2022

Design Document

INDIVIDUAL PROJECT

MITOV,LACHEZAR

Contents

Introduction

The Design document focuses on the technical structure of the individual project. It will give insights into what methodology will be followed, what frameworks will be used and how the quality will be assured.

Project Methodology

This project will follow the Agile methodology. This is due to the Agile method's emphasis on using practical and efficient approaches to generate high-quality products.

Frameworks

Frameworks that will be used on this project:

- **Java Spring Boot** for the back end of the application
- **React** for the front end of the application
- **MySQL** database
- **Spring Data JPA** for working with the database
- **JWT** technology for the authentication and authorization

Why do I use these technologies?

Java Spring Boot – This framework helps with removing a lot of boiler plate code, which makes the solution comprehensive and understandable. Moreover, Spring Boot makes implementing Dependency Injection quick and easy. Finally, this framework is required to be used for this project.

React – React is a JavaScript library, which allows developers to create websites and application. The JSX syntax that React implements is revolutionary and is very handy when writing the components of the application. This brings me to another positive, which is that components can be reused and thus the application consistency much better. Moreover, this makes further support and optimization a piece of cake. React is used by many, which makes it very easy to find materials and documentation online.

MySQL – I have worked with MySQL databases since I started learning at Fontys and therefore I chose to work with it again. I have a good understanding of this system and it gives me confidence.

Spring Data JPA – Even though I don't mind writing queries and creating databases myself using ORM's like the JPA helps automate and make the process faster. Moreover, ORM's are widely used in the industry, which makes it more appealing to use it now, so that I can get used to working with it.

JSON Web Tokens- Security is very important in the Internet world. This can be achieved through authentication and authorization. Very common way of implementing these functionalities is by using JSON Web Tokens. I chose JWT technology because it offers great security, it is very compact and can incorporated with Restful Services and it is easy to process.

Quality Assurance

The quality of the project will be assured through:

- **Unit test** through Junit
- **Integration tests** through Junit
- **Acceptance tests**
- **Front-end testing** through Cypress
- **SOLID** principles

For more information about the testing of the project, refer to this document : Test Plan

How is SOLID applied

Single-Responsibility principle

Firstly, the Single-responsibility principle says that every class created should have only one responsibility. This is implemented in the project by having services, controllers and repositories containing methods only about their respectable entities. For example, UserService and UserController classes only contain methods about the User Entity. Additional configuration classes have only one responsibility. For example, the WebSecurityConfig class is only responsible for configuring the security of the application.

Open-Closed Principle

“Open for extension and closed for modification” -this is what this principle follows. In the application classes are made so that it is possible to add fields to the data structures it contains, or new elements to the set of functions it performs without the need to change other classes or modules.

Liskov Substitution Principle

The principle states that an object (such as a class) and a sub-object (such as a class that extends the first class) must be interchangeable without breaking the program. At present there is no inheritance used in the project.

Interface Segregation Principle

Clients shouldn't rely on interfaces they don't utilize, according to the principle of Interface Segregation. This is achieved in the project by properly splitting interfaces and making sure there are no unnecessary methods that belong to them.

