MMA TourSafe
Analysis and Design Document

Student: Halmai Erik
Group: 30432

Table of Contents

1. Requirements Analysis 3

  1.1 Assignment Specification 3
  1.2 Functional Requirements 3
  1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 4

4. UML Sequence Diagrams 6

5. Class Design 6

6. Data Model 7

7. System Testing 8

8. Bibliography 9

1. Requirements Analysis

1.1 Assignment Specification

Using a OOP language, design and implement an application that helps MMA tournament managers organize scheduled fights while ensuring COVID safety standards throughout the tournament.

1.2 Functional Requirements

The manager can create tournaments and invite fighters to sign up. Each tournament requires a match scheduling strategy of weekly or monthly matches. To ensure that the COVID safety standards are met, after fighters sign up, they are tested. If the test is positive, they are moved to quarantine, else they are moved to the tournament bubble. The application will populate the schedule of a tournament with matches between fighters from the tournament bubble.

1.3 Non-functional Requirements

The information about tournaments, fighters, matches and COVID tests will be stored to a database, using MariaDB. The Client-Server Architectural Pattern will be used to organize the application and separate its concerns. The Client side will be implemented using Angular, while for the Server side, Spring Boot will be used. The website should offer a seamless experience, without noticeable load times. It should be easily extensible and be available on any web browser.

2. Use-Case Model

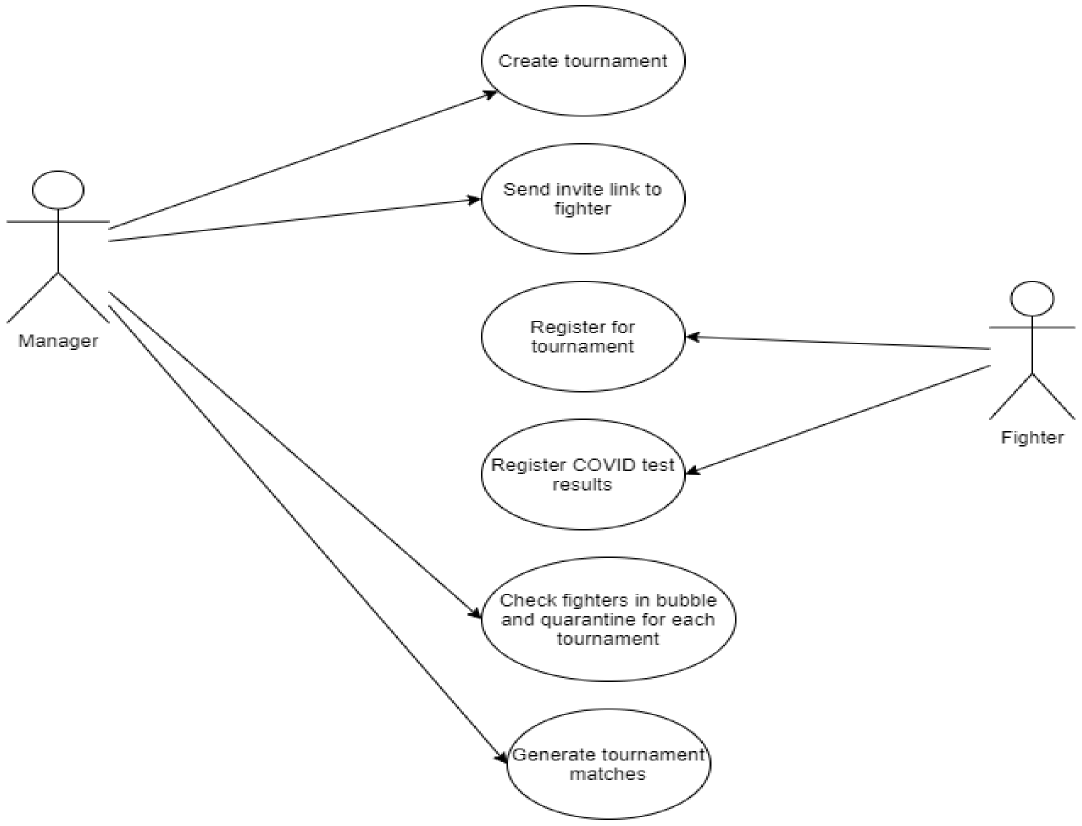
Create tournament
Send invite link to fighter
Register for tournament
Register COVID test results
Check fighters in bubble and quarantine for each tournament
Generate tournament matches
Manager
Fighter

Figure 1 Use case

Use case: Register for tournament
Level: user-goal level
Primary actor: Fighter
Main success scenario: The fighter will be saved to the database for the specific tournament. He/she will need to provide the result for the COVID test.
Extensions: The submit button will be disable until all the fields are completed.

