Assignment 1

Analysis and Design Document

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.

# 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. 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 between 2 and 3 fights a week 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 asses 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.

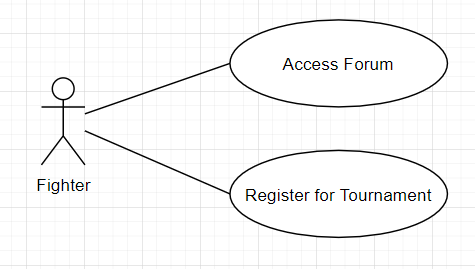
Referring to Deployability, the layered architecture of this assignment’s software poses a problem again. Also, because of my choice to implement this assignment as a desktop application, ease of deployability as in deploying software updates for my users is very hard, almost impossible, compared to what it would be in case I implemented it as a Web Application. The newly updated application would have to be reinstalled on a user’s computer and the last version would not matter at all.

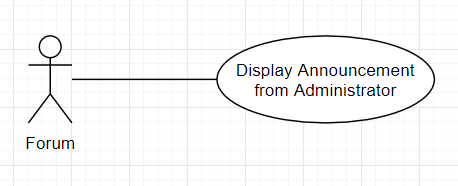
When talking about scalability, the layered architecture poses a problem again, the application’s ability to manage an increased demand of users or be more adaptable to user’s needs or changes being difficult to uphold.

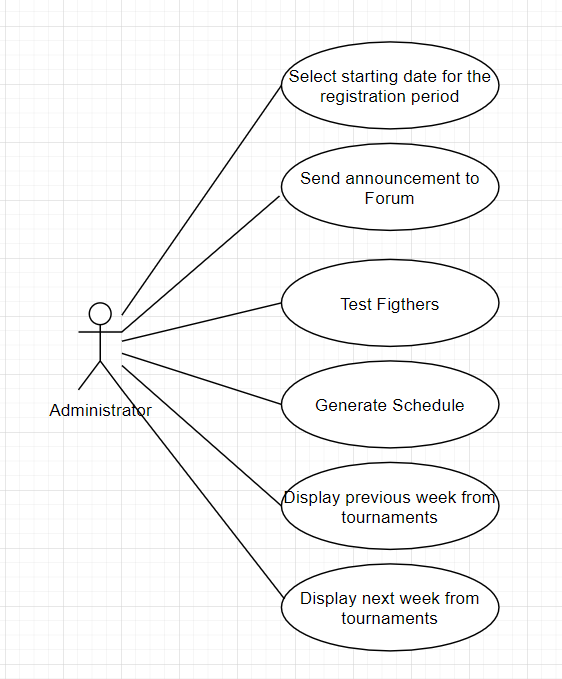
Referring to testability, the application does great. Because of most of the logic being implemented in the lower layers of the application, one can easily test only the final functionalities of the project and see their correctness.

The last of the non-functional requirements discusses refers to Usability. The application is very easy to use and has a very user-friendly Graphical User Interface. 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**

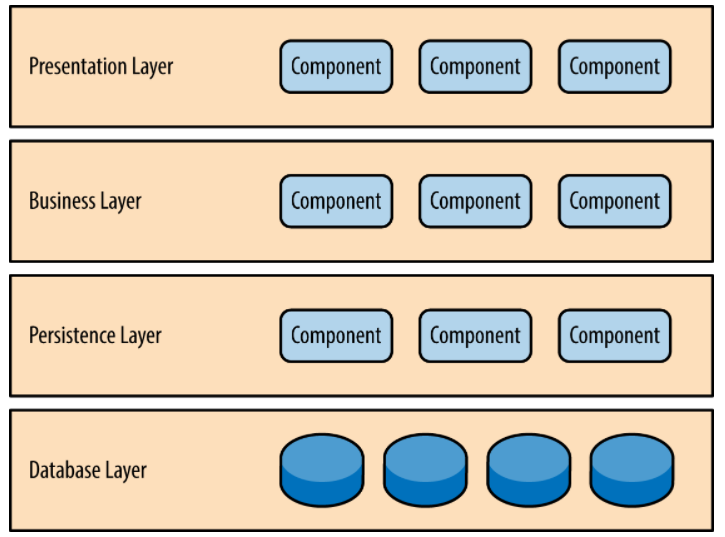
Main success scenario : The Administrator’s main job revolves around the creation of the tournament. He will select the date for the registration period, he will send the announcement about the start of the registration period on the forum, he will start the testing period on all fighters and continue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