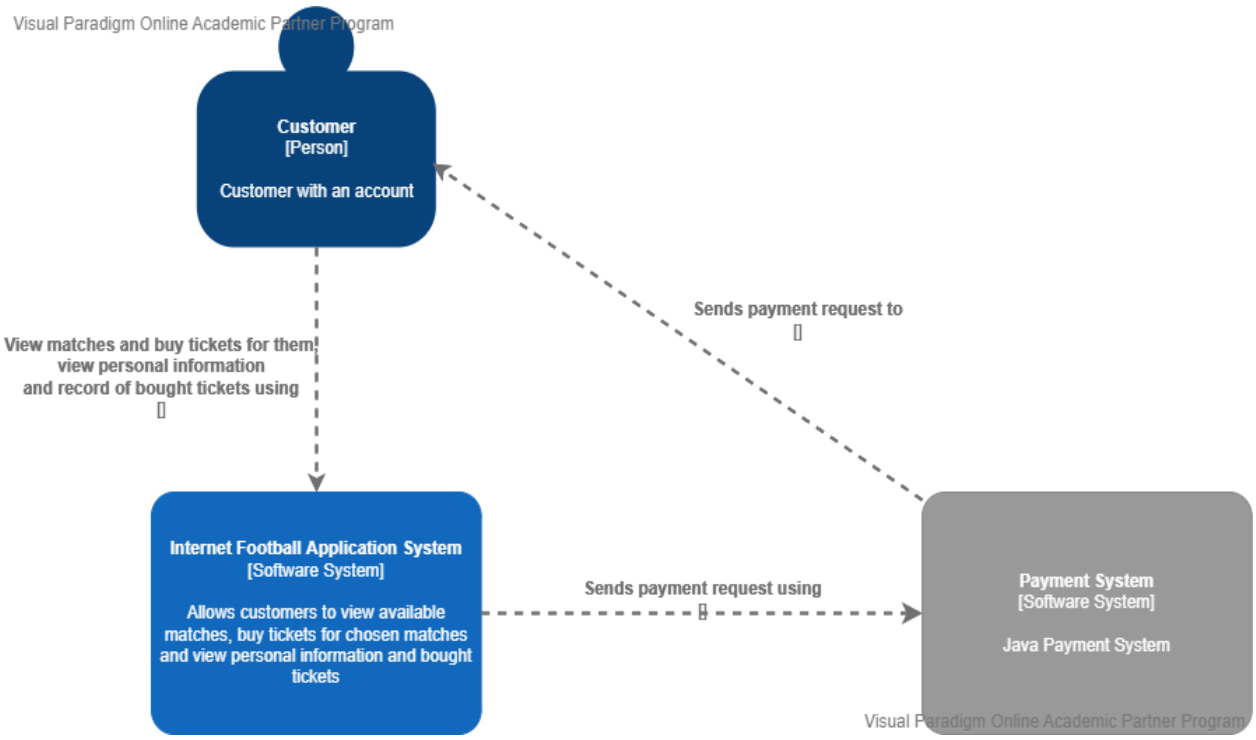
Dependency Inversion Principle

The Dependency Inversion principle states that classes should depend on abstraction and not concretions. And this principle is implemented in the solution by implementing interfaces. That way the classes use abstractions, which creates a barrier preventing coupling to dependencies.