3. System Architectural Design

3.1 Architectural Pattern Description

The application uses the Client-Server architectural pattern. This pattern allows the separation of the business logic from the presentation layer into two cooperating programs. The server component provides the services for one or many clients, which initiate requests for such services.
The Server side of the application is implemented using Spring Boot and some modules that it offers: Lombok, Spring Data JPA along with Hibernate, Spring Web and ModelMapper. For the database MariaDB is used which offer fast queries. The Server side also obeys the Layered architecture pattern by separating the layers into different packages.
The Client side is implemented using Angular and can be deployed with Angular CLI's **ng serve** command. This way, by using the Client-Server architecture, the final project consists of two separately deployable programs. To communicate between the two programs Rest API is used. This way we define some endpoints in the Server side's Controller objects and send the appropriate HTTP requests to those addresses.

3.2 Diagrams

The Layered Architectural Pattern is applied during the package structuring of the program. The application will have 4 packages representing the layers described above.

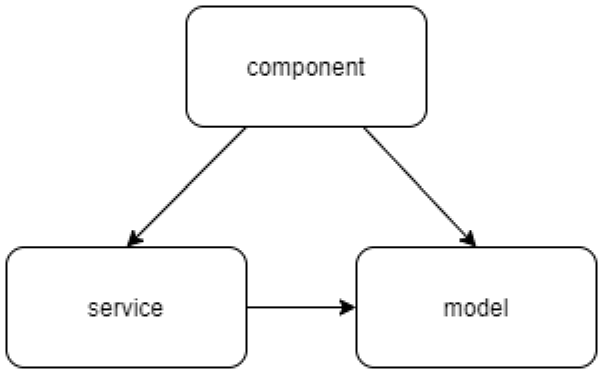

Figure 2 Package diagram
Client side

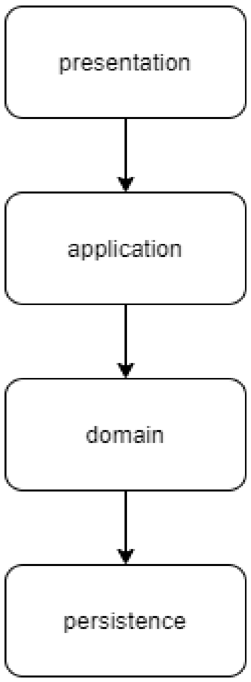

Figure 3 Package diagram
