Tennis Tournament Application, Assignment 1

Analysis and Design Document

Student: Andrei Gog

**Group:30433**

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

### Objective

The objective of this assignment is to allow students to become familiar with architectural patterns.

### Application Description

Use JAVA/C# API to design and implement an application for a ping-pong association that organizes tournaments on a regular basis. Every tournament has a name and exactly 8 players (and thus 7 matches). A match is played best 3 of 5 games. For each game, the first player to reach 11 points wins that game, however a game must be won by at least a two-point margin. The application should have two types of users: a regular user represented by the player and an administrator user. Both kinds of uses have to provide an email and a password in order to access the application.

The regular user can perform the following operations:

- View Tournaments

- View Matches

- Update the score of their current game. (They may update the score only if they are one of the two players in the game. The system detects when games and matches are won)

The administrator user can perform the following operations:

- CRUD on player accounts

- CRUD on tournaments: He creates the tournament and enrolls the players manually.

### Application Constraints

● The data will be stored in a database. Use the Layers architectural pattern to organize your application. Use a domain logic pattern (transaction script or domain model) / a data source hybrid pattern (table module, active record) and a data source pure pattern (table data gateway, row data gateway, data mapper) most suitable for the application.

● All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.

### Requirements

- Create the analysis and design document (see the template).

- Implement the application.

- Write at least one Unit Test for each method in the business layer (e.g. Test that a game ends if the score is 11-8).

### Deliverables

The following files will be uploaded on your personal github account in a new repository:

- Analysis and design document.

- Implementation source files.

- SQL script for creating and populating the database with initial values.

- Build file (e.g. pom.xml)

The link to this repository must be sent to vlad.buzea@student.utcluj.ro after presenting the application at the laboratory. In the absence of this email the grade of the assignment is 0, regardless of the circumstances.

# Functional Requirements

* login for both the player and the tournament organizer
* CRUD operations on the players, tournaments, etc.
* enroll players to tournaments
* update current game score
* update and detect match end
* update and detect tournament end

# Non-functional Requirements

Performance: a response time of maximum 2 seconds

Availability: 16hours/day

Security: personal encrypted account for each user

Usability: the application will work on any computer that has java installed

2. Use-Case Model

Use case: Enroll in a tournament

Level: user-goal level

Primary actor: Player

Main success scenario: open application, enter personal login information, select user type, click the login button, player interface opens, select the available tournaments, check status for each tournament, press the enroll button

Extensions:

-Wrong login information, and error is shown, have to retype the information

-The tournament is already full or the player is already registered to that tournament

3. System Architectural Design

**3.1 Architectural Pattern Description**

The three-tier architecture is a special case of the Multitier architecture, for the three-tier we have 3 distinct levels: Data Access Level, Business Level and the User Interface Level, this architecture allows any one of the three tiers to be upgraded or replaced independently.

