POLITECNICO
MILANO 1863

Code Kata Battle project Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli
Version:	1.0
Date:	07-January-2024
Download page:	https://github.com/Angelo7672/AttivissimoBelardinelliChiodaroli
Copyright:	Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli– All rights reserved

Contents

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

List of Figures

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

List of Tables

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

1 Introduction

1.1 Purpose

The Design Document has the main goal of illustrate a more technical and specific indications to help developers implement what RASD describes.

In particular, where the previous mentioned document is more focused on the requirements, goals, general context description and their theoretical correctness, DD defines main guidelines to concrete those ideas and principles, making the program real.

The procedures and phases here reported are, namely, implementation, integration and testing plans.

Finally, is reported the singular effort spent to realize the Document.

1.2 Scope

CodeKataBattle is an application thought to train coding skills using the "kata Battle" principles.

This technique, typical of martial arts, consists of repeating an action to improve it, until a good level of confidence, correctness and precision is achieved.

In this context, the application provides two different type of user that will interact with it, which are Students and Educators.

Educator is the one who oversee the Students' work and learning: he/she designs Tournaments and their corresponding Battles, which are the last online arenas in which Students compete to develop a particular code. Furthermore, he/she specify potential badges that could be given to the Students who deserve them the most during each Tournament.

Instead, the Student is a user who subscribes to CKB in order to advance his or her abilities in software development; as a result, he or she may participate in Tournaments alone or in Teams with other Students. Through challenging other colleagues, he or she will gain experience in developing and solving real-world programming problems.

The Platform will assist the Educator in overseeing Tournaments and Battles, and it will automatically evaluate, based on certain standards, the code that the Students provide. Furthermore, it will facilitate Students' learning by simplifying the user interface and making it simple to peruse the list of all accessible Tournaments, distinctly outlining the prerequisites and goals of each of them.

The app's services are accessible through a Web App, that you reach through browser, and among them are included the creation, for Educators, and registration, for Students, to Tournaments and Battles, code evaluation, creation of Teams of Students and invitations to collaborate to a Tournament management for Educators. The architecture of the product is divided in 4 tiers: client, web server, application server and data server. Three layer are adopted: presentation layer, application layer and data layer. The app's services described before are implemented as microservices: there are microservices that operate to a correct functioning of the managing of the user's account and microservices to managed Battles and Tournaments with everything is concerned with them.

1.3 Definitions, acronyms, abbreviations

To better understand the Document and the Actors involved here are reported some Definitions.

- **Platform**
  Represents the CodeKataBattle System's behavior as a whole, describes its workflows and interactions.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

- **User**
  With User is intended, depending on the situation, Educator or Student. The entity, in particular will model behaviors and interactions common to both main Actors.
- **Student**
  Student is one of the main Actors. The entire Platform is build with the aim to improve his/her coding skills, joining Battles, forming Teams, inviting other Students to join Battles, receiving Badges.
- **Educator**
  Educator is one of the main Actors. He/She manages Tournaments, alone or inviting other Educators, creating them and adding Battles. Educator creates Badges and can add extra point to Student's Score.
- **Battle**
  Within a Battle Students compete to improve their skills. It is the base entity on which is realized the entire Platform.
- **Team**
  This entity describes behaviors and interactions common to Students' Teams, that compete in a Battle, and a set of Educators, that manages a Tournament.
- **Badge**
  In order to gamify the Platform Badges can be created. These are assigned to a Student that achieves a specific goal.
- **Tournament**
  Tournaments are the context within Battles take place. They are managed by Educators and defines a specific topic for code developed for a hosted Battle.
- **Repository manager Platform**
  Repository management Platform is an external Platform that oversees code developed for a particular Battle.
- **Score**
  Score is the quantified evaluation of the uploaded code. It is automatically assigned by the Platform or can be defined by an Educator.
- **Application Programming Interface**
  The Application Programming interface is a set of functions that denote the interface of a software system to other ones.

1.3.1 Repository manager Platform specific definitions

These definitions describe the technical jargon used while talking about repository management Platform functionalities.

- **Version Control System**
  A system that allows to save data in different versions keeping track of changes and authors.
- **Repository**
  A folder in which code is stored and managed by a version control system.
- **Action**
  Code that gets executed by the RMP on the repository once a certain condition is met (i.e. new upload, time of the day, etc...).

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

- **Commit**
  Act in which changes are saved in the VCS. Commits have details consisting in a short textual description known as Message, and author data consisting in RMP handle and e-mail address.
- **Push**
  Act in which commits are uploaded to the RMP Platform.
- **Fork**
  Act of duplicating repositories created by others in a personal one

1.4 Abbreviations and Acronyms

- **CKB**
  CodeKataBattle, the name of the Platform.
- **RMP**
  Repository Manager Platform
- **VCS**
  Version Control System
- **API**
  Application Programming Interface

1.5 Revision History

- **Version 1.0** - 2024-01-07
  - First release

1.6 Document Structure

1. **Introduction** This section provides a general description of the document purpose, structure and goals associated useful definitions, acronyms and abbreviations related to the software to develop. Moreover, is reminded to the reader the general description and purpose of the app for which is intended to provide instructions.
2. **Architectural Design** Here are reported the main architectural components and qualities to implement in the software. A more detailed description of the app's functioning appears with respective modules and physical components that host them. In order to do this, the section is enriched with diagrams and specification to organize the code in the architecture making it work as intended.
3. **User Interface Design** User Interface is crucial for the app's usability. In this section are provided principles to apply to realize the intended interface, providing mockups, examples and descriptions. It would be clarified how to connect interface components to the code.
4. **Requirements Traceability** The purpose of the Document is to provide details to implement the software as intended. This means that is crucial to respect requirements and goals at all levels defined in the RASD. Each module and software components has to be matched with the requirement it brings to live. This procedure, reported in this section, is at the base of the project's coherence, and it's essential for its correct realization.
5. **Implementation, Integration and Test Plan** DD provides a more detailed analysis and description of what is needed to finally shape the project. In order to do that is vital to have a plan with the intention of achieving goals for the prefixed time without renouncing to quality. Proceeding

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

step by step and following a logic based on software characteristics, is possible to make the app progressively grow meeting stakeholders, clients and developers needs, involving every actor in the process without compromising the code. For these reasons Implementation, Integration and Test plans are provided to build a coherent, robust, complete code.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2 Architectural Design

2.1 Overview

CKB is conceptualized from the ground basis as a Client-Server Platform. In fact, Users access services through a Web App, which constitutes the only virtual device able to receive commands to interact with CKB.

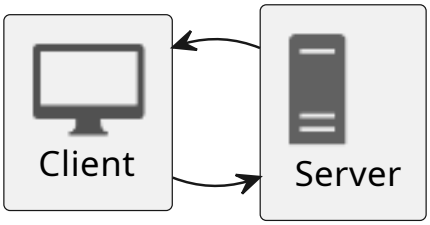

Figure 1: Client-Server paradigm

The main logic is hosted in the server that manages all the functionalities of CKB, while the client is only a graphic representation for the User.

Because the client, constructed based on its thin version, only displays the GUI to the User in order to expose the Platform's services and functionalities with their corresponding interfaces that it obtains by communicating with the Server through the Network. In fact the client-server software organization that was chosen is Remote Representation.

The Web Browser hosting the Web App should ideally handle client communication. It must enable safe and efficient communication with the server via the HTTP and TCP protocols.

To be used with all types of browsers, CKB User Interface code needs to be written properly. However, after their kind and privileges, servers would be assigned to protect Users' data, and the Platform's model would change in accordance with the guidelines.

2.2 Microservices Architecture

One of the fundamental skills of CKB is the ability to simultaneously manage and conduct multiple Battles, Tournaments, and receive multiple inputs from various clients, based on their type. The server must manage several requests at once, recognizing this through concurrency.

The Microservices architectural approach is one that can be selected. This option, which is supported by the availability of multiple servers, would allow each machine to host multiple service components that are properly replicated even in other computers. These components would each elaborate particular inputs pertaining to a particular topic or collection of related operations that make up a service. Components can also be allocated according to what are the needs of the moment, if the number of requests increases, the number of components can be increased, and vice versa.

The concept would include DBMS services, provided thought proper APIs, as long as communication ones and the dispatcher itself, that would account all requests from Users redirecting them to the right Microservice. This architecture guarantees concurrency, modularity, which positively affects maintenance, and Platform reactivity.

The method ensures scalability and improvements by enabling developers and authors to update CKB with new functionalities without altering the User interface in any way. It also allows the Platform to retain data and determine resilience while ensuring appropriate duplication of each component to prevent failures.

CKB is thought to be implemented as a Three Layer Architecture:

- **Presentation Layer:** this layer's only purpose is to display to the User the application's product functionalities through a graphical interface.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

- **Application Layer:** this layer is the central component of the Platform since it contains the entire application's logic and controls its functionalities.
- **Data Layer:** the application's whole data set is contained in this layer.

Figure 2: Three Layer Architecture

CKB, once realized, would be a Four Tier Application:

1. **Client:** The task assigned to this Tier is to gather input from Users—both Educators and Students—and report their instructions into the system. It functions as a GUI.
2. **Web Server:** It counts the parts assigned to create a secure connection between the Application Server and the Clients, which are the major actors. Web Server is outfitted with suitable firewalls and a Demilitarized Zone (DMZ) to ensure the security and integrity of Application Server data.
3. **Application Server:** It houses the elements that make up the Platform itself, the modules that carry out the primary needs and objectives outlined in RASD. This Tier can be defined as the application Model, realized through a dispatcher-coordinated Microservices Architecture.
4. **DBMS Server:** Data security, integrity, and preservation are the purview of DBMS. It will provide them with the relevant modules' requests according to their authorization for that particular data.

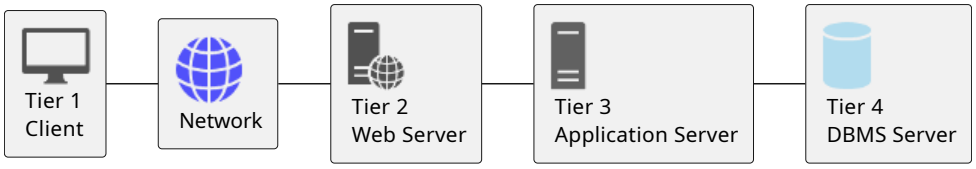

Figure 3: Four Tier Application

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.3 Component View

2.3.1 Microservices Architecture

CKB's Application Layer is based on Microservices, that constitutes components in the system's architecture.

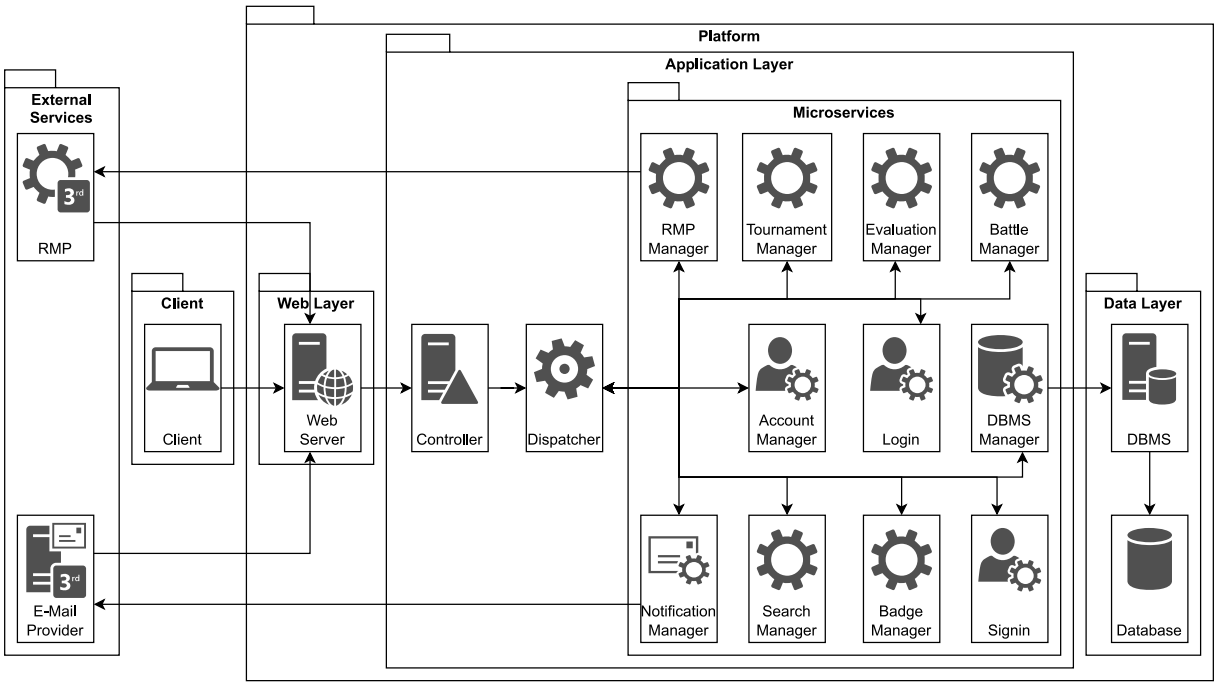

Figure 4: Microservices Architecture

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.3.2 Component Diagrams

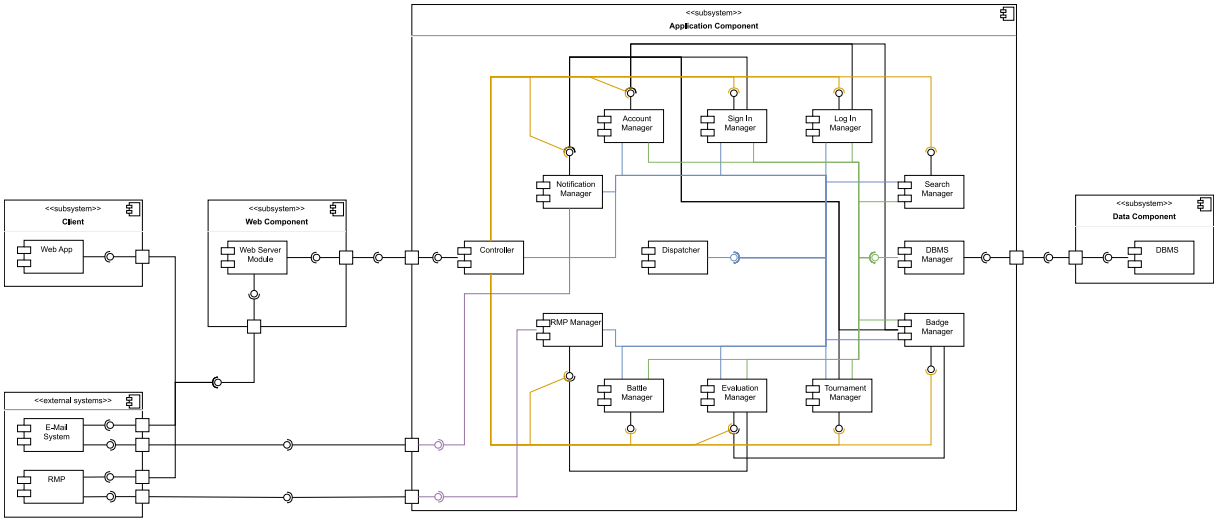

Figure 5: Component Diagram

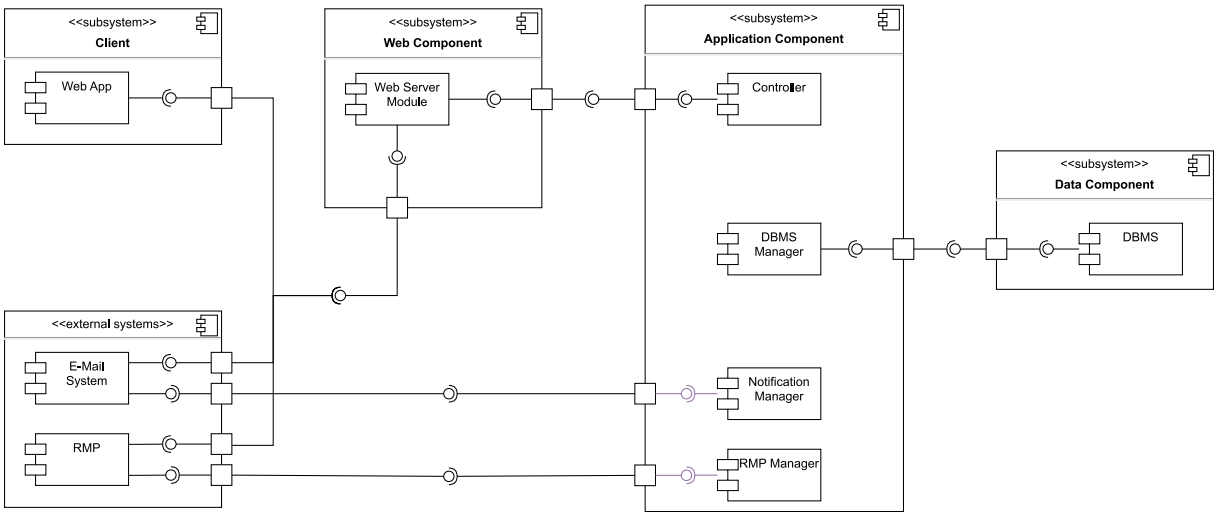

Figure 6: Simplified Component Diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Client The Web Browser module is the only module contained in the Client in this representation. To establish a network connection between the Client and the CKB Server, a Web Browser is required. The Web Browser can also control the graphical elements that the Server sends to the client, which the User's GUI will employ.

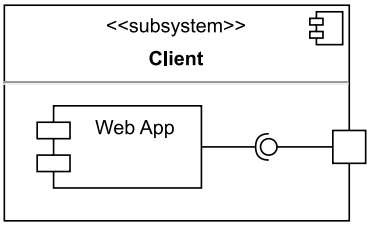

Figure 7: Client Component Diagram

Web Server The Web Browser's job is to route browser requests to the Application Server and get back responses from the Application Server. The Web Server just needs to route requests from and to the Network, a result of being located in the DMZ. It is composed of the following module:

- **Web Server Module** This is in charge of providing the Web browser of the necessary files to run the Client and an interface to the client to communicate with the Application Tier. The API implemented here is one of the RESTFul type, as commented here RESTFul API. This allows to expose only the needed functionalities to the client, masking the internal logic of the Platform.

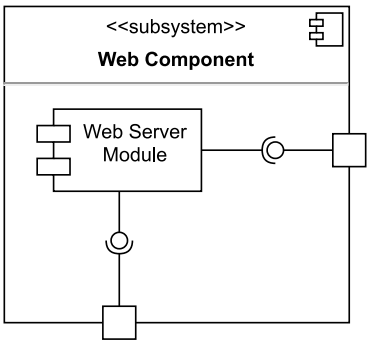

Figure 8: Web Server Component Diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Application Server

1. **Controller** The Controller is the component that manages the requests from the Web Server. It is invoked by the Web Server, which provides the request and the parameters. The Controller will interface with the Dispatcher to re-route the request to the right Microservice.

2. **Dispatcher** The Dispatcher is the component that routes the requests to the right Microservice. It is invoked by the Web Server, which provides the request and the parameters. The Dispatcher will call the right Microservice, passing the parameters. The functionality of this component will be better shown in the dedicated section: Dispatcher Component.

3. **Sign In Manager** It is the manager in charge of allow Users to sign in into the Platform through proper interface.

4. **Log In Manager** This module is the access door to CKB's Platform. It memorizes all Users which are currently interacting with the system. The component exposes a 'logIn' Interface used by the User to log in and by other components to verify if an existing User is, in fact, logged in. Finally, the interface allows logging out too.

5. **RMP Manager** The manager allows the Platform to perform pull requests from corresponding repos, acting as a Client in face of RMP. It is used by Evaluation Manager which access 'pullRequests' Interface.

6. **Notification Manager** This component manages particular notifications to be sent to Users, communicating with E-mail provider, which performs as a Server for this module. Requests are sent on other components' indications, which are expressed through 'sendNotification' Interface.

7. **Evaluation Manager** It is in charge of running test cases for every code file submitted by each Team. Basing on given parameters it would assign the corresponding Score. This is done automatically via a particular process that is explained here: Battle Evaluation.

8. **Badge Manager** It assigns Badges to Students who can be awarded. Badge Evaluation

9. **Battle Manager** It allows managing Battles, updating the score of each Team.

10. **Tournament Manager** It allows to manage Tournaments, create Teams, invite Students and Educators, create Battles and Badges, update the Tournament score.

11. **Account Manager** It is in charge of recruiting all information about a single Account, allowing the owner to update the Account with new personal data.

12. **Search Manager** It performs all searches of Users and Tournaments within the Platform. It organizes the data properly from results got by DBMS and makes them available via its interface.

