Ping Pong Tournament App

Analysis and Design Document

Student: Stanciu Alexander

**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

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 games). ​A game is played best 3 of 5 matches. For each match, the first player to reach 11 points wins that match, however a match 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.

# Functional Requirements

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:

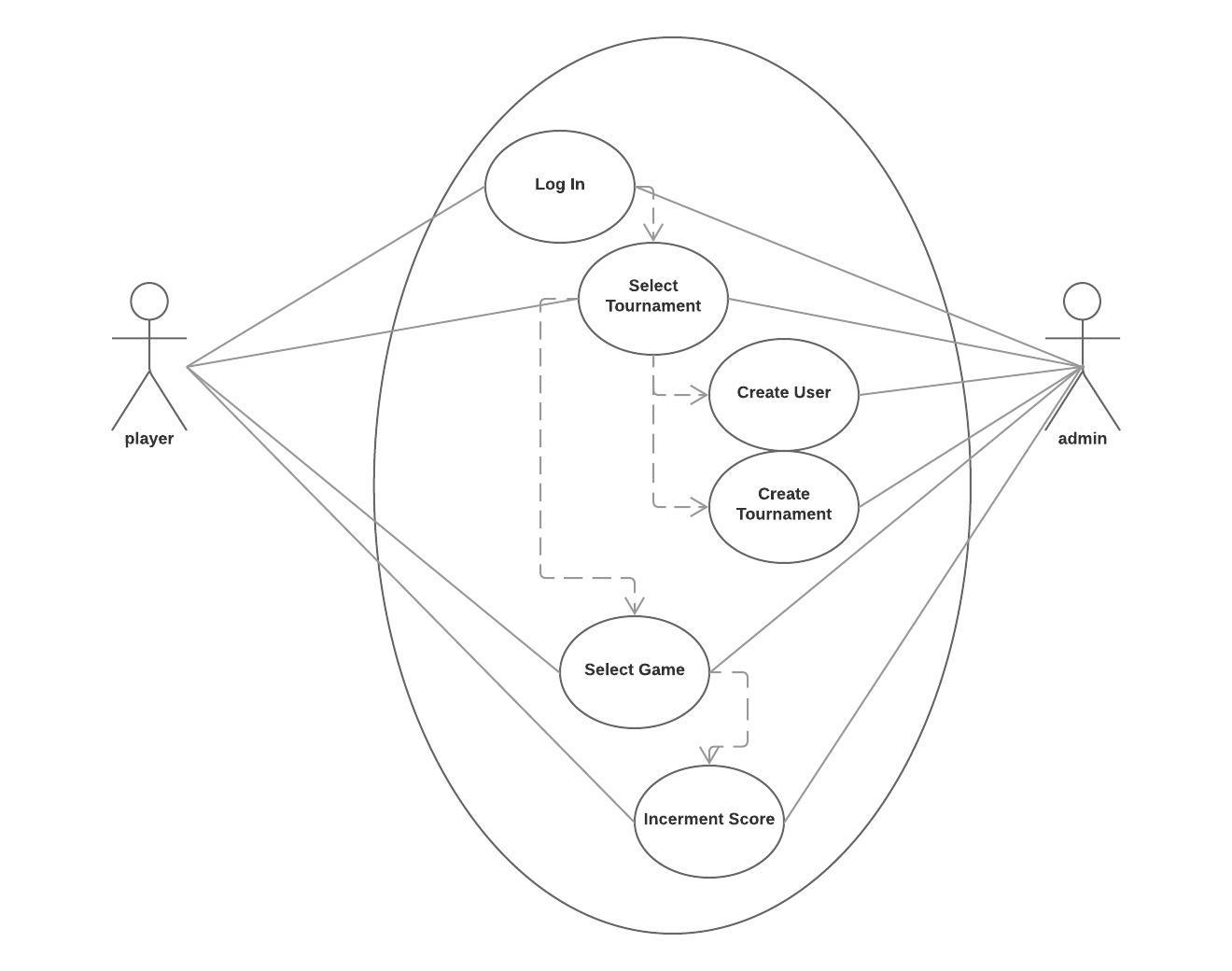
* + CRUD on player accounts
  + CRUD on tournaments: He creates the tournament and enrolls the players manually

# Non-functional Requirements

* The database is a cloud database
* Database answer less than 10 seconds
* Project done with maven
* Use 3 layer architecture

2. Use-Case Model

Use-case Diagram:



Use case: Log in

Level: user-goal level

Primary actor: user (can be player or admin)

Main success scenario: Introduce a valid email-password combination and clicking “Log In”

Extensions: Introducing anything else than a valid email-password combination results in an error message.

