Tennis Club

Analysis and Design Document

Student: Georgescu Vlad

**Group: 30434**

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

The aim of this project is to design an application for a tennis club that organizes matches. In order to win a tennis match, one should first get to 3 sets won. In order to win a set, the player must first reach minimum 6 games by a margin of 2.

The application was design to have 2 types of users, an admin and a regular user represented by a player. The player can visualize the matches and can update the score of a game in a desired match. The admin can see all the matches, can add a new player, can edit an existing player and can delete one. Moreover, the admin can see all the matches, can update the score of a specific match, can create a new set for a specific match and can create a new match between two existing players. Also, the admin can generate match reports in pdf or txt format.

# Functional Requirements

Here are the functional requirements of the application:

- the regular user:

\*) press Show Match button and can see the all the matches

\*) insert the match number to update score, insert the set number in which to update score, enter the games for the player 1 and the games for the player 2 and press UpdateGame button.

- the admin:

\*) press ShowPlayers button and see all players

\*) enter player name, mail and password and press AddPlayer button to add a new player

\*) enter player name for which the admin wants to update mail or password or both and enter password and mail in the textfields and press EditPlayer.

\*) enter player name which admin wants to delete and press DeletePlayer button to delete a player

\*) press ShowMatches button to see all matches

\*) enters two existing player1 and player2 names and press the AddMatch to add a new match.

\*) enters the match number and press DeleteMatch button to delete a match

\*) enters the match number and press CreateNewSet button to create a new set.

\*) insert the match number to update score, insert the set number in which to update score, enter the games for the player 1 and the games for the player 2 and press UpdateScore button

\*) insert the desired format of the report and press GenerateReport button to generate reports of mathces

# Non-functional Requirements

## 1.3.1 Availability

- Definition: Is the extent to which an application is operational, functional and usable for completing or fulfilling a user’s or business's requirements

- Source of stimulus: Designing of the app

- Stimulus: Crash, timing, response

- Environment: Normal, Operation

- Artifact: Process, Storage, Communication

- Response: Record, notify

- Response measure: Available, no downtime

## 1.3.2 Performance

- Definition:  Performance is about timing. Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them.- Source of stimulus: Internet connection and speed, designing of the app

- Stimulus: Periodic events arrive

- Environment: Normal

- Artifact: System

- Response: Processes stimuli

- Response measure: Latency, deadline, throughput, miss rate, data loss

## 1.3.3 Security

- Definition:  Security is a measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users.

- Stimulus: Tries to display data, change/delete data, access system services, reduce availability to system services

- Environment: disconnected

- Artifact: System services; data within system

- Response: Authenticates user; hides identity of the user

- Response measure: Time/effort/resources required to circumvent security measures with probability of success;

## 1.3.4 Testability

- Definition: Software testability refers to the ease with which software can be made to demonstrate its faults through testing.

- Stimulus: Software testability refers to the ease with which software can be made to demonstrate its faults through testing.

- Environment: At design time, at development time, at compile time, at deployment time

- Artifact: Piece of design, piece of code, complete application

- Response: Provides access to state values; provides computed values; prepares test environment

- Response measure: Percent executable statements executed, probability of failure if fault exists, time to perform tests

## 1.3.5 Usability

- Definition: Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides

- Stimulus: Wants to learn system features; use system efficiently; minimize impact of errors; adapt system; feel comfortable

- Environment: At runtime or configure time

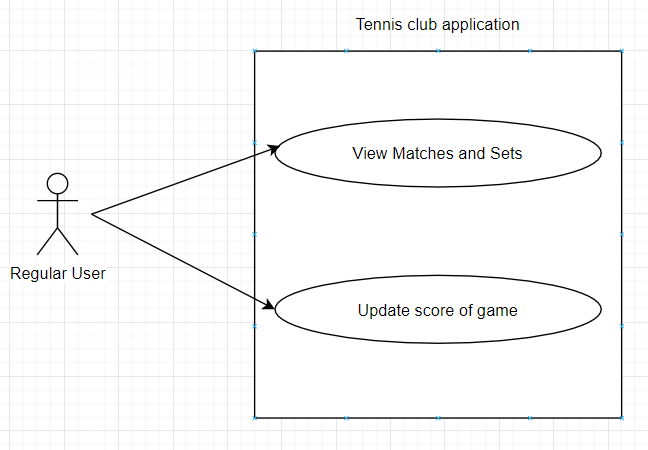
- Artifact: System

- Response: "use system efficiently": aggregation of data and/or commands; re-use of already entered data and/or commands; support for efficient navigation within a screen; distinct views with consistent operations;

