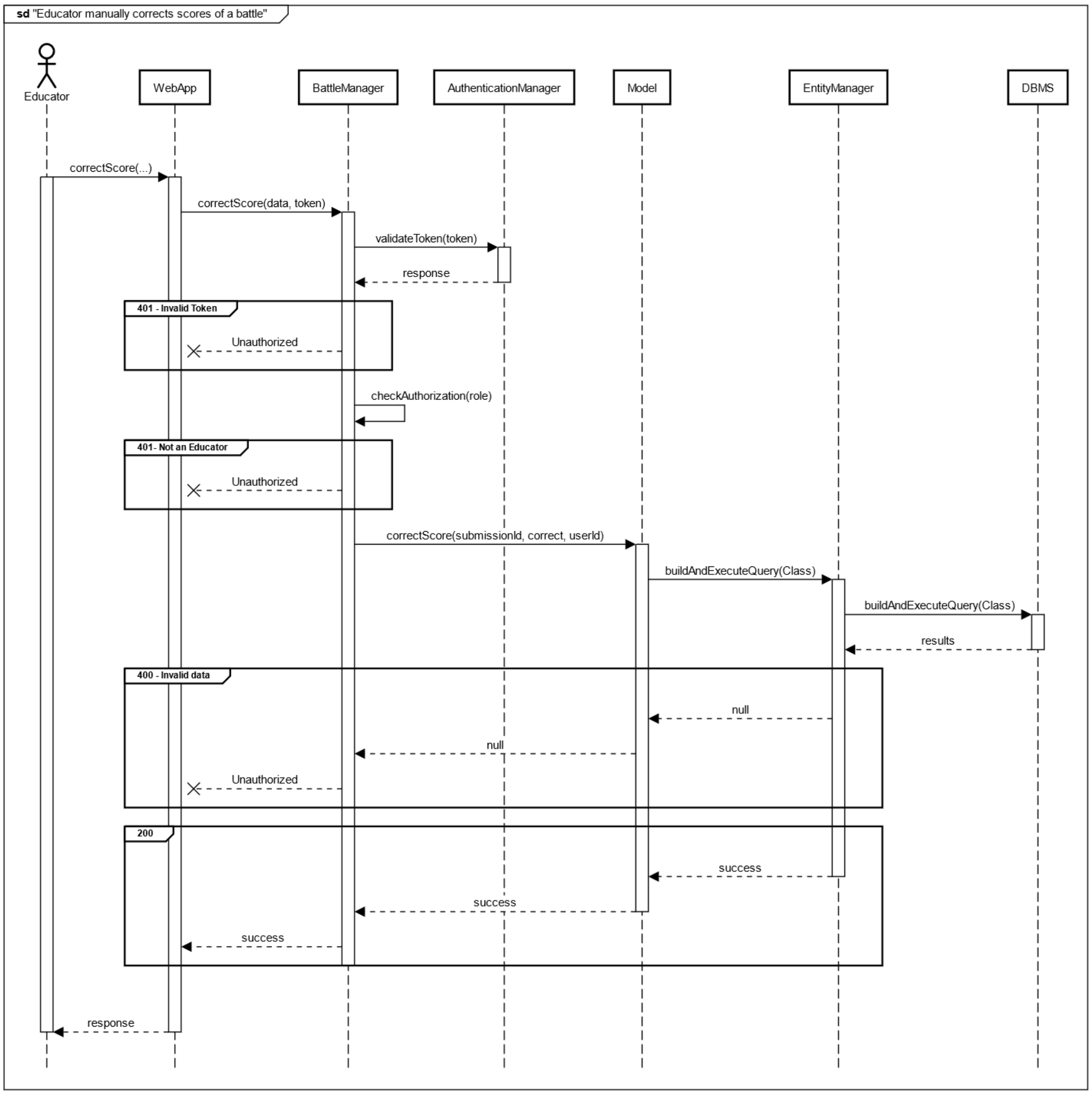


Figure 2.19: An Educator manually corrects score of a battle

Educator ends the consolidation stage of a battle

The diagram represented down below shows the process of an Educator ending the consolidation stage of a Battle. The process begins with the Educator triggering the Battle closure via the WebApp, providing the Battle ID. Upon successful token validation and role authorization, an invalid Battle ID triggers error handling, resulting in a null response. Otherwise, success indications flow from the EntityManager to the Model, concluding with the BattleManager transmitting the success response back to the WebApp.

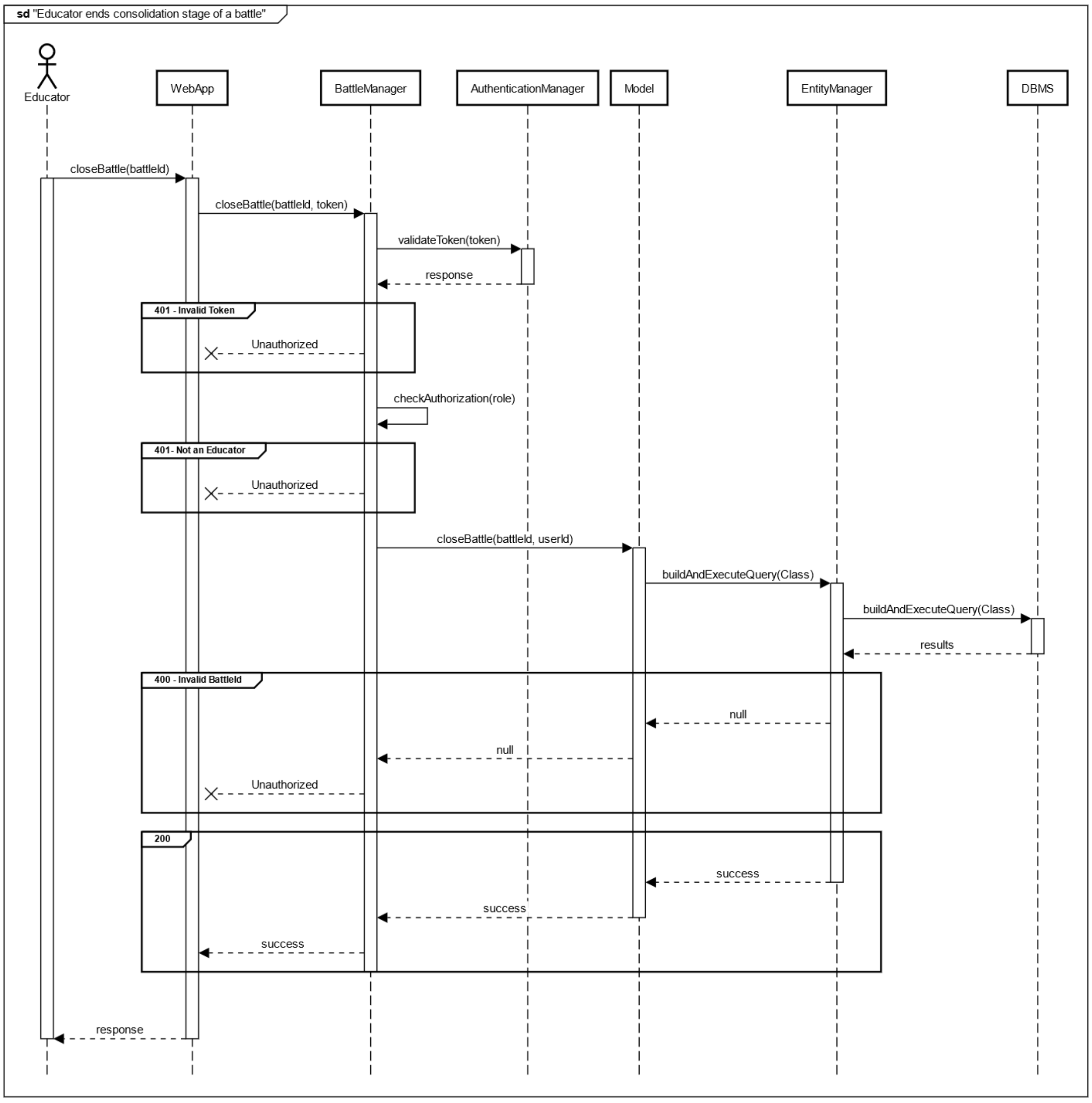


Figure 2.20: An Educator ends the consolidation stage of a battle

Student enrolls in a battle

The diagram represented down below shows the process of a Student enrolling into a Battle. The process commences as the WebApp initiates the enrollment, providing the Battle ID. Upon successful token validation and role authorization, the BattleManager interacts with the Model to enroll the Student in the specified Battle using the Battle ID and Student's ID. An invalid Battle ID triggers error handling, resulting in a null response. In a successful scenario, the enrollment proceeds smoothly, and success indications flow from the EntityManager to the Model, concluding with the BattleManager transmitting the success response back to the WebApp.

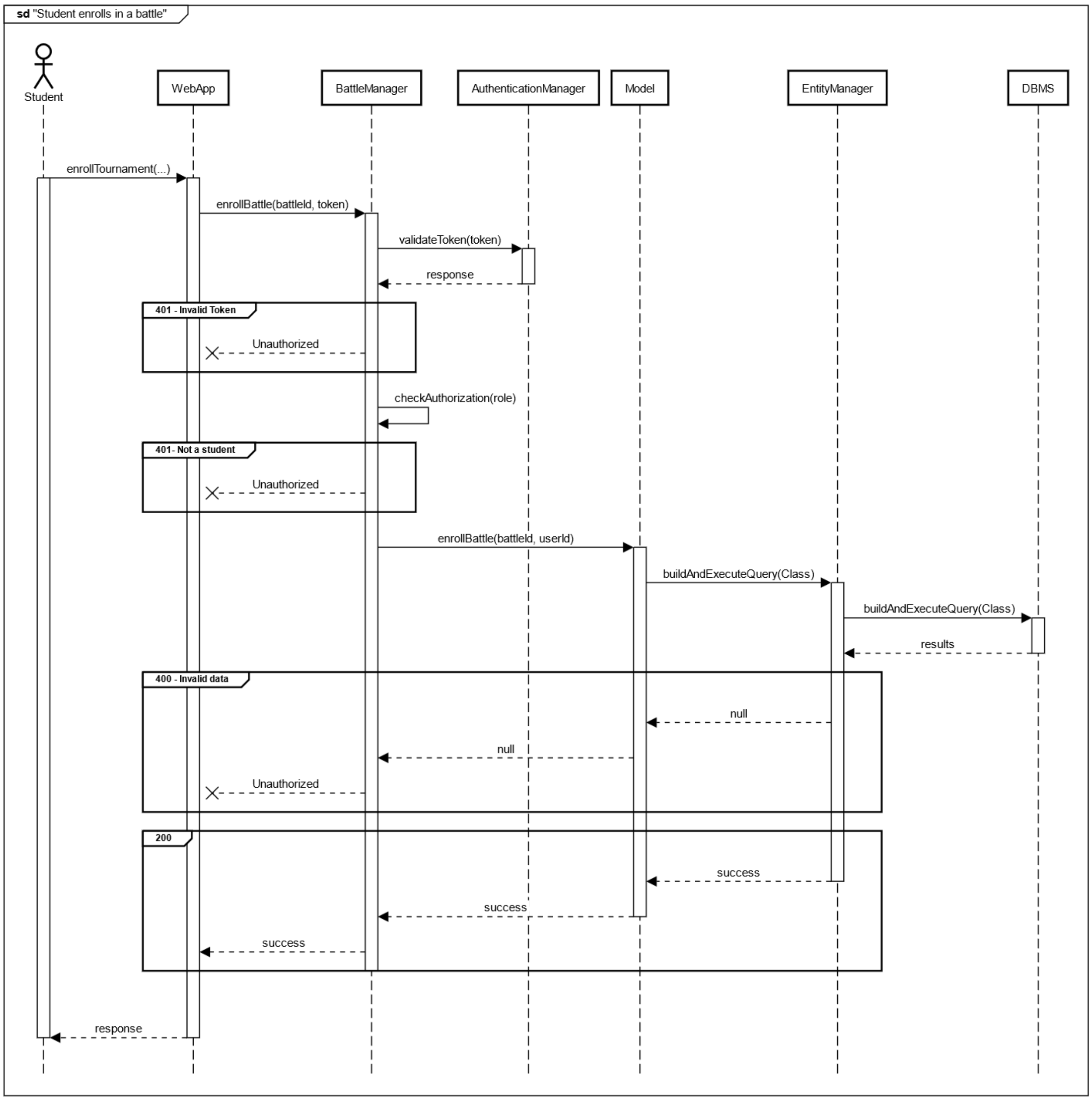


Figure 2.21: A Student enrolls in a battle

User sees the ranking of a battle

The diagram represented down below shows the process of a Student visualizing the ranking of a Battle. It begins as the WebApp initiates a request for the Battle's ranking, providing the Battle ID. Upon successful authentication, the BattleManager retrieves the Battle's ranking from the Model based on the provided Battle ID and User ID. An invalid Battle ID triggers error handling, resulting in a null response. In a successful scenario, the ranking flows from the EntityManager to the Model, concluding with the BattleManager transmitting the ranking to the WebApp.