C4 Architecture

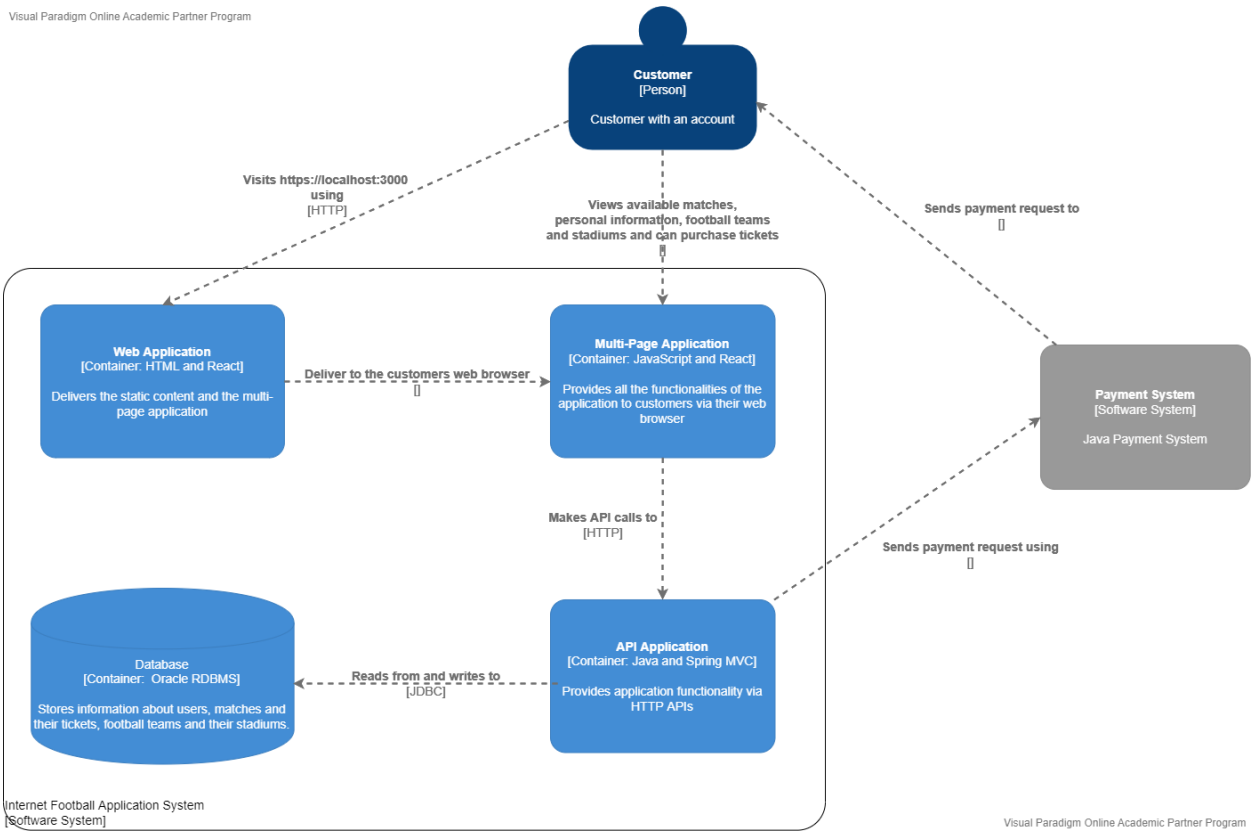
C1 Diagram

The System Context or so-called C1 diagram provides a starting point, showing how the software system in scope fits into the world around it