13. **DBMS Manager** It is the component that manages the DBMS, allowing the Platform to perform operations on the DBMS. It directly manages connections to the database, keeping them few allowing to lighten the load to the DBMS without losing any functionality.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

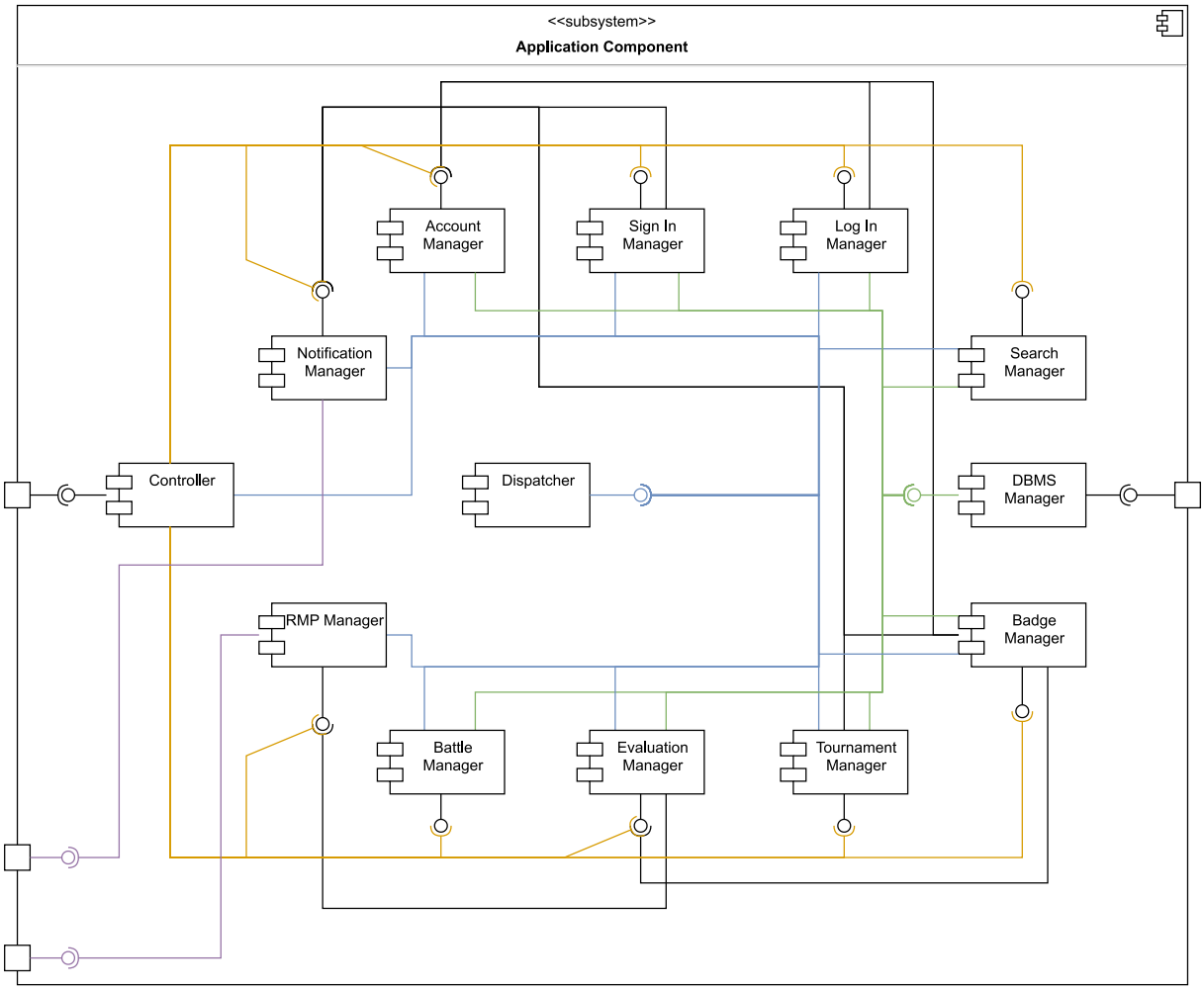

Figure 9: Application Server Component Diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

DBMS Server The DBMS server's job is to keep all of the Platform's data safe and always accessible. Information about Users, Tournaments, Battles and Badges are contained in the data.

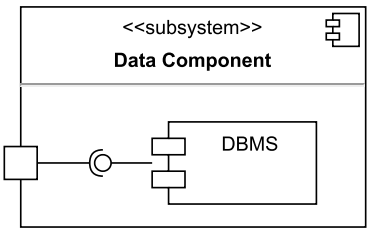

Figure 10: Client Component Diagram

External Services The role of the External Services is to provide the Platform with the functionalities that it needs to work properly. In particular, the E-mail Provider is used to send notifications to Users, while the RMP is used to perform pull requests.

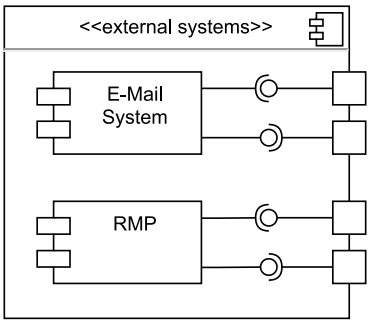

Figure 11: External Services Component Diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.4 Deployment View

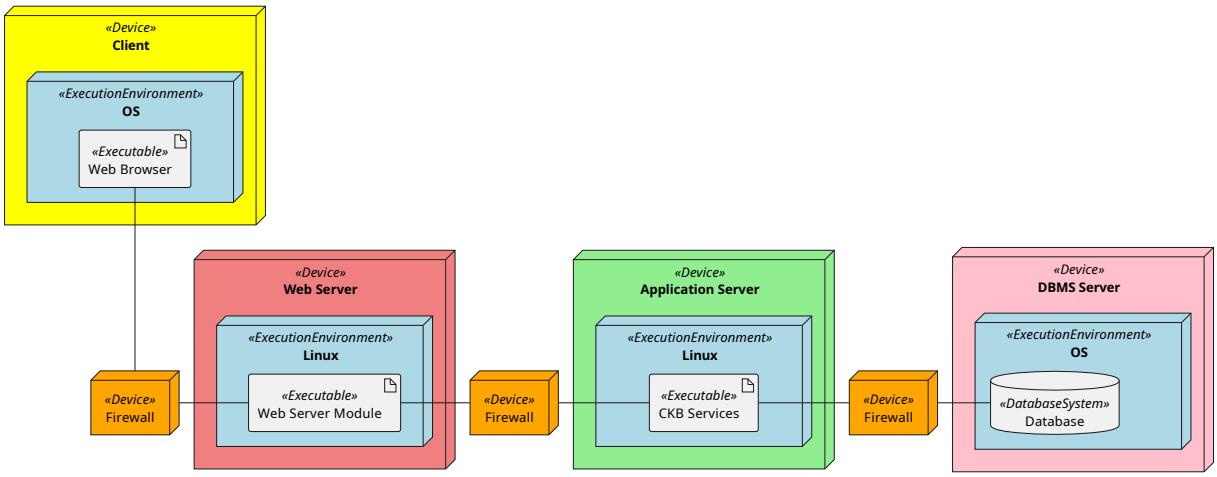

Figure 12: Deployment Diagram

As previously said, when CKB is deployed, it will include a Four Tiers Architecture that both reflects and extends upon the three Layers that rationally make up CKB. In fact, additional components will be needed for the implementation in order to handle User and Platform interactions as well as data module interaction.

- **Client:** Users input data into the Client, which displays responses from the Server and enables Users to navigate and interact with the Platform. In fact this Tier has its corresponding logical layer in the presentation one. Since a thin client only has communication APIs with the server, it transmits the components needed to create the most recent one as well as the necessary data to display the GUI. The appropriate Student and Educator variants make up the User visualizing interface, representing, as specified before, the actual presentation layer; differences between them exist exclusively at the architectural level, which is irrelevant to the operation of the system. However, this is true for other aspects of CKB functioning.

- **Web Server:** The primary entry point to the Platform is through the Web Server. It takes into account the interfaces that provide system access, the firewalls, the DMZ, and the module assigned to communicate with clients. In particular input from Clients, once achieved the Server, enter into the DMZ and as first step pass thought a first firewall that analyzes them to verify the provenience of the same. If it is granted them access, Web Server registers Client that sent it, parsing the content of the request, evaluating in addition their correctness. Furthermore, Web Server it is responsible for providing clients with information that updates the GUI and reflects changes made to the Platform. Web Server and Application Server are divided by a second firewall which role is to avoid undesired access to the actual model. Between Students, Educators, and CKB, this Tier serves as a bridge. Web Server constitutes part of the implementation of the Application Layer.

- **Application Server:** The Platform, with its essential features, is the Application Server. It is constructed using a Microservices Architecture, which is made up of lone software modules that oversee a single "service", or collection of tasks associated with a particular subject. Each module that performs a Microservice is better explained and analyzed in the previous and following section of this document. After parsing the requests, a dispatcher module that routes client requests to the appropriate service comes before the structure. Managing the interactions of several Users at once will be possible with a proper distribution of the Application Server across Microservices. A firewall delimits this Tier with the DMZ as described above, a second one separates the App with Email-Provider and RMP and a third defines the boarder with DMBS. This last one avoids

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

intrusions and damage into DBMS, while the fist two are deputed to protect Application Tier from external attacks or bad requests. Finally, this Server constitutes the reaming implementation of the Application Layer.

- **DBMS Server:** CKB keeps track of both User data and Platform operational data, such as Battle and Tournament records. DBMS is responsible for managing, securing, and storing the data. It permits access to Microservices that need it, and it updates it as needed in compliance with regulations. This Tier is the actual implementation of logic Data Layer and it is protected by a Firewall that filters requests and operations from Application Server.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5 Runtime View

2.5.1 Introduction to the Runtime View

The following sequence diagrams are meant to show the interactions between the components of the system. But some components are so crucial to the functioning of the system that the interactions are very frequent and would clutter these diagrams, making them of difficult fruition. For this reason the following interactions have to be intended as inserted each time there is a suitable communication.

Dispatcher Component This interaction is intended as placed everywhere there is a communication between components.
Where ComponentA and ComponentB are two generic components, Somewhere is a microservice, or Web Server or API manager or any component that triggers MicroserviceA to do something in a procedure, and Dispatcher is the dispatcher component.

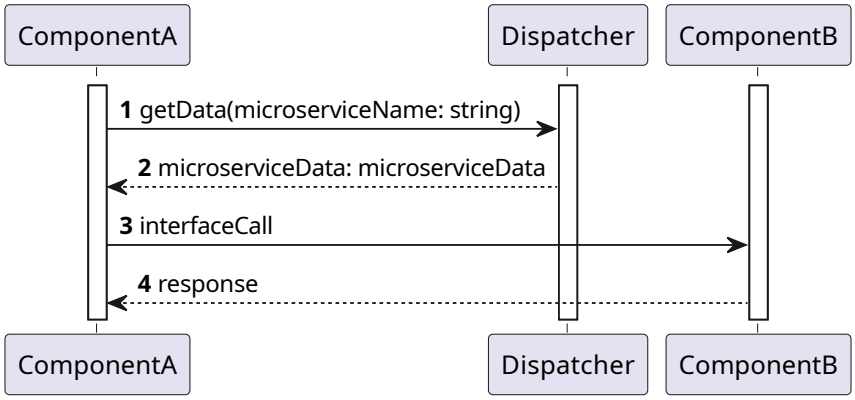

Figure 13: Dispatcher Sequence Diagram

DBMS Manager Component The interaction, here reported, is intended as placed any time there is a communication between this Manager and the DBMS. Microservice is the component that needs to perform a query to the database, while the component mediates it to the DBMS.

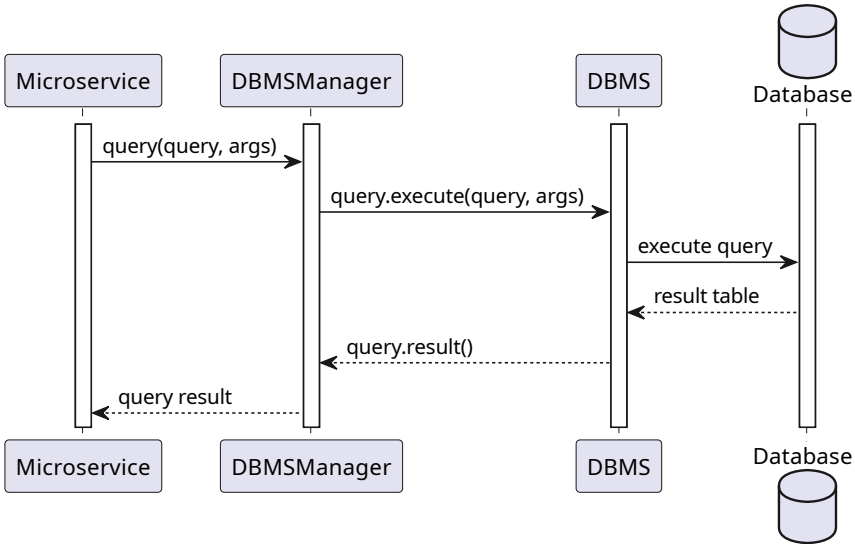

Figure 14: DBMS Manager Sequence Diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

API call authorization check Any time would be needed a communication between an external service, or client and the Web server module, it will happen as described below. Where the API call can be of any kind within the context described, request is any action the Platform has to perform as result of that call.

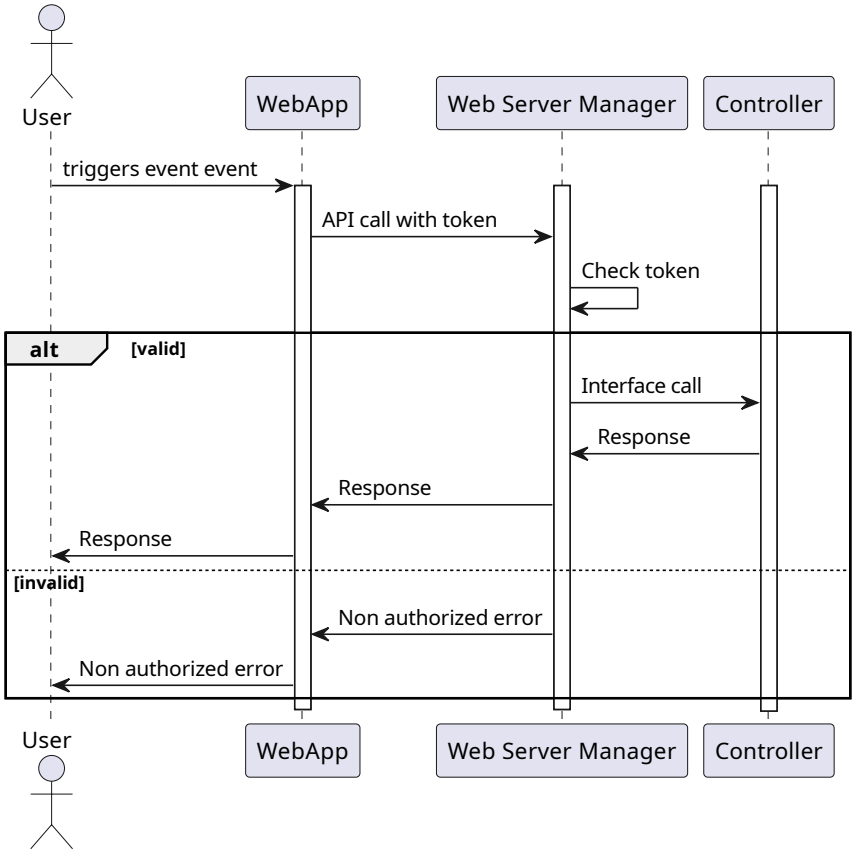

Figure 15: API call authorization check Sequence Diagram

Exception handling Some methods at the occurrence of certain conditions, can throw an exception. The error is propagated to the caller, which can handle it in the way it prefers. The exceptions represented in the sequence diagrams are used by the platform to notify to the User that something went wrong. The exceptions are not always represented in the sequence diagrams, but only when they are relevant to the understanding of the diagram.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.2 User signs in to the Platform

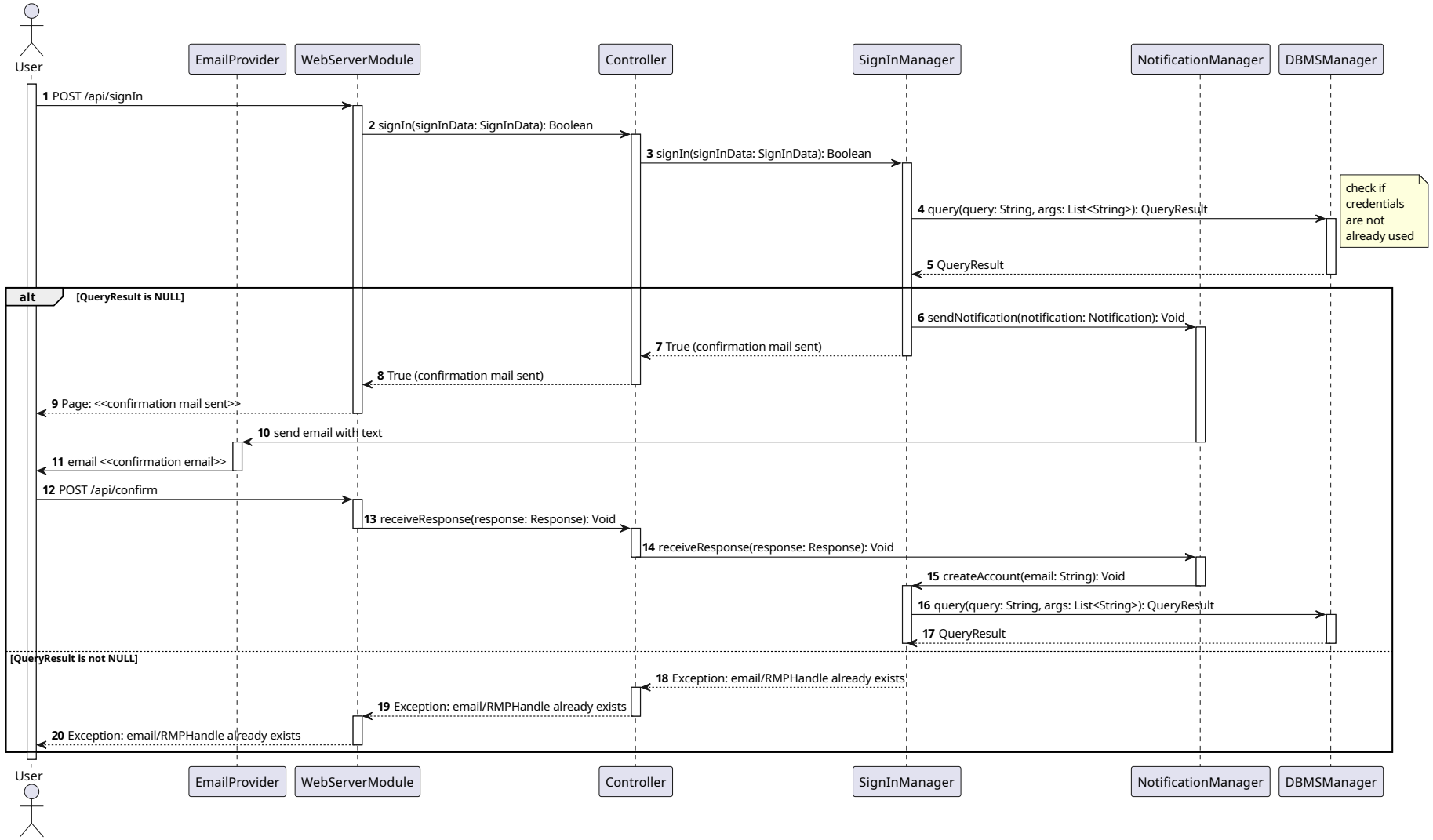

Figure 16: Sign In Sequence Diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

This sequence diagram shows the interactions between the components of the system when a User signs in into the Platform.
After determining that the user's credentials are valid, SignInManager displays to the user a page instructing them to confirm the operation by clicking the button on the email they received.
The User is registered and able to log in to the Platform following the confirmation.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.3 User logs in to the Platform