3. System Architectural Design

**3.1 Architectural Pattern Description**

Three-tier architecture is a client–server software architecture pattern in which the user interface (presentation), functional process logic ("business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms. It was developed by John J. Donovan in Open Environment Corporation (OEC), a tools company he founded in Cambridge, Massachusetts.

**Three-tier architecture:**

**Presentation tier**

This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing and shopping cart contents. It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network. In simple terms, it is a layer which users can access directly (such as a web page, or an operating system's GUI).

**Application tier (business logic, logic tier, or middle tier)**

The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application’s functionality by performing detailed processing.

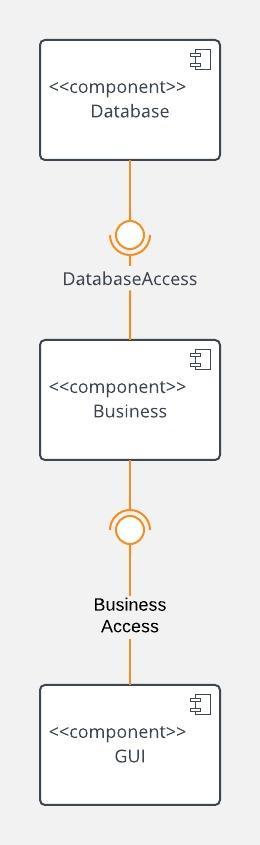
**Data tier**

The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The data access layer should provide an API to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change. As with the separation of any tier, there are costs for implementation and often costs to performance in exchange for improved scalability and maintainability.