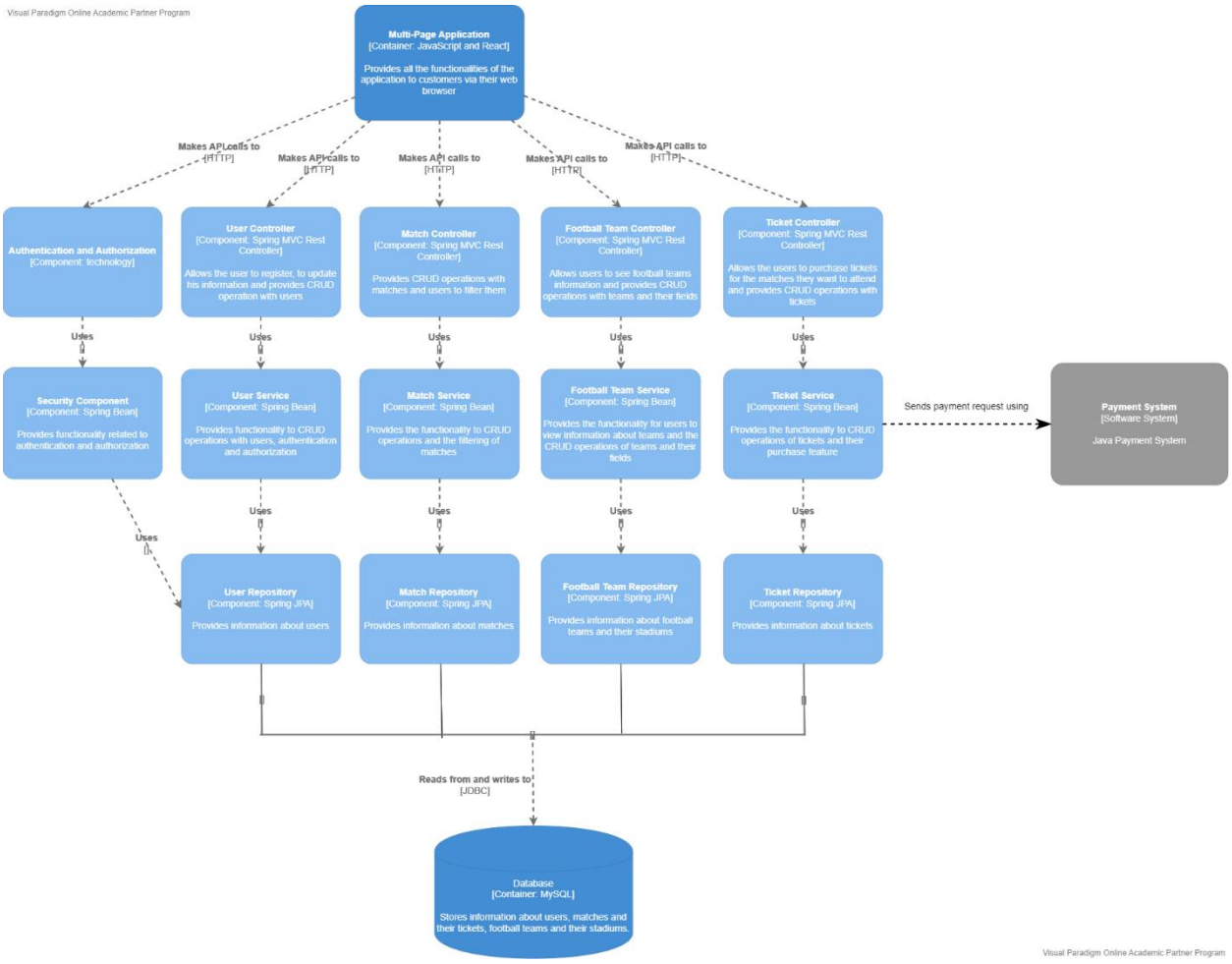
C2 Diagram

The Container or otherwise known as C2 diagram zooms into the software system in scope, showing the high-level technical building blocks