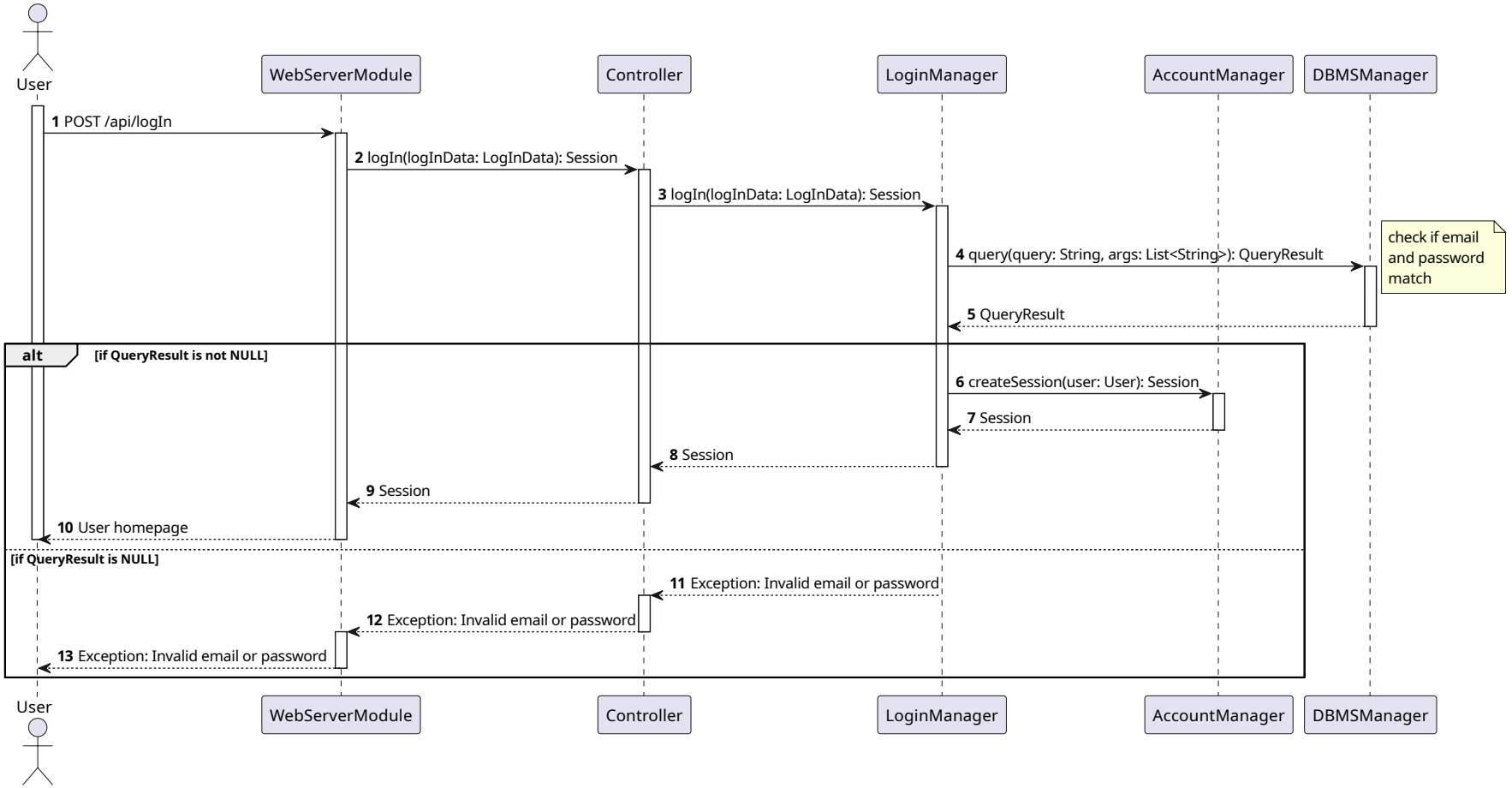

Figure 17: Log In Sequence Diagram

This sequence diagram shows the interactions between the components of the system when a User logs in into the Platform.
The LogInManager checks if the User's credentials are correct, and if so, it generates a session token and sends it to the Web Server.
If the credentials are not correct, the User is notified and the process ends.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.4 Student subscribes to Tournament

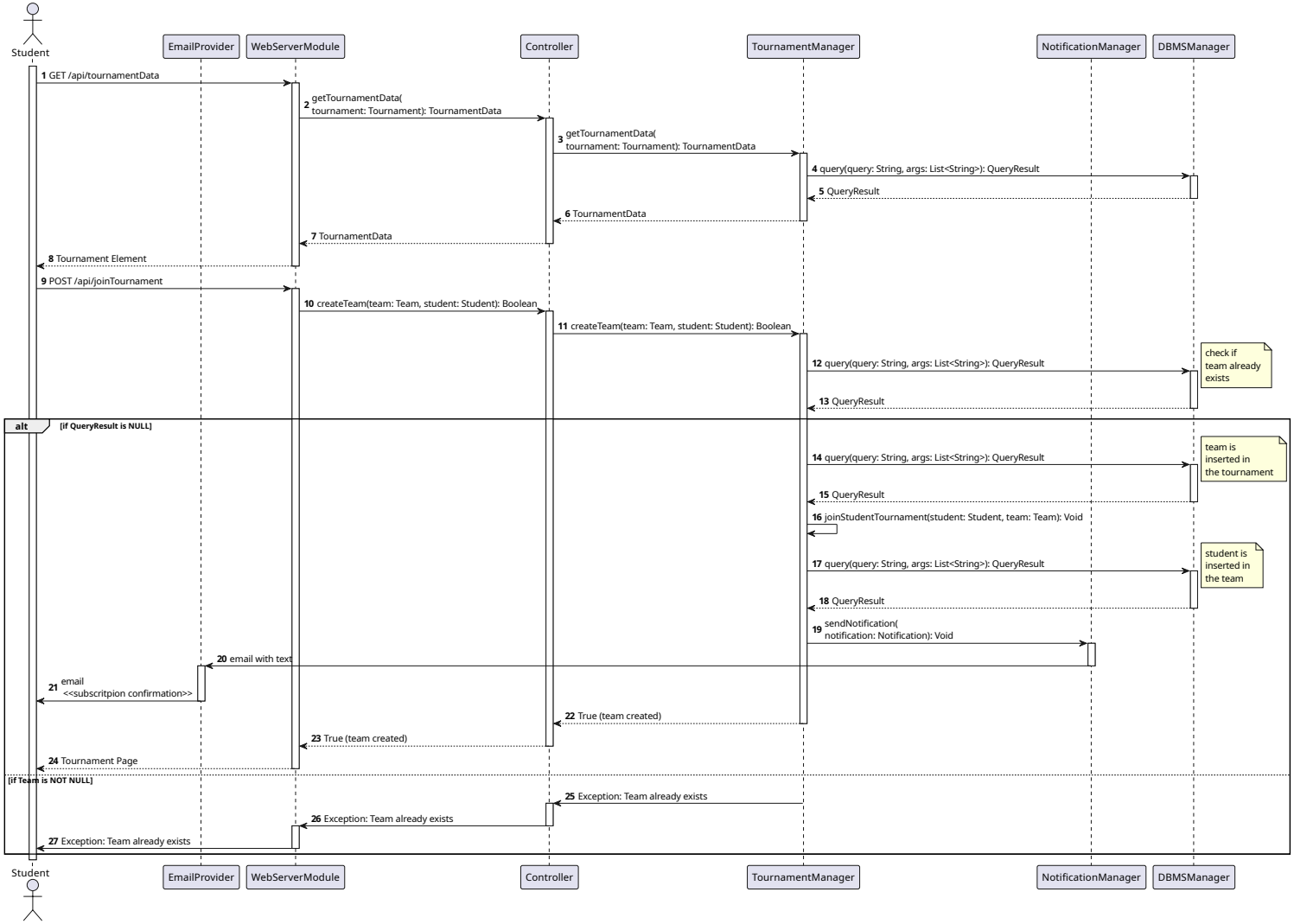

Figure 18: Join a Tournament Sequence Diagram

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

This sequence diagram shows the interactions between the components of the system when a Student subscribes to a Tournament.
First the Student searches for the Tournament they want to join, then they click on the "subscribe" button.
The TournamentManager first checks if the Team already exists, and if not, it creates it. We assume that a Student that is already subscribed to a Tournament, when he/she visits the Tournament page, he/she doesn't see the "subscribe" button, but just see the name of his/her Team, for this reason it is not checked if a Student is already subscribed to the Tournament. In other, because the Student's information and the Tournament's information are not provided by the User, but they are automatically inserted, Student and Tournament existence is not checked but assumed true.
Then the TournamentManager adds the Student to the new Team and email him/her to notify the subscription.
If the Team name already exists, the inviting Student is notified and the process ends.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.5 Educator invites other Educator to co-manage Tournament

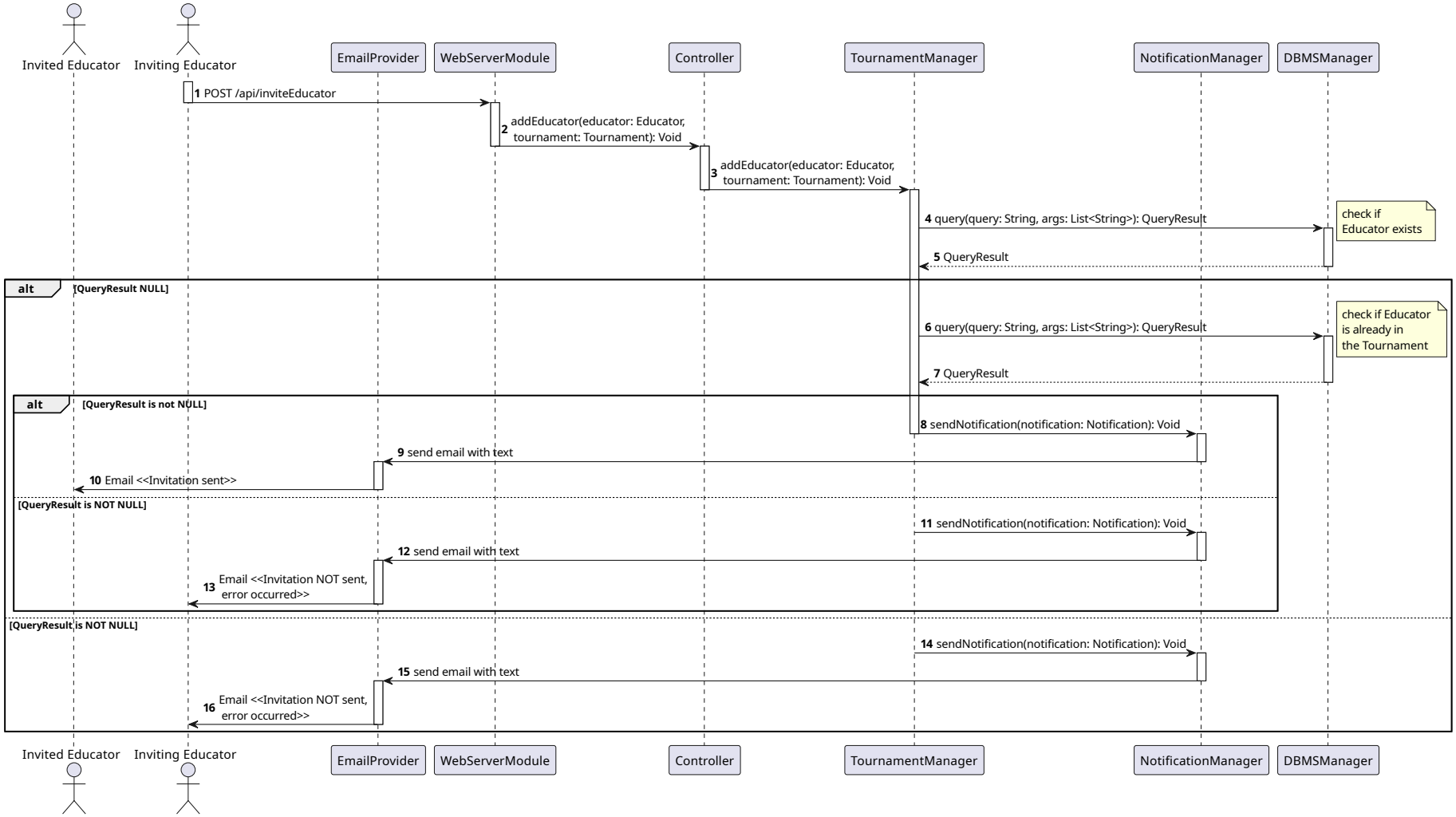

Figure 19: Invite an Educator Sequence Diagram

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

This sequence diagram shows the interactions between the components of the system when an Educator invites another Educator to co-manage a Tournament. First the TournamentManager checks if the invited Educator exists. Then it checks if the invited Educator is already involved in the Tournament, and if not, it sends an invitation to the Educator. The Tournament's existence is not checked and assumed true because the Tournament name is not provided by the User, but it is automatically added.
If the invited Educator provided doesn't exist or he/she is just in the Tournament, the Educator is notified and the process ends.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.6 Student invites other Student to participate in the Tournament as a Team

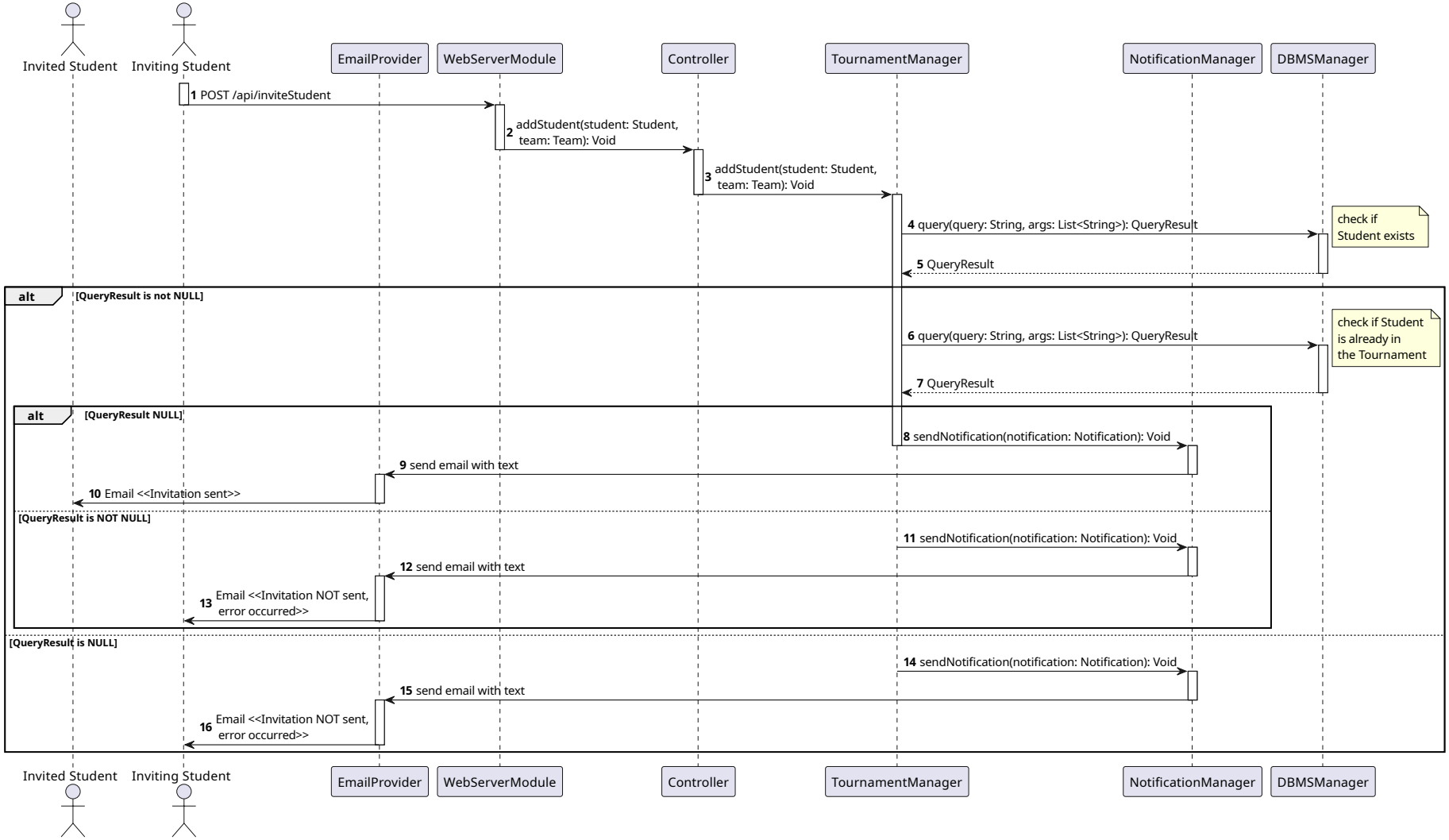

Figure 20: Invite a Student Sequence Diagram

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

This sequence diagram shows the interactions between the components of the system when a Student invites another Student to join his/her Team. The Tournament's existence is not checked and assumed true because the Tournament name (inserted in the Team object) is not provided by the User, but it is automatically added.
First the TournamentManager checks if the invited Student exists. Then it checks if the invited Student is already subscribed to the Tournament, and if not, it sends an invitation to the Student.
If the invited Student provided doesn't exist or he/she is just in another, the User is notified and the process ends.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.7 Receive Educator Invitation

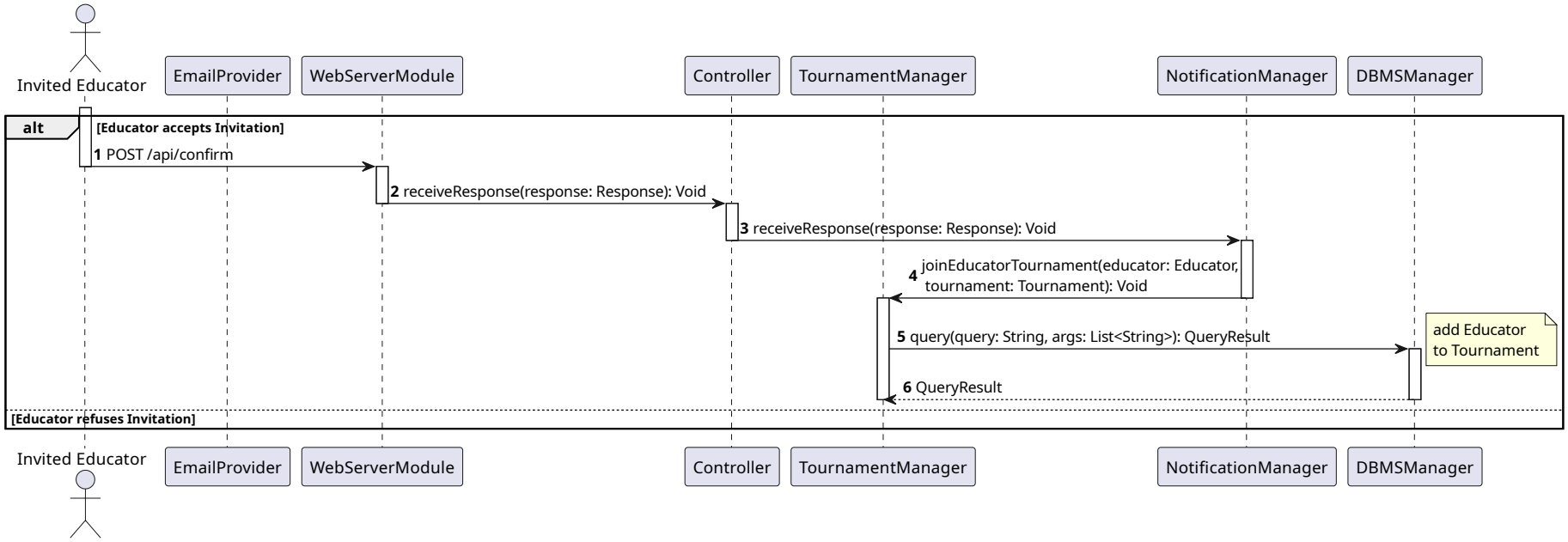

Figure 21: Receive an Educator invitation Sequence Diagram

This sequence diagram shows the interactions between the components of the system when an Educator receives an invitation to co-manage a Tournament. If the invited Educator click on the "accept" button, the TournamentManager adds the Educator to the Tournament.

