Assignment 2

Student: Tofan Emil-Mihai

**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 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

For this assignment, we had to Design and implement an application that helps MMA tournaments manage their scheduled fights better while ensuring covid safety standards. The goal was to have an administrator who would notify fighters about an upcoming UFC Tournament. Fighters would presents themselves with a covid test, and further testings would be done to ensure safety standards. The fighters who have at least 3 weeks of negative test history would then be eligible for participation and they would get matched with fighters of the same caliber. The goal of the application was to help the administrator generate and view in real time the schedule of the tournament based on all the criteria presented above.

Different from the other assignment was the creation of the events and the difference in time between the scheduled fights. The administrator could create both weekly and monthly matches which would require at least bi-weekly or bi-monthly fights respectively to generate traction and revenue.

# Functional Requirements

The functional requirements of this project are the following :

My application is designed to work for the administrator of the UFC Tournament as well as for the fighter. The administrator can notify fighters of a date for the starting of the preparation period for the next tournament (Period in which fighters can register for the tournament and in which they will be tested against Covid to ensure safety standards). Before notifying them, the administrator will also choose the type of the tournament to be generated, the choice being between a tournament with at least bi-weekly events and one with at least bi-monthly events.

A forum is available at all times on which fighters will see the announcement of the administrator. The fighters can then start registering for the event and the administrator can choose to start the testing period. The administrator will test all the fighters at least once. If they are negative from the start, they shall not be tested again, but if their tests come back as positive, they will keep getting tests every three weeks until they are negative. Once all the fighters are healthy, the administrator can choose to start the tournament and generate a schedule.

The tournament’s starting day will be the day in which the last positive fighter got his negative three weeks result. The scheduler will generate 3 fights per week or per month (depending on the type of the tournament) until all the fighters of matching calibers have found an opponent. When the schedule is all done, the administrator can move forward and backward on each week on the tournament and assess the situation.

# Non-functional Requirements

For the non-functional requirements, I am going to focus on 6 main ones when talking about my application : Extensibility, Maintainability, Deployability, Scalabillity, Testability and Usability.

One of the requirements regarding the design of this assignment was the use of a layered architecture so the non-functional requirements revolve highly around the constrains imposed by this architecture.

When talking about extensibility, the ability to extend such a system and the level of effort required to implement the extension is great because for a new extension of the software, one would have to start the implementation at the lowest layer and gradually implement it for all the layers, which is quite a difficult and time-consuming task.

When talking about maintainability, the same problems as with extensibility arise. Those two non-functional requirements could have even been grouped together into one non-functional requirement called Agility. Referring to the ease with which this software system can be modified to correct faults or to add improvements is not great because one would have to implement the correction or improve the system across all the layers of the application.

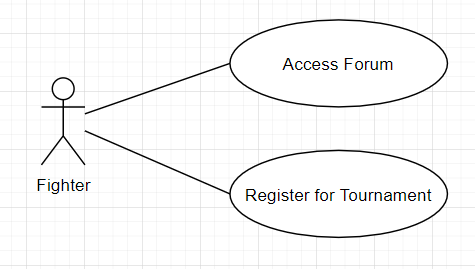
Referring to Deployability, the application performs great. Another constraint imposed on the development of the project, was that a frontend would be created using the client-server architectural style, making. Deployment requirements describe the precise, desired configuration of a software system. They relate the system’s non functional requirements to its architecture, providing a basis for making decisions about design trade-offs in terms of the resulting system's non functional properties. The client-server architecture thus makes my project easy to deploy.

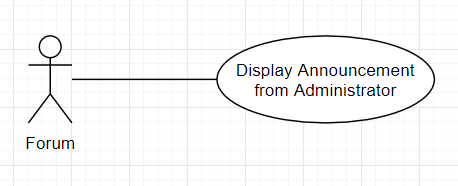
When talking about scalability, the layered architecture poses a problem again, the application’s ability to manage an increased demand of users or be more adaptable to user’s needs or changes being difficult to uphold.