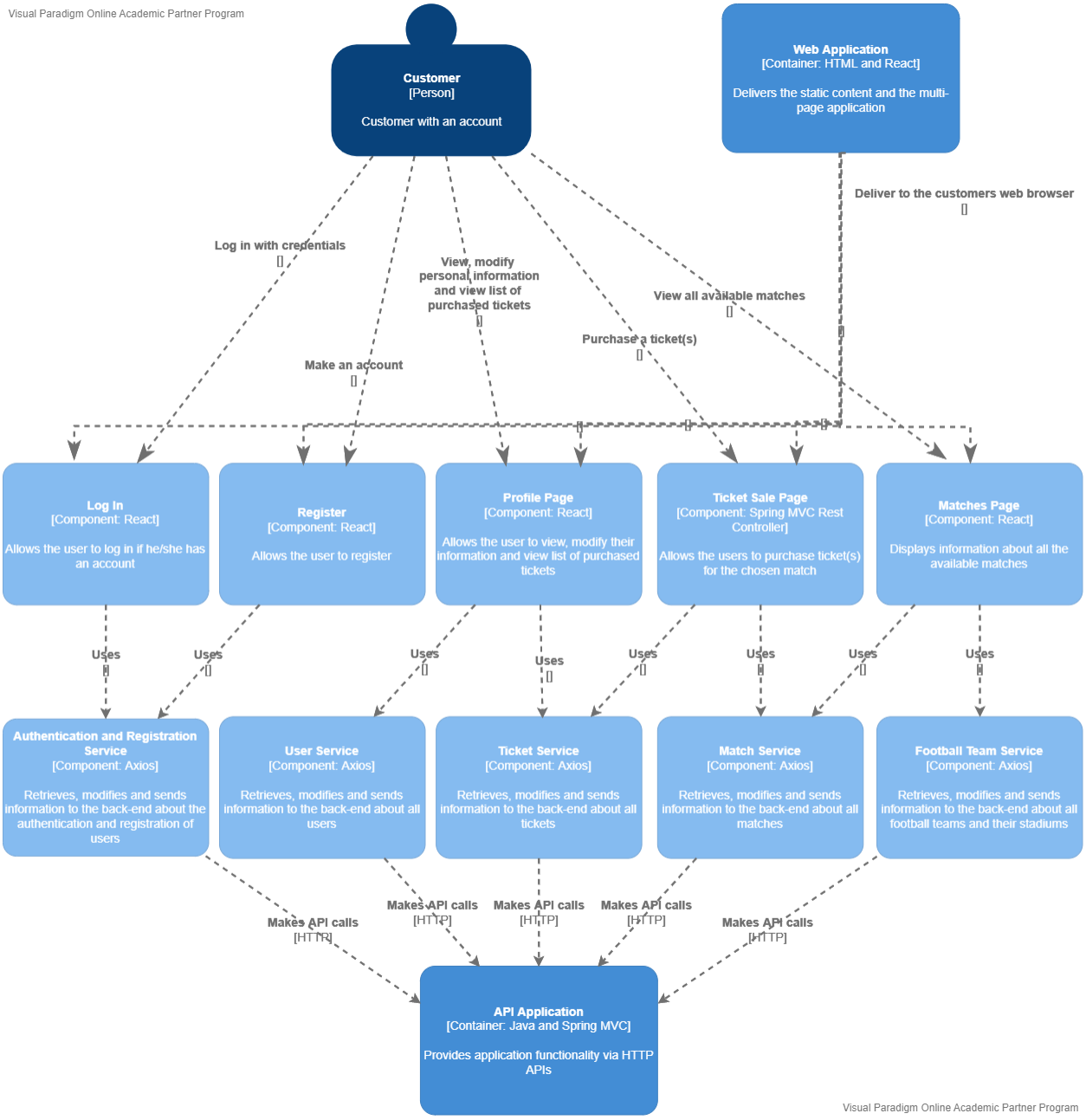
C3 API Application

A Component or C3 diagram zooms into an individual container, showing the components inside it. This diagram shows how the components of the API Application work.



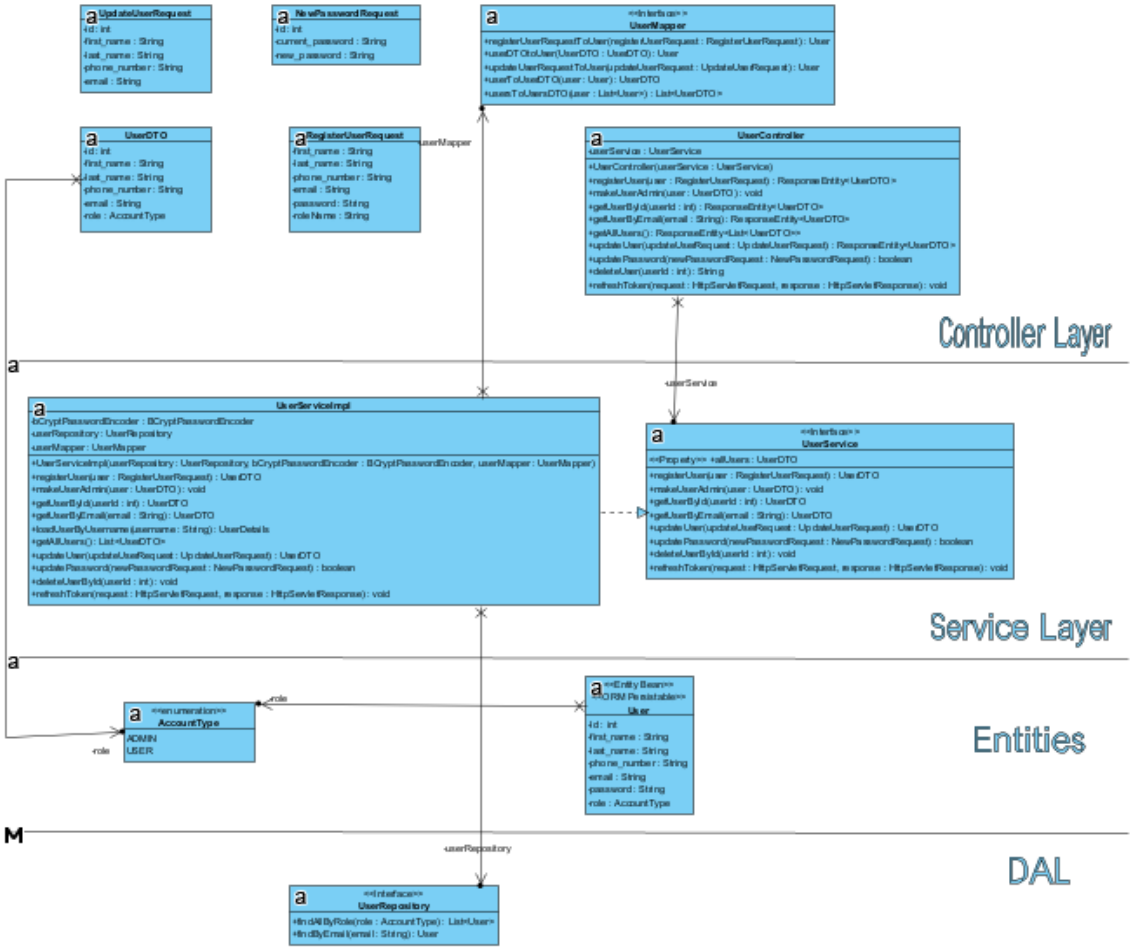
C3 Multi-Page Application

Just like above this is another C3 diagram, which however displays how the components of the Multi-Page Application work.