Server side

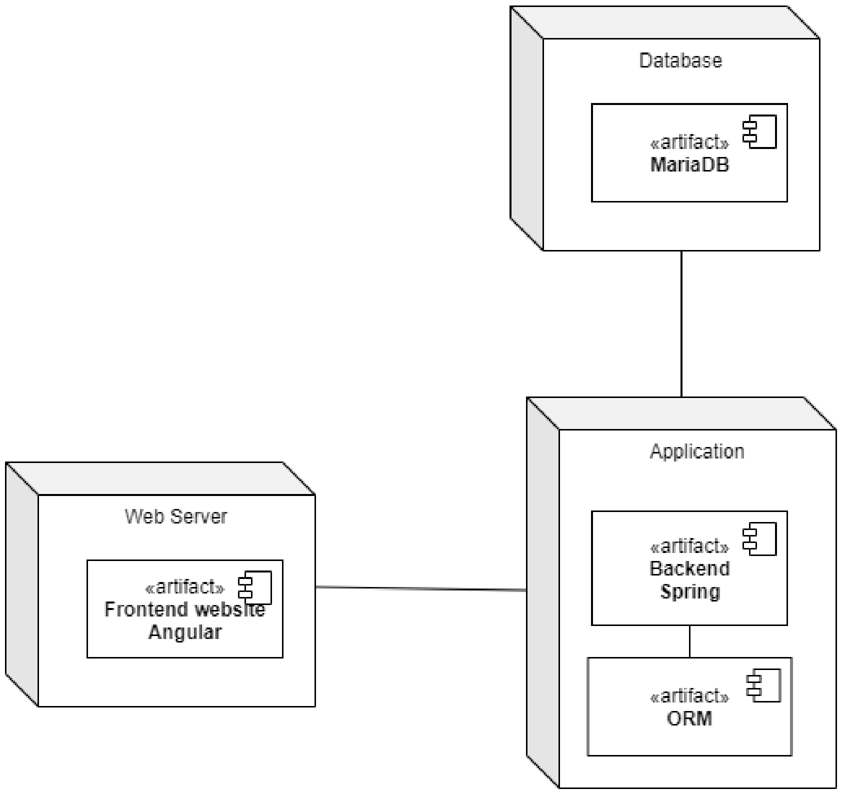

Figure 4 Deployment diagram

4. UML Sequence Diagrams

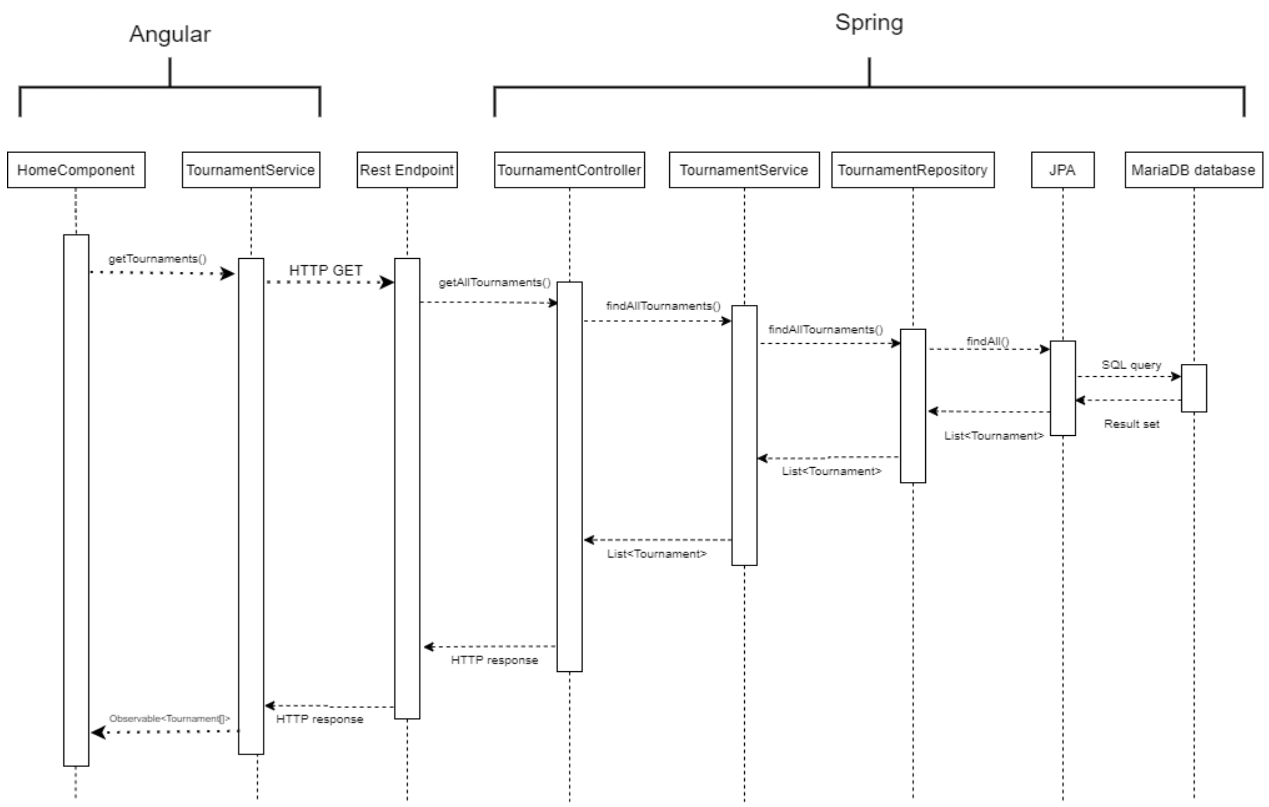

Figure 5 Fetch all tournaments sequence diagram

5. Class Design

5.1 Design Patterns Description

The application uses the Strategy design pattern in the context of match scheduling of a tournament. A tournament can have a weekly or a monthly match scheduling strategy that is selected during run time when the manager creates a new tournament. The Strategy pattern allows to change the behavior of scheduling matches during run time and it is perfect for this situation.

By creating a ScheduleStrategy interface, having one method, scheduleMatch() and two @Component classes, MonthlySchedule and WeeklySchedule that implement this interface, we can have one field inside the Tournament entity of type ScheduleStrategy. To use the Strategy pattern, I created a Map<String, ScheduleStrategy> field in the MatchService and added names to the @Component classes. Spring automatically wires these names to the corresponding component.

Whenever we want to add a new match, we first get the ScheduleStrategy corresponding to the MatchDTO object's tournament and create the appropriate MatchDTO object and set its date by obeying the strategy.

5.2 UML Class Diagram

The following picture presents the class diagram. It is presented for an entity, the diagram being the same for all the others.