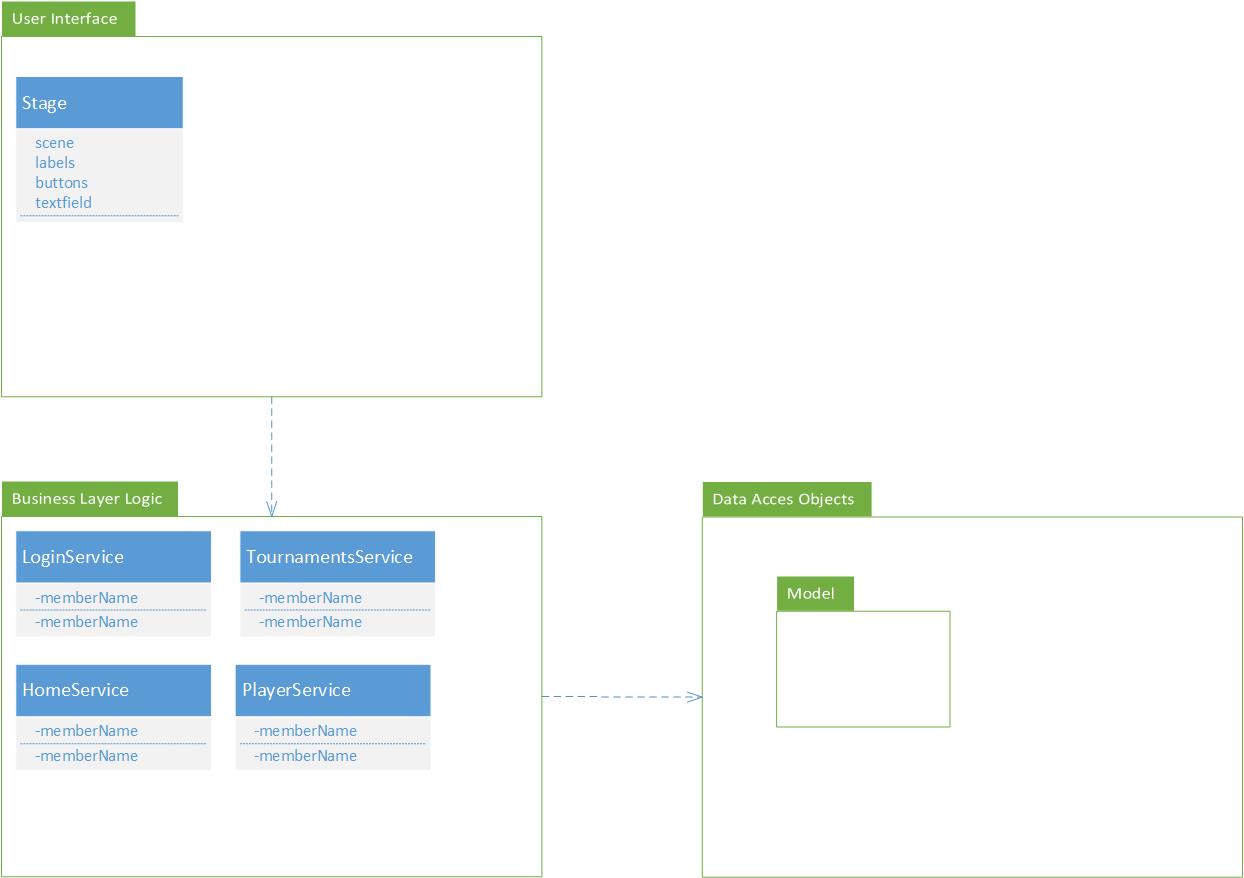
In the Data Access Layer, we have the model-classes of the tables from the database where we just define the attributes, the database-connection class and the DAO-classes where the find and CRUD operations are implemented (also here you can find the DAO interfaces).

In the Business Layer, we have a class containing all the functions that use the Data Access Layer, like login which uses the find method.

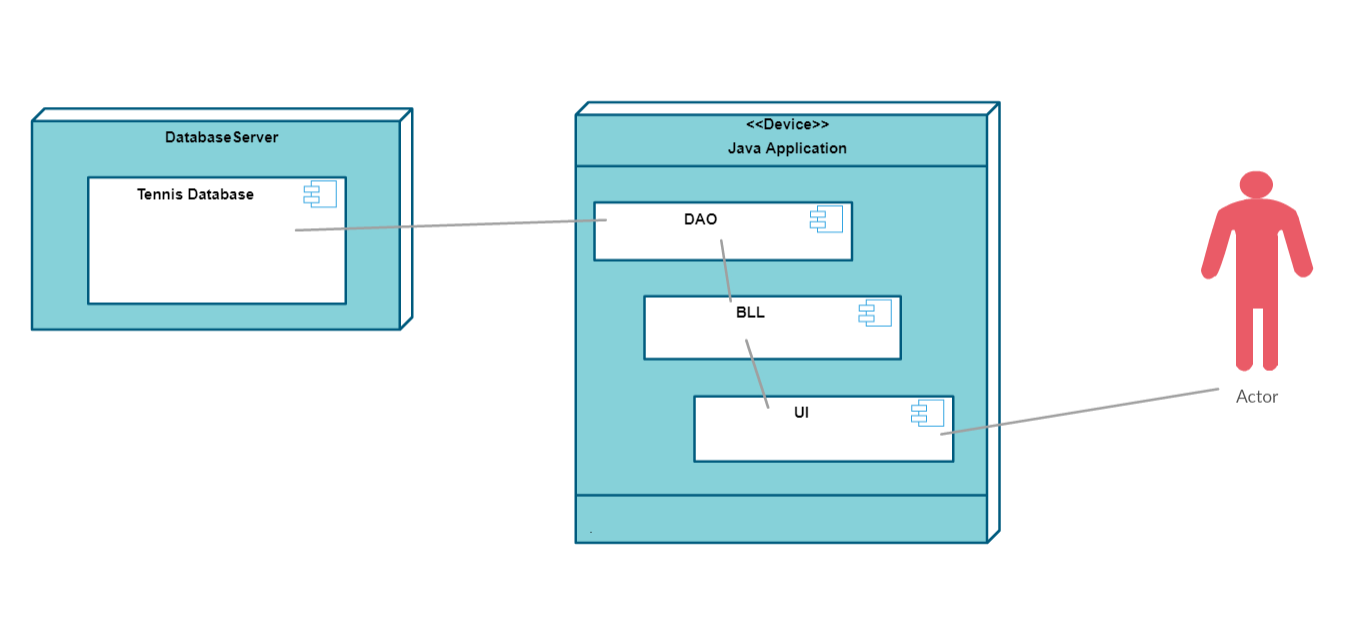
In the User Interface Layer, in my implementation there is one class that contains all the components of the user interface like labels, buttons, scenes, panes, etc. and their attributes.