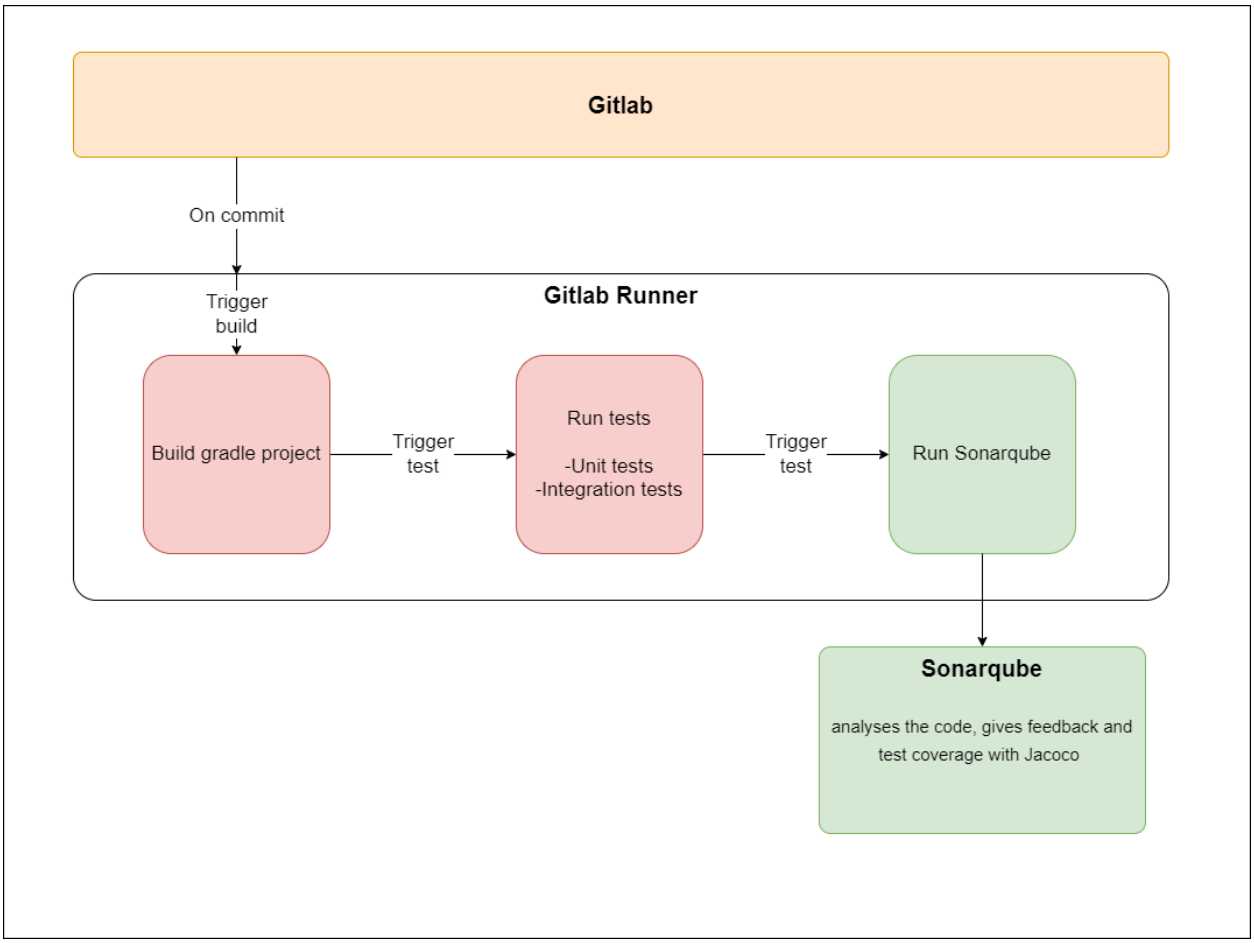


C4 Class Diagram

The diagram displayed below presents a class diagram for the flow of registering a user to the system. When a user wants to register a RegisterUserRequest is sent by the UserController and received by the UserService. Afterwards, the service creates a user account and sends a UserResponse to the Controller.