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.8 Receive Student Invitation

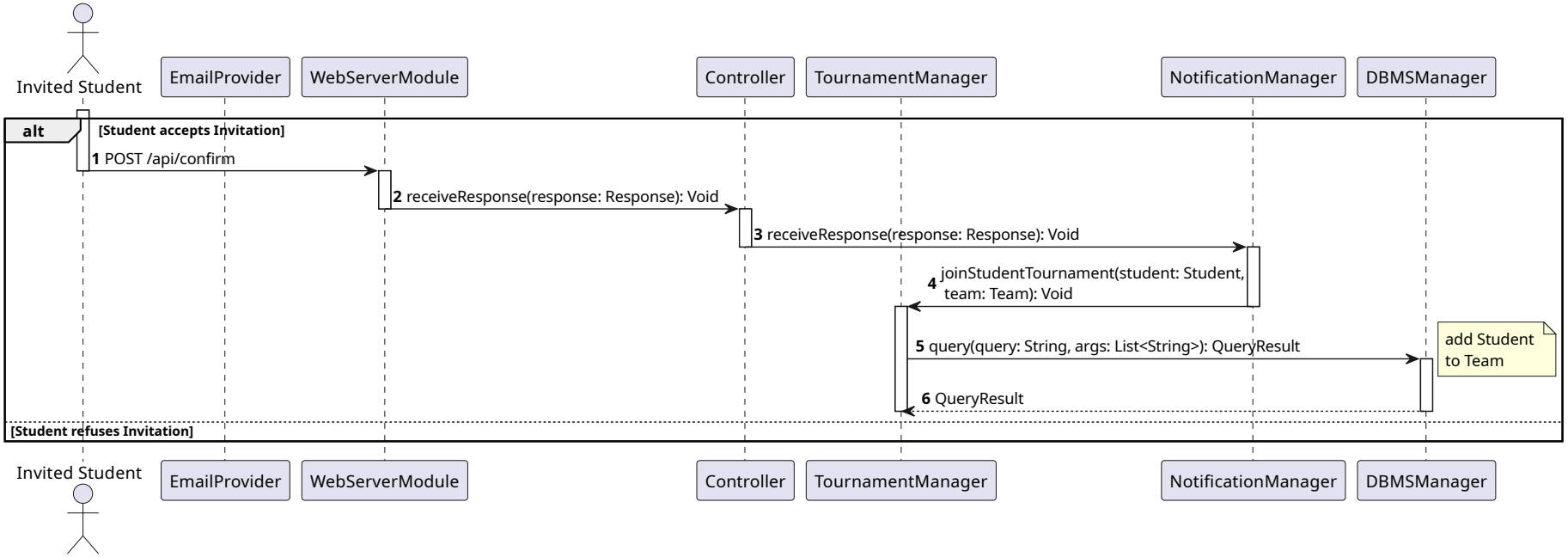

Figure 22: Receive a Student Sequence Diagram

This sequence diagram shows the interactions between the components of the system when a Student receives an invitation to join a Team.
If the invited Student click on the "accept" button, the TournamentManager adds the Student to the Team.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.9 Create a Tournament

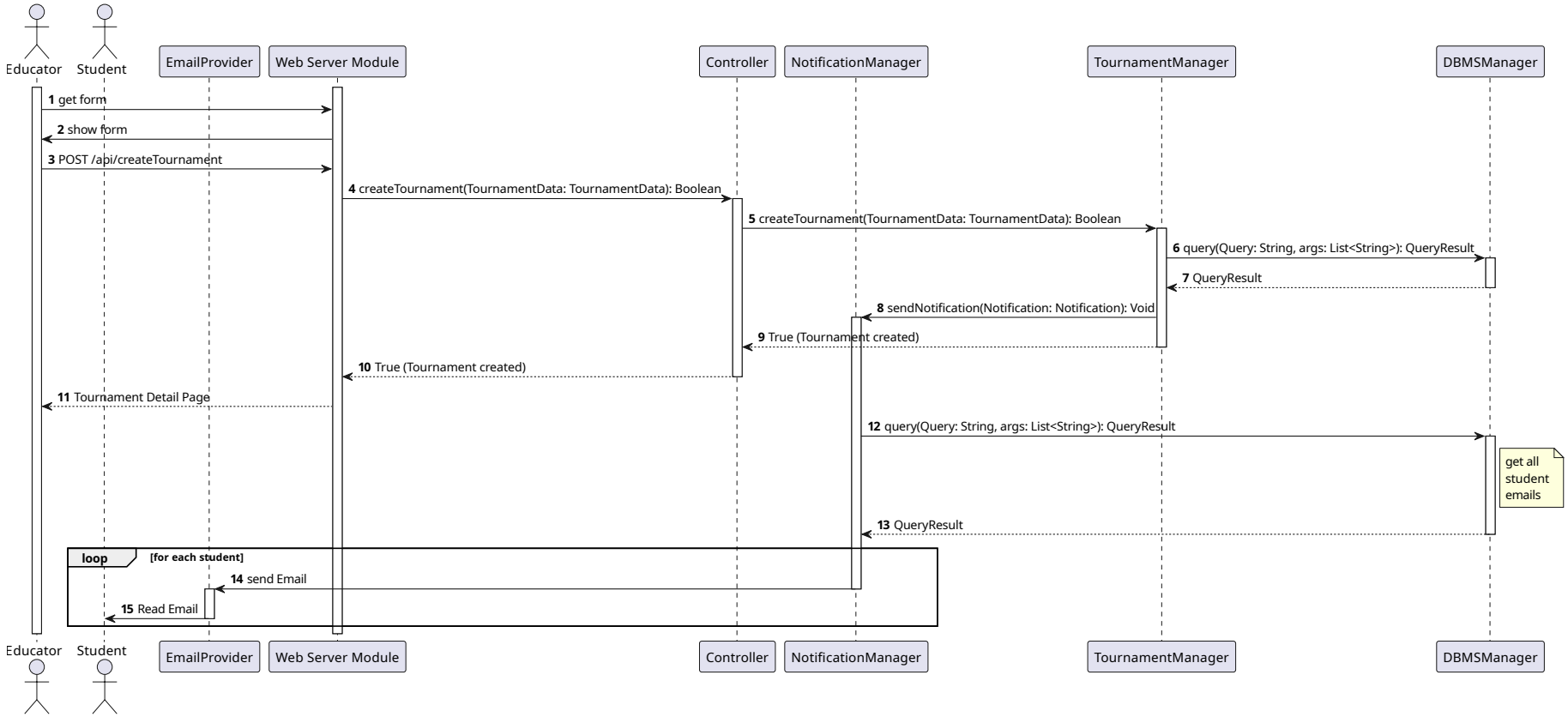

Figure 23: Create a Tournament Sequence Diagram

This sequence diagram shows the interactions between the components of the system when an Educator creates a Tournament.
After the creation the NotificationManager asks the DBMS manager the email of all the Students to notify them about the new Tournament.

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.10 Create a Battle

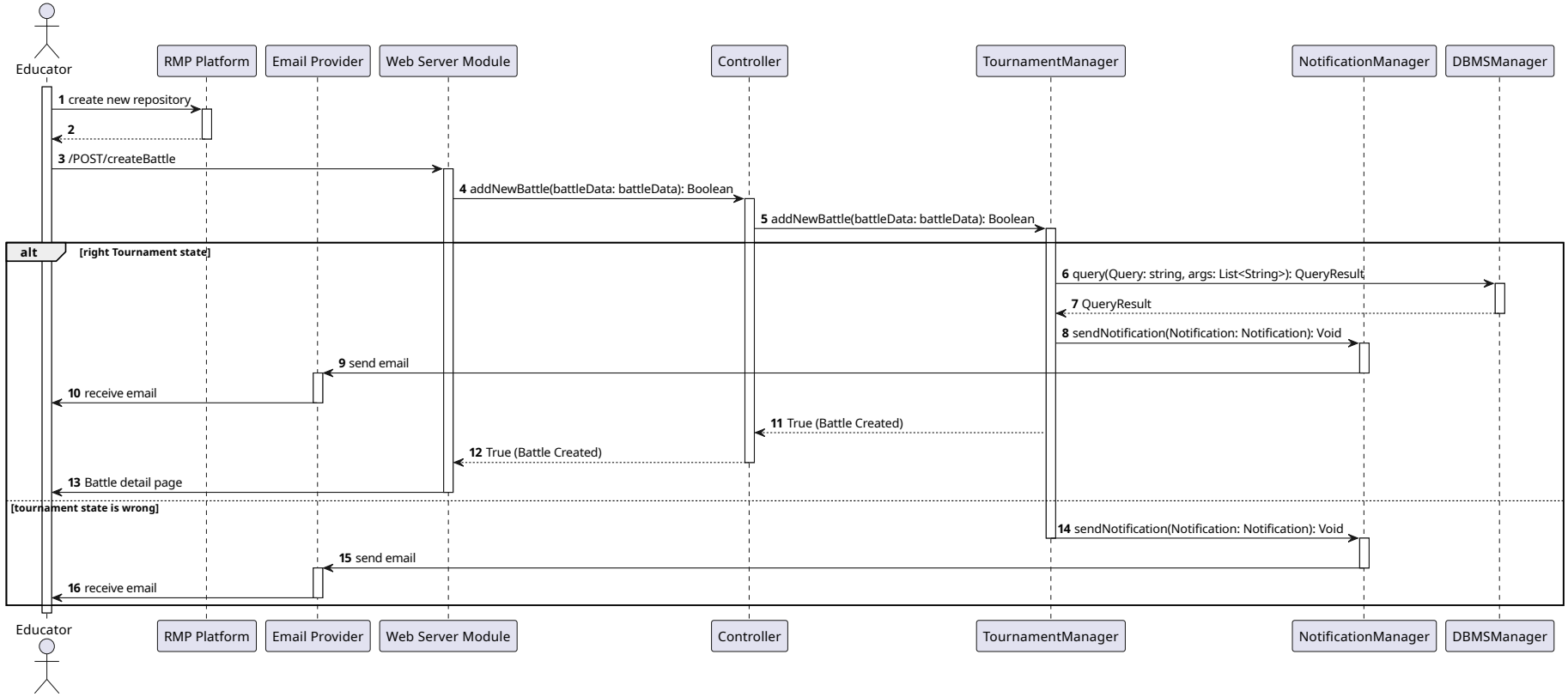

Figure 24: Create a Battle Sequence Diagram

This sequence diagram shows the interactions between the components of the system when an Educator creates a Battle.

2.5.11 Create a Badge

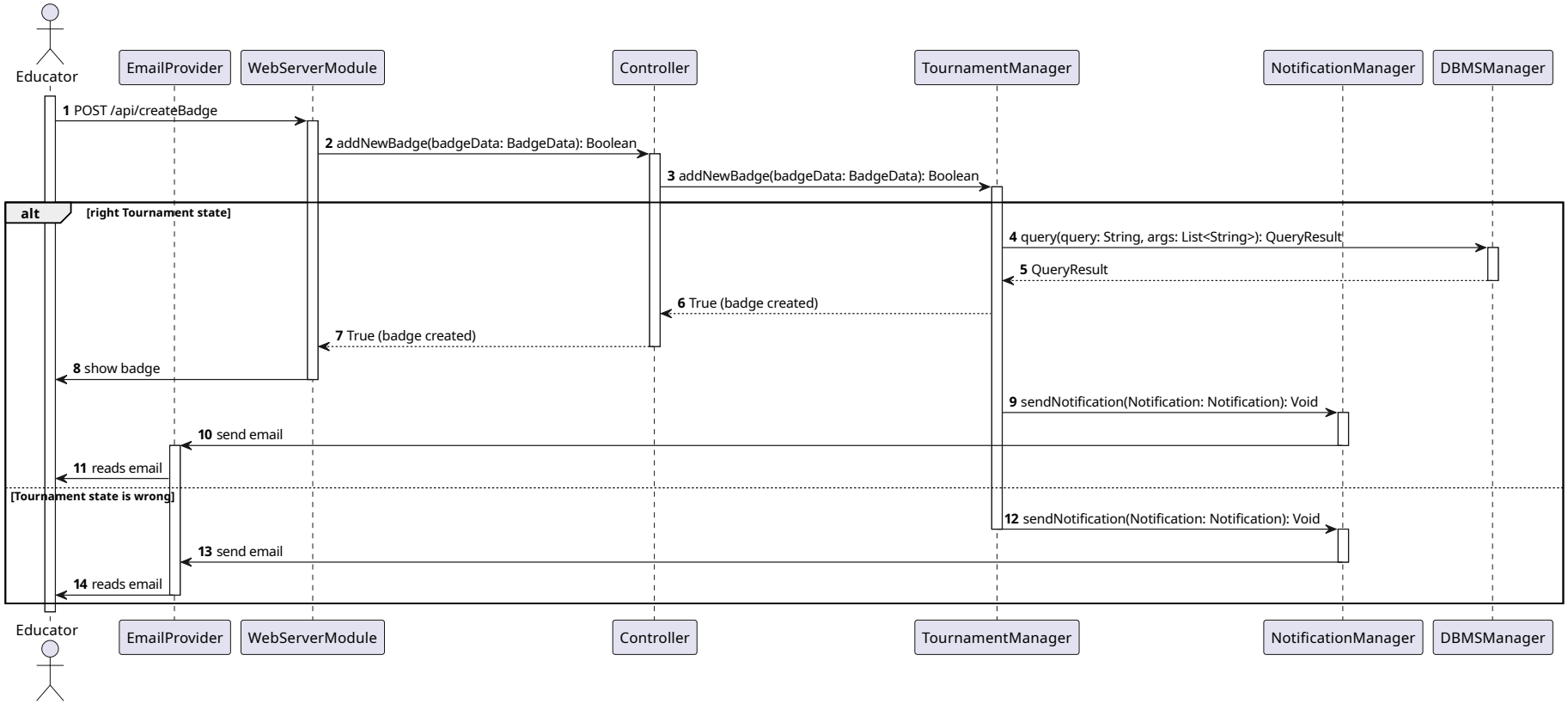

Figure 25: Create a Badge Sequence Diagram

This sequence diagram shows the interactions between the components of the system when an Educator creates a Badge.

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.12 Assign a Badge

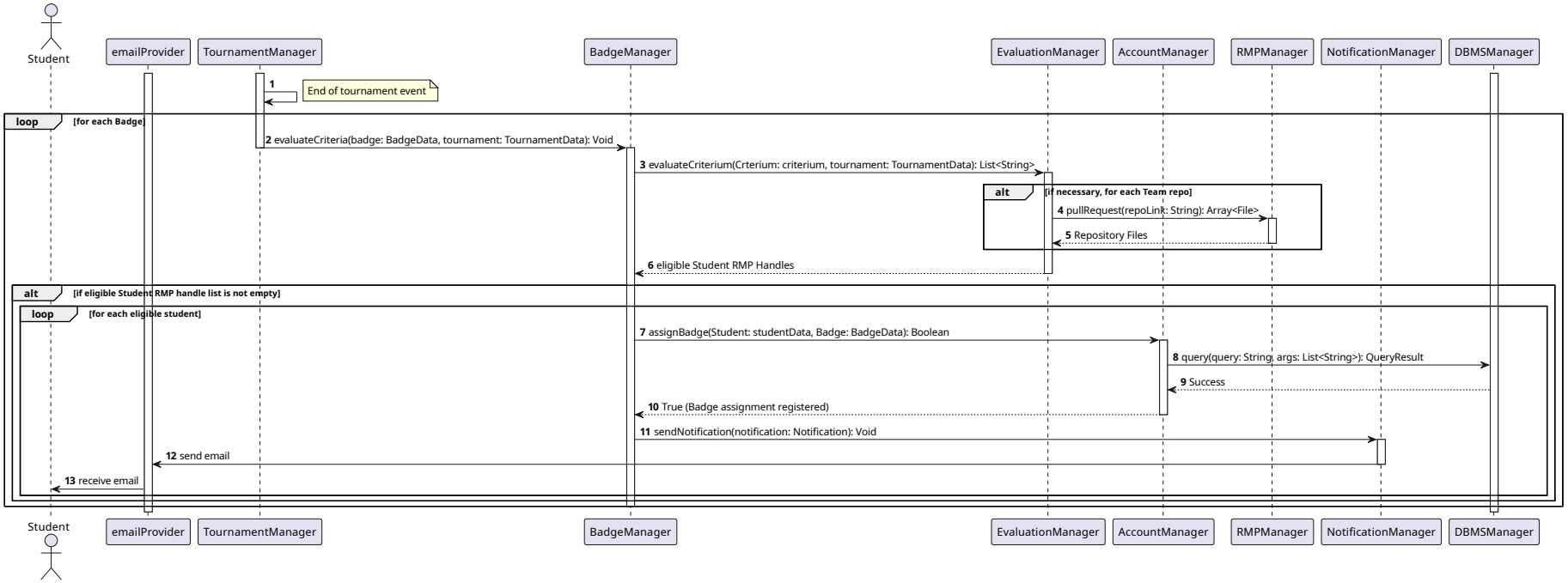

Figure 26: Assign a Badge Sequence Diagram

This sequence diagram shows the interactions between components of the system when the tournament ends and assigns Badges. The Tournament manager delegates the evaluation of the badge criteria to the Badge Manager, who making use of the Evaluation manager, finds eligible students to assign the badge. After that assigns it and notifies them via the Notification Manager.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.13 Evaluate Code

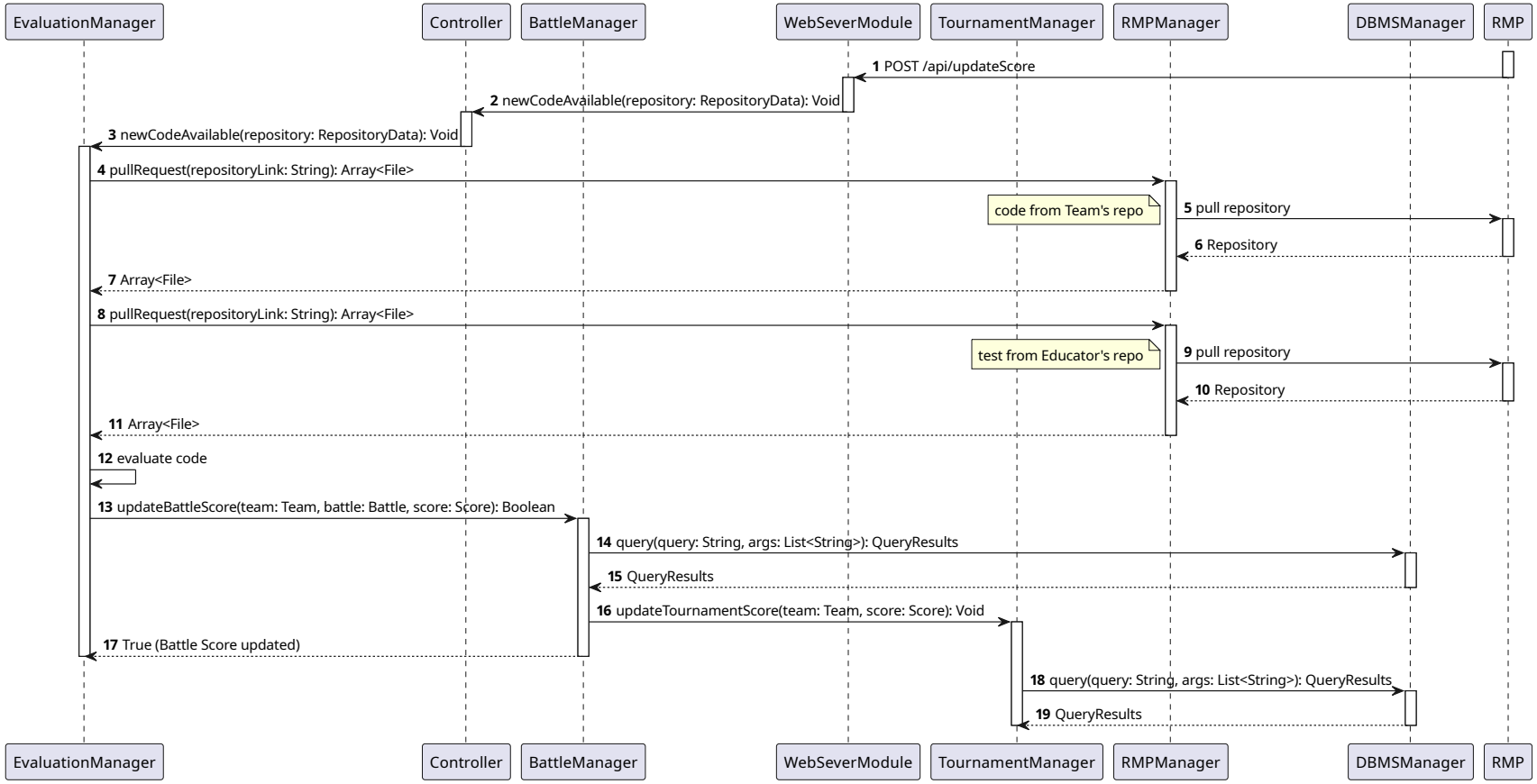

Figure 27: Evaluate Code Sequence Diagram

This sequence diagram shows the interactions between the components of the system when the Tournament Manager evaluates the code of a Team.
The Evaluation Manager requires two different repo: the one of the Team for the code and the one of the Educator for the tests in order to avoid that Students tamper them.
The Tournament Score is updated when the Battle Score has been updated.