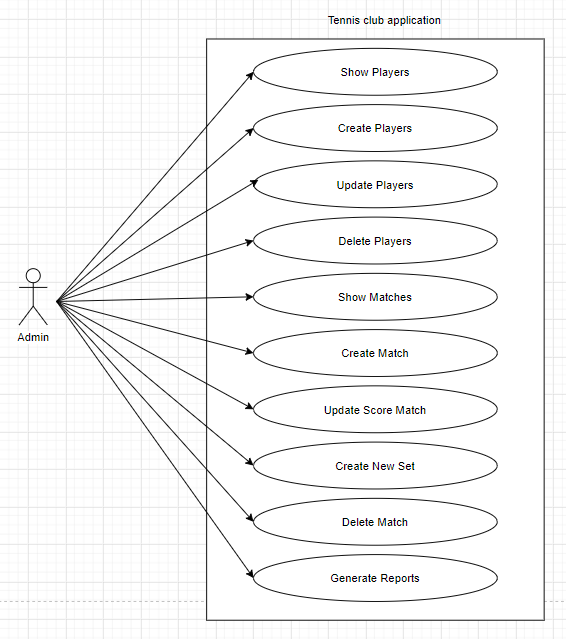
- Response measure: Task time, number of errors, number of problems solved, user satisfaction.

2. Use-Case Model

The Use case diagram for the Regular user (player) is the following one:



The Use case diagram for the Admin is the following one:



Use case: <Create players. Its purpose is to describe how players are being created and the importance of this scenario and what happens in case of the success or failure scenarios.>

Level: <Admin goal>

Primary actor: <Administrator>

Main success scenario: <The administrator wants to perform create player operation. He inserts the name, mail and password for that player and the system checks if the data is valid. In case of success, the system inserts the player in the database and display it in the user interface.>

Extensions: < The administrator wants to perform create player operation. He inserts the name, mail and password for that player and the system checks if the data is valid. In case of success, the system inserts the player in the database and display it in the user interface. In case of failure, an error message is being displayed and the player is not created.>