CI/CD Diagram



Design Decisions

URL Design

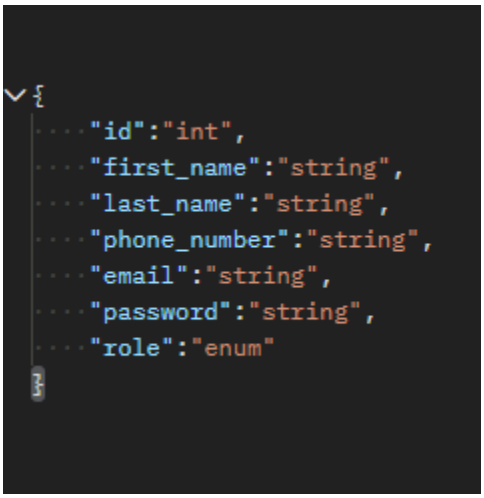
Below in the table are listed the URLs that will be used to service the software solution.

URL	Resource	Operation	
/matches/	Matches	GET	Read a list with all available matches
/matches/123	Matches	GET	Read the match with ID 123
/matches/456	Matches	DELETE	Delete the match with ID 456
/matches/789	Matches	PUT	Update match with ID 789
/matches/	Matches	POST	Add a new match
/tickets/	Tickets	GET	Read all tickets in the system
/tickets/123	Tickets	GET	Read the ticket with ID 123
/tickets/	Tickets	POST	Add a new ticket
/users/	Users	GET	Real all users from the system
/users/123	Users	GET	Read the user with ID 123
/users/456	Users	DELETE	Delete the user with ID 456
/users/789	Users	PUT	Update user with ID 789
/users/	Users	POST	Create new user

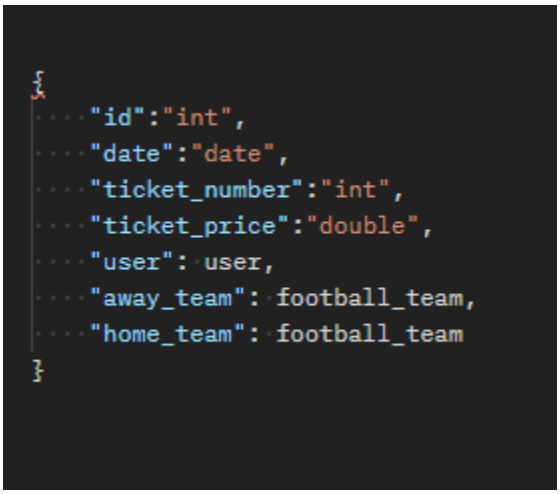
Entity Design

The project consists of 5 entities (User, Match, Ticket, Football Team, and Stadium). Below each of them are explained.

