MMA TourSafe
Analysis and Design Document

Student: Halmai Erik
Group: 30432

Table of Contents

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 should have at least bi-weekly matches to generate traction and revenue within the tournament period.
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 Layered Architectural Pattern will be used to organize the application and separate its concerns. An ORM and a DI Container will be used, using Spring Boot. 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

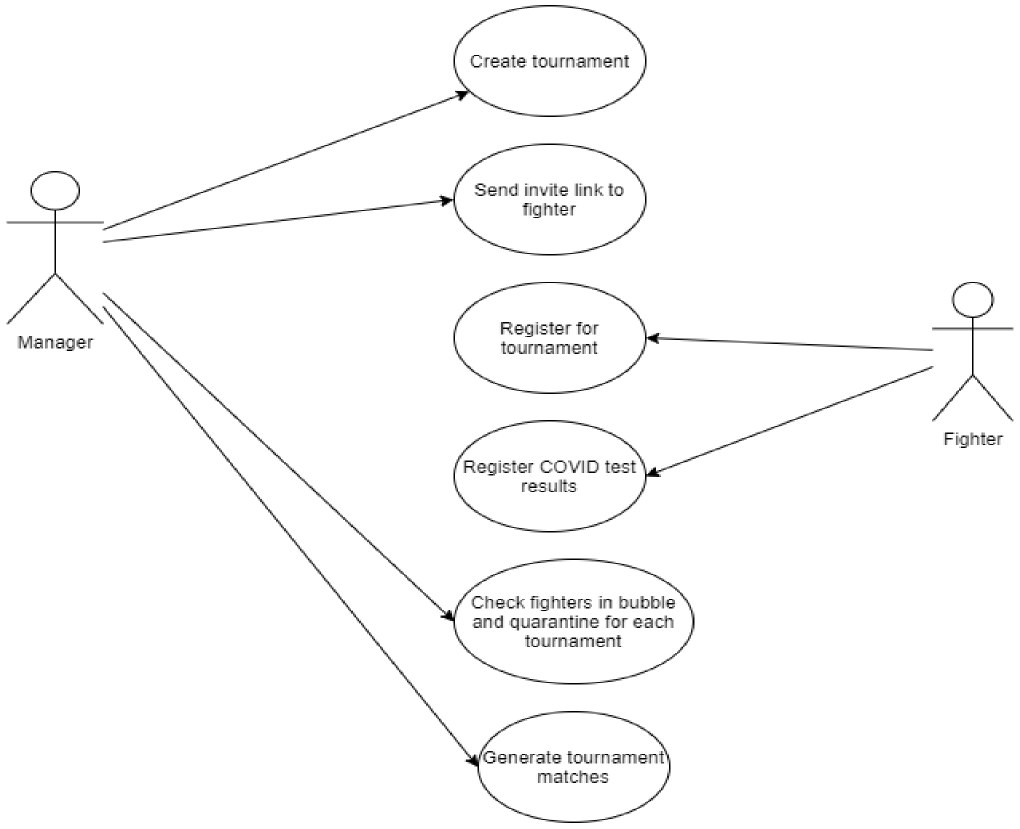

Create tournament
Send invite link to fighter
Register for tournament
Register COVID test results
Check fighters in bubble and quarantine for each tournament
Generate tournament matches
Manager
Fighter

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 Layered Architectural Pattern, which helps organize the project's structure into four main categories: presentation, application, domain, and persistence. Each of the layers contains objects related to the particular concern is represents.

- Presentation layer: contains the controllers that get the requests from the front end and the Data Transfer Objects that represent the entities in the outer world.
- Application layer: contains the services and components that represent the logic that is required by the application to meet its functional requirements.
- Domain layer: contains domain entities that represent the database's tables.
- Persistence layer: contains the repositories that persist the data in the database.

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.