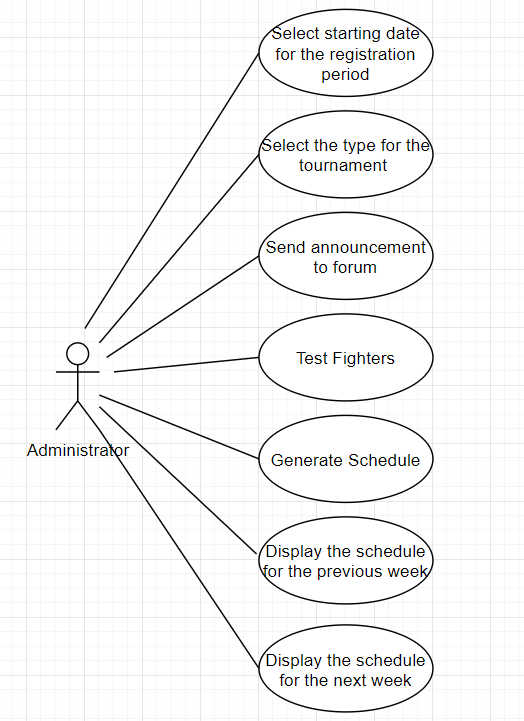
Referring to testability, the application does great. Because of most of the logic being implemented in the lower layers of the application, one can easily test only the final functionalities of the project and see their correctness.

The last of the non-functional requirements discusses refers to Usability. The application is very easy to use and has a very user-friendly frontend. All the buttons and functionalities are labeled and each time a user inserts wrong data for instance, they are instantly notified of their mistake and what they can do to correct it.

2. Use-Case Model







**Use case : Administrator’s use case for the application**

**Level : summary level**

**Primary actor : Administrator**

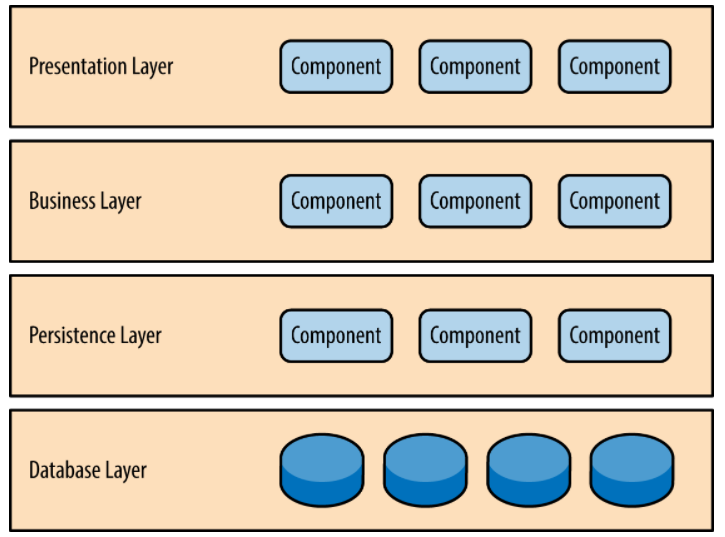
Main success scenario : The Administrator’s main job revolves around the creation of the tournament. He will select the date for the starting of the registration period, select the type of tournament that will be generated, will send the announcement about the start of the registration period to the forum, he will start the testing period on all fighters and continue it until they are all covid free, he will start the generation of the tournament and he will be the one for which the schedule will be displayed and the one who can move back and forth on this schedule based on weekly or month events

Extensions: In case the administrator panel is refreshed by mistake, all the modifications that were done on the database will be kept. In case the fighters are already healthy, he will be able to just start the generation of the schedule. In case the schedule is already generated, he will be able to move back and forth on it without generating another scheudle. All the operations are done on the database.

3. System Architectural Design

**3.1 Architectural Pattern Description**

Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. The scheme for this architecture is shown below.



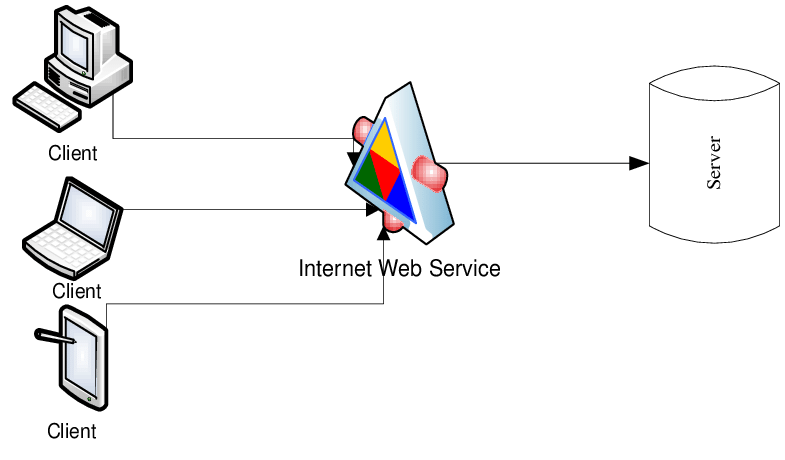
**3.1.2. Client-Server Architecture**

**Client Server Architecture** is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection. This system shares computing resources. Client/server architecture is also known as a networking computing model or client/server network because all the requests and services are delivered over a network.