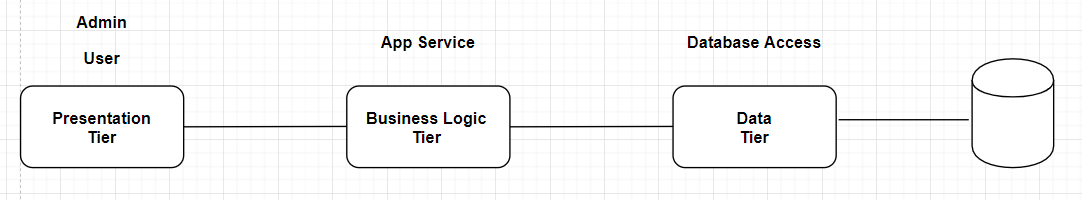
3. System Architectural Design

**3.1 Architectural Pattern Description**

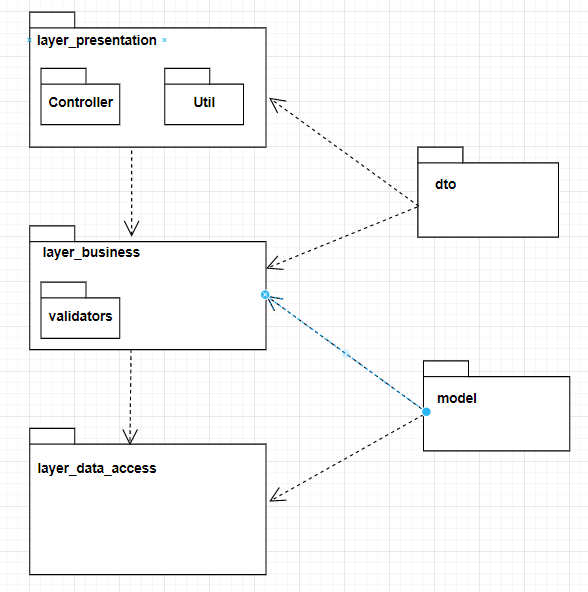
When it comes to the architectural patterns, the one used in the application is the Layers architectural pattern in conjunction with the MVC design pattern. Layered architecture patterns are n-tiered patterns where the components are organized in horizontal layers. This is the traditional method for designing most software and is meant to be self-independent. This means that all the components are interconnected but do not depend on each other. It is divided into Presentation, Business and Data Layers. The Model View Controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information.

**3.2 Diagrams**

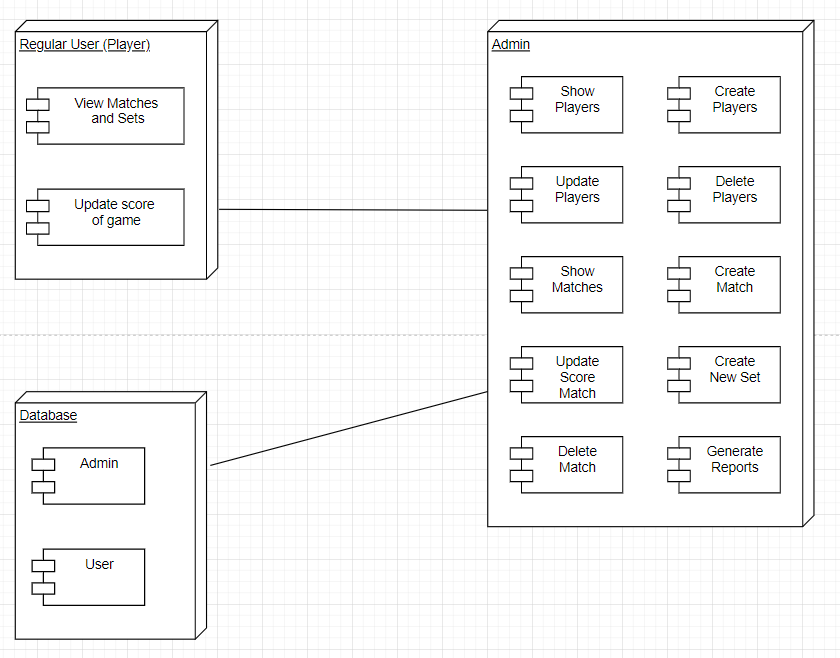
Below is the conceptual architecture diagram of the application:



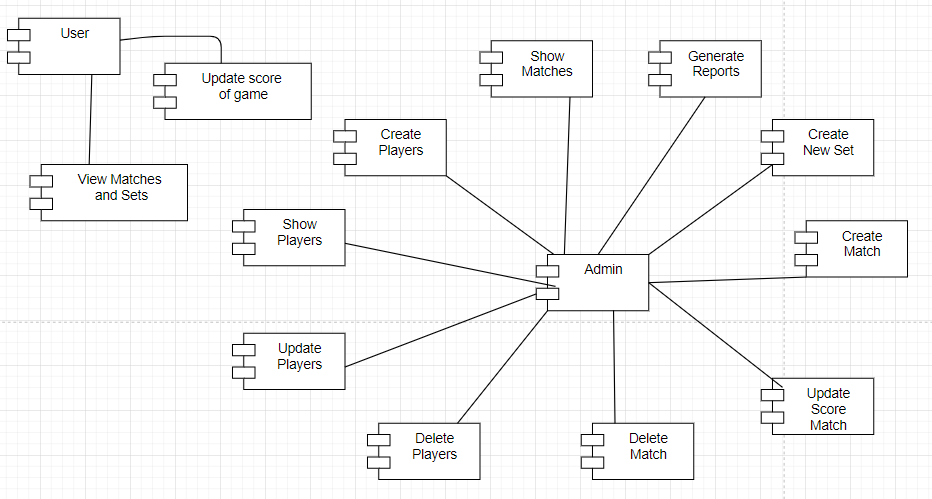
Here is the system package diagram:



This is the system Deployment diagram:

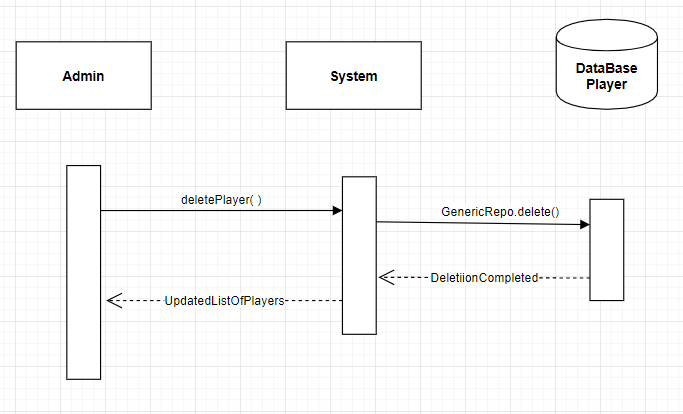


This is the Component diagram of the system:



4. UML Sequence Diagrams

The sequence diagram for delete player scenario:



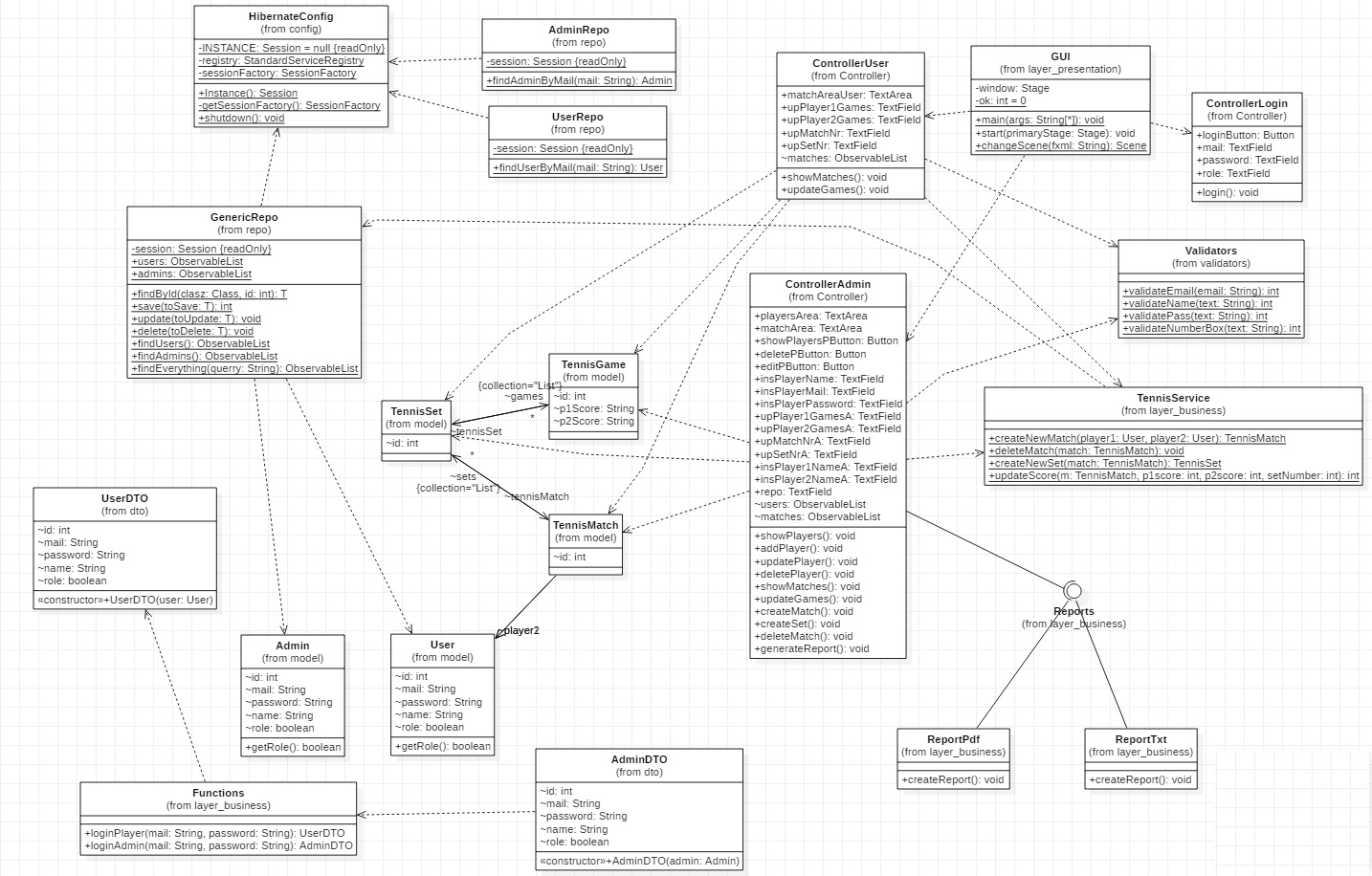
5. Class Design

**5.1 Design Patterns Description**

The design patterns used in the system are the following ones: Layered architecture design pattern, mvc design pattern and Factory Method design pattern. Whereas the first two were used in the structure of the application, since they are architectural deign pattern, the Factory Method design pattern was used to create the matches reports. The first two were described at the beginning of the documentation. Factory Method design pattern is a creational design pattern and it uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

**5.2 UML Class Diagram**

The UML class diagram of the system is the following one:



6. Data Model

The data models used in the system are the following ones:

- The admin: has an id, a name, a password and a mail associated to it. Moreover, it has a role field set to true, which differentiate it from the user which is set to false. From the model class, these fields are linked to the database generating the admin table by the @Column(name = ...) addnotation above each field.

- The user: has an id, a name, a password and a mail associated to it. Moreover, it has a role field set to false, which differentiate it from the user which is set to true. From the model class, these fields are linked to the database generating the admin table by the @Column(name = ...) addnotation above each field.