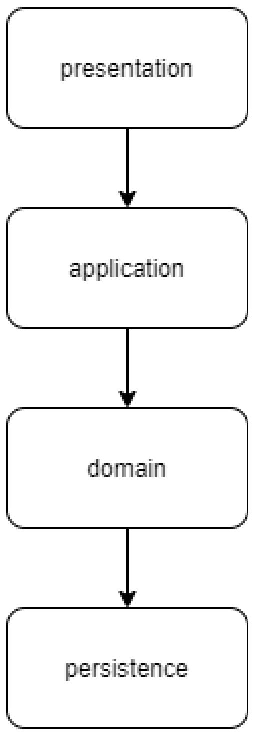


Figure 2 Package diagram

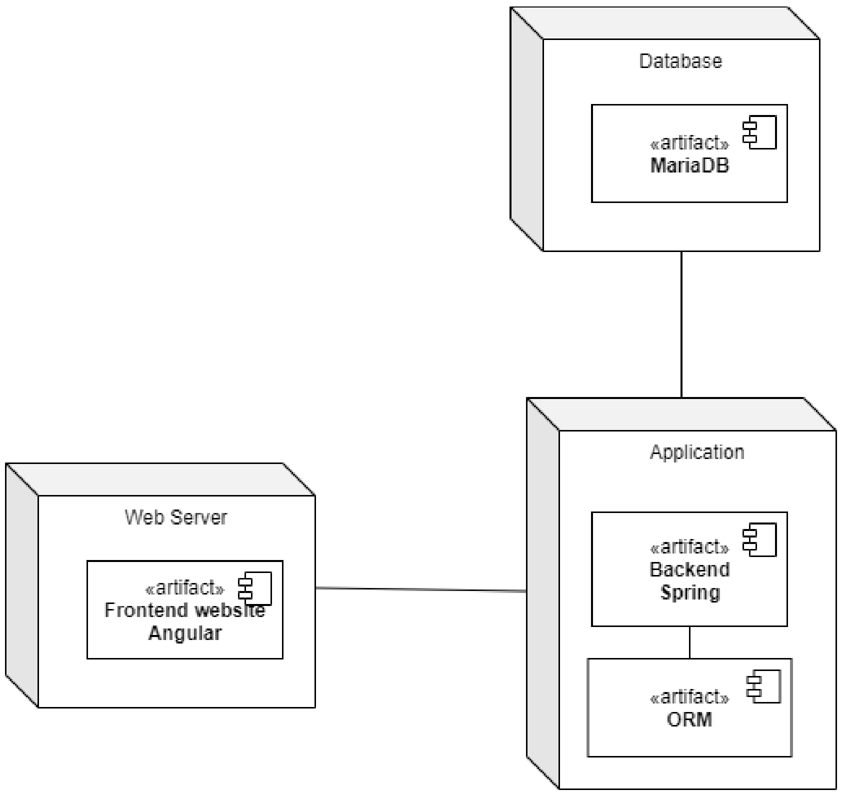


Figure 3 Deployment diagram

4. UML Sequence Diagrams

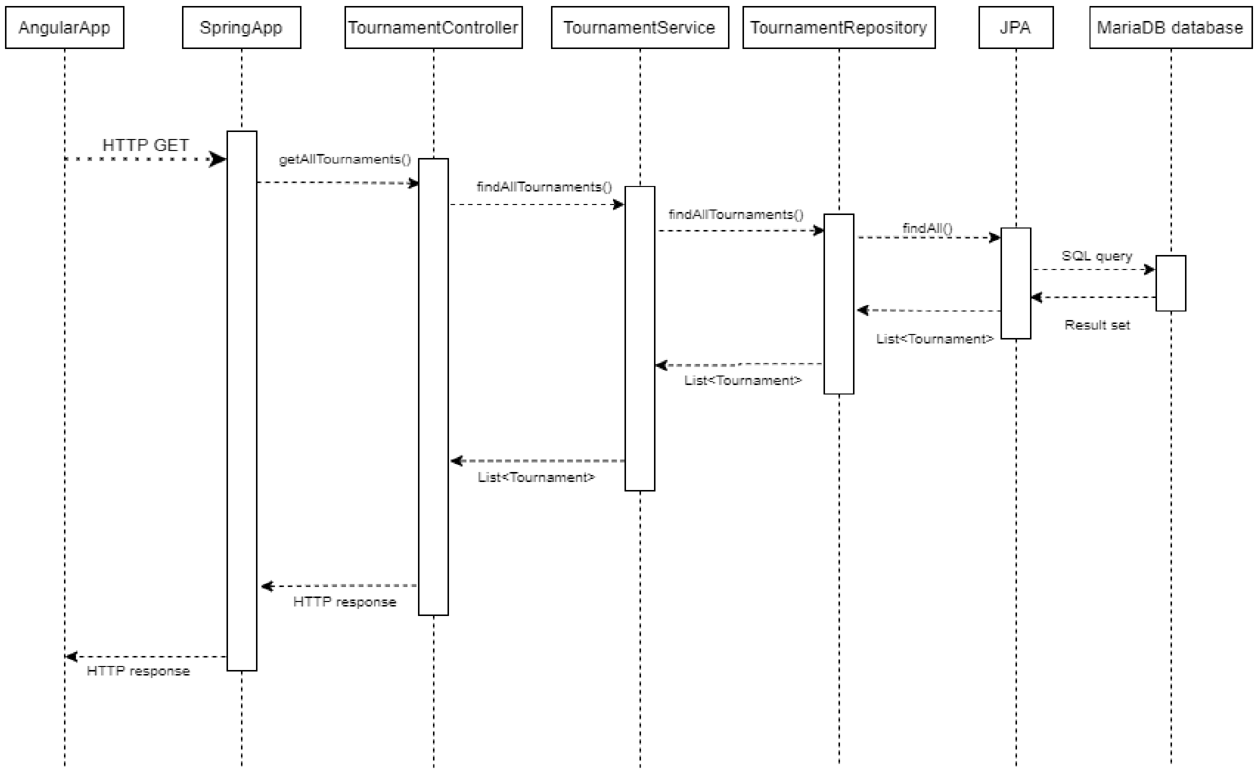


Figure 4 Fetch all tournaments sequence diagram

5. Class Design

5.1 Design Patterns Description

The application uses the Builder design pattern. Its intent is to separate the construction of a complex object from its representation. Creating and assembling the parts of a complex object directly within a class is inflexible. It commits the class to creating a particular representation of the complex object and makes it impossible to change the representation later independently from the class. The Builder design pattern describes how to solve such problems:

- Encapsulate creating and assembling parts of a complex object in a separate Builder object
- A class delegates object creation to a Builder object instead of creating the objects directly

This pattern is used in the process of creating a new Match object. A MatchBuilder object is created, which given a Tournament, and two Fighter object, it creates an appropriate Match object. It checks for the best time for the Match to take place at:

- If there are no matches scheduled, schedule it at the first day of the tournament
- Else schedule it a week after the last scheduled match, if it is still in the tournament's period
- If it is not, schedule it for the last day of the tournament

5.2 UML Class Diagram

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

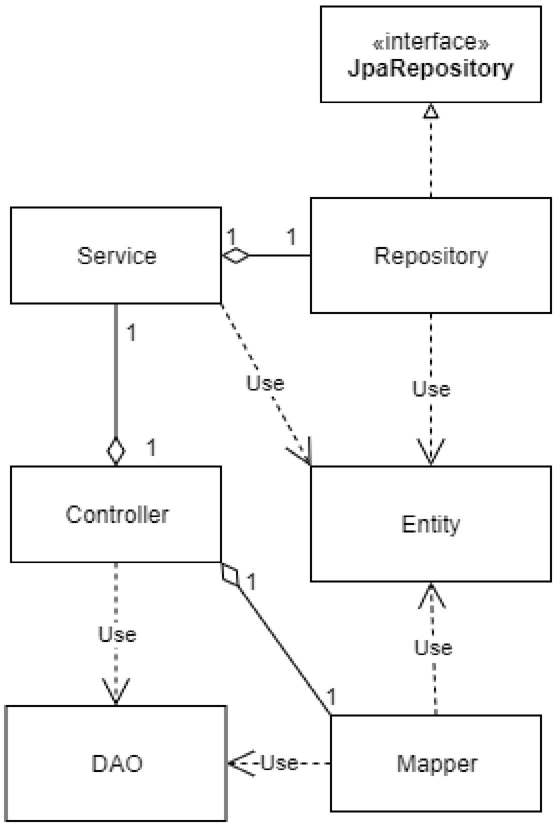
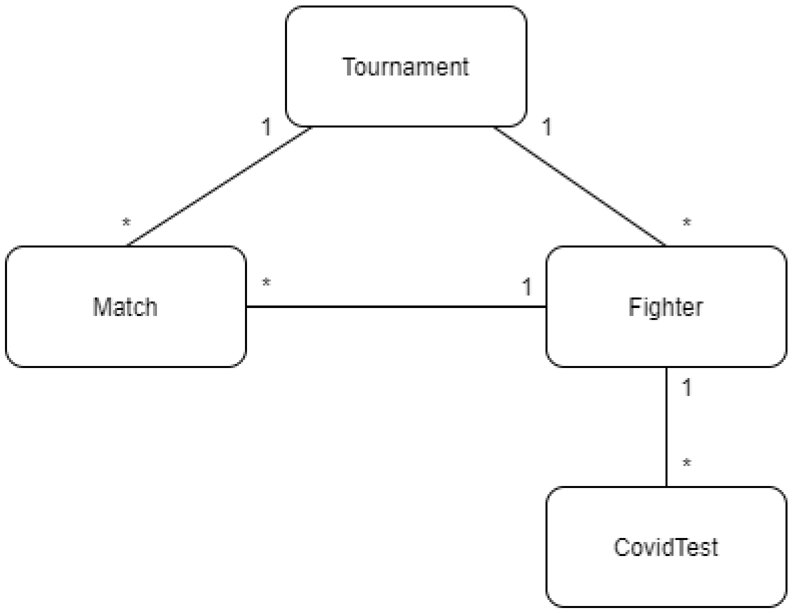


Figure 5 Class diagram

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.



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, created in Angular, 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.

8. Bibliography

https://en.wikipedia.org/wiki/Builder_pattern#:~:text=The%20builder%20pattern%20is%20a,Gang%20of%20Four%20design%20patterns.

https://dzone.com/articles/layered-architecture-is-good#:~:text=What%20Is%20Layered%20Architecture%3F,the%20particular%20concern%20it%20represents

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

https://cli.angular.io/