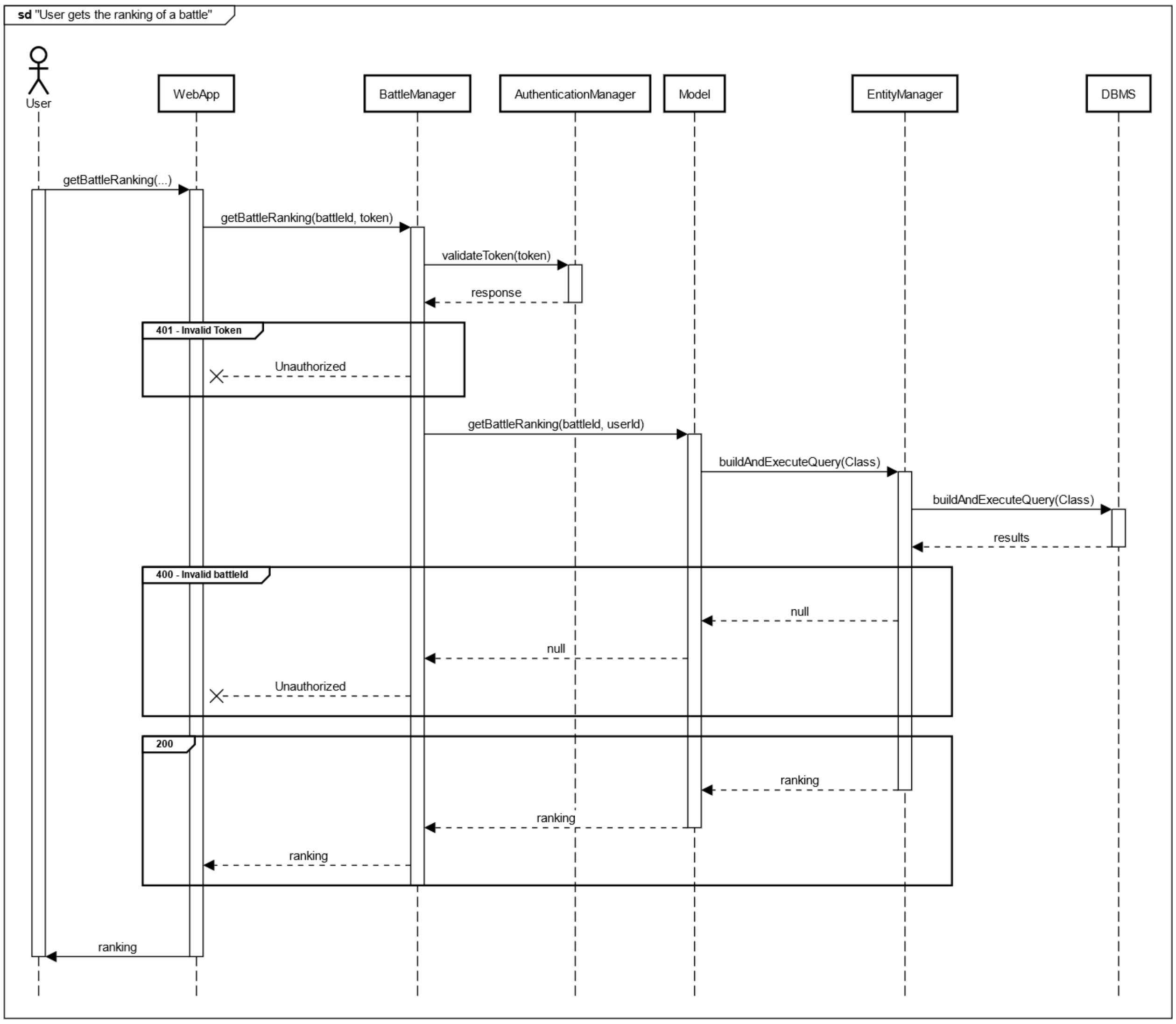


Figure 2.22: A User sees the ranking of a battle

User sees the submissions of a battle

The diagram represented down below shows the process of a Student visualizing the submissions performed by their team during the Battle. It begins as the WebApp initiates a request for the Battle's ranking, providing the Battle ID. Upon successful authentication, the BattleManager retrieves the Battle's submissions of the team from the Model based on the provided Battle ID and User ID. An invalid Battle ID triggers error handling, resulting in a null response. In a successful scenario, the ranking flows from the EntityManager to the Model, concluding with the BattleManager transmitting the submissions to the WebApp.

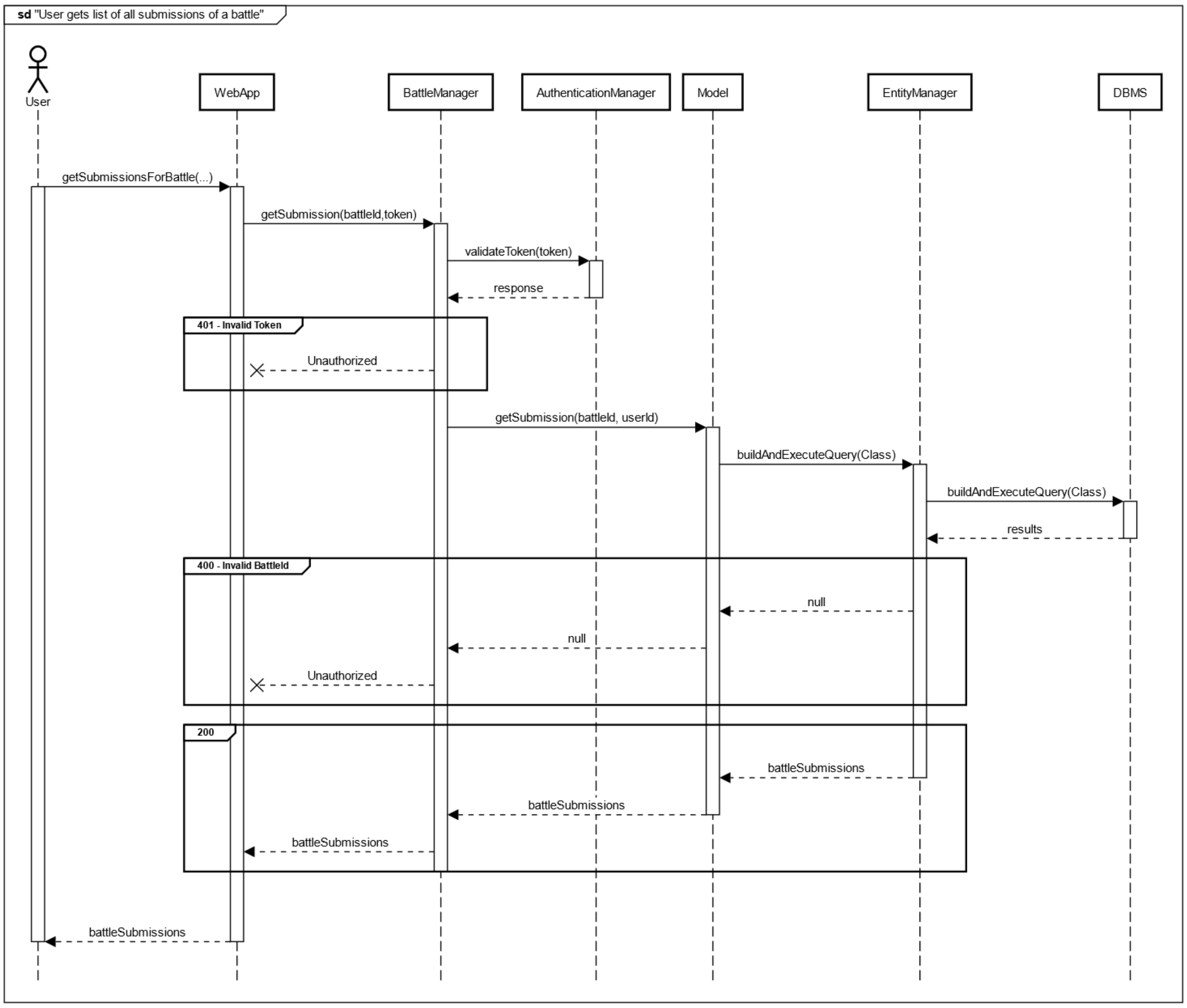


Figure 2.23: A User sees the submissions of a battle

Student sees the battles they are enrolled in within a tournament

The diagram represented down below shows the process of a Student visualizing the Battles they are enrolled in within the context of a certain Tournament. It begins as the WebApp initiates a request for Battles, providing the Tournament ID. Upon successful token validation and role authorization, the BattleManager retrieves the Battles the Students are enrolled in from the Model based on the provided Tournament ID. An invalid Tournament ID triggers error handling, resulting in a null response. In a successful scenario, the Battles flow from the EntityManager to the Model, concluding with the BattleManager transmitting the submissions to the WebApp.

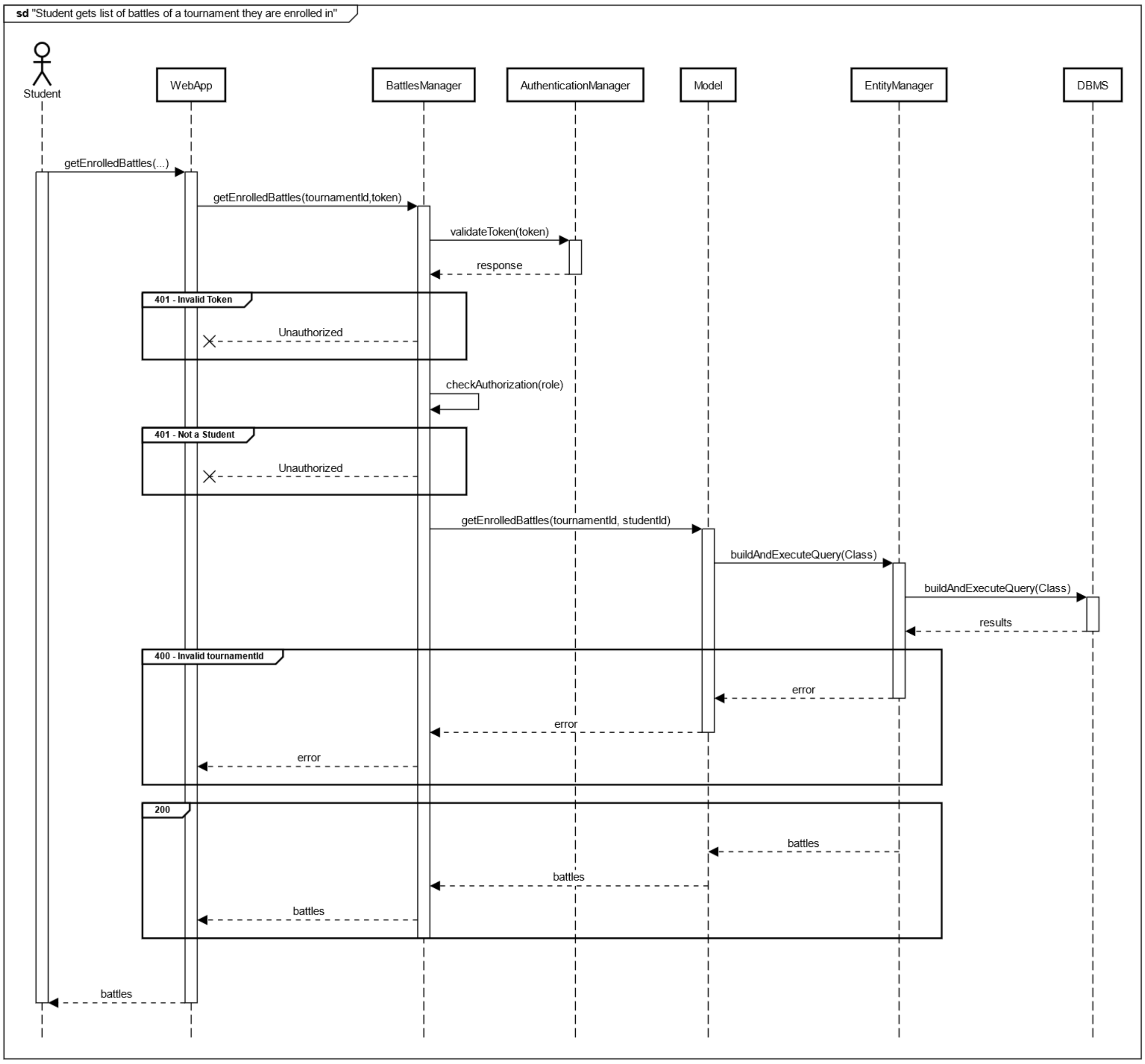


Figure 2.24: Student sees the battles they are enrolled in within a tournament

Educator sees the battles of a tournament they can manage

The diagram represented down below shows the process of an Educator visualizing the list of Battles within the context of a Tournament that they can manage. It begins as the WebApp initiates a request for the Battles, providing the Tournament ID. Upon successful authentication, the BattleManager retrieves the Battles from the Model based on the provided Tournament

ID. An invalid Tournament ID triggers error handling, resulting in a null response. In a successful scenario, the Battles flow from the EntityManager to the Model, concluding with the BattleManager transmitting the submissions to the WebApp.