**3.2 Diagrams**

Package diagram

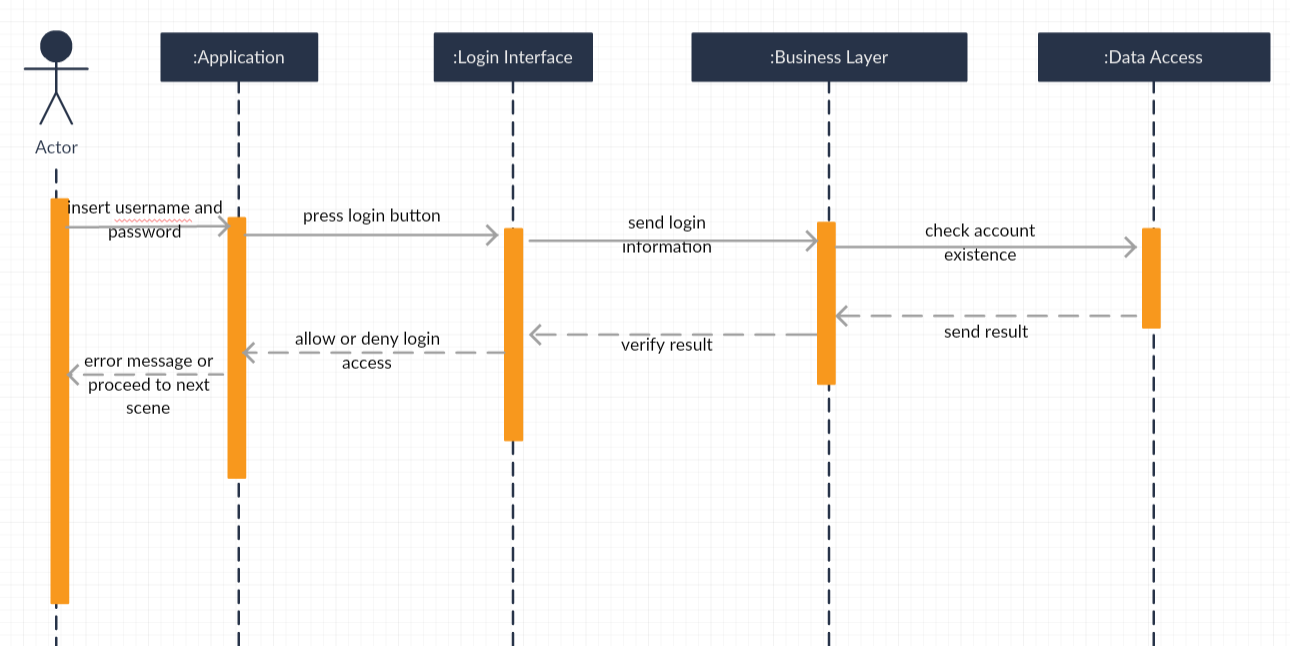


Deployment Diagram



4. UML Sequence Diagrams

Here, is the sequence diagram for a basic player login.