The "client–server" characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide. For example, a web server serves web pages, and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a *service*.

Characteristics of the client – server architecture include:

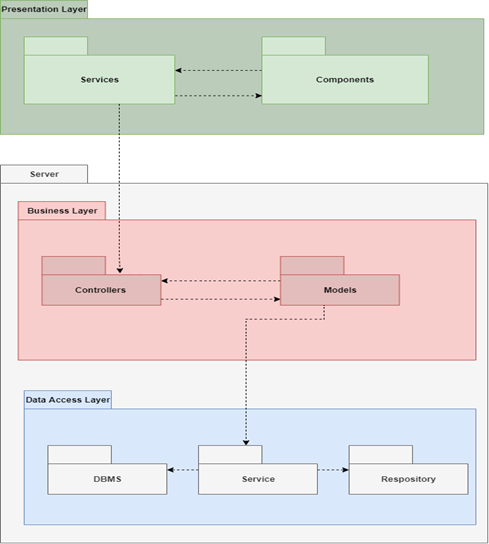
* Client and server machines need different amount of hardware and software resources.
* Client and server machines may belong to different vendors.
* A client or server application interacts directly with a transport layer protocol to establish communication and to send or receive information.
* The transport protocol then uses lower layer protocols to send or receive individual messages. Thus, a computer needs a complete stack of protocols to run either a client or a server.
* A single server-class computer can offer multiple services at the same time; a separate server program is needed for each service.



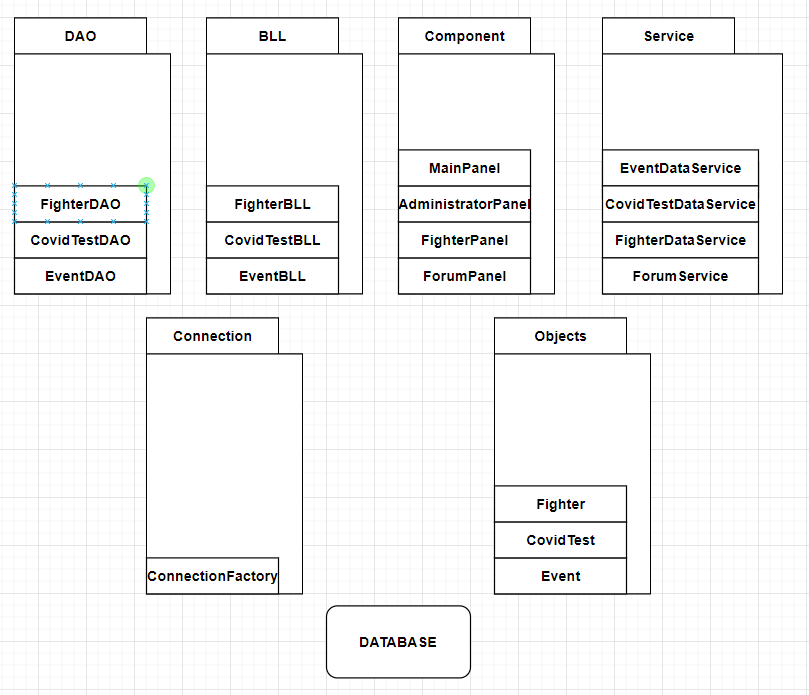
**3.2 Diagrams**

When talking about the style in which I implemented my specific architecture, I constructed my application in a presentation layer, which contains the frontend of my architecture together with the services for the communication with the server, the server representing the business layer, containing the classes that encapsulate the application logic, the data access layer which contains the classes containing the queries and the database connection and the database layer which is represented by the database used for this application.