Extensions: In case fighters are already registered in the database, he will not have to send another announcement, he will just start the testing. In case the fighters are already healthy, he will be able to just start the generation of the schedule. 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.

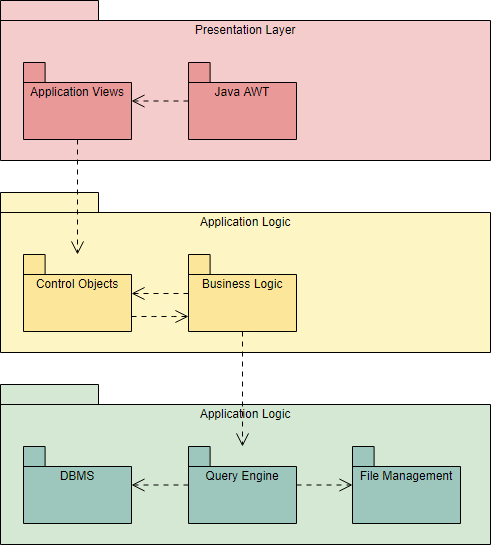


**3.2 Diagrams**

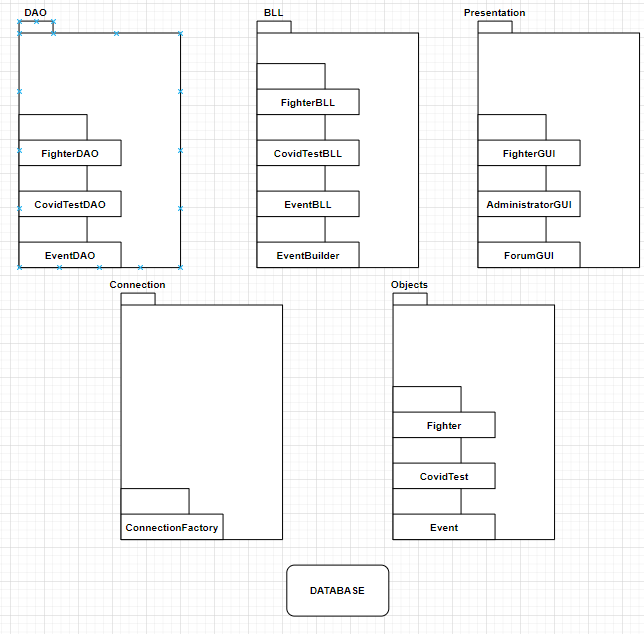
When talking about the style in which I implemented my specific architecture, I used a 4-tier( or 4-layered) layered architecture. This architecture helps us split the application is different layers, each layer having a special purpose . The four layers and their respective functionalities are :

* + The presentation layer, which contains the classes defining the user interface
  + The business layer, which contains the classes that encapsulate the application logic
  + The Data Access Layer, which contains the classes containing the queries and the database connection
  + The Database Layer, which is represented by the database used for this application