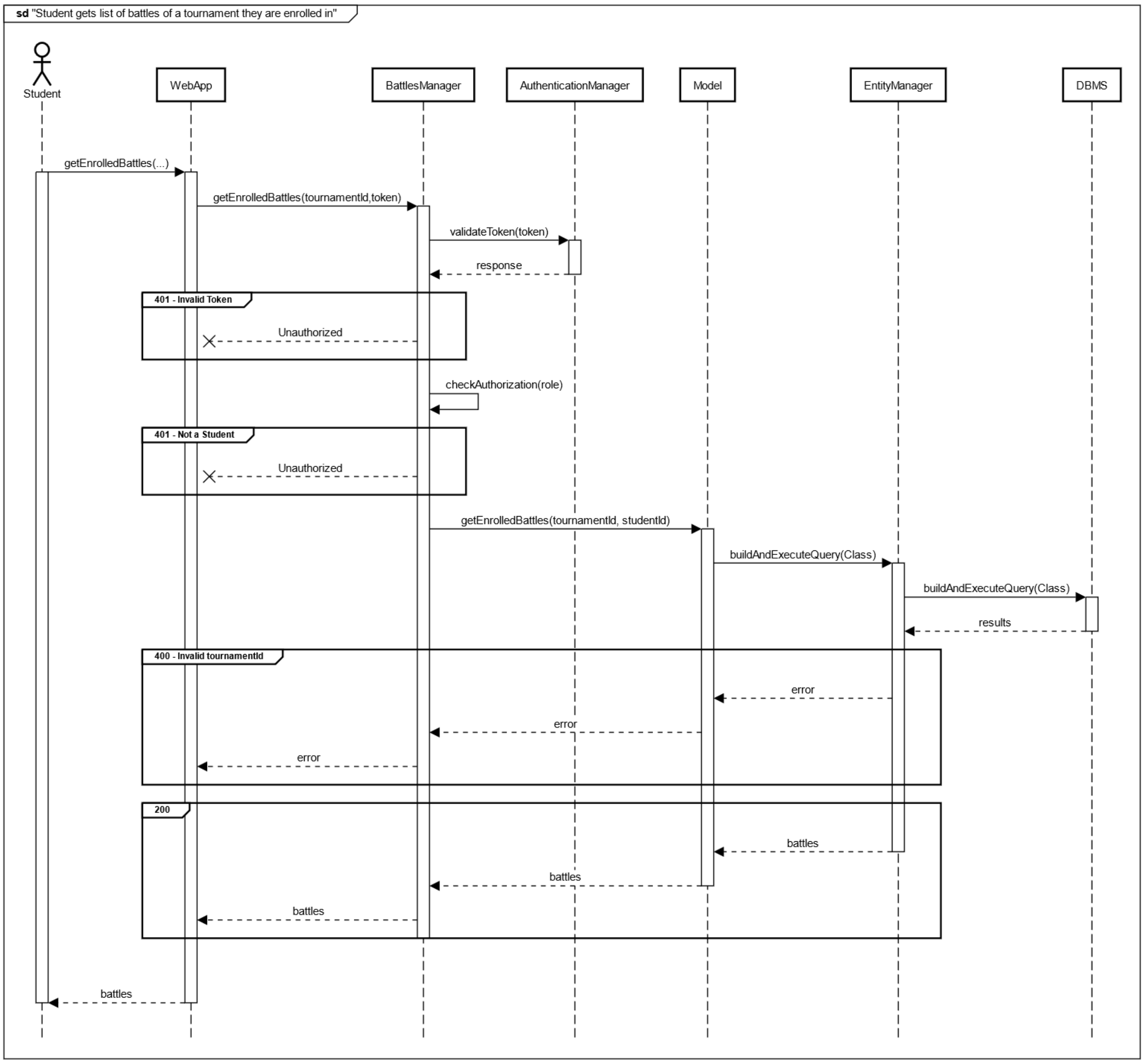


Figure 2.25: An Educator sees the battles of a tournament they can manage

GitHub Notifies CKB of a new push in a main branch

The diagram represented down below shows the process of a new submission being performed. In this scenario, GitHub asynchronously sends a POST HttpRequest to the system, subsequently handled by the BattleManager. Upon successful token validation, the BattleManager registers the commit within the Model, utilizing the provided commit data and the User ID associated with the action. This interaction triggers the EntityManager to execute a query on the DBMS to store the commit information. Error handling mechanisms are in place to manage instances of invalid or missing data. If such issues occur, the EntityManager returns a null response, otherwise success indications flow from the EntityManager to the Model, concluding with the BattleManager transmitting the success response back to the WebApp.

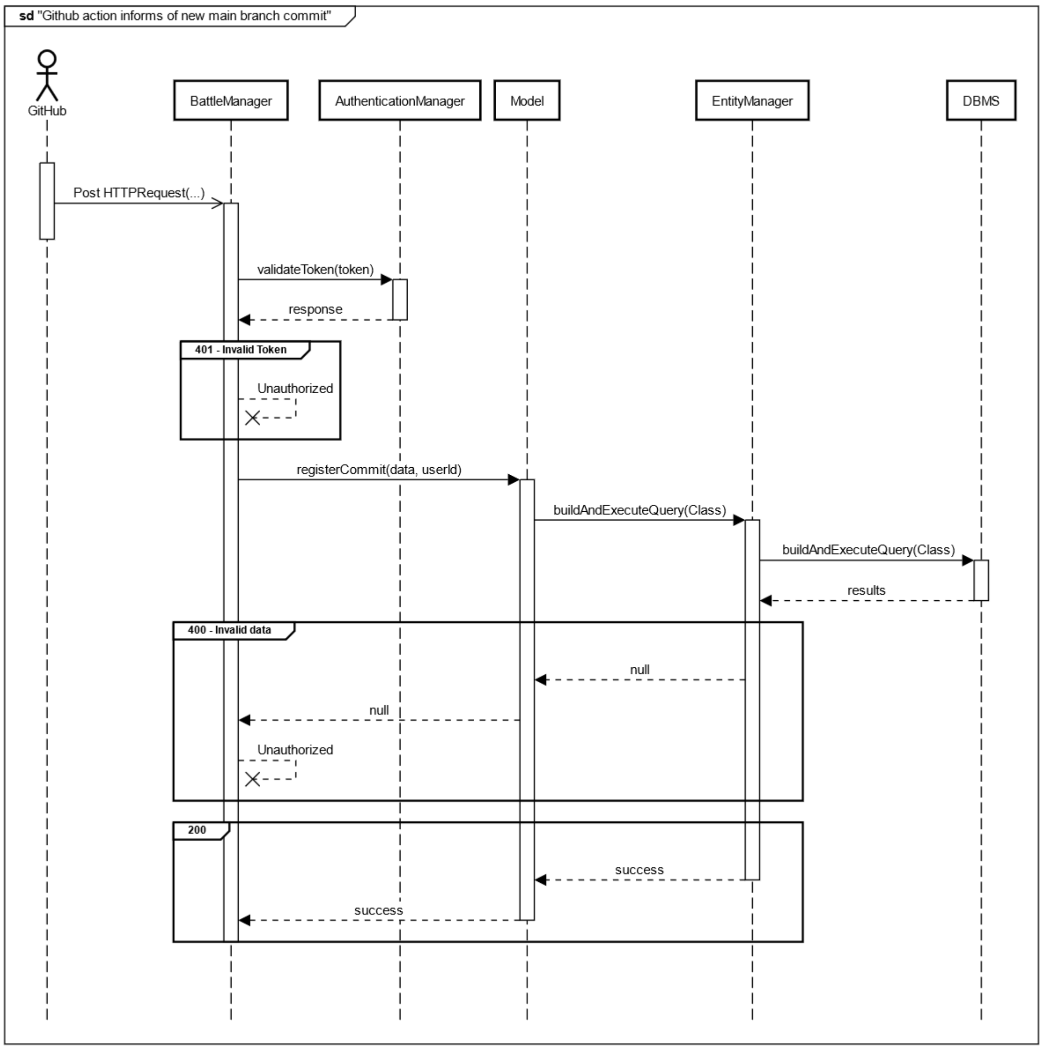


Figure 2.26: GitHub Notifies CKB of a new push in a main branch

User checks a profile details

The diagram represented down below shows the process of a User visualizing the profile of a Student. Upon successful token validation, error handling comes into play if the provided Student ID is invalid. If such an issue arises, the EntityManager returns a null response. In a successful scenario, the details of the Student are retrieved and flow back through the components, concluding with the BattlesManager transmitting the Battle details back to the WebApp.

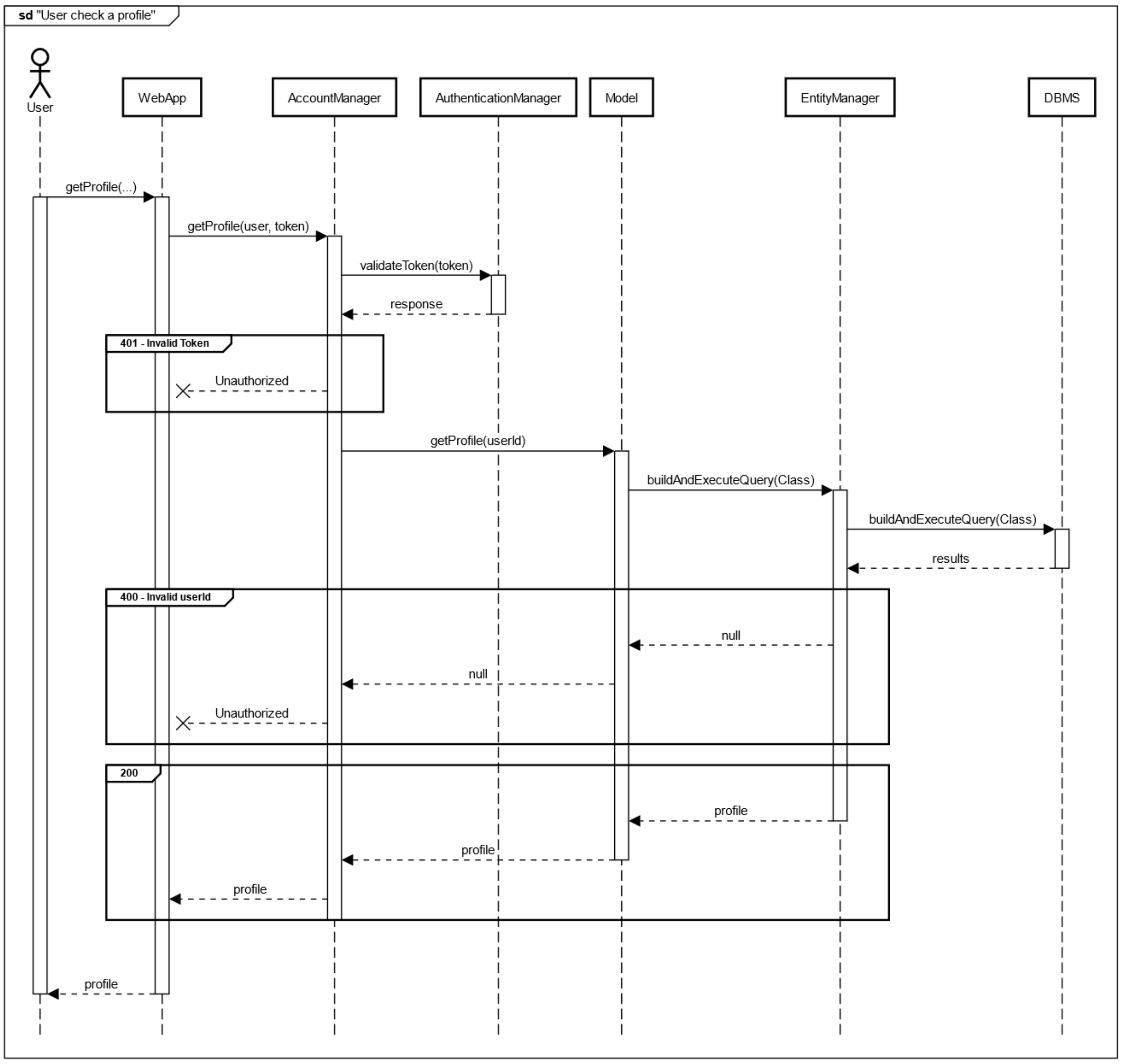


Figure 2.27: A User checks a profile details

User searches for a profile

The diagram represented down below shows the process of a User searching for the profile of a Student through keyword search. Upon successful token validation, the list of Students is retrieved and flow back through the components, concluding with the BattlesManager transmitting the list to the WebApp.