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.14 Educator Manual Evaluate Code

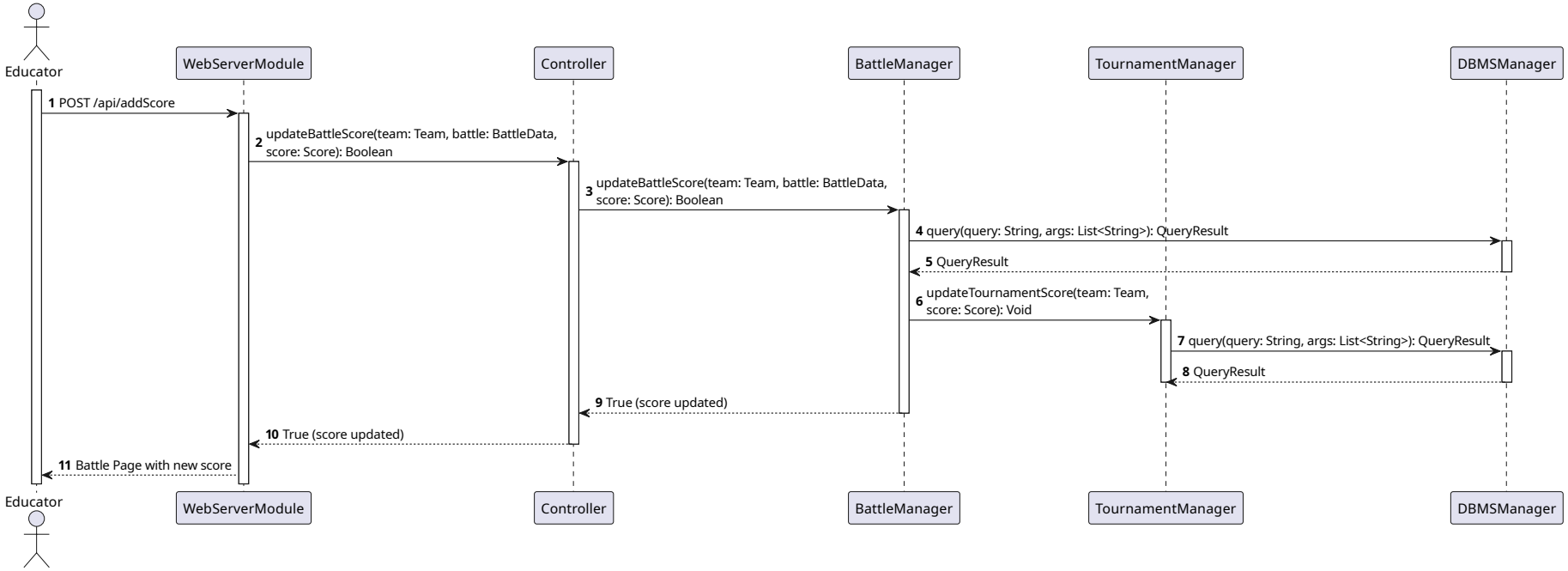

Figure 28: Educator Manual Evaluate Code Sequence Diagram

This sequence diagram shows the interactions between the components of the system when an Educator manually evaluates the code of a Team. The Tournament Score is updated when the Battle Score has been updated.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.15 User searches for Users

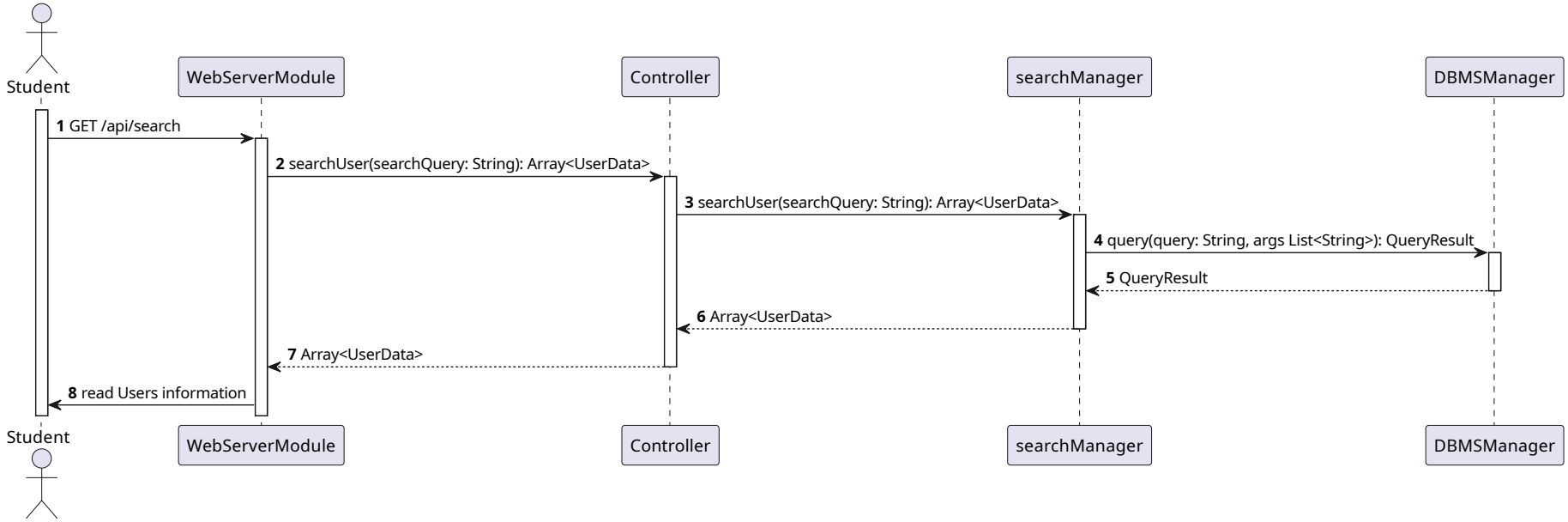

Figure 29: User searches for Users Sequence Diagram

This sequence diagram shows the interactions between the components of the system when a User searches for Users.

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.16 Student searches for Tournaments

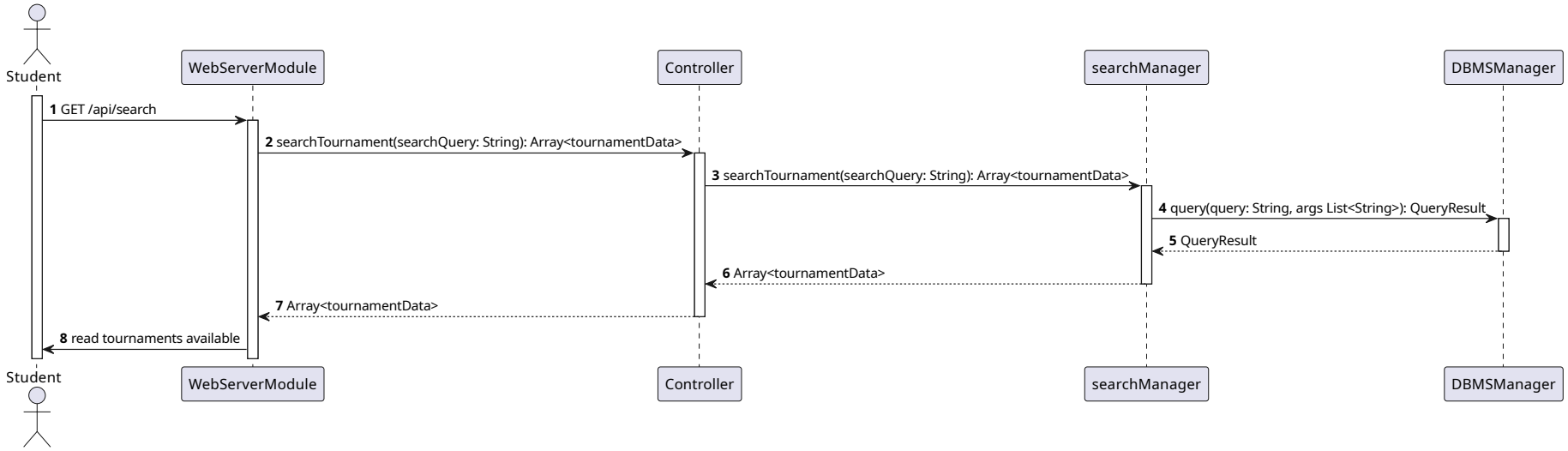

Figure 30: Student searches for Tournaments Sequence Diagram

This sequence diagram shows the interactions between the components of the system when a Student searches for Tournaments.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.5.17 Simple getters of Battle and Student Data

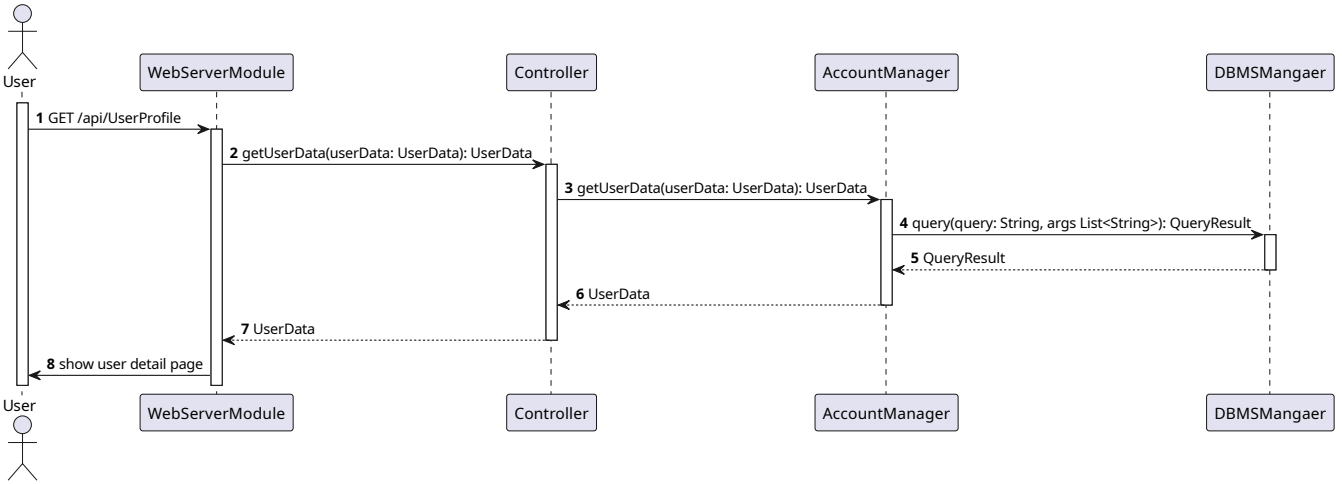

Figure 31: User data getter Sequence diagram

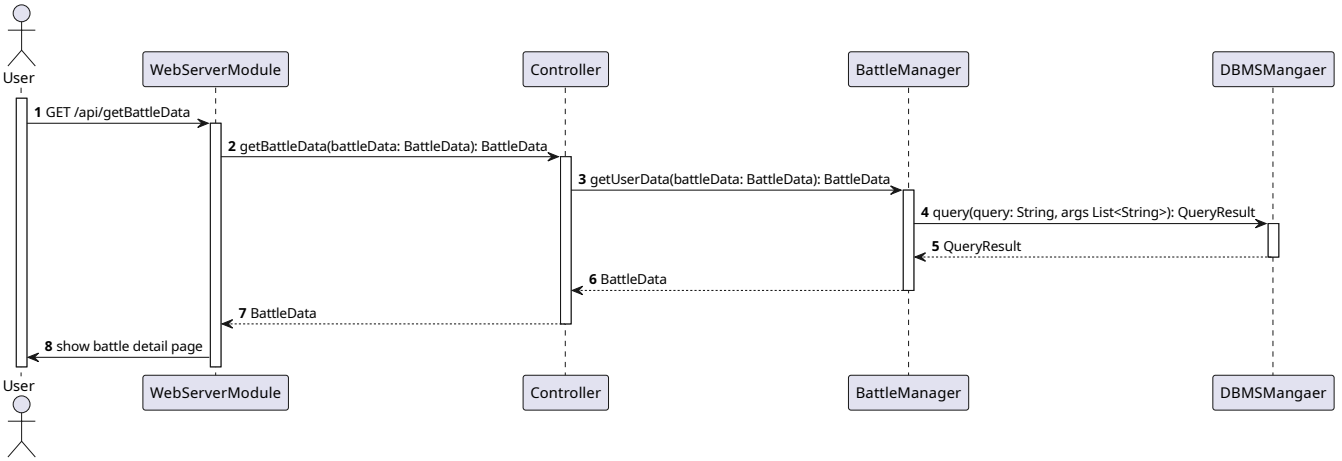

Figure 32: Battle data Sequence diagram

CodeKataBattle project by Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.6 Component Interfaces

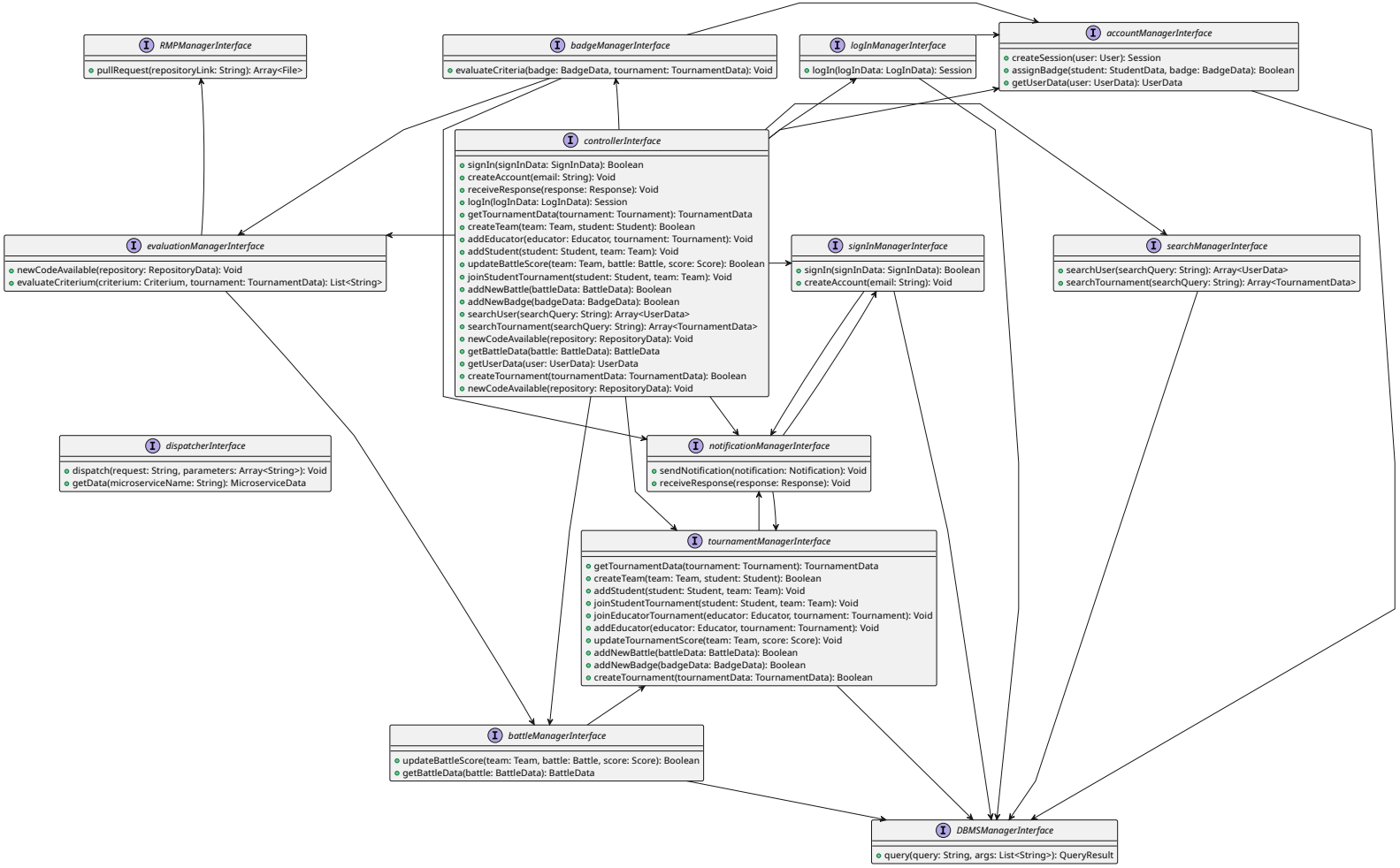

Figure 33: Component Interfaces Diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

For each component here are explained functions associated with respective interfaces.

1. **Controller Interface** This Component serves as a bridge between the Web Server and the Dispatcher. It allows for the Web Server to have only one interface to communicate with the Application Server, masking any possible deployment or implementation change to the latter. Given the role, it implements no logic, and its methods have all the same signature as the ones implemented in the microservices, which are the ones that contain the actual logic. Mapping between the API and this interface is given in the following section: API - Controller mapping.
   For this reason here follows only the list of the methods implemented by the Controller, while the description of the logic is provided in the corresponding microservice.

   - **signIn(signInData: SignInData): Boolean**
   - **createAccount(email: String): Void**
   - **receiveResponse(response: Response): Void**
   - **logIn(logInData: LogInData): Session**
   - **getTournamentData(tournament: Tournament): TournamentData**
   - **createTeam(team: Team, student: Student): Boolean**
   - **addEducator(educator: Educator, tournament: Tournament): Void**
   - **addStudent(student: Student, team: Team): Void**
   - **updateBattleScore(team: Team, battle: Battle, score: Score): Boolean**
   - **joinStudentTournament(student: Student, team: Team): Void**
   - **addNewBattle(battleData: BattleData): Boolean**
   - **addNewBadge(badgeData: BadgeData): Boolean**
   - **searchUser(searchQuery: String): Array<UserData>**
   - **searchTournament(searchQuery: String): Array<TournamentData>**
   - **newCodeAvailable(repository: RepositoryData): Void**
   - **getBattleData(battle: BattleData): BattleData**
   - **getUserData(user: UserData): UserData**
   - **createTournament(tournamentData: TournamentData): Boolean**
   - **newCodeAvailable(repository: RepositoryData): Void**

2. **SignInManager Interface**

   - **signIn(signInData: SignInData): Boolean** This method allows User to register to the Platform providing name, email, password and a RMP handle. All the information are contained in the SignInData object.
   - **createAccount(email: String): Void** This method create the account for the new User, providing the email. The other information are provided by the User in the signIn method.

3. **LogInManager Interface**

   - **logIn(logInData: LogInData): Session** This method allows User to log in into the Platform providing email and password. All the information are contained in the LogInData object. The return value is a Session, from it the User can operate as himself/herself on the Platform.

4. **RMPManager Interface**

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

- **pullRequest(repositoryLink: String): Array<File>** In order to perform a Pull request to the right RMP repo by RMPManager, the component, through the method described here, asks for the repo as a parameter. The return value is the code pulled from the repo itself.

5. **NotificationManager Interface**

   - **sendNotification(notification: Notification): Void** The method is invoked to send a notification to a User. The notification object contains the information about the notification to send and the content message.
   - **receiveResponse(response: Response): Void** The method is invoked to provide the User's answer. The response object contains the information about the User's answer, with the corresponding ConfirmationToken.

6. **BattleManager Interface**

   - **updateBattleScore(team: Team, battle: Battle, score: Score): Boolean** The method is invoked to update the Battle's score. The team object contains the Team's name, but also the Tournament's name in which the Team is. The battle object contains the Battle's information, which is the one that has to be updated. The score object contains the integer value of the score. The return value is a Boolean that confirms the update of the Battle's score.
   - **getBattleData(battle: BattleData): BattleData** The method is invoked to get all data that the Platform has about a Battle. The battle object contains enough information about the Battle that is requested, to be able to find the right one.

7. **TournamentManager Interface**

   - **getTournamentData(tournament: Tournament): TournamentData** The method is invoked to get all data that the Platform has about a Tournament. The tournament object contains the information about the Tournament that is requested.
   - **createTeam(team: Team, student: Student): Boolean** The creation of the Team is deputed to this method. The team object contains the Team's name, but also the Tournament's name in which the Team wants to subscribe. The student object contains the Student's information, which is the creator of the Team. The return value is a Boolean that confirms the creation of the Team.
   - **addStudent(student: Student, team: Team): Void** This method allows a Student to invite another Student to join a Team. The student object contains the information about the Student that is inviting, while the team object contains the information about the Team in which the Student is inviting and the Tournament's name in which the Team is.
   - **joinStudentTournament(student: Student, team: Team): Void** This method allows a Student to join a Tournament in a Team. The student object contains the Student's information, which is the one who wants to join the Tournament and the team object contains the Team's name, but also the Tournament's name in which the Student wants to subscribe.
   - **joinEducatorTournament(educator: Educator, tournament: Tournament): Void** This method adds an Educator with the given email to the Tournament. The educator object contains the information about the Educator that is joining, while the tournament object contains the information about the Tournament in which the Educator is joining.
   - **addEducator(educator: Educator, tournament: Tournament): Void** This method allows an Educator to invite another Educator to co-manage a Tournament. The educator object contains the information about the Educator that is inviting, while the tournament object contains the information about the Tournament in which the Educator is inviting.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

- **updateTournamentScore(team: Team, score: Score): Void** The Score of the Tournament has to be updated via this function. The team object contains the Team's name, but also the Tournament's name in which the Team is. The score object contains the integer value of the score.
- **addNewBattle(battleData: BattleData): Boolean** The method allows to add a new Battle to the current Tournament. It would be required the Battle's information comprised with evaluation criteria. The return value is a Boolean that confirms the creation of the Battle.
- **addNewBadge(badgeData: BadgeData): Boolean** The method allows to add a new Badge to the current Tournament. It would be required the Badge's information comprised with assignment criteria. The return value is a Boolean that confirms the creation of the Badge.
- **createTournament(tournamentData: TournamentData): Boolean** The method allows to create a new Tournament. It would be required the Tournament's information and the return is a Boolean that confirms the creation of the Tournament.

8. **AccountManager Interface**

   - **createSession(user: User): Session** The method is invoked to create a new session for the User. The User object contains the information about the User that is logging in and requesting the session.
   - **assignBadge(student: StudentData, badge: BadgeData): Boolean** Via this function is assigned a Badge to the Student, providing the corresponding BadgeManager. The return value is a Boolean that confirms the assignment of the Badge.
   - **getUserData(user: UserData): UserData** The method is invoked to get all data that the Platform has about a User. The user object contains enough information about the User to get the right one.

9. **SearchManager Interface**

   - **searchUser(searchQuery: String): Array<UserData>** To search one or more Users, SearchManager, throught the method presented here requires the email or name or surname or RMP handle of the User to search and returns the list of the Students'/Educators' data of the ones who matched the criteria.
   - **searchTournament(searchQuery: String): Array<TournamentData>** To search one or more Tournaments, SearchManager, through the method presented here requires the name of the Tournament or its properties to search and returns the list of the Tournaments' data of the ones who matched the criteria.

10. **BadgeManager Interface**

    - **evaluateCriteria(badge: BadgeData, tournament: TournamentData): Void** BadgeManager is invoked via this method to verify if some tests results meet criteria to assign a Badge to all eligible Students.