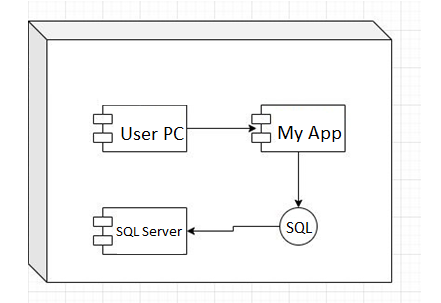
**Package Diagram**



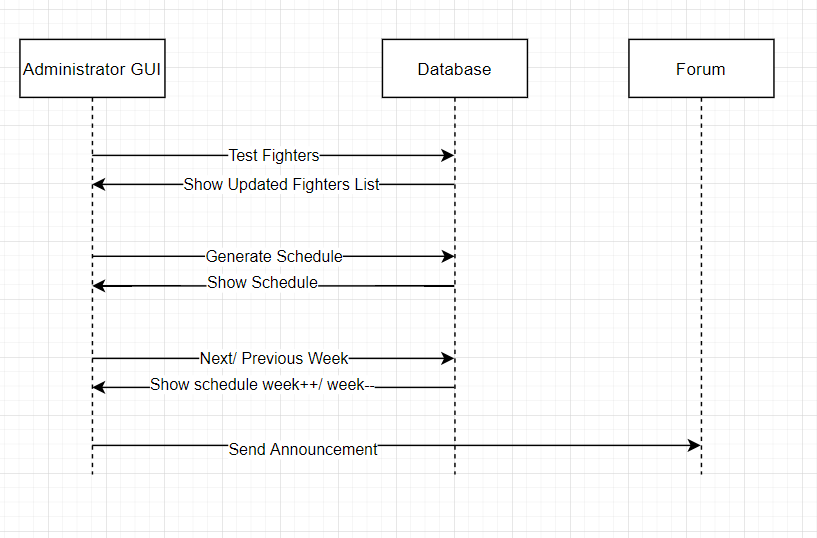
**Component Diagram**

****

**Deployment Diagram**

****

4. UML Sequence Diagrams

**

5. Class Design

**5.1 Design Patterns Description**

In my application I have used two design patterns : The observer design pattern and the builder design pattern.

The observer design pattern is a software design pattern in which an object, named the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

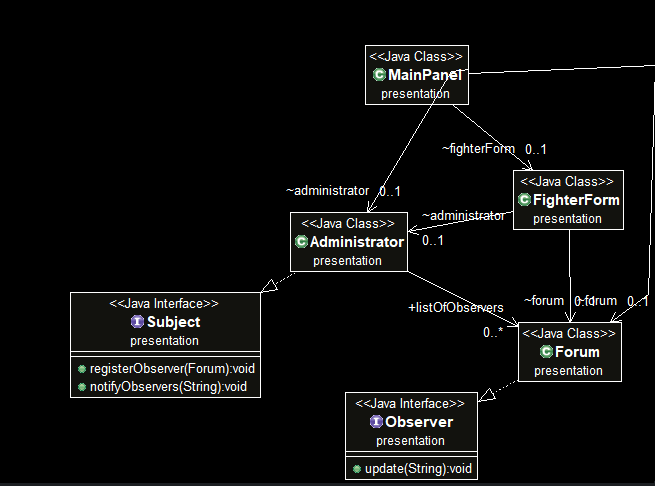
The builder pattern is a design pattern designed to provide a flexible solution to various object creation problem. The main advantages of Builder Pattern are as follows: It provides clear separation between the construction and representation of an object. It provides better control over construction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

**5.2 UML Class Diagram**

****

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.



The Observer Pattern was used to notify the Forum Object of the announcements made by the administrator. The Main panel, which is the GUI from which all the other GUI’s are accessed keeps track of the administrator, the fighter and the forum so that one announcement will always be visible for a forum observer even if the forum is closed. Any fighter seeing the forum will be able to access it over and over again and the message sent by the observable administrator will be kept at all times, as long as the instance of Main Panel is active.

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 using regex validation and other conditions 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/171D9FdPZ_h0vI1w4l_bVOYVTKxDMkuozLIfU1w-SXVk/edit>

<https://drive.google.com/drive/folders/1ZjCkIhx7wotNglN6Zypu6Dk>

<https://en.wikipedia.org/wiki/Builder_pattern>

<https://en.wikipedia.org/wiki/Observer_pattern>

https://openclassrooms.com/en/courses/6397806-design-your-software-architecture-using-industry-standard-patterns/6896176-layered-architecture