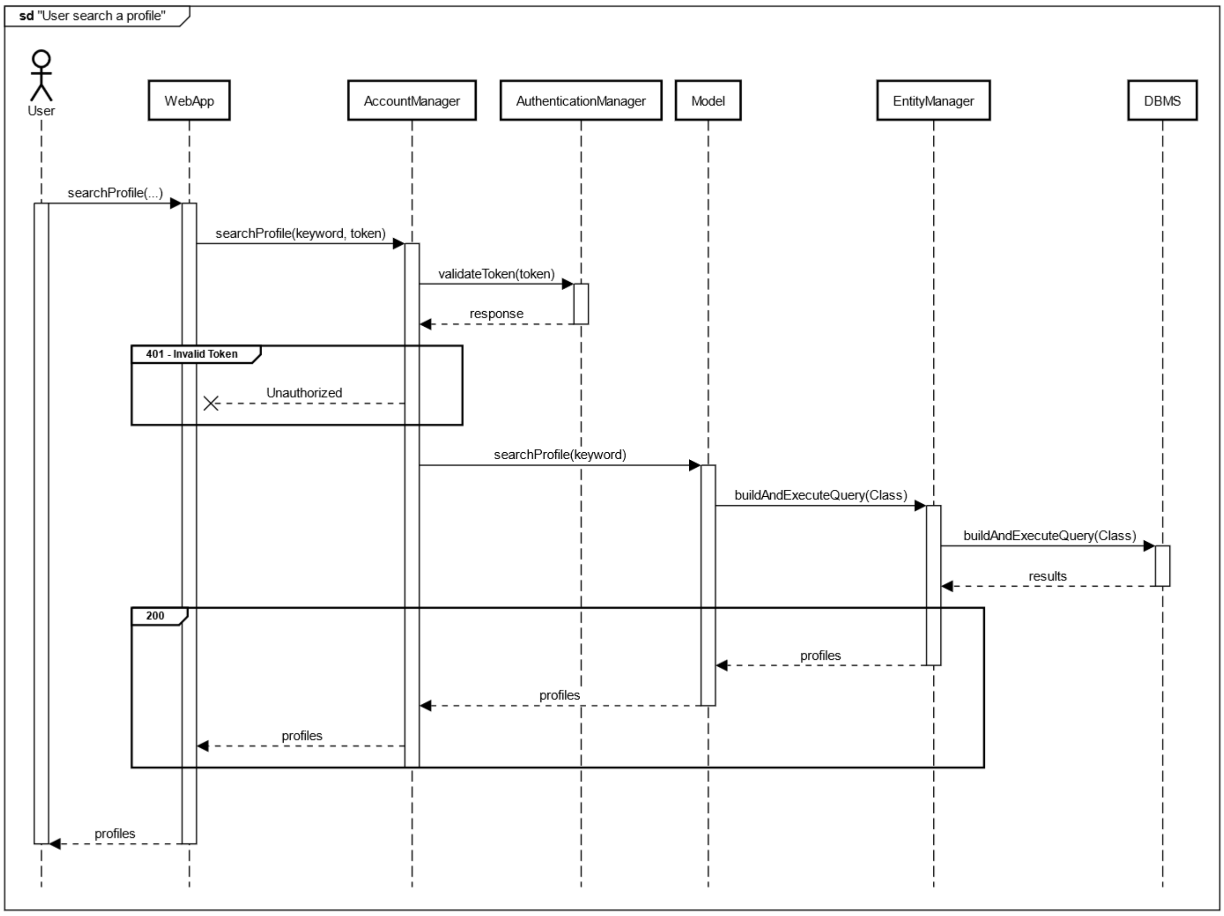


Figure 2.28: A User searches for a profile

User updates its profile

The diagram represented down below shows the process of a User update their own profile with new data. Upon successful token validation, the AccountManager interacts with the Model to update the profile information based on the provided user data. This interaction triggers the EntityManager to execute a query on the DBMS to modify the necessary information. Error handling mechanisms are implemented to manage cases where invalid data is provided. If such an issue arises, the EntityManager returns a null response. In a successful scenario, the profile information is updated successfully. Indications of success flow from the EntityManager to the Model, concluding with the AccountManager transmitting the success response back to the WebApp.

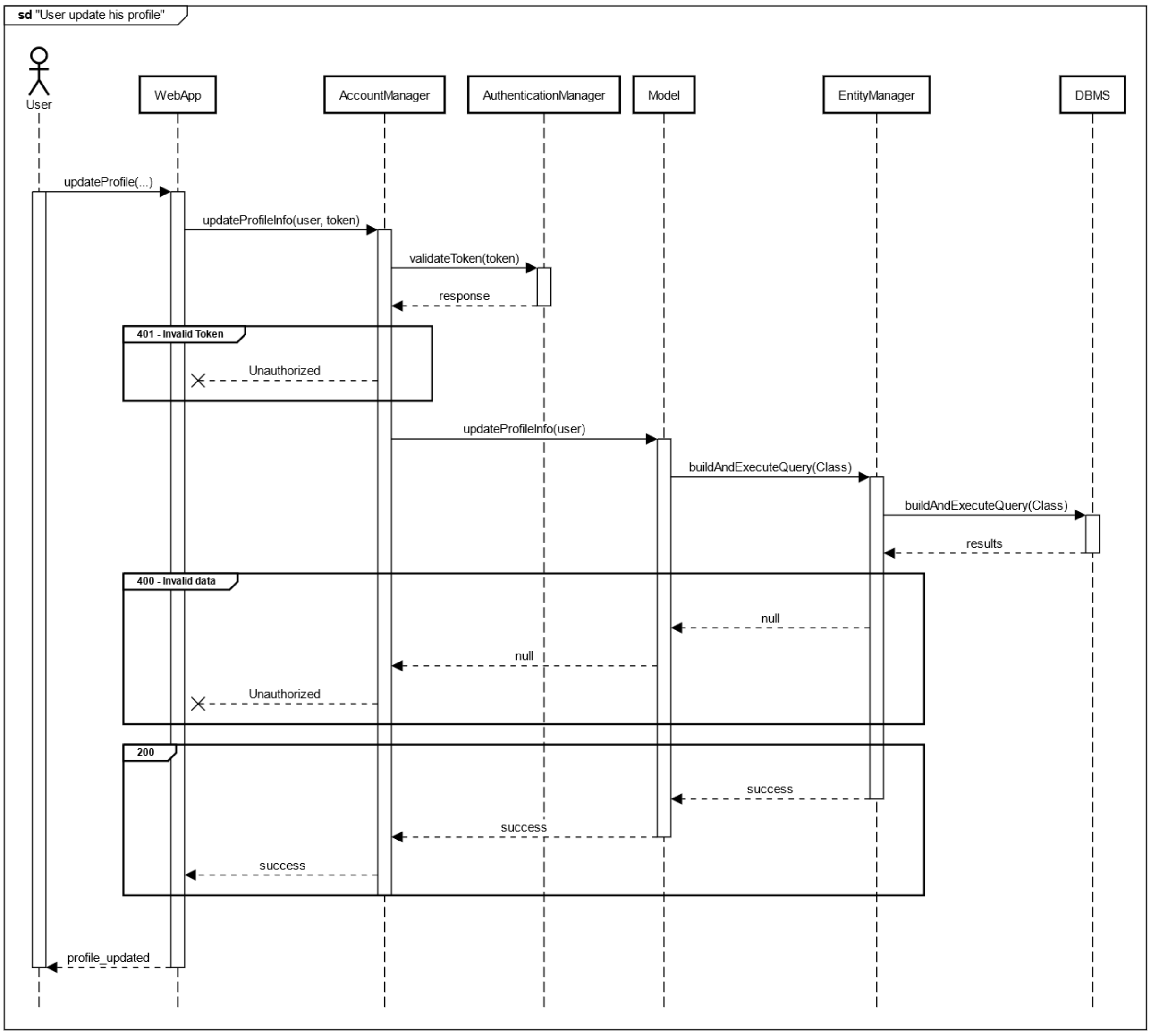


Figure 2.29: A User updates its profile

2.5. Component Interfaces

In this section, we use a class diagram to illustrate the component interfaces. The methods are presented as clearly as possible, and we expect their functionality to be easily understandable.