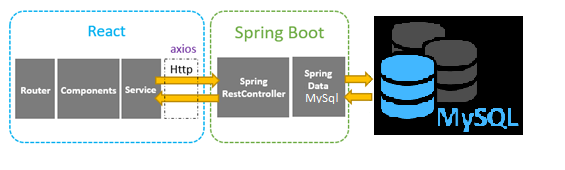
**Package Diagram**



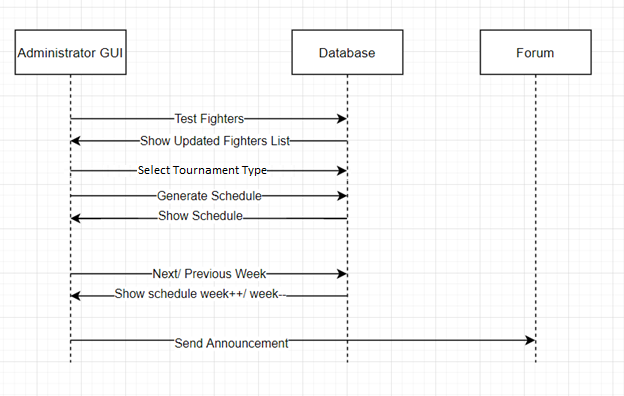
**Component Diagram**

****

**Deployment Diagram**



4. UML Sequence Diagrams



5. Class Design

**5.1 Design Patterns Description**

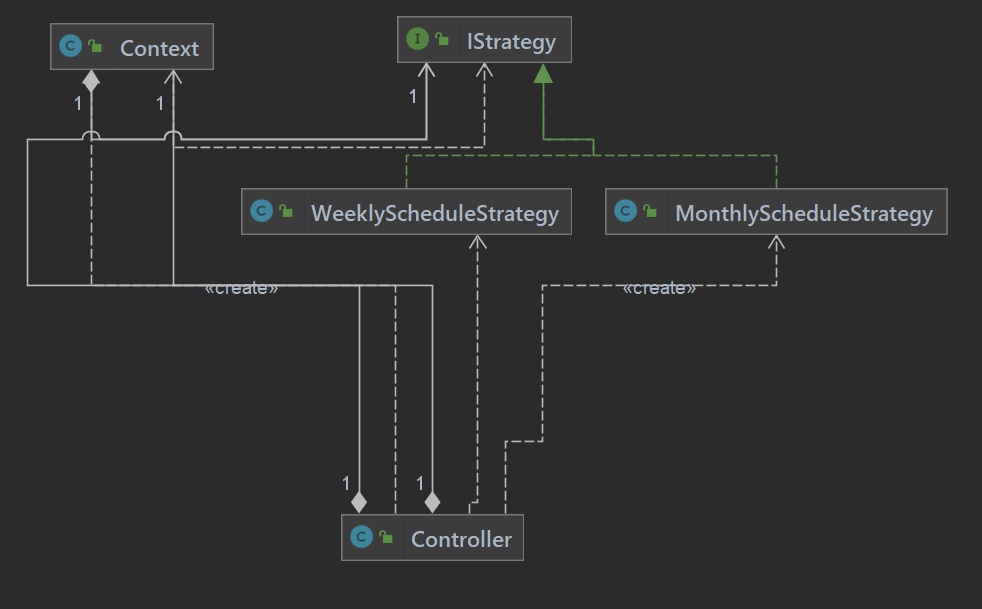
In my application I have used two design patterns : The Strategy Behavioral Design Pattern and the Builder Creational Design Pattern.

The Strategy Design Pattern is a design pattern which enables a class’ behavior to be changed at run time. It is useful because instead of implementing a single algorithm directly, code receives run time instructions as to which in a family of algorithms to use.

The builder pattern is a design pattern designed to provide a flexible solution to various object creation problem. The main advantages of Builder Pattern are as follows: It provides clear separation between the construction and representation of an object. It provides better control over construction process. It supports to change the internal representation of objects.

**5.2 UML Class Diagram**

**Strategy Class Diagram**

**

The Strategy Design Pattern was implemented to enable the generation of different types of tournaments at run time. The administrator will chose the type of tournament to be generated and the strategy pattern will enable the right algorithm to be executed. The implementation of the pattern was done as follows :

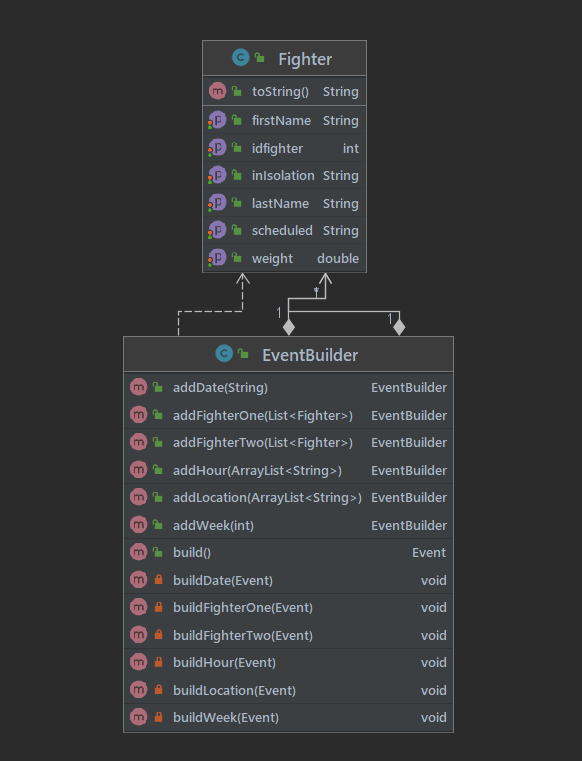
An interface called IStrategy was implemented containing one method called addWeekFlag() which returns an integer.