11. **EvaluationManager Interface**

    - **newCodeAvailable(repository: RepositoryData): Void** The method allows to evaluate the code provided as input, basing on the evaluation parameters. Both the code and the test files are required. The return value is the score of the code.
    - **evaluateCriterium(criterium: Criterium, tournament: TournamentData): List<String>** The method allows to evaluate the criterium on the tournament data, considering, if necessary, also student's code. The return value is the list of the Students' RMP handles that meet the criterium.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

12. **Dispatcher Interface**

    - **dispatch(request: String, parameters: Array<String>): Void** The Dispatcher is the component that routes the requests to the right Microservice. It is invoked by the Web Server, which provides the request and the parameters. The Dispatcher will call the right Microservice, passing the parameters.
    - **getData(microserviceName: String): MicroserviceData** The method is invoked by a microservice to know the data of another microservice as discussed in: Dispatcher Component. The parameter is the name of the microservice that is requesting the data.

13. **DBMS Manager Interface**

    - **query(query: String, args: List<String>): QueryResult** The method is invoked by a microservice to perform a query to the database. The first parameter is the Query to be precompiled, the second an array of arguments to be inserted in the query.

2.7 Platform API specification

In this section will be proposed the specification of the Platform API, that will be used by the Web App to communicate with the Platform. For convenience this API is shown divided in three parts: firstly the part dedicated to the Web App, secondly the one dedicated to the E-Mail confirmations, thirdly the one dedicated to the RMP actions.

2.7.1 Web App API

- **POST /api/signIn** This call allows a User to sign in into the Platform. The body of the request must contain the following fields: Name, Surname, Email, Password, RMP Handle, UserType. It returns the confirmation of successful sign in, or an error message.

- **POST /api/logIn** This call allows a User to log in into the Platform. The body of the request must contain the following fields: Email, Password. It returns the confirmation of successful log in with the session token, or an error message.

All of the following have a common argument, that is the User session token. This token is used to authenticate the User and to check if he/she has the right to perform the call. So all the following calls are valid if and only if the User is logged in.

- **GET /api/search** This call allows a User to search for other Users or Tournaments. The body of the request must contain the following fields: SearchType, SearchParameters. It returns the list of Users or Tournaments that match the search parameters.

- **GET /api/UserProfile** This call allows a User to get all data that the Platform has about a User. The body of the request must contain the following fields: Email. It returns the User public data, and, in the case of Students, also the list of Badges.

- **GET /api/tournamentData** This call allows a User to get all data that the Platform has about a Tournament. The body of the request must contain the following fields: TournamentName. It returns the Tournament data.

- **POST /api/createTournament** This call allows an Educator to create a Tournament. The body of the request must contain the following fields: TournamentName, TournamentDescription, TournamentLanguage, TournamentMaxTeamSize, SubscriptionDeadline, TournamentDuration. It returns the confirmation of successful creation, or an error message.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

- **POST /api/joinTournament** This call allows a Student to join a Tournament. The body of the request must contain the following fields: Tournament, Team, Student. It returns the confirmation of successful join, or an error message.

- **POST /api/inviteStudent** This call allows a Student to invite another student to his team. The body of the request must contain the following fields: Tournament, TeamName, StudentEmail. It returns the confirmation of successful invitation, or an error message.

- **POST /api/inviteEducator** This call allows an Educator to invite another Educator to his Tournament. The body of the request must contain the following fields: Tournament, EducatorEmail. It returns the confirmation of successful invitation, or an error message.

- **POST /api/createBadge** This call allows an Educator to create a new Badge in the context of a Tournament. The body of the request must contain the following fields: Tournament, BadgeName, BadgeDescription, BadgeCriteria. It returns the confirmation of successful creation, or an error message.

- **POST /api/createBattle** This call allows an Educator to create a new Battle in the context of a Tournament. The body of the request must contain the following fields: Tournament, BattleName, BattleDescription, BattleRMPRepo. It returns the confirmation of successful creation, or an error message.

- **GET /api/getBattleData** This call allows a User to get all data that the Platform has about a Battle. The body of the request must contain the following fields: Tournament, BattleName. It returns the Battle data, including the Team Scoreboard.

- **POST /api/addScore** This call allows an Educator to assign a custom score to a Team in a Battle. The body of the request must contain the following fields: Tournament, BattleName, TeamName, Score. It returns the confirmation of successful assignment, or an error message.

2.7.2 E-mail API

- **POST /api/confirm** This call allows a User to confirm the request. The request can be: e-mail confirmation and invitation confirmation. The body of the request must contain the following fields: Email, ConfirmationToken. It returns the confirmation of successful confirmation, or an error message.

2.7.3 RMP Action API

- **POST /api/updateScore** This call allows the Platform to perform a pull request from a RMP repo with the objective to evaluate the code. The body of the request must contain the following fields: RepoLink, ValidyToken. It returns the confirmation of successful call, or error to be displayed in the Actions panel of the RMP.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.8 API - Controller mapping

API	Controller Interface
POST /api/logIn	login
POST /api/signIn	signIn
GET /api/search	searchUser, searchTournament
GET /api/UserProfile	getUserProfile
GET /api/tournamentData	getTournamentData
POST /api/createTournament	createTournament
POST /api/joinTournament	createTeam
POST /api/inviteStudent	inviteStudent
POST /api/inviteEducator	inviteEducator
POST /api/createBadge	addNewBadge
POST /api/createBattle	addNewBattle
GET /api/getBattleData	getBattleData
POST /api/addScore	updateBattleScore
POST /api/confirm	receiveResponse
POST /api/updateScore	newCodeAvailable

Table 1: API - Controller mapping

2.9 Notable observations

2.9.1 Authorization Management

The management of the validity of certain API calls is done via the use of Tokens. These tokens are considered as of three types:
User Session Token This token is used to authenticate the User and to check if he has the right to perform the call. This is generated after a successful login and is valid for the whole session, or until the User logs out. It contains information about the User, such as e-mail and role.
Confirmation Token This token is used to confirm the validity of a certain action, such as the confirmation of an e-mail or the confirmation of an invitation. This is generated when is requested a confirmation action from the User. It contains the e-mail of the User and the type of confirmation.
Validity Token This token is used by the Platform to understand if the request to evaluate some code is legitimate or not. Once registering the repo to the Platform, this token is generated and saved in the repo. It contains the Tournament end date, the Battle.
All of the above tokens have to be encrypted to ensure their security and authenticity. In the Runtime View is shown the corresponding sequence diagram for the authorization check of an API call: API call authorization check

2.9.2 Battle Evaluation

For the automatic evaluation of the Battles is advisable to use an automatic evaluation system based on containers. This kind of evaluation system is useful and satisfies different requirements such as:
Security: the testing environment is isolated and has no access to the rest of the application. A vulnerability at this component would be catastrophic since it bypasses all security measures placed before.
Reusability: the testing environment, via the use of a suitable script, can be used to test any kind of code in a completely automatic manner.
In order to work well, is needed that all Battles repos have a RMP action active, which scope is to alert the Platform of any new update of the repo. This will trigger two pushes from the Platform, one from

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

the Team Repository, the other from the Educator repository. This is done to ensure that Teams don't tamper with the evaluation tests, that are saved in the Battle repository, while being able to use them for local evaluation. After saving everything, a new container is spawned that will have to run the evaluation script from the repository. At the end a special file will be created, that will contain the result of the evaluation, and most importantly the score of the team in that Battle to be saved in the database in order to update corresponding Tournament and Battle scores. All of this will be done internally by the Evaluation Manager.

2.9.3 Badge Evaluation

For the automatic evaluation of the Badges is advisable to define a set of rules with which define the criteria for the assignment of it. These criteria will be saved upon badge creation and will be used by the Badge Manager at the end of the Tournament to evaluate tho whom assign it. The definition of the rules to be used to write the criteria allows to reuse the same evaluation system for different Badges with same rules, in a standardized manner without the need of exposing too much of the internal logic of the system.

2.9.4 Dispatcher Component

The dispatcher is the backbone of the whole Microservice Architecture, because it manages load balancing, routing and communication between components and if necessay spawn or kill instances of components. Components are assumed to share interfaces between each other but before any kind of communication, they need to call the dispatcher, who will provide them with valid reference to an instance of a component suitable to receive the request. This allows the dispatcher to do load balancing, so it can decide which instance make the request go to, and consequently which components replicate. The Dispatcher also manages the DBMS Manager component that will be better described below. There can be multiple DBMS Managers occurrences but is advisable to keep their lives long and to not spawn and kill them too frequently. This is done to allow future scalability and flexibility, since the proposed solution involves the use of only one DBMS and one Database, but, if necessary, in future can be expanded without changing anything in the architecture. From this the dispatcher interface has also a method **getData()** to get a reference to a component, necessary to allow components to communicate between each other.

2.9.5 External Actors different from the User

This Platform has to interact with RMP and E-mail provider, which are external actors that do not interact with the system via the Web App, but via APIs. In fact, RMP repos implement an action whose scope is to call the Platform's API to notify it of any update to the repository, similarly Users use e-mail to respond to notifications sent by the Platform, again via simple buttons present in the notifications that allows to send confirmation or denial answers via the Platform's API. Is also useful to denote that the Platform, making use of those two services, behaves as a client for them. This is done via the use of the respective RMP Manager and E-mail Manager components.

2.9.6 DBMS manager

This component is in charge of performing operations to the data tier. One of the most heavy actions on the database is the opening and the closing of connections, needed to interact with the components, so if all of them were to dynamically open and close connections to the database, the load on the DBMS would be too high, and the system would be slow. Inserting this manager allows to keep the number of connections low and keep them more time open, reducing the load on the DBMS while not losing any functionality for other modules. So DBMS manager is to be intended as a proxy between the DBMS and the other components, that will use it to perform queries to the database, and for each access to the

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

database the following Sequence diagram can be referenced:

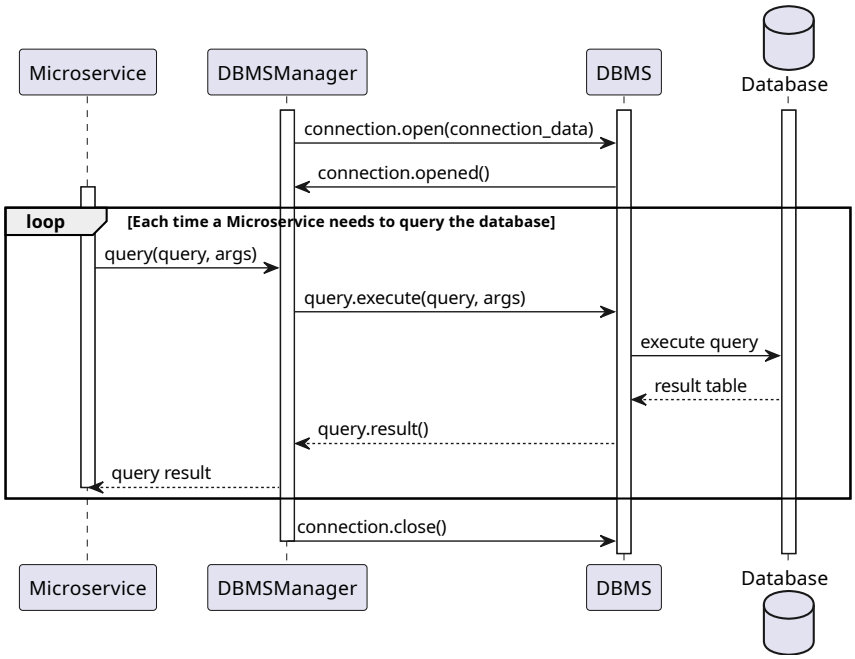

Figure 34: DBMS Manager Sequence Diagram

Where DBMS Manager is the component, and MicroserviceA is the component that needs to perform a query to the database.

2.10 Selected architectural styles and patterns

2.10.1 Four-Tier Architecture

This kind of architecture as discussed above distinguishes the Platform in four Tiers. This allows to have better code differentiation, so that each Tier has its own responsibility and can be developed independently of the others. Moreover, allows for a more secure system, since before accessing saved data, any kind of attack shall pass through each tier to reach the last one.

2.10.2 Model View Controller

A popular software design pattern for creating user interfaces, model-view-controller, or MVC, splits the associated program logic into three interconnected parts. This is done in order to keep information presented to and accepted from the user separate from internal representations of that information.

- **Model** It is the core of the application, it is the part that manages the data, logic and rules of the application. It receives the input from the Controller and updates the View. The module that belongs to this component are all the Managers of the Application Server and the DBMS.
- **View** It is the part that manages the user interface, it receives the input from the Controller and shows the result to the User. The module that belongs to this component is the Web Server that elaborate the graphical elements and send them to the Web App of the Client.
- **Controller** It is the part that manages the interaction between the Model and the View. It receives the input from the View and sends it to the Model, then it receives the result from the Model and sends it to the View. The module that belongs to this component is the Controller of the Application Server.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.11 Other design Decisions

2.11.1 Relational Database

The DBMS chosen for this Platform is a relational database. This choice is due to the fact that the data that the Platform has to store is structured and has a well-defined schema.

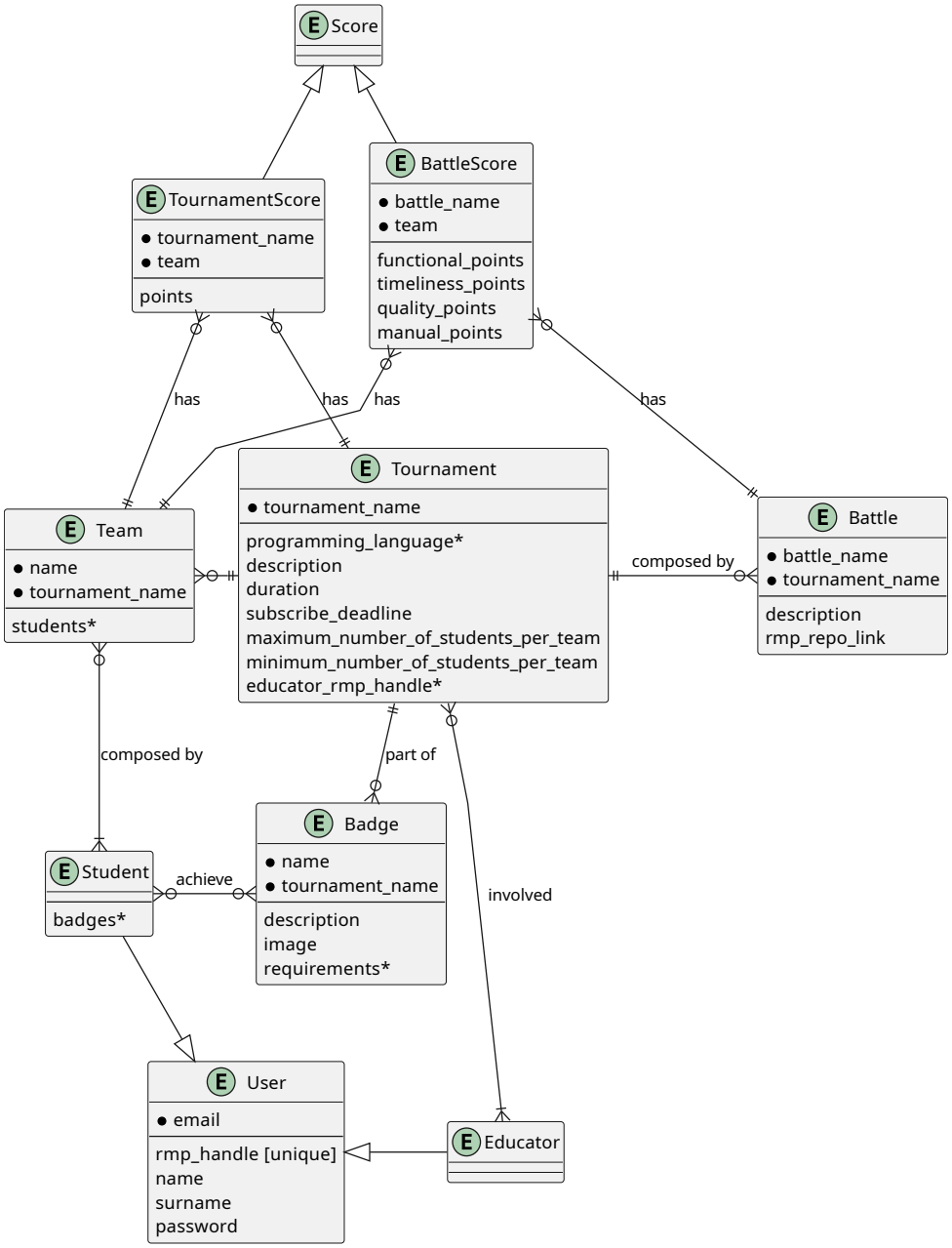

Figure 35: E-R Diagram

Legend

- **|o–** Zero or One
- **||–** Exactly One
- **}o–** Zero or Many
- **}|–** One or Many

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

2.11.2 RESTFul API

For this Platform has been thought to use a thin client which interacts with a RESTFul API. This allows for a secure system since the API Tier exposes to the client a very limited set of functionalities. Moreover, the use of a RESTFul API allows for easy future scalability, since it is possible to replicate services in different machines, and locations without changing the API, nor having problems about keeping track of session data, allowing to implement a truly free load balancer.

2.11.3 Scale-Out

With Microservices Architecture, the Platform is able to scale-out, in order to manage multiple requests at the same time. In fact, each Microservice is able to run on different machines, and the Platform is able to replicate them in order to avoid failures. And as cited before, if needed, the Platform can be entirely replicated to different locations.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

3 User Interface Design

In this section will be presented the user interface design of the application. The design will be presented both by means of diagrams and mockups. The diagrams will be presented in a top-down approach, starting from the main diagram and then going into the details of each sub diagram. After them will be shown the corresponding mockups. The scope of the diagrams is to give a schematic approach to see what pages shall show and allow users to do, while user interface mockups give a possible example on how components inside the diagrams can graphically be managed inside the webpage. Ultimately both complete each other describing user interface.
In the diagrams:

- Big titled rectangles represent singular web pages
- Smaller rounded rectangles represent UI components
- Hexagons represent action performed after an event
- Circles represent events; for example a click on a button, or the ending of an action

Colors are used to distinguish between different pages or actions, in order to render easier the understanding of the connection between them.
Since it is big and difficult to see, it has been subdivided in sub diagrams.

3.1 Complete user interface structure

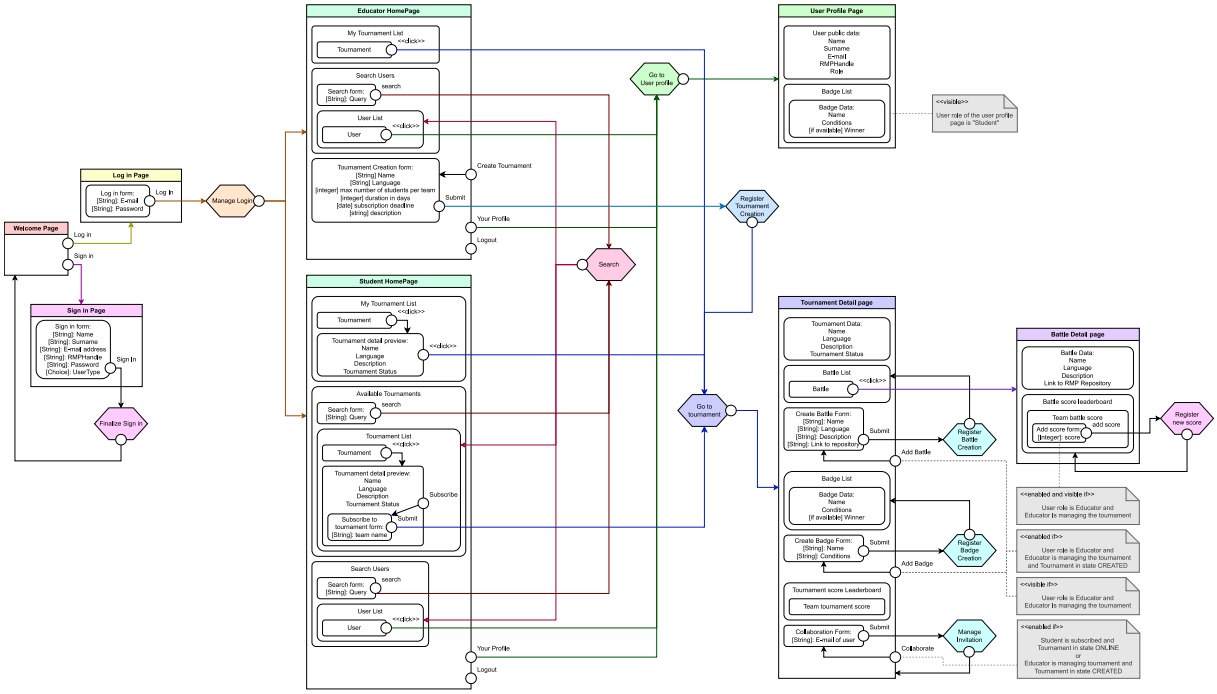

Figure 36: Main user interface structure diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

3.2 Welcome, Log in and Sign in pages

These pages are the once that a user or potential one sees when he/she first opens the application. Here he/she can choose to log in or sign in.