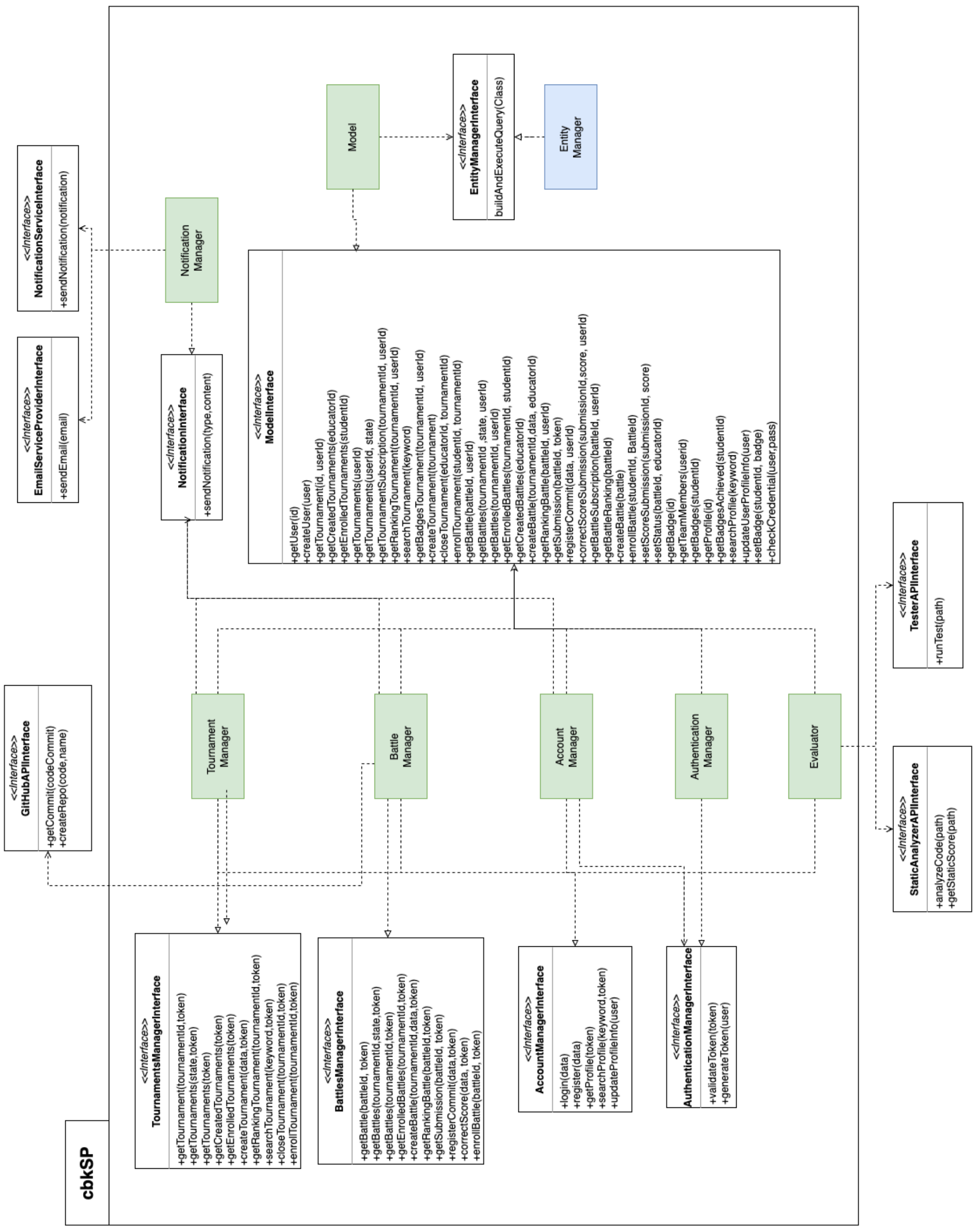

cbkSP
<<Interface>> NotificationServiceInterface
+sendNotification(notification)
<<Interface>> EmailServiceProviderInterface
+sendEmail(email)
<<Interface>> GitHubAPIInterface
+getCommit(codeCommit)
+createRepo(code,name)
Notification Manager
<<Interface>> NotificationInterface
+sendNotification(type,content)
Model
<<Interface>> EntityManagerInterface
buildAndExecuteQuery(Class)
Entity Manager
<<Interface>> ModelInterface
+getUser(id)
+createUser(user)
+getTournament(id, userId)
+getCreatedTournaments(educatorId)
+getEnrolledTournaments(studentId)
+getTournaments(userId)
+getTournaments(userId, state)
+getTournamentSubscription(tournamentId, userId)
+getRankingTournament(tournamentId, userId)
+searchTournament(keyword)
+getBadgesTournament(tournamentId, userId)
+createTournament(tournament)
+closeTournament(educatorId, tournamentId)
+enrollTournament(studentId, tournamentId)
+getBattle(battleId, userId)
+getBattles(tournamentId ,state, userId)
+getBattles(tournamentId, userId)
+getEnrolledBattles(tournamentId, studentId)
+getCreatedBattles(educatorId)
+createBattle(tournamentId,data, educatorId)
+getRankingBattle(battleId, userId)
+getSubmission(battleId, token)
+registerCommit(data, userId)
+correctScoreSubmission(submissionId,score, userId)
+getBattleSubscription(battleId, userId)
+getBattleRanking(battleId)
+createBattle(battle)
+enrollBattle(studentId, BattleId)
+setScoreSubmission(submissionId, score)
+setStatus(battleId, educatorId)
+getBadge(id)
+getTeamMembers(userid)
+getBadges(studentId)
+getProfile(id)
+getBadgesAchieved(studentId)
+searchProfile(keyword)
+updateUserProfileInfo(user)
+setBadge(studentId, badge)
+checkCredential(user,pass)
Tournament Manager
Battle Manager
Account Manager
Authentication Manager
Evaluator
<<Interface>> TesterAPIInterface
+runTest(path)
<<Interface>> StaticAnalyzerAPIInterface
+analyzeCode(path)
+getStaticScore(path)
<<Interface>> TournamentsManagerInterface
+getTournament(tournamentId,token)
+getTournaments(state,token)
+getTournaments(token)
+getCreatedTournaments(token)
+getEnrolledTournament(token)
+createTournament(data,token)
+getRankingTournament(tournamentId,token)
+searchTournament(keyword,token)
+closeTournament(tournamentId,token)
+enrollTournament(tournamentId,token)
<<Interface>> BattlesManagerInterface
+getBattle(battleId, token)
+getBattles(tournamentId,state,token)
+getBattles(tournamentId,token)
+getEnrolledBattles(tournamentId,token)
+createBattle(tournamentId,data,token)
+getRankingBattle(battleId,token)
+getSubmission(battleId, token)
+registerCommit(data,token)
+correctScore(data, token)
+enrollBattle(battleId, token)
<<Interface>> AccountManagerInterface
+login(data)
+register(data)
+getProfile(token)
+searchProfile(keyword,token)
+updateProfileInfo(user)
<<Interface>> AuthenticationManagerInterface
+validateToken(token)
+generateToken(user)

Figure 2.30: Class diagram with interfaces of the CodeKataBattle System

2.5.1. API Endpoints

In this section the API endpoints are presented. The focus is on the method used, on the parameters required and on the response.

POST api/user/login

Request Body: User: Object

200 Response: UserObject + token

400 Response: String ("eMail not valid, "Password must contain at least 8 characters (at least one uppercase), at least 1 special symbol and at least 1 number")

401 Response: error: String ("Email or Password are wrong")

410 Response: error: String ("User not verified. Register a new account.")

POST api/user/register

Request Body: User: Object

200 Response: String("User registered")

400 Response: error :String("Invalid parameters")

GET api/tournaments

Request Parameters: state: String {optional}

200 Response: ongoingTournaments: Array of Objects

401 Response: error: String ("Unauthorized")

GET api/tournaments/created

200 Response: createdTournaments: Array of Objects

400 Response: error: String ("Invalid or missing tournamentId")

401 Response: error: String ("Unauthorized")

GET api/tournaments/enrolled

200 Response: enrolledTournaments: Array of Objects

400 Response: error: String (“Invalid or missing tournamentId”)

401 Response: error: String (“Unauthorized”)

POST api/tournaments/enroll

Request Body: tournamentId: int, token : String

200 Response: String("User enrolled")

400 Response: error :String("Invalid parameters"

401 Response: error: String (“Unauthorized”)

POST api/battles/enroll

Request Body: BattlesId: int, token : String

200 Response: String("User enrolled")

400 Response: error :String("Invalid parameters"

401 Response: error: String (“Unauthorized”)

GET api/tournaments/{tournamentId}

200 Response: tournament: Object

400 Response: error: String (“Invalid or missing tournamentId”)

401 Response: error: String (“Unauthorized”)

GET api/tournaments/{tournamentId}/battles

Request Parameters: state: String {optional}

200 Response: battles: Array of Objects

400 Response: error: String (“Invalid or missing tournamentId”)

401 Response: error: String (“Unauthorized”)

GET api/tournaments/{tournamentId}/battles/created

200 Response: createdBattles: Array of Objects

400 Response: error: String (“Invalid or missing tournamentId”)

401 Response: error: String (“Unauthorized”)

GET api/tournaments/{tournamentId}/battles/enrolled

200 Response: enrolledBattles: Array of Objects

400 Response: error: String (“Invalid or missing tournamentId”)

401 Response: error: String (“Unauthorized”)

GET api/tournaments/{tournamentId}/ranking

200 Response: ranking: Array of Objects

400 Response: error: String (“Invalid or missing tournamentId”)

401 Response: error: String (“Unauthorized”)

GET api/battles/{battleId}

200 Response: Battle: Object

400 Response: error: String (“Invalid or missing BattleId”)

401 Response: error: String (“Unauthorized”)

GET api/battles/{battleId}/submissions

200 Response: submissions: Array of Objects

400 Response: error: String (“Invalid or missing BattleId”)

401 Response: error: String (“Unauthorized”)

GET api/battles/{battleId}/ranking

200 Response: ranking: Array of Objects

400 Response: error: String (“Invalid or missing BattleId”)

401 Response: error: String (“Unauthorized”)

POST api/battles/{battleId}/commit

Request Body: commit Object

400 Response: error: String (“Invalid or missing BattleId”)

401 Response: error: String (“Unauthorized”)

POST api/tournaments/create

Request Body: educatorId: int, tournamentName: String, registrationDeadline: Date, badges: Array of Objects, educatorsInvited: Array of Strings

200 Response: message: String "Tournament created successfully."

400 Response: error: String("Missing values", "Values not valid")

POST api/tournaments/close

Request Body: tournamentId: int, educatorId: int

200 Response: message: String "Tournament closed successfully."

400 Response: error: String("Missing values", "Values not valid")

POST api/tournaments/{tournamentId}/createBattle

Request Body: educatorId: int, BattleName: String, registrationDeadline: Date, endBattleDeadline: Date, minGroupSize: int, maxGroupSize: int, codingLanguage: String, codeKata: Object

200 Response: message: String "Battle created successfully."

400 Response: error: String("Missing values", "Values not valid")

POST api/battles/closeBattle

Request Body: battleId: int, token

200 Response: message: String "Battle created successfully."

400 Response: error: String("Missing values", "Values not valid")

POST api/submissions/correctScore

Request Body: submissionId: int, score: int, token

200 Response: message: String "Success"

400 Response: error: String("Missing values", "Values not valid")

GET api/tournaments/search

Request Parameters: keyword: String, state: String

200 Response: enrolledTournaments: Array of Objects

GET api/profile/search

Request Parameters: keyword: String

PUT api/profile/personal-data

Request Body: User: Object

200 Response: message: "Data updated successfully"

400 Response: error: String("Missing values", "Values not valid")

GET api/profile/{profileId}

200 Response: profile: object

400 Response: error: String ("Invalid or missing profileId")

2.6. Selected Architectural Styles and Patterns

1. Three-Tier The 3-tier architecture divides the system into three logical layers: presentation (Web Server), business logic (Application Server), and data storage (Database). Each tier has specific functions, ensuring separation of concerns and efficient management of the system. It allows scalability, simplifies maintenance and enhancing control and security.

2. RESTful APIs RESTful APIs (Representational State Transfer) follow a design paradigm based on specific principles for creating web services. They use standard HTTP methods (GET, POST, PUT, DELETE) and are stateless, allowing easy data exchange between systems. It provides seamless integration with various platforms and systems, it is simple to implement and reduces server load.

3. On-Cloud The system is hosted on cloud infrastructure, leveraging services provided by any cloud provider. This offers on-demand scalability, cost efficiency and ensures high availability.

2.7. Other Design Decisions

2.7.1. Scale-out

The decision to implement a scale out design in the software was taken to enhance its ability to scale, its availability, and its performance. This design approach enables the system to expand its capacity to cope with increased demand by adding more resources, such as servers or machines, as needed. This can be more cost-effective than upgrading individual components and avoids the need for downtime. The scale out design also improves the system's reliability by providing redundant resources that can take over if any component fails. In addition, the design allows for flexibility, as resources can be added or removed as required. Lastly, the scale out design improves the system's performance by allowing workloads to be processed in parallel rather than sequentially.

2.7.2. Relational Database

We selected a relational database for our system design because it is effective at storing structured data, enforcing data integrity, and providing fast query performance. It can also scale to handle large amounts of data and support many concurrent users. The database allows us to store and retrieve information efficiently, while also ensuring that the data is accurate and consistent.

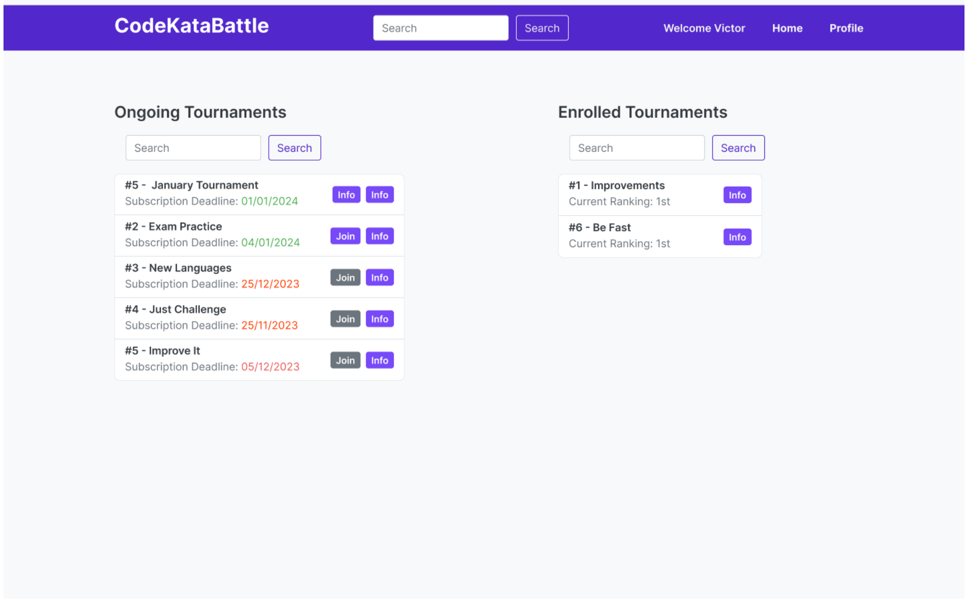
2.7.3. Token-Based Authentication and Authorization

The authentication and authorization will be implemented using a token-based process. These tokens are sent to the users each time they log in and they must be included with each request that requires authentication or authorization.