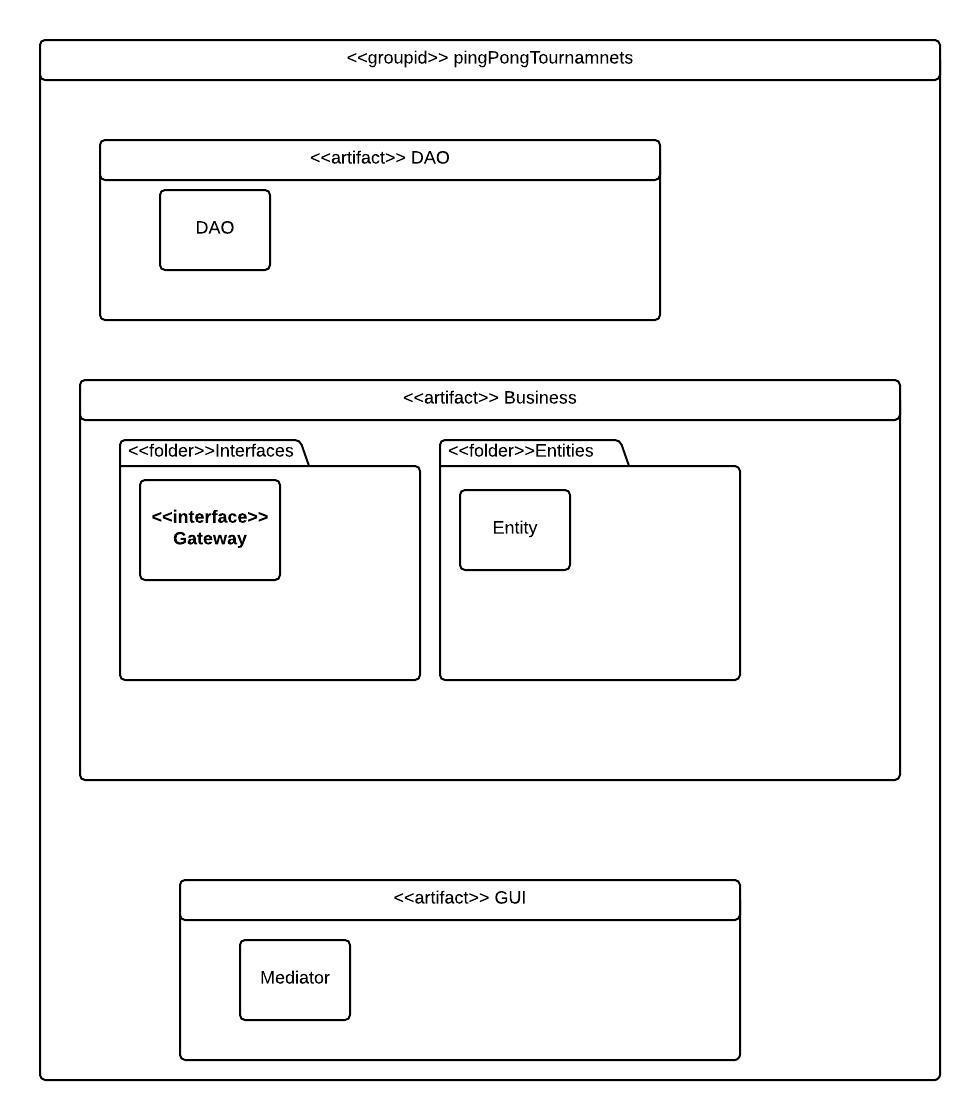
**3.2 Diagrams**

*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]*

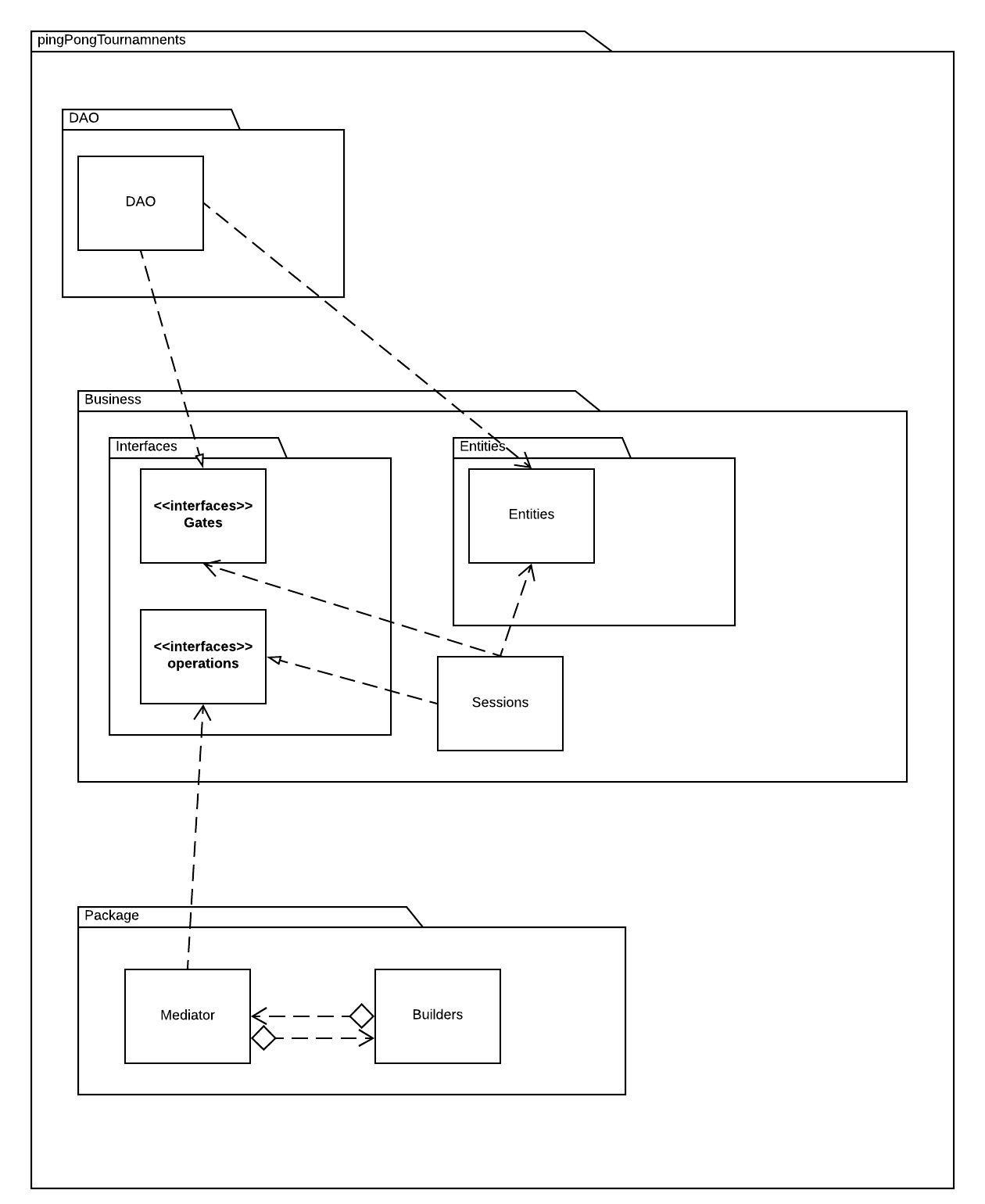
Component Diagram:



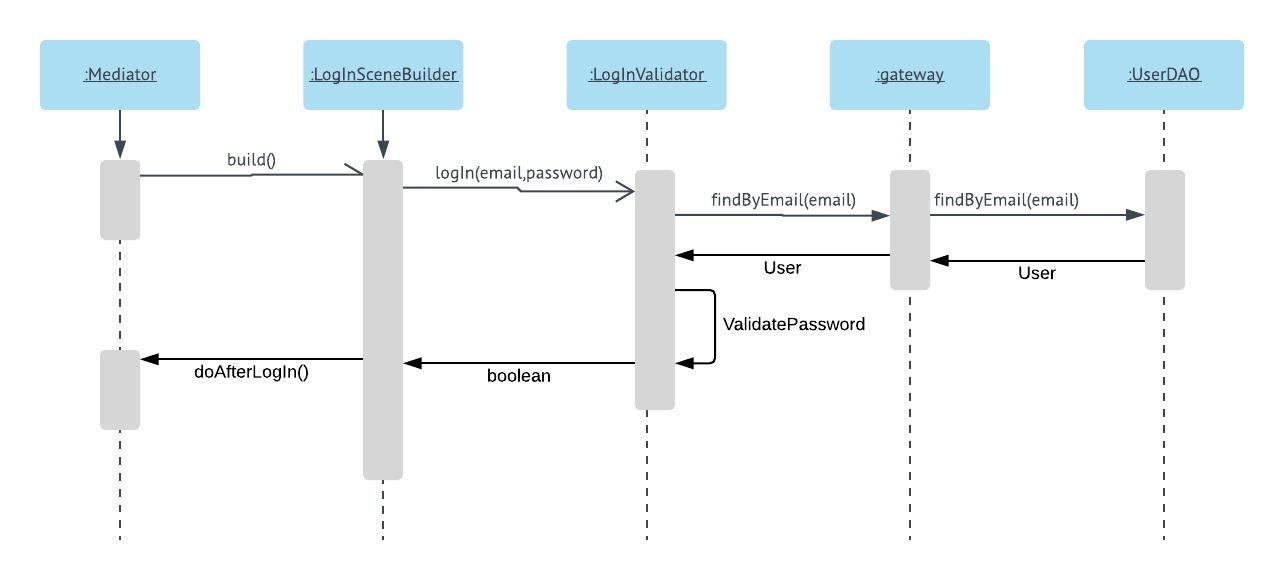
Deployment diagram:



Package diagram:



4. UML Sequence Diagrams

**

5. Class Design

**5.1 Design Patterns Description**

**Mediator pattern**

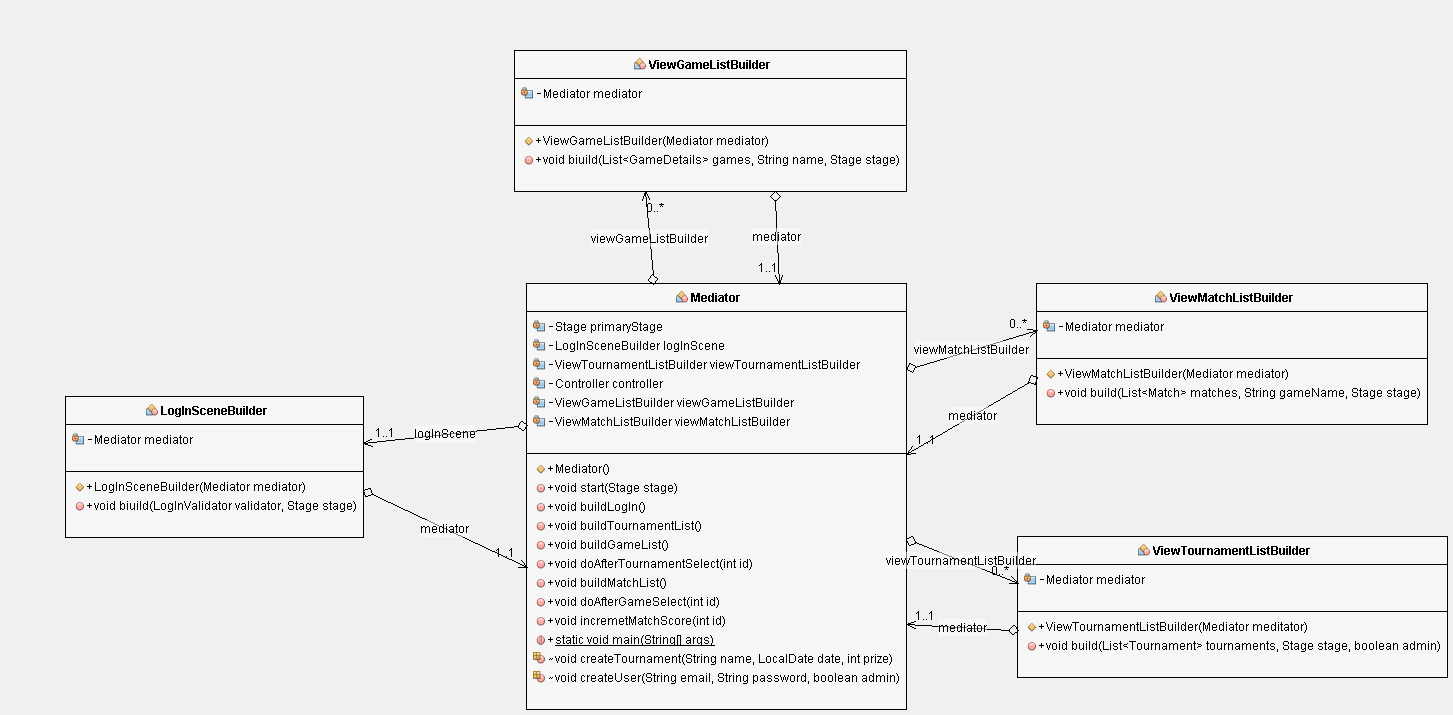
In software engineering, the mediator pattern defines an object that encapsulates how a set of objects interact. This pattern is considered to be a behavioral pattern due to the way it can alter the program's running behavior.

Usually a program is made up of a large number of classes. Logic and computation are distributed among these classes. However, as more classes are added to a program, especially during maintenance and/or refactoring, the problem of communication between these classes may become more complex. This makes the program harder to read and maintain. Furthermore, it can become difficult to change the program, since any change may affect code in several other classes.