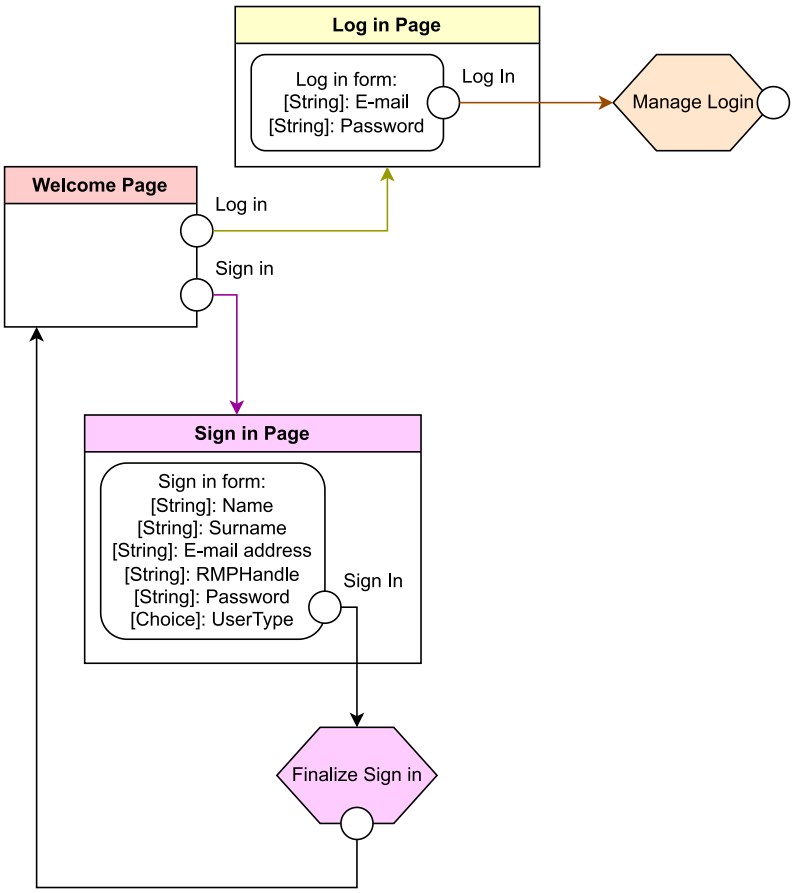

Figure 37: Welcome, Log in and Sign in pages diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Figure 38: Welcome page

Figure 39: Sign in page

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Figure 40: Log in page

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

3.3 Educator Homepage

This page is the first page an Educator sees once logged in. Here he/she can see the list of Tournaments he created, search for a user, and check his profile. Moreover he/she can create a new Tournament by clicking on the "Create Tournament" button.

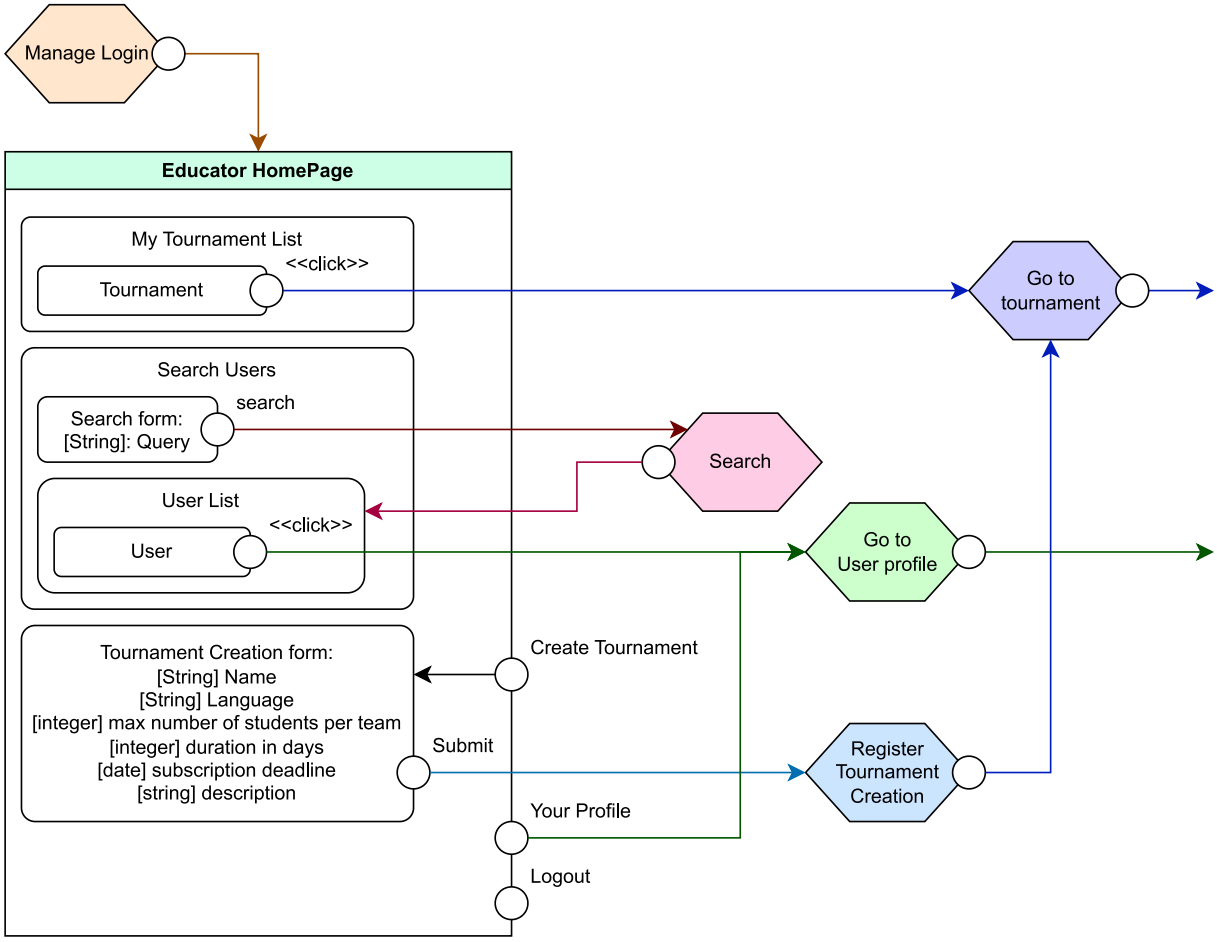

Figure 41: Educator Homepage diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

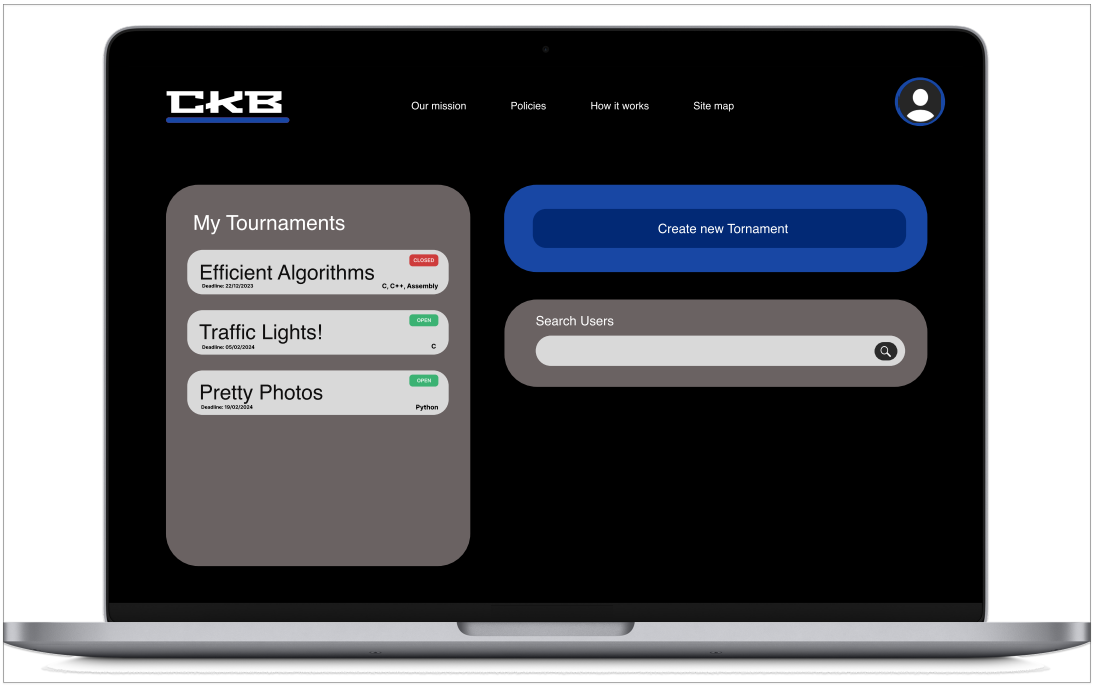

Figure 42: Educator Homepage mockup

Figure 43: Educator Homepage with create Tournament form mockup

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

3.4 Student Homepage

Similarly to the Educator Homepage, the Student Homepage is the first page a Student sees once logged in. Here he/she can see the list of Tournaments he/she is subscribed to, search for a user, and check his/her profile. Moreover, the Student can see the list of available Tournaments with a specific search form, in which it is possible to search for a Tournament, and once selected one, see details and subscribe to it.

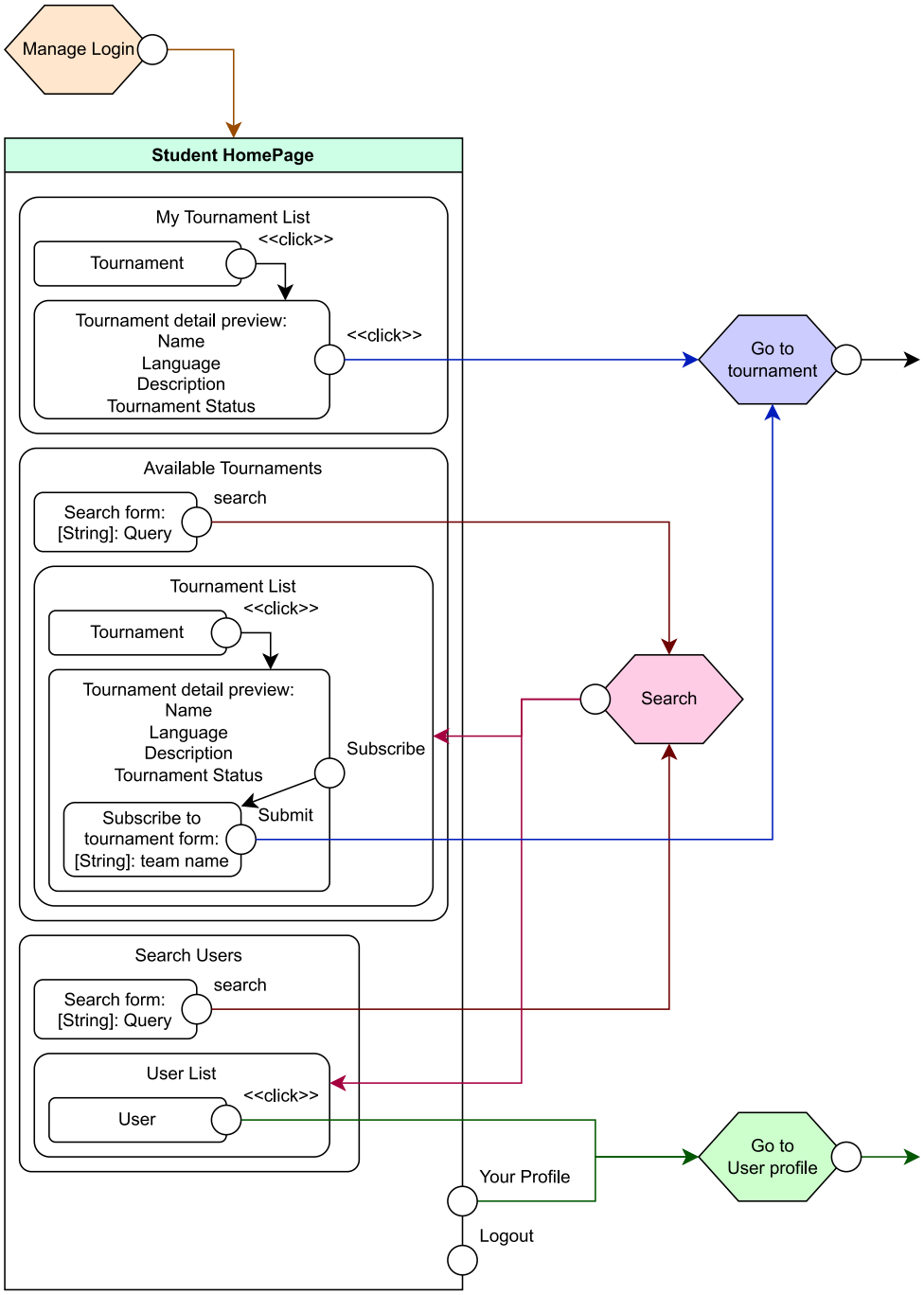

Figure 44: Student Homepage diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

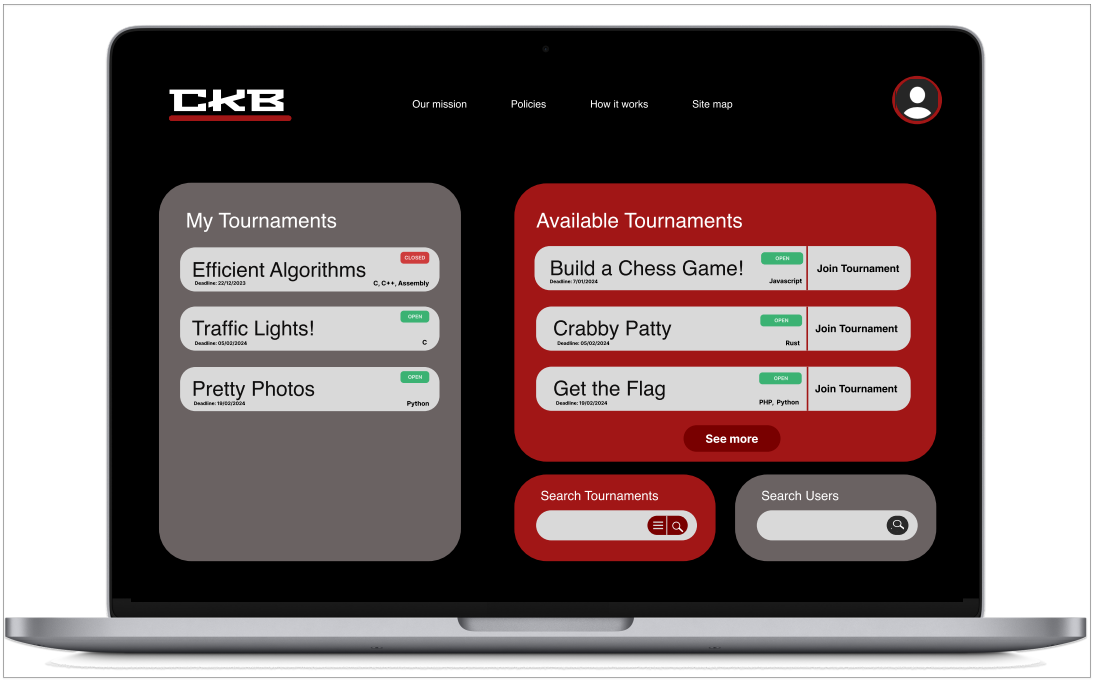

Figure 45: Student Homepage mockup

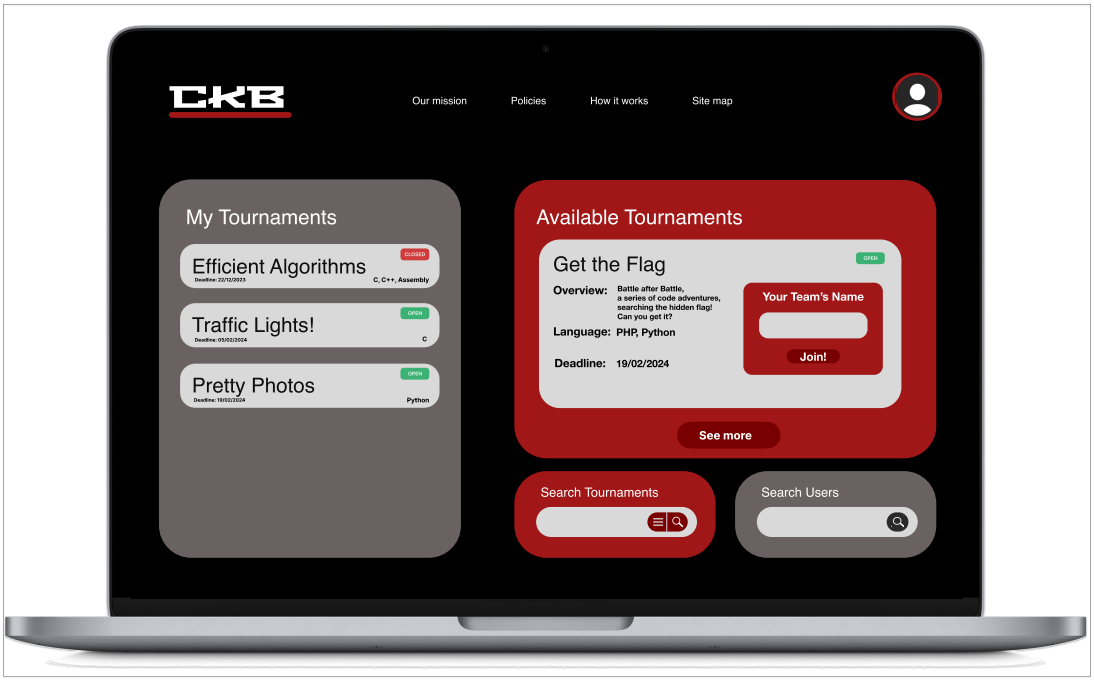

Figure 46: Student Homepage with more Tournament detail and subscription form mockup

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

3.5 Tournament Detail Page and Battle Detail Page

Tournament detail page displayes information about a Tournament. It differs significantly between Student and Educator, and the number of available actions varies according to the state of the Tournament. Also the Battle detail page is shown here, since Battles are strongly related to Tournaments. In the Battle detail page, if the user is an Educator, is also presented the form to add a score to the Battle of a specific Team.