2.7.4. Distributed MVC Pattern

Our decision to implement the Distributed MVC pattern was driven by the need for a modular and maintainable software architecture. This pattern divides the application into three core components: Model, View, and Controller. The Model manages the data and business logic, the View presents this data to users, and the Controller acts as an intermediary, processing user input and system responses. This division enhances the clarity and organization of our codebase, allowing for simultaneous development of distinct parts of the system. The distributed aspect of this pattern particularly aids in handling complex, scalable applications by facilitating efficient data processing and user interface management across different systems or networks.

3 | User Interface Design



Ongoing Tournaments

Search Search

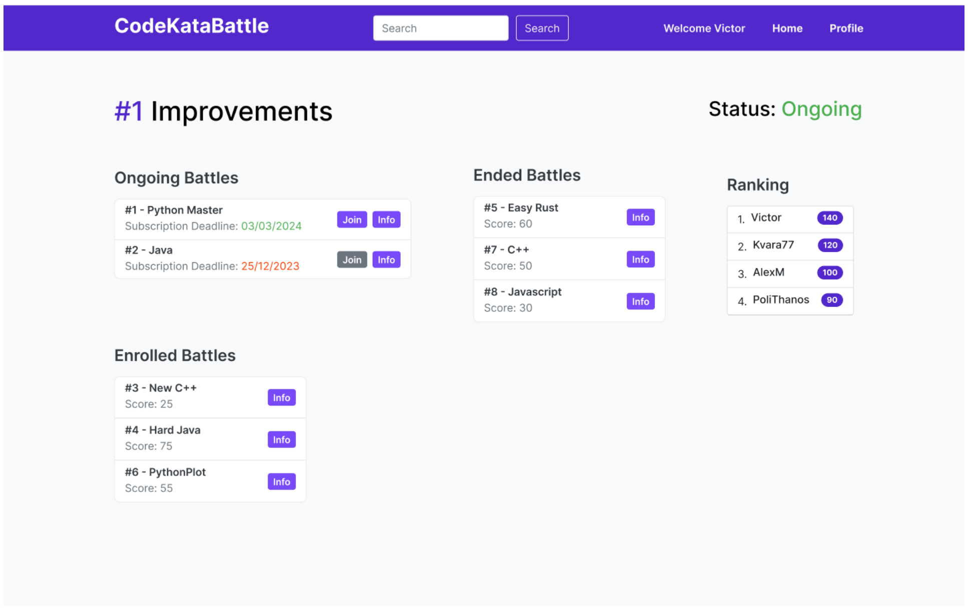
#5 - January Tournament Subscription Deadline: 01/01/2024	Info Info
#2 - Exam Practice Subscription Deadline: 04/01/2024	Join Info
#3 - New Languages Subscription Deadline: 25/12/2023	Join Info
#4 - Just Challenge Subscription Deadline: 25/11/2023	Join Info
#5 - Improve It Subscription Deadline: 05/12/2023	Join Info

Enrolled Tournaments

Search Search

#1 - Improvements Current Ranking: 1st	Info
#6 - Be Fast Current Ranking: 1st	Info

Figure 3.1: Home Page (Student)


CodeKataBattle Search Search Welcome Victor Home Profile

#1 Improvements

Status: Ongoing

Ongoing Battles

#1 - Python Master
Subscription Deadline: 03/03/2024 Join Info

#2 - Java
Subscription Deadline: 25/12/2023 Join Info

Ended Battles

#5 - Easy Rust
Score: 60 Info

#7 - C++
Score: 50 Info

#8 - Javascript
Score: 30 Info

Ranking

1.	Victor	140
2.	Kvara77	120
3.	AlexM	100
4.	PoliThanos	90

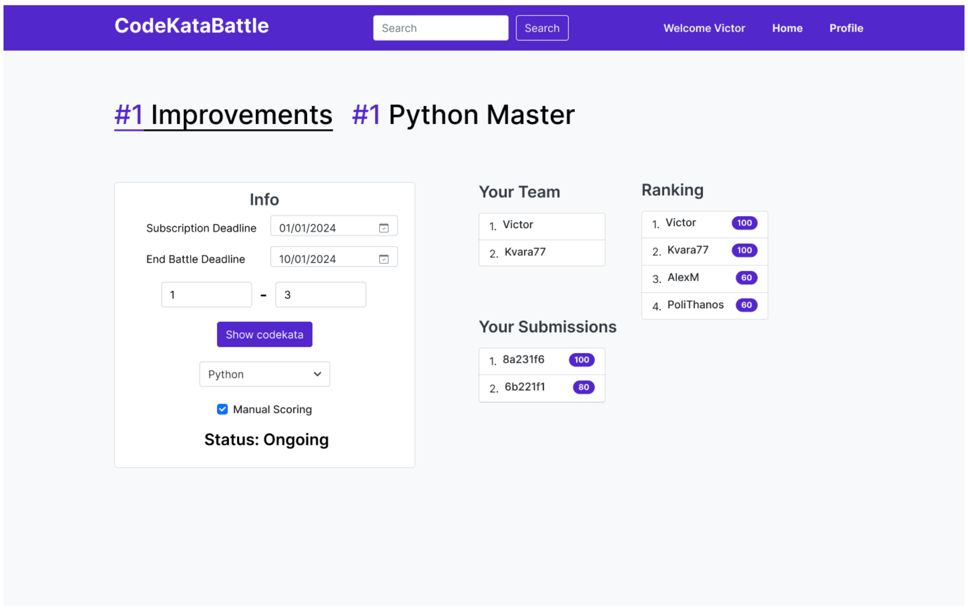
Enrolled Battles

#3 - New C++
Score: 25 Info

#4 - Hard Java
Score: 75 Info

#6 - PythonPlot
Score: 55 Info

Figure 3.2: Tournament Page (Student)


CodeKataBattle Search Search Welcome Victor Home Profile

#1 Improvements #1 Python Master

Info

Subscription Deadline: 01/01/2024
End Battle Deadline: 10/01/2024
1 - 3
Show codekata
Python
✓ Manual Scoring

Status: Ongoing

Your Team

1. Victor
2. Kvara77

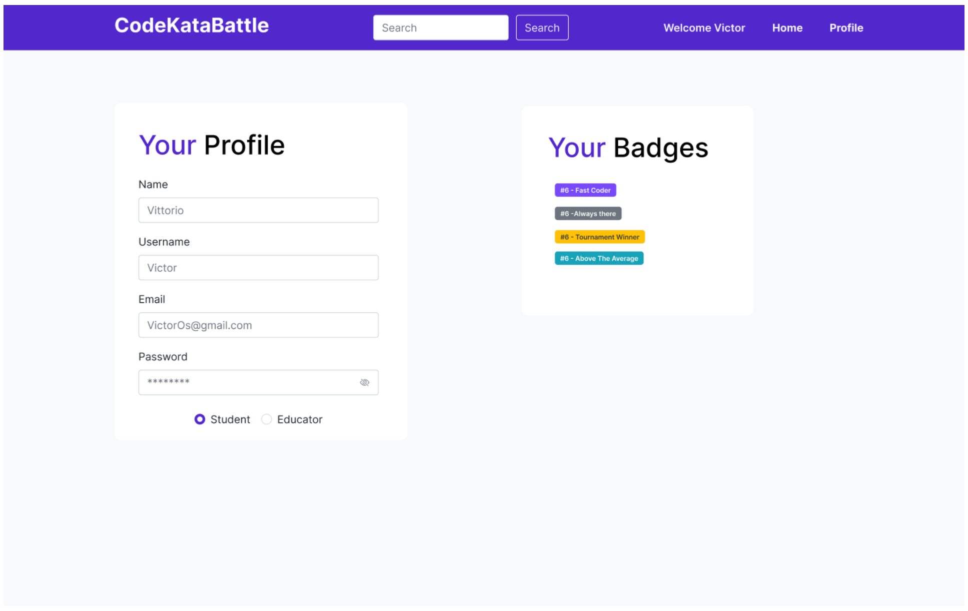
Your Submissions

1.	8a231f6	100
2.	6b221f1	80

Ranking

1.	Victor	100
2.	Kvara77	100
3.	AlexM	60
4.	PoliThanos	60

Figure 3.3: Battle Page (Student)



Your Profile

Name: Vittorio

Username: Victor

Email: VictorOs@gmail.com

Password: ********

[✓ Student] Educator

Your Badges

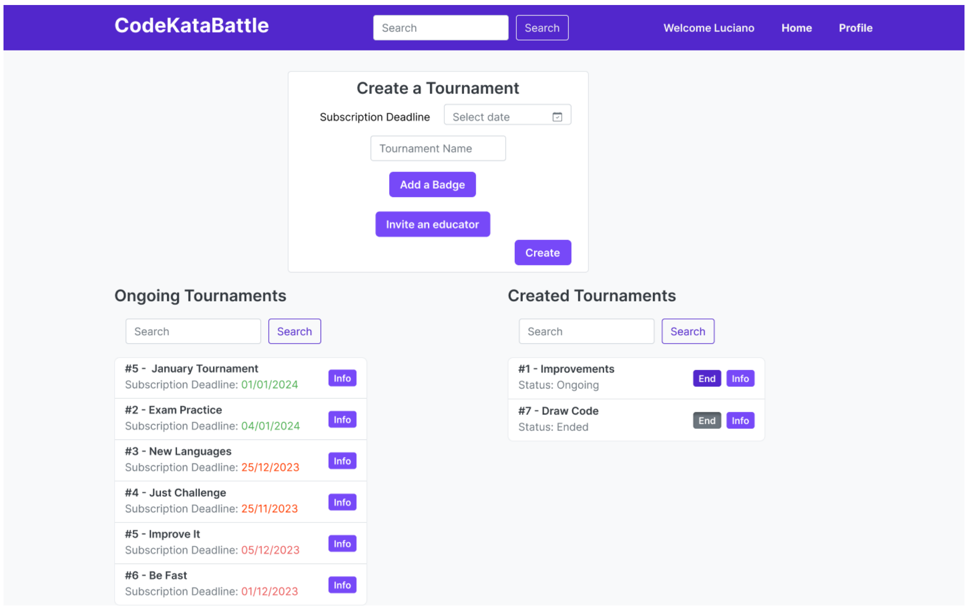
#6 - Fast Coder

#6 -Always there

#6 - Tournament Winner

#6 - Above The Average

Figure 3.4: Profile Page (Student)



Create a Tournament

Subscription Deadline: Select date

Tournament Name

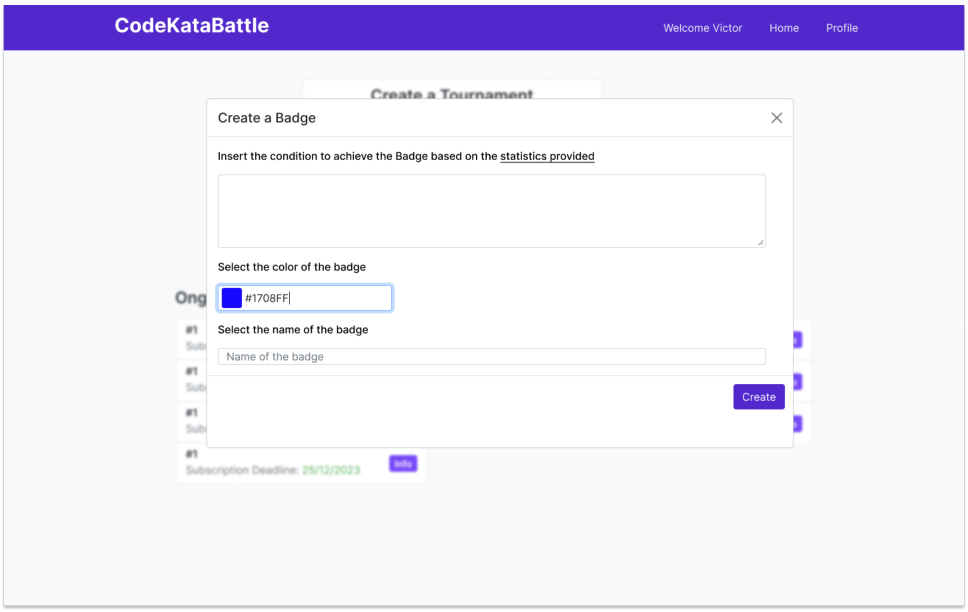
Ongoing Tournaments

#5 - January Tournament Subscription Deadline: 01/01/2024	Info
#2 - Exam Practice Subscription Deadline: 04/01/2024	Info
#3 - New Languages Subscription Deadline: 25/12/2023	Info
#4 - Just Challenge Subscription Deadline: 25/11/2023	Info
#5 - Improve It Subscription Deadline: 05/12/2023	Info
#6 - Be Fast Subscription Deadline: 01/12/2023	Info