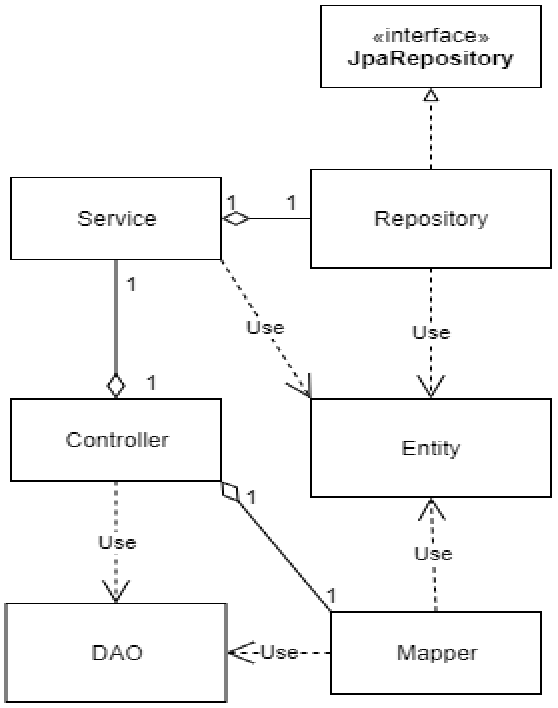

Figure 6 Class diagram – Server side

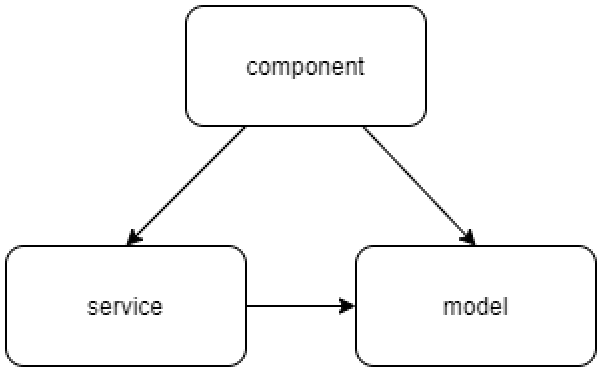

Figure 7 Class diagram - Client side

6. Data Model

The application consists of four entities: Tournament, Fighter, Match, and CovidTest, representing the database's tables. The relationships between the entities are presented below:

- The Tournament entity has OneToMany relationships with the Fighter and the Match entities.
- The Fighter entity has a OneToMany relationship with the CovidTest entity
- The Match entity has two ManyToOne relationships with the Fighter entity.

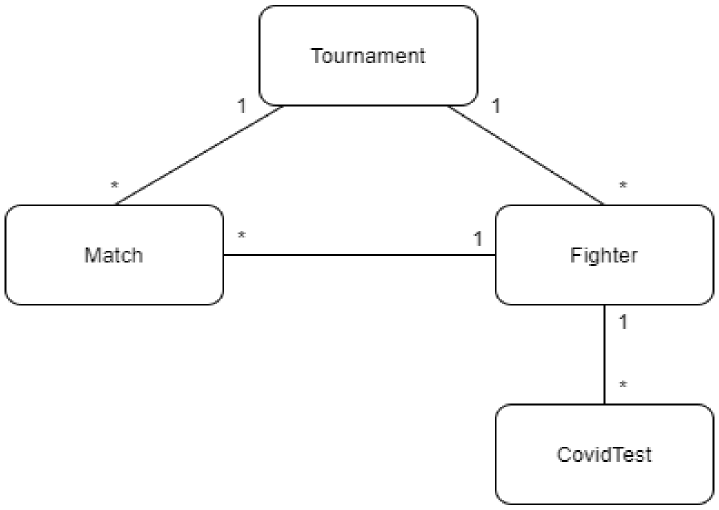

Figure 8 Data model

The application uses DTO objects in the presentation layer. These are copies of the entity objects that are used to communicate the objects between the frontend and backend. The frontend application, also has these DTO objects.

7. System Testing

To the test the application I used Postman. Postman lets you create and save simple and complex HTTP request and read their responses. This way, after implementing a feature in the backend, I could test it by simply sending the appropriate GET or POST request to that end point. In such a way, I was able to test the addition and getting of elements from the database.
I also used IntelliJ's built in Database editor, to check the data inside of the tables. I also used Hibernate's feature to show the SQL queries it performs and checked those as well.

Presentation: https://www.youtube.com/watch?v=TWg36Qc0sGc

8. Bibliography

https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm

https://en.wikipedia.org/wiki/Client%E2%80%93server_model

https://spring.io/projects/spring-boot

https://cli.angular.io/

https://getbootstrap.com/docs/5.0/getting-started/introduction/