Two classes implementing that interface were created : WeeklyScheduleStrategy and MonthlyScheduleStrategy, each with it’s own implementation of the addWeekFlag method. the WeeklyScheduleStrategy would add one to the current week flag and the MonthlyScheduleStrategy would add four, both returning the newly modified week flag. Based on this week flag, the schedule would be created.

A context class for the strategy was created called Context. The constructor for a context object would create the new object based on the strategy chosen and the executeStrategy method inside the Context Class would return the output of the addWeekFlag() method for that specific strategy.

Inside the controller, when the administrator would press the generate schedule button, the controller would initialize the context with the strategy chosen by the administrator and the scheduler would start building events with the correct time interval.

**Builder Class Diagram**

****

The Builder Design pattern was used to generate an event type Object having a great attention to the logic of implementing such an object. For instance, any event contains two fighters participating in the event(fighting). If the first fighter has been selected, the second fighter for the event has to be picked in a precise way such that he or she is of the same caliber as the first one. For that, the eventBuilder keeps track of all the fighters in the database, and only creates events based on +-5 kilogram range, so that the fight would be equilibrated. Other logic that went into the design of the event builder is, selecting only the healthy and not already scheduled fighters in the database, not generating a fight between the same two fighters, not generating an event at a location, date and hour where an event is already taking place etc.

As we can observe, the Builder Design pattern was very useful in this case because of the complexity of the logic for the object we had to build.

6. Data Model

For this system’s implementation, three data models have been used and they are the fighter, the event and the covidTest. Each of them has its separate table in the database.

The fighter contains six columns :

idFighter which is the unique identifier for any fighter

firstName which is the first name of the fighter

lastName representing the last name of the fighter

weight representing the fighter’s weight

inIsolation which tells us if the fighter is in Isolation or not

scheduled which tells us if the fighter has an event assigned to it or not

The covidTest contains seven columns :

idCovidTests which is the unique identifier for any test

fighterFirstName representing the first name of the fighter for which that test has been made

fighterLastName representing the last name of the fighter for which that test has been made

arrivalTest representing the result of the test with which the fighter came from home

arrivalTestDate which is the date on which the arrival test was done

secondTest representing the result of the test made at the tournament

secondTestDate which is the date on which the second test was done

The event contains seven columns :

idEvent which is the unique identifier for any event

location which represents the location at which the event will take place

date which represents the date on which the event will take place

hour which represents the hour at which the event will take place

fighterOne which represents the first fighter taking part in that event

fighterTwo which represents the second fighter taking part in that event

week which represents the week of the tournament in which the event takes place

7. System Testing

In order to test my application I have tested it for any new operation/ object that was added. The user of my application also has to pass some validation tests in order for the application to run as it should, otherwise he will not be able to use it. For instance, when a fighter is registered, the button will not work unless the administrator has already selected a date for the registration period. All the inputs for the fighter’s data are parsed so the user cannot enter anything else than a string for the first and last names and a double value for the weight. Also, if the fighter that is to be inserted is already in the database, the user will be notified and the fighter will not be inserted again.

For the administrator, certain validation tests are in place. No button on the administrator GUI works unless the administrator has already selected a starting date for the registration. The generate schedule button will not appear on the GUI until all the fighters are covid free.

8. Bibliography

<https://docs.google.com/document/d/1UF_DMNeyUNhqCafZFlSRIIUCN-25TsFC/edit>

<https://refactoring.guru/design-patterns/strategy>

<https://cio-wiki.org/wiki/Client_Server_Architecture>

<https://www.springboottutorial.com/spring-boot-react-full-stack-crud-maven-application?fbclid=IwAR1_EEVsMkGDDM1qEVtTXEC2puVLmiWAatgQzOl1kDD6OiZeIUfLvvaydus>