Created Tournaments

#1 - Improvements Status: Ongoing	End Info
#7 - Draw Code Status: Ended	End Info

Figure 3.5: Home Page (Educator)



Create a Badge

Insert the condition to achieve the Badge based on the statistics provided

Select the color of the badge

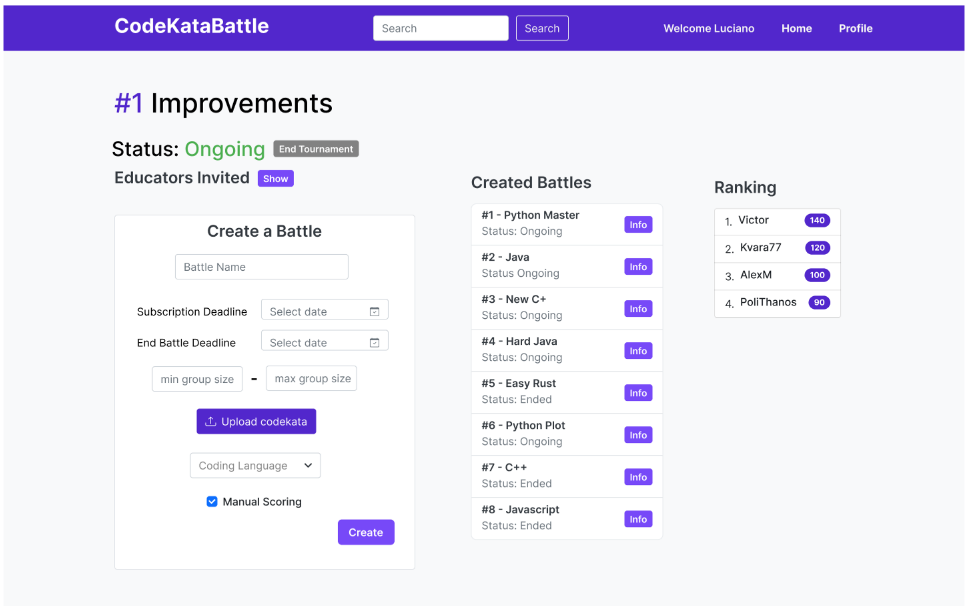
#1708FF

Select the name of the badge

Name of the badge

Subscription Deadline: 25/12/2023

Figure 3.6: Badge definition for a tournament (Educator)



#1 Improvements

Status: Ongoing

Educators Invited

Create a Battle

Subscription Deadline

End Battle Deadline

Manual Scoring ✓

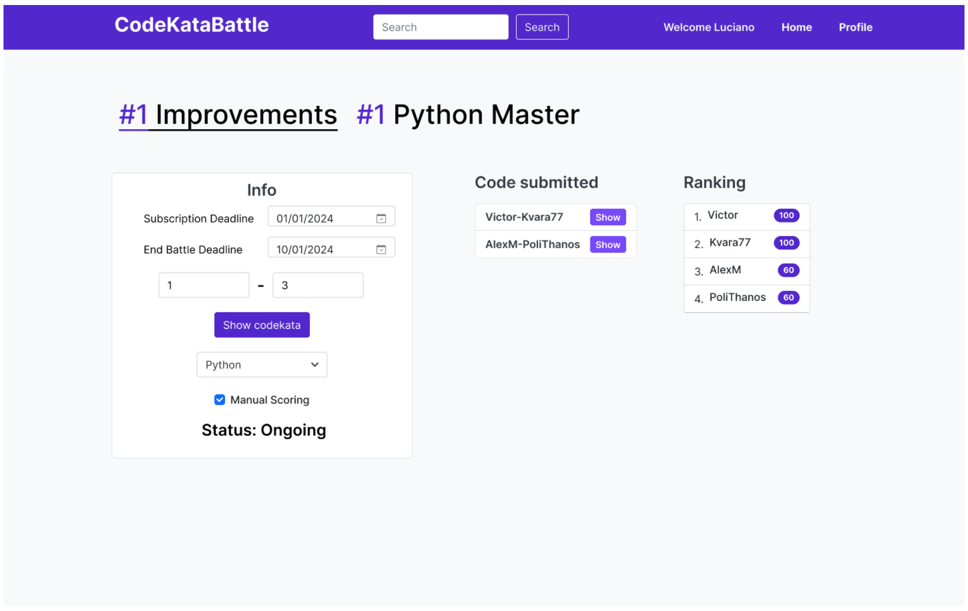
Created Battles

#1 - Python Master Status: Ongoing	Info
#2 - Java Status Ongoing	Info
#3 - New C+ Status: Ongoing	Info
#4 - Hard Java Status: Ongoing	Info
#5 - Easy Rust Status: Ended	Info
#6 - Python Plot Status: Ongoing	Info
#7 - C++ Status: Ended	Info
#8 - Javascript Status: Ended	Info

Ranking

1.	Victor	140
2.	Kvara77	120
3.	AlexM	100
4.	PoliThanos	90

Figure 3.7: Tournament Page where the educator is the creator of the tournament (Educator)



#1 Improvements #1 Python Master

Info

Subscription Deadline 01/01/2024

End Battle Deadline 10/01/2024

1 - 3

Show codekata

Python

Manual Scoring

Status: Ongoing

Code submitted

Victor-Kvara77 Show

AlexM-PoliThanos Show

Ranking

1.	Victor	100
2.	Kvara77	100
3.	AlexM	60
4.	PoliThanos	60

Figure 3.8: Battle Page where the educator is the creator of the tournament (Educator)

4 | Requirements Traceability

R1	
R1	**The system must allow an unregistered Educator to sign up.**
C1	Account Manager
C4	Authentication Manager
C6	Notification Manager
C7	Model
C8	Entity Manager
E4	Email Service Provider

R2	
R2	**The system must allow an unregistered Student to sign up.**
C1	Account Manager
C4	Authentication Manager
C6	Notification Manager
C7	Model
C8	Entity Manager
E4	Email Service Provider

R3	
R3	**The system must allow a registered User to log in.**

C1	Account Manager
C4	Authentication Manager
C7	Model
C8	Entity Manager

R4	
R4	**The system must allow registered Educators to start the creation process of a Tournament of Code Kata Battles.**
C3	Tournament Manager
C7	Model
C8	Entity Manager

R5	
R5	**The system must provide registered Educators of a list of Tournament-related statistics for the Badges definition, during the Tournament creation process.**
C3	Tournament Manager
C7	Model
C8	Entity Manager

R6	
R6	**The system must provide registered Educators of a specific language which lets them define the Badges, during the Tournament creation process.**
C3	Tournament Manager
C7	Model
C8	Entity Manager

R7	
R7	**The system must allow registered Educators to grant other registered Educators the permission to manage the Tournament, during the Tournament creation process.**
C3	Tournament Manager
C7	Model
C8	Entity Manager

R8	
R8	**The system must allow registered Educators to end the creation process of a Tournament that they started themselves.**
C3	Tournament Manager
C7	Model
C8	Entity Manager

R9	
R9	**The system must be able to send notifications to every registered User.**
C6	Notification Manager
C7	Model
C8	Entity Manager
E5	Notification Service

R10	
R10	**The system must allow a registered Educator to create a Battle in a Tournament if and only if he is the creator of the Tournament or if he was granted the permission to by the latter.**
C3	Tournament Manager

C7	Model
C8	Entity Manager

R11	
R11	**The system must allow a registered Student to create a group for a Battle in a Tournament.**
C2	Battle Manager
C7	Model
C8	Entity Manager

R12	
R12	**The system must allow a registered Student to accept an invitation to a group for a Battle in a Tournament.**
C2	Battle Manager
C6	Notification Manager
C7	Model
C8	Entity Manager
E5	Notification Service

R13	
R13	**The system must allow registered Students to see the list of ongoing Tournaments and join any of those if its subscription deadline is not expired yet.**
C3	Tournament Manager
C7	Model
C8	Entity Manager

R14	
R14	**The system must allow registered Students who are enrolled in a Battle to perform code submissions.**
C5	Evaluator
C7	Model
C8	Entity Manager
E1	GitHub API

R15	
R15	**The system must be provided of proper APIs to let registered Students perform code submissions through GitHub Actions.**
C2	Battle Manager
C5	Evaluator
C7	Model
C8	Entity Manager
E1	GitHub API

R16	
R16	**The system must update the Battle ranking when a valid code submission is performed.**
C2	Battle Manager
C7	Model
C8	Entity Manager

R17	
R17	**The system must set the consolidation stage of a Battle when its submission deadline expires.**

C2	Battle Manager
C7	Model
C8	Entity Manager

R18	
R18	**The system must let Educators to end the consolidation stage of a Battle if and only if he is the creator of the Tournament or if he was granted the permission to by the latter.**
C2	Battle Manager
C7	Model
C8	Entity Manager

R19	
R19	**The system must update Tournament ranking when a Battle exits the consolidation stage.**
C3	Tournament Manager
C7	Model
C8	Entity Manager

R20	
R20	**The system must let Educators who are either the creator of the Tournament or who have been granted the permission to by the latter to close a Tournament if and only if there is no Battle such that either their subscription or submission deadline is not expired yet or such that they are still in the consolidation stage.**
C3	Tournament Manager
C7	Model
C8	Entity Manager

R21	
R21	**The system must assign an achievements' Badge for a given Tournament to any student who satisfied the conditions defined by the creator of the Tournament.**
C1	Account Manager
C7	Model
C8	Entity Manager

R22	
R22	**The system must allow every User to see the Badges which were ever obtained by a given Student.**
C2	Account Manager
C7	Model
C8	Entity Manager

R23	
R23	**The system must notify every registered Student about the creation of a new Tournament.**
C6	Notification Manager
C7	Model
C8	Entity Manager
E5	Notification Service

R24	
R24	**The system must notify every registered Student about the creation of a new Battle within a Tournament they are enrolled in.**

C6	Notification Manager
C7	Model
C8	Entity Manager
E5	Notification Service

R25	
R25	**The system must notify every registered Student about the end of a Battle they are participating in.**
C1	

R26	
R26	**The system must notify every registered Student about the end of a Tournament they are enrolled in.**
C6	Notification Manager
C7	Model
C8	Entity Manager
E5	Notification Service

5 | Implemenation, Integration and Testing Plan

5.1. Development Process and Approach

The system will be implemented, integrated, and tested using a bottom-up approach, taking into account the dependencies between components of the system. This strategy will be used for both the server and client sides, which will be developed and tested at the same time. Incremental integration testing will be applied to try to find and fix bugs as soon as possible during the development cycle. Since Entity Manager and external services are assumed to be reliable, they don't need unit testing.

5.2. Implementation & Integeration Plan

This section will describe the implementation and integration plan of both sides of our system.

5.2.1. Server Side

In the first step, the Model is implemented and unit tested. The model, which makes use of the EntityManager, is responsible for all the interactions with the database.

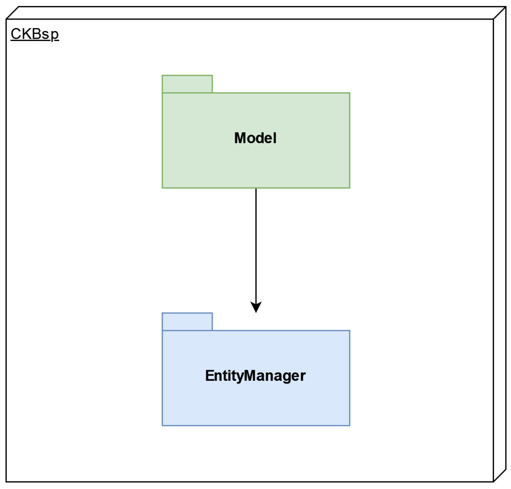


Figure 5.1: Implementation and testing of the Model

In the second step, the controllers are implemented and unit tested using Drivers and Stubs for not yet implemented or external modules. Integration tests with the Model are done as well.