5. Class Design

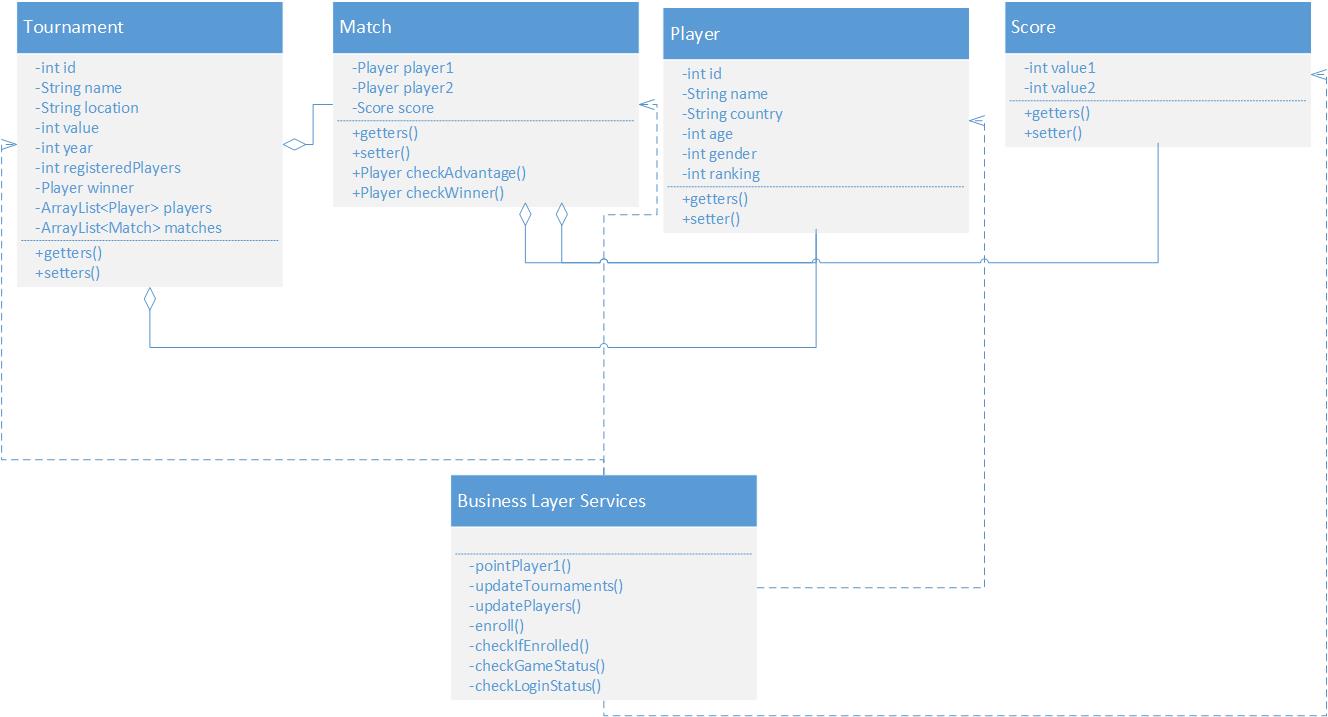
**5.1 Design Patterns Description**

The three-tier architecture is an architecture in which the functional process logic, data access and user interface are developed and maintained as independent modules on separate platforms. Three-tier architecture is a software design pattern and a well-established software architecture.

Data Access Object Pattern or DAO pattern is used to separate low level data accessing operations from high level business services. Following are the participants in Data Access Object Pattern.

* **Data Access Object concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
* **Model Object or Value Object** - This object is containing get/set methods to store data retrieved using DAO class.

**5.2 UML Class Diagram**

**

6. Data Model

The system has in background a simple MySQL database where it will store most of the information about the Tournaments, Players, Users, Matches, etc. and the relationship between them. As an example, on of the most important relationship is the many to many relation between the tournaments and the players, because one tournaments has many players and also one player can participate to many tournaments. This is done using an auxiliary table called “tournamentplayers”. The fields in the database correspond to the fields existent in the Model classes from the DAO.

7. System Testing

The main testing strategy until the implentation of the Junit Testing is by displaying the results in the userinterface and checking if it corresponds to what should normally be extracted from the database.

8. Bibliography

[1] Project idea and tutorials: <https://github.com/buzea/SoftwareDesign2018>

[2] 3-tier architecture: <https://en.wikipedia.org/wiki/Multitier_architecture>

[3] DAO pattern: <https://martinfowler.com/eaaCatalog/domainModel.html>