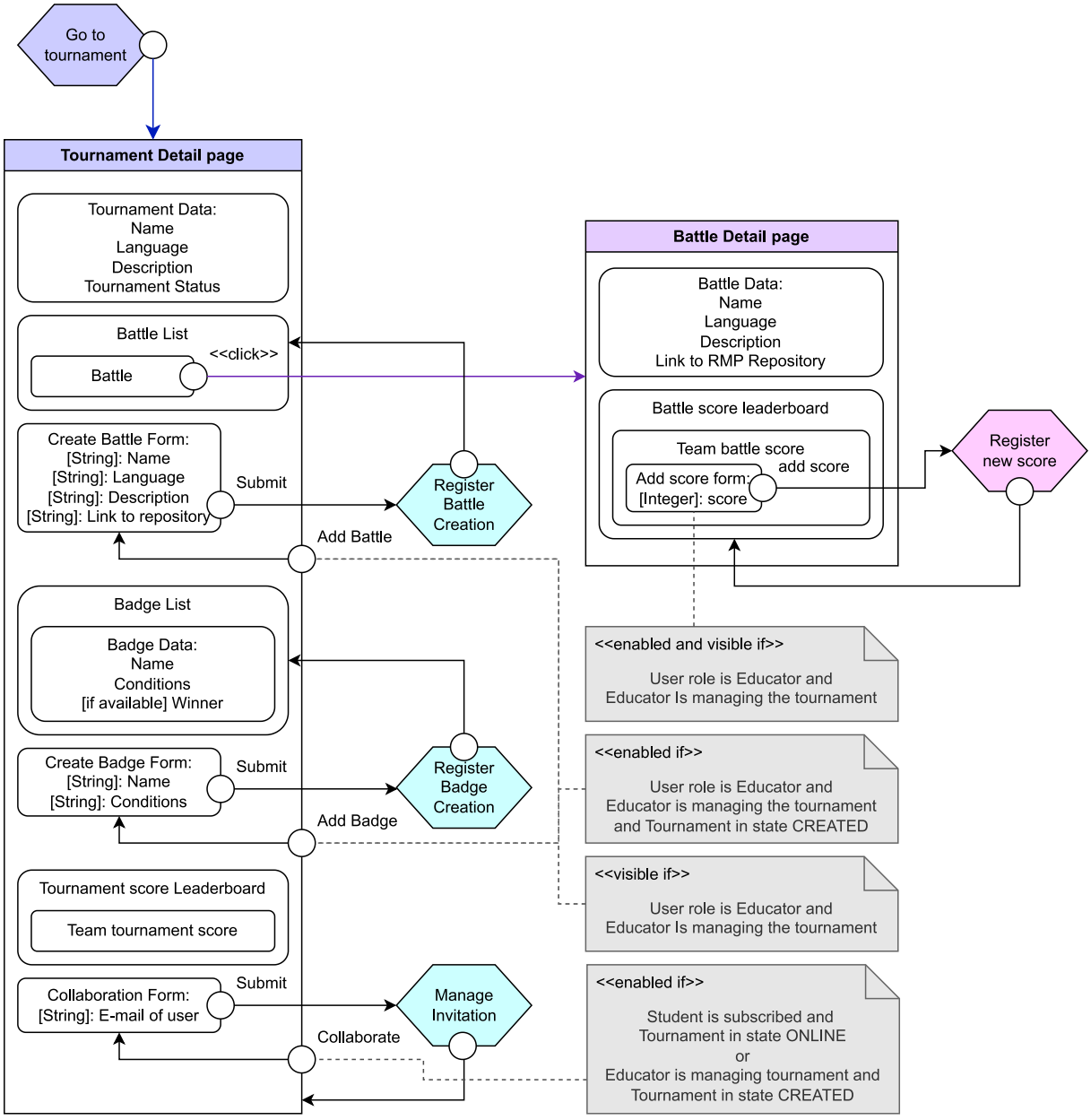

Figure 47: Tournament Detail diagram

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

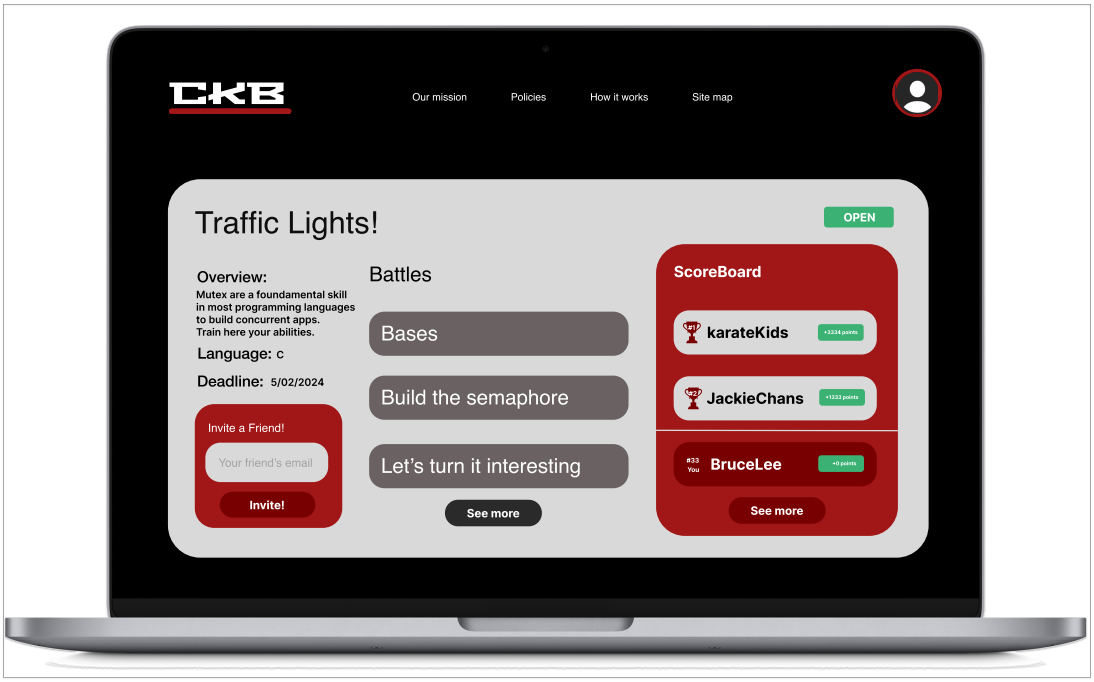

Figure 48: Tournament just started page mockup

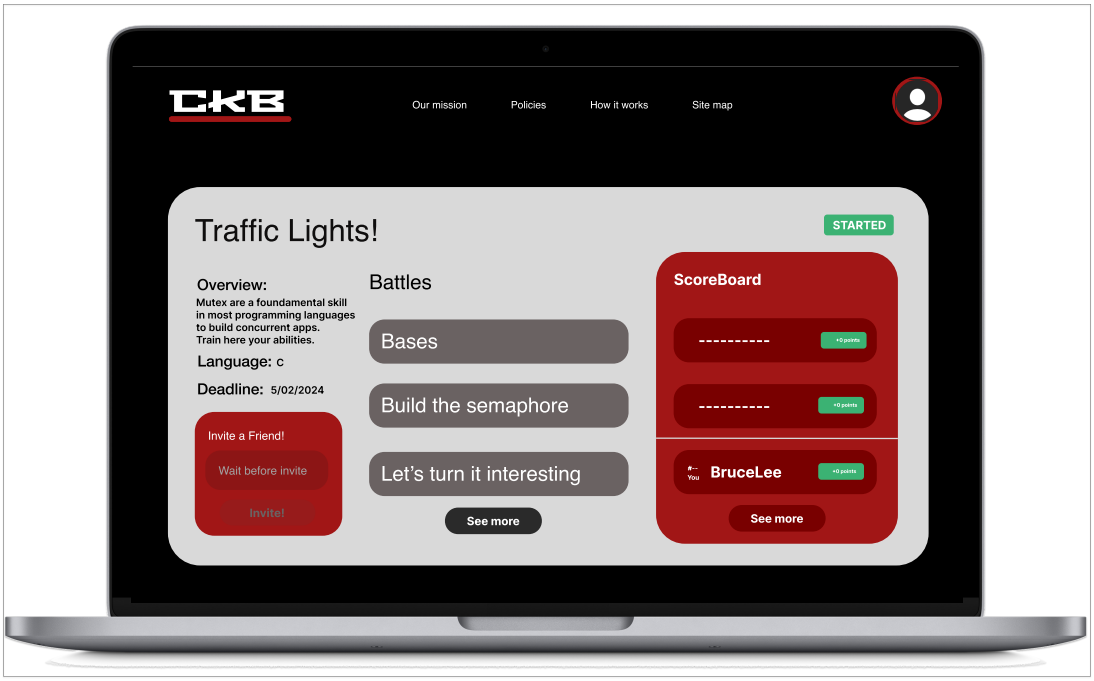

Figure 49: Standard Tournament page mockup

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Figure 50: Battle page mockup

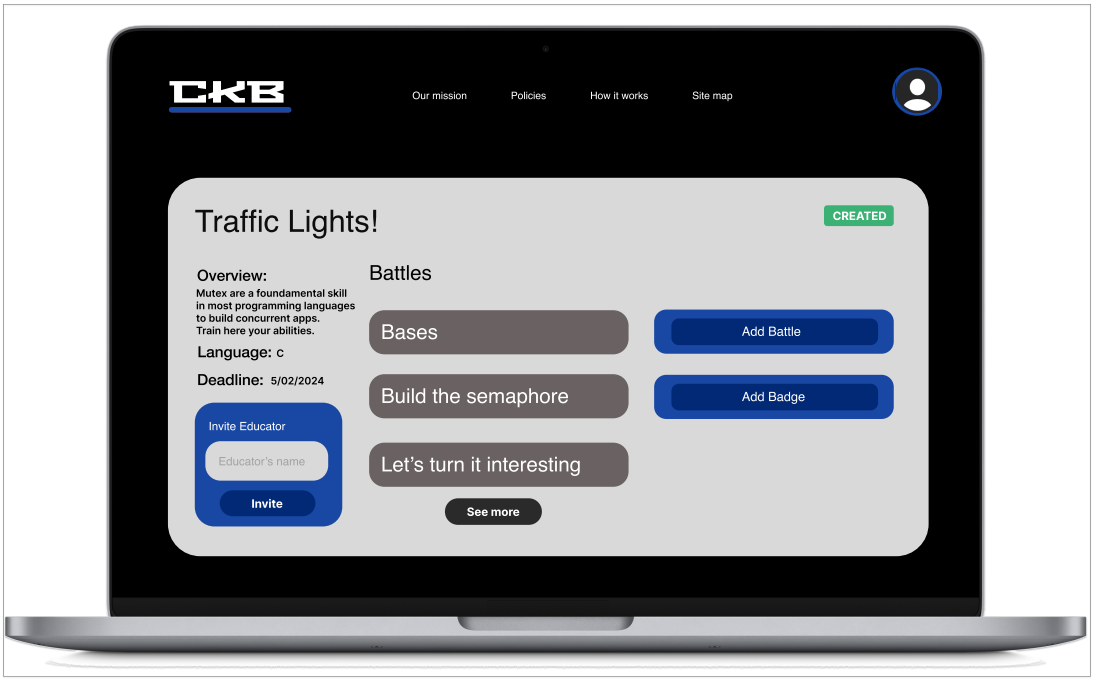

Figure 51: Educator Tournament Page mockup

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Figure 52: Educator Create Badge Page mockup

Figure 53: Educator Create Battle Page mockup

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

Figure 54: Educator Battle detail Page with score leaderboard mockup

Figure 55: Battle detail page with add score form activated mockup

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

3.6 User Profile Page

Pretty simple page where user profile data are shown and, in case of a Student, also his/her badges.

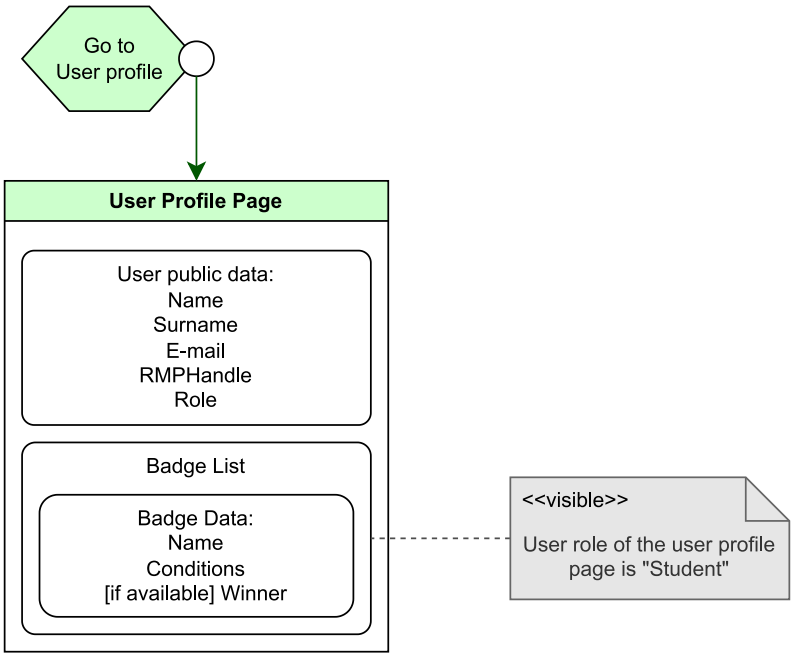

Figure 56: User Profile Page diagram

Figure 57: Profile Page within a searchUser Context

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

3.7 Multiple device UI mockups

The UI shall adapt to the screen size of the device it is being displayed on. Here are some mockups of how the UI will look on different devices.

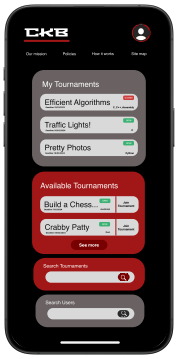

Figure 58: Smartphone UI mockup

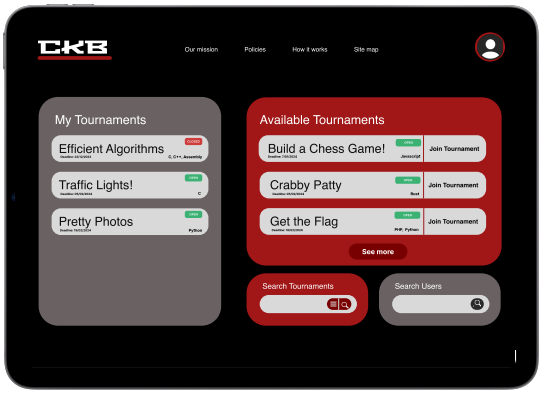

Figure 59: Tablet UI mockup

Figure 60: PC UI mockup

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

4 Requirements Traceability

In this section is presented a mapping between the two documents, the Requirement Analysis and Specification Document and the Design Document. The mapping is done between the requirements and the components that implement them.
In the following table are reported the requirements and the components that implement them.

4.1 Component - Sequence Diagram - Requirement Mapping

Component	Sequence Diagram	Requirement
Notification Manager	2.5.2, 2.5.4, 2.5.5, 2.5.6, 2.5.7, 2.5.8, 2.5.9, 2.5.11, 2.5.12	R1, R7, R8, R14
Account Manager	2.5.3, 2.5.12	R2, R17
Sign in Manager	2.5.2	R1
Log in Manager	2.5.3	R2
Search Manager	2.5.15, 2.5.16	R4, R15
Badge Manager	2.5.12	R17
Tournament Manager	2.5.4, 2.5.5, 2.5.6, 2.5.7, 2.5.8, 2.5.9, 2.5.11, 2.5.12, 2.5.13, 2.5.14	R3, R5, R6, R7, R8, R9, R10, R11, R13, R14
Evaluation Manager	2.5.12, 2.5.13	R12
Battle Manager	2.5.13, 2.5.14	R10, R12
RMP Manager	2.5.12, 2.5.13	R12, R16

Table 2: Component - Sequence Diagram - Requirement Mapping Table

4.2 Non-functional requirements' traceability

The presented design of the platform allows to satisfy the non-functional requirements pointed in the RASD. In particular:

4.2.1 Reliability and Availability and Maintainability

One of the advantges of using a microservice architecture is that the platform has an intrinsically high level of reliablilty and availability. In fact, this kind of architecture allows for replication of services: this allows that issues to one instance of them can be promptly solved by another one serving the same feature. Moreover, although has been presented as all hosted on the same server, the application subsystem, structured in this way, can be easily distributed, allowing for a higher level of reliability and availability. Finally, the use of a microservice architecture allows for a high level of maintainability, since the system is composed of many small services, that can be easily modified, updated or replaced without affecting the rest of the system.

4.2.2 Security

The Security of the system is guaranteed thanks the partitioning of the network in three subnetworks, where the once nearest to the internet are also the once that contain the less sensible data and application logic. In fact the use of the Demilitarized zone to host the Web Server, is the best solution to keep high both security and availability, since it masks the internal logic from the hostilities from the Internet.
Another possible point of attack could be the testing of user created code, since the testing environment, if badly implemented, could allow for the execution of malicious code bypassing all protection directly

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

inside the application logic. Has already been described the use of containers for testing purposes, to avoid this.

4.2.3 Portability

Since the application is developed as a Web Application, it can be accessed via a Browser, which is a kind of software that is available for almost all kinds of computers capable of some software development. Moreover, this nature of the platform, allows for easy integration with smartphone and tablet systems, which, even if not capable of software development, can be used anyway for interfacing with the platform. The use of the RMP allows for users not to be forced to use always the same computer for participating to tournaments, since they can access to their code and their standings from the internet.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

5 Implementation, Integration and Test Plan

Following the idea of the Three Layers architecture, also the implementation of the platform can be grossly divided in three main chunks. These parts interact with each other thanks to the use of interfaces, this allows to implement them independently and simultaneously. So the first step is to implement rigorously their definition, specifying how messages are composed and how they are exchanged between the components.

5.1 Data Layer

The Data Layer comprises database implementation and the implementation of the interfaces that allow the Application Layer to interact with it.
Each component of this layer has to be tested independently, then once they are all implemented, can be integrated and tested together, using also Drivers that emulate the behavior of the Application Logic Layer.

5.2 Application Logic Layer

Given the chosen architecture, this Layer needs a more detailed plan. Among Microservices can be identified some dependencies shown in the following graph:

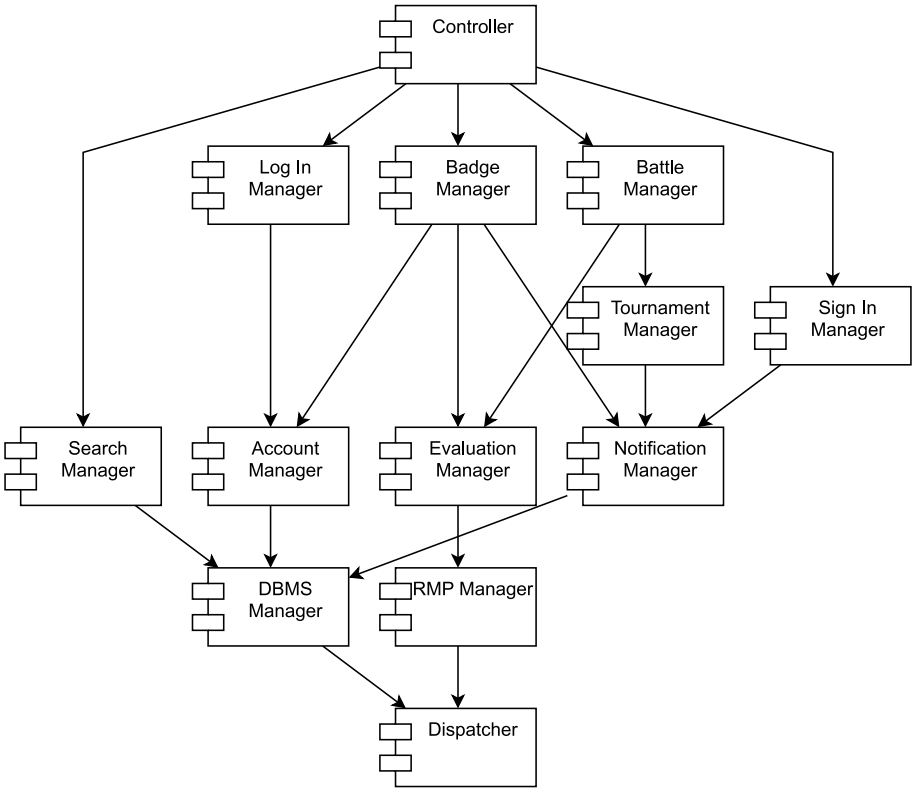

Figure 61: Application Logic Layer Dependency Graph

Given the internal structure of this Layer, all Microservices depend on the Dispatcher. This is a Key component, since each communcation between Microservices is mediated by it, so it has to be implemented first.
Then can be implemented the DBMSManager and the RMPManger, these are the second most important

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

components since they are used by a variety of others. Then starting from them the dependency graph can be followed from bottom to the top, until reaching the Controller Component.
Each component has to be subjected to unit testing, then can be integrated first with the Dispatcher and then with other components. Unit testing can be done by means of necessary Stubs and Drivers, needed to mimic interactions with other components.
After Unit Testing, Integration testing can be done on the layer as a whole, checking that interactions follow the desired behavior without generating any side effects.
Given the fact that the Dispatcher is the component that does load balancing, components have to be stressed for proving that the Dispatcher is able to allocate more resources, and the dispatcher itself has to be tested with a high amount of similar requests to check if it manages them equally among resources.

5.3 External Logic Layer

The External Logic Layer comprises the implementation of the Web Server, The API, the Web App that runs on client's browser, the RMP action and the confirmation links; so mainly all the logic needed to interact with the Platform.
This part is formed by fewer components than other Layers, but the complexity of each one is higher. Here follows the dependency graph of the components:

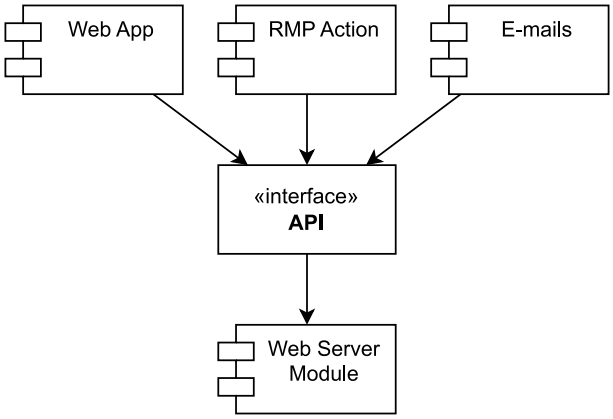

Figure 62: External logic Layer Dependency Graph

Can be noticed how the Web Server component, having the job of connecting the Application Logic to Presentation Level is the most critical Layer, and has to be implemented first. In fact it contains all logic needed to manage the API calls and forward them to the Application Logic.
Then after implementing the API, the other components are totally separated between each other and can be realized independently and in parallel.
Given the Nature of this layer, Each component needs to be tested via Unit testing and integration testing, but, most importantly, has to be tested for possible security flaws, since one of the jobs of this layer is to provide basic security to the platform.

5.4 Integration of the Layers

Thanks the use of the necessary Drivers and Stubs, each Layer, once implemented, can be tested independently. Then, once all Layers are implemented and tested, the system can be integrated and tested as a whole, via means of test accounts, which job is to firstly test if functionalities work, secondly to test edge cases and possible issues that can arise during production.

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

6 Effort Spent

Name	Section 1	Section 2	Section 3	Section 4	Section 5
Angelo Attivissimo	2 h	30 h	0 h	1 h	1 h
Isaia Belardinelli	1 h	30 h	20 h	0 h	1 h
Carlo Chiodaroli	0 h	30 h	10 h	2 h	4 h

Table 3: Effort Spent

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved

7 References

[1] Specification Document: RASD and DD Assignment A.Y. 2023-24

[2] CodeKataBattle RASD

[3] UML official specification: https://www.omg.org/spec/UML/

Copyright © 2024, Angelo Attivissimo, Isaia Belardinelli, Carlo Chiodaroli – All rights reserved