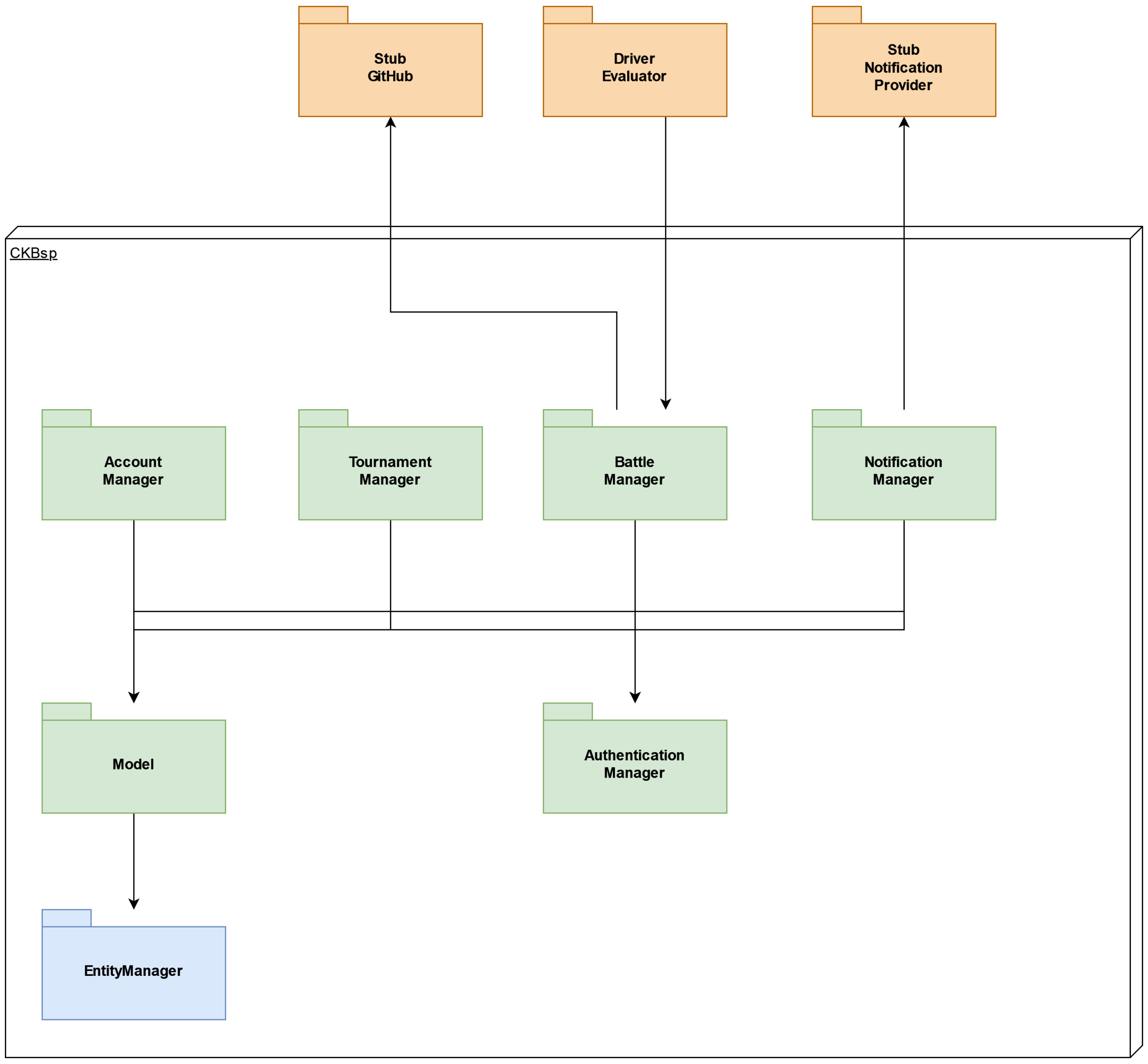


Figure 5.2: Implementation and testing of the controllers

In the third step, the Evaluator is implemented and unit tested using Drivers and Stubs for not yet implemented or external modules. Integration tests between the controllers are done at this stage.

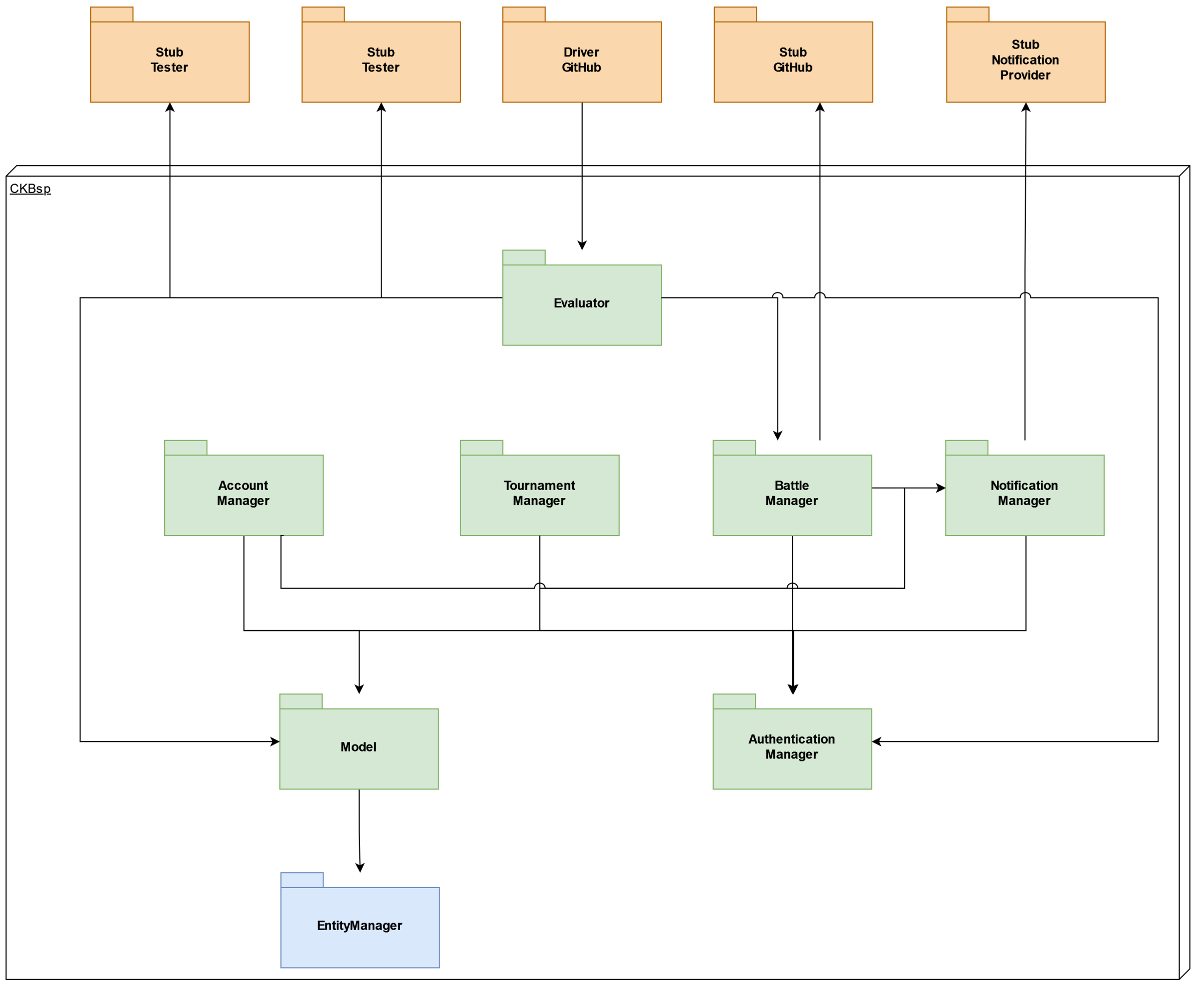


Figure 5.3: Implementation and testing of the Evaluator

Once all the components have been implemented and unit tested, full integration tests with Drivers and Stubs for external services are done. As the last step, integration tests with the real services take place.

5.2.2. Client Side

Each component of the View is rigorously unit tested using a stub of the REST API, enabling parallel development of the frontend and backend.

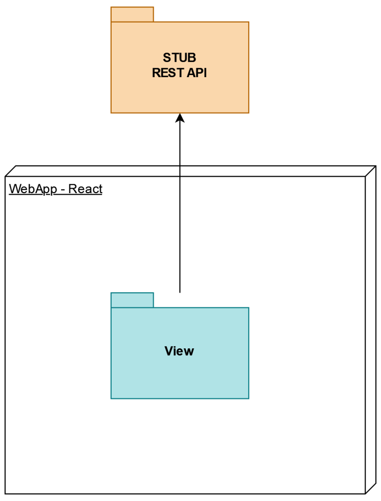


Figure 5.4: View testing using API Stub

5.2.3. Final Integration Test

Once the backend and the frontend have been implemented, final tests can take place.

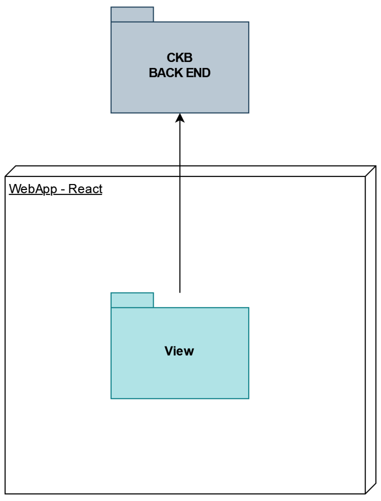


Figure 5.5: View testing using CKB Backend

5.3. Technologies

5.3.1. Development Technologies

- Server Side Components: Server-side components will be developed using Java, utilizing the widely-used Spring framework. This framework facilitates the efficient and straightforward development of REST web applications through JakartaEE and TomEE APIs. Our decision to choose this framework is primarily influenced by the extensive range of robust and well-documented Spring libraries available. These libraries provide developers with the tools needed to effortlessly implement state-of-the-art paradigms.

- Client Side Components: The front end of the web app will be developed using JavaScript and the React framework. This approach allows us to build encapsulated components and later compose them to create complex user interfaces. Each component can be developed and maintained independently, thus enhancing the reusability, testability, and separation of the application's codebase. This modular structure makes it easier to manage and scale applications while also facilitating collaboration among team members, as different components of the application can be worked on simultaneously without significant overlap.

- Database: In order to efficiently and easily access, persist, and manage data mapping between Java objects and a relational database, without the need for manually writing SQL code, we have decided to utilize the Java Persistence API (JPA). Any relational DBMS supporting those APIs, such as MySQL or PostgreSQL, can be used.

- Token-based Authentication and Authorization: The authentication and authorization will be implemented using the JSON Web Token (JWT) standard. These tokens are sent to the users each time they log in and must be included with each request that requires authentication or authorization. In a JWT, the token itself contains all the relevant information about the user, digitally signed using a secret, which makes it trustworthy and stateless. The server's memory doesn't need to store session information about each user, as all necessary data is contained in the token. This is particularly useful for scalability, as it reduces the load on the server and is ideal for distributed systems. To keep the system secure, tokens should be carefully transmitted using secure protocols such as HTTPS and stored carefully locally.

5.3.2. Testing Technologies

- JUnit: JUnit provides a modern foundation for developer-side testing on the JVM. It will be used for unit and integration testing of server side classes.
- Jest: Jest is a JavaScript Testing Framework with a focus on simplicity. It will be used for unit and integration testing of client side codebase.
- Postman: Postman is an API platform for building and using APIs. It will be used to efficiently test the REST API endpoints while the front-end is still under development.

6 | Effort Spent

6.1. Effort Spent per Unit

This section shows the amount of time that each member has spent to produce the document. Please notice that each unit is the result of coordinated work among all the members.

UNIT	**MEMBERS**	**HOURS**
SetUp	Piccinato	2h
Component View Diagrams	Puglisi, Piccinato, Piazzalunga	9h
DB Schema and ER Diagram	Piazzalunga	6h
Deployment View Diagram	Piccinato	1h
Runtime View Sequence Diagrams	Piazzalunga, Piccinato, Puglisi	25h
Interfaces Diagrams	Puglisi	4h
Endpoints	Puglisi	4h
Requirements Traceability	Piccinato	1h
Chapter 2 Redaction	Piccinato	2h
Chapter 4 Redaction	Piccinato	2h
Chapter 5	Piazzalunga	4h

7 | References

7.1. References and Tools

1. GitHub: https://www.github.com
2. GitHub Actions: https://github.com/features/actions
3. The UI Mockups have been made with: https://www.figma.com
4. Sequence Diagrams have been made with: https://sequencediagram.org
5. Other Diagrams have been made with: http://draw.io