- TennisMatch, which has as fields 2 users, player1 and player2, and a list of TennisSet associated to each match.

- TennisSet, which has as fields a tennis match and a list of TennisGames associated to each set.

- TennisGame, which has two Strings as fields representing the score of player1 and player2, and a tennis set of which the games belong.

7. System Testing

In order to test the application, JUnitTest framework was used. There were tested the methods from TennisService class.

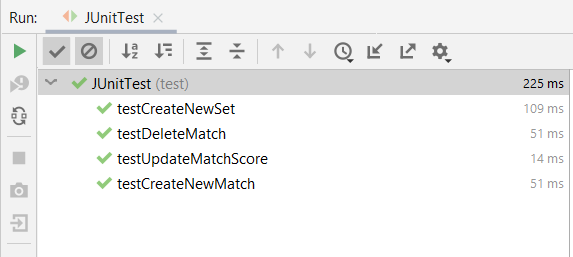
- for the testCreateNewSet method, it tests for creating a new set in match number 1

- for method testDeleteMatch, it tests the deletion of the match number 3.

- for method testUpdateMatchScore, it tests the update of the score in match number 2 and the score updated to 5-4.

- for method testCreateNewMatch, it tests the creation of a new match with the newly created players x and y.

Below, it can be observed the results of the tests:



8. Bibliography

- https://medium.com/@priyalwalpita/software-architecture-patterns-layered-architecture-a3b89b71a057

- https://en.wikipedia.org/wiki/Factory\_method\_pattern

- https://medium.com/@priyalwalpita/software-architecture-patterns-layered-architecture-a3b89b71a057

- https://www.guru99.com/functional-requirement-specification-example.html

- https://www.tutorialspoint.com/itext/itext\_adding\_paragraph.html