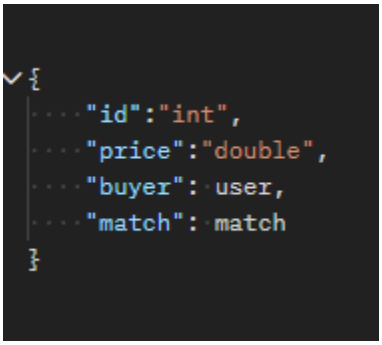
User



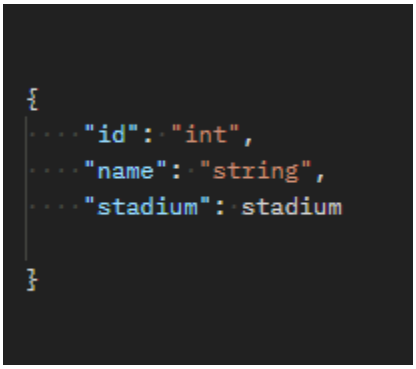
Match



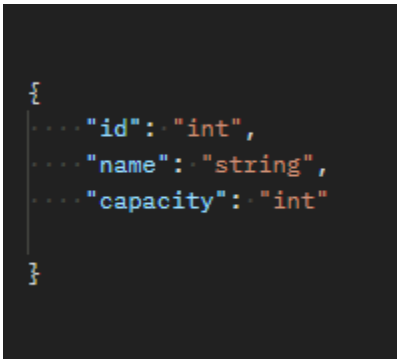
Ticket



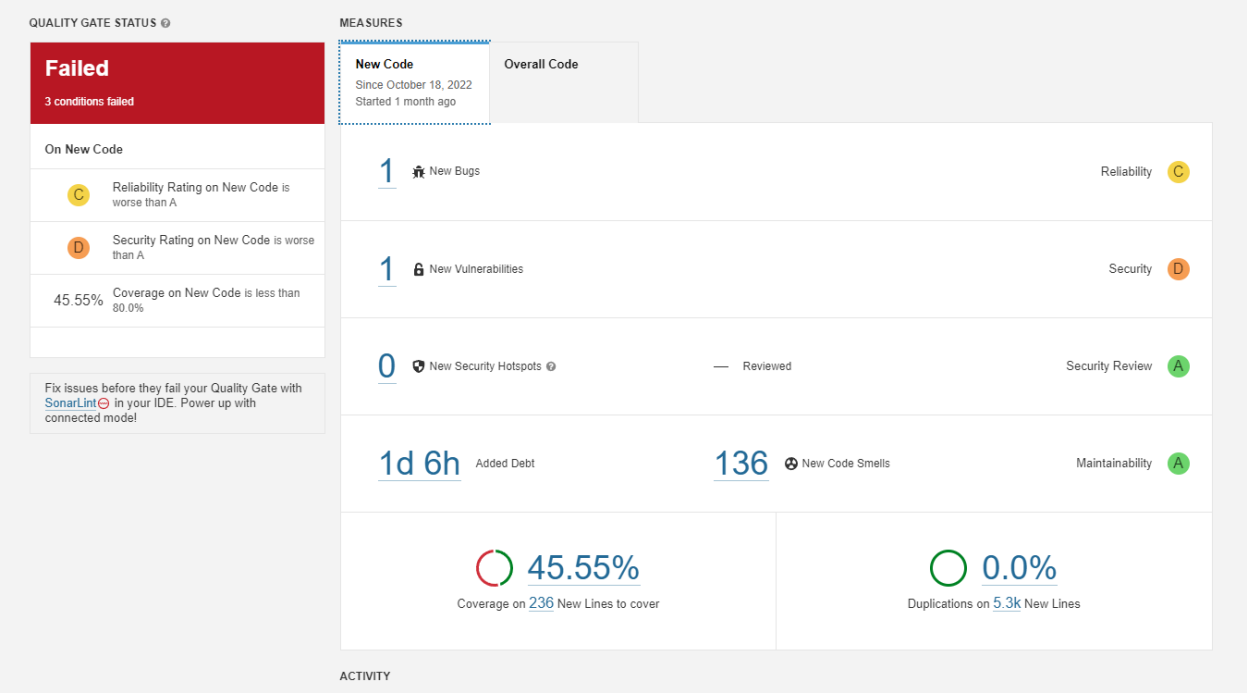
Football Team



Stadium



Sonarqube


QUALITY GATE STATUS
Failed
3 conditions failed
On New Code
C
Reliability Rating on New Code is worse than A
D
Security Rating on New Code is worse than A
45.55%
Coverage on New Code is less than 80.0%
Fix issues before they fail your Quality Gate with SonarLint in your IDE. Power up with connected mode!
MEASURES
New Code
Since October 18, 2022
Started 1 month ago
Overall Code
1
New Bugs
Reliability
C
1
New Vulnerabilities
Security
D
0
New Security Hotspots
Reviewed
Security Review
A
1d 6h
Added Debt
136
New Code Smells
Maintainability
A
45.55%
Coverage on 236 New Lines to cover
0.0%
Duplications on 5.3k New Lines
ACTIVITY