With the mediator pattern, communication between objects is encapsulated within a mediator object. Objects no longer communicate directly with each other, but instead communicate through the mediator. This reduces the dependencies between communicating objects, thereby reducing coupling.

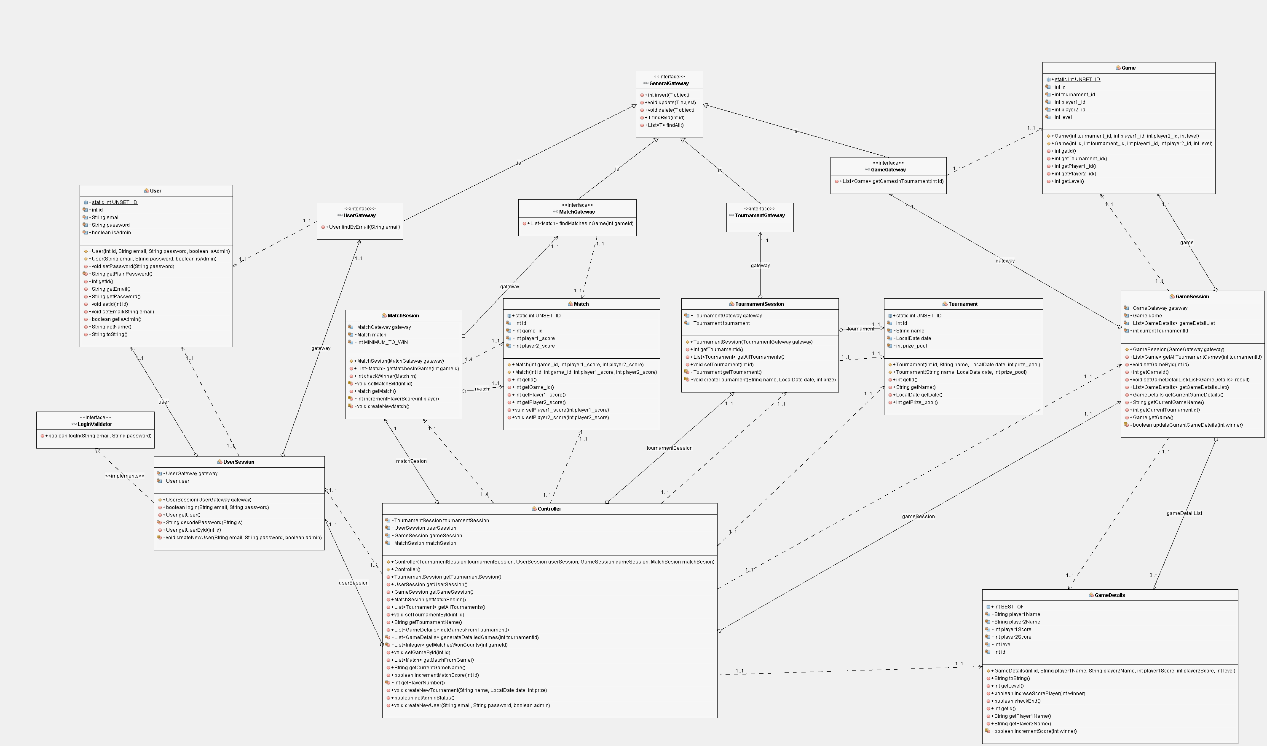
**5.2 UML Class Diagram**

GUI layer class diagram:



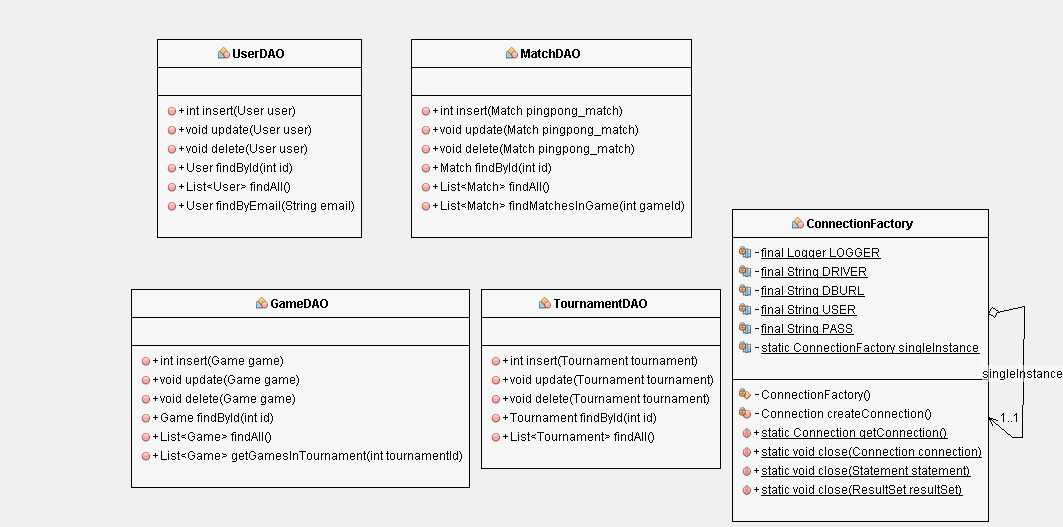
The mediator controls the flow of the events. (Picture: https://gyazo.com/5ba3ca5d89af2608f168a37d3c0910c6)

Business layer class diagram:



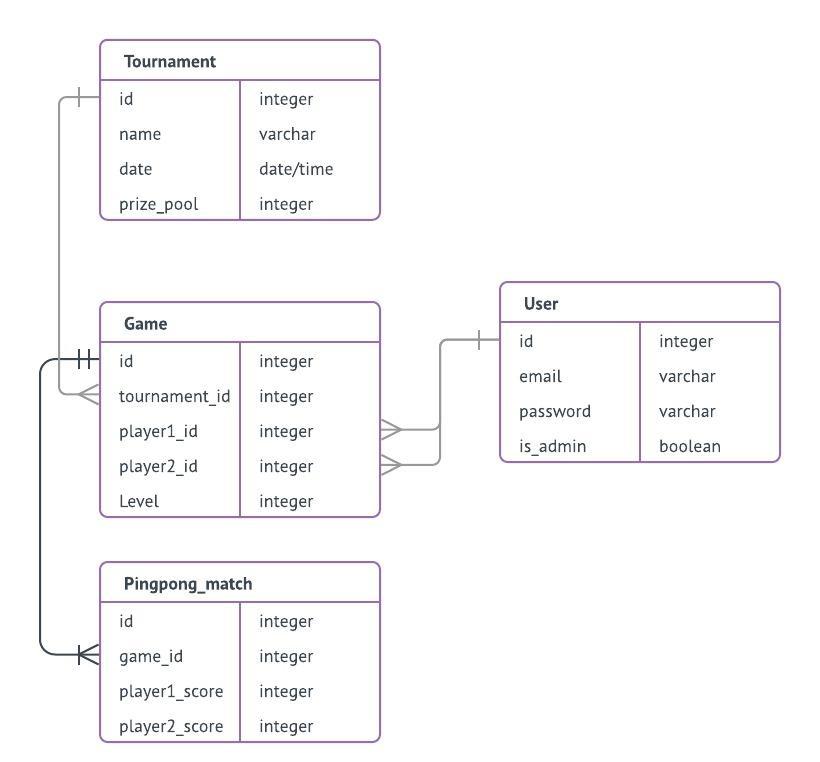
(Picture: https://gyazo.com/2d2d990533f9873c0eac1afb704abc60)

Data layer class diagram:



6. Data Model

Database diagram



7. System Testing

Sadly, all the testing was done manually. On the other hand, all the tests were successful.

Tests:

* Increment score works correctly and doesn`t update when it shouldn`t.
* Creates work correctly, but there is no check for valid email
* Scene transitions work correctly between scenes
* End match and end game are detected correctly

8. Bibliography

1. <https://en.wikipedia.org/wiki/Multitier_architecture>
2. <https://en.wikipedia.org/wiki/Mediator_pattern>
3. <https://stackoverflow.com/>
4. https://github.com/buzea/SoftwareDesign2018