Ping-pong tournament application

Student: Gyarmathy Tímea
Group: 30433

Table of Contents

1.1 Assignment Specification

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.

1.2 Functional Requirements

FR1: The application should have two types of users: a regular user represented by the player and an administrator user.
FR2: Both kinds of uses must provide an email and a password to access the application.
FR3: The regular user should be able to 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)

FR4: The administrator user can perform the following operations:

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

1.3 Non-functional Requirements

- Availability: system must be available 98% per year.
- Performance: response time for any event should be less than 2 seconds.
- Security: only system administrators should be allowed to execute CRUD operations on database data.
- Testability: system must have a greater than 90% code coverage realized by unit tests.
- Usability: system must be clear and intuitive for types of users involved: ping-pong players, system administrators; any user should be able to log in 3 clicks;

NFR1: The data will be stored in a database.
NFR2: 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.
NFR3: All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.

2. Use-Case Model

Use case: Login as Administrator
Level: User-goal level
Primary actor: System Administrator
Main success scenario:

1. User opens the app, the login window appears with possibilities to enter credentials
2. Administrator enters his e-mail
3. Administrator enters his password
4. Administrator clicks Log in button
5. System verifies credentials, sends request to database to verify if credentials exist
6. System verifies if user credentials have administrator privileges
7. System displays the Administrator Window with operations only available to the system administrator

Extensions:
In case administrator enters wrong credentials:

5. System displays a pop-up window stating, “Wrong credentials or user doesn’t exist!”
6. Return to step 2 (correct entered credentials)

In case user doesn’t have administrator privileges:

7. System opens Player Window with operations available to a registered player

Use case: Log in as Player
Level: User-goal level
Primary actor: Registered ping pong player
Main success scenario:

1. User opens the app, the login window appears with possibilities to enter credentials
2. Player enters his e-mail
3. Player enters his password
4. Player clicks Log in button
5. System verifies credentials, sends request to database to verify if credentials exist
6. System verifies if user credentials (don't) have administrator privileges
7. System displays the Player Window with operations available to every player

Extensions:
In case player enters wrong credentials:

5. System displays a pop-up window stating, "Wrong credentials or user doesn't exist!"
6. Return to step 2 (correct entered credentials)

In case user is not a player but an administrator, having system administrator privileges:

7. System opens Administrator Window with operations available only to system administrator

Use case: View Tournaments and associated matches
Level: User-goal level
Primary actor: Registered ping pong player
Main success scenario:

1. Player successfully logs in
2. System displays player view: a list of tournaments on the left, the matches on the right
3. Player clicks on one of the tournaments from the list
4. Player clicks select button
5. System displays players and their matches on the match view, in a tree-like structure

Extensions:

6. In case player is one of the players in a match, he can input the score of the match
7. Player clicks

Use case: Input final score of match
Level: Sub-function
Primary actor: Registered ping pong player
Main success scenario:

1. Player successfully logs in
2. Player successfully displays a tournament where he is one of the players of a match
3. In case player is one of the players in a match, he can input the score of that match
4. Player clicks Play button near the match
5. System registers the score and verifies if there is a winner
6. If there is a winner, system propagates winner to the next level of matches

Extensions:
There is no winner (neither of the scores is three, and sum of scores is less than 5):

5. System registers the results but does not give user feedback

Wrong user input (sum of scores is greater than 5 or nonnumeric character):

5. System displays "Wrong user input!"
6. Return to step 3.

Use case: Register a player
Level: User-goal level
Primary actor: System administrator
Main success scenario:

1. Administrator successfully logs in

2. System displays administrator view with possibilities to register/retrieve/update/delete player data on the left, and for tournaments on the right
3. Administrator inputs player's name, email, and password to the fields on the left
4. Administrator clicks "Add player" button
5. System displays confirmation window
6. System verifies input data
7. System accepts input data and writes into the database

Extensions:
In case input data is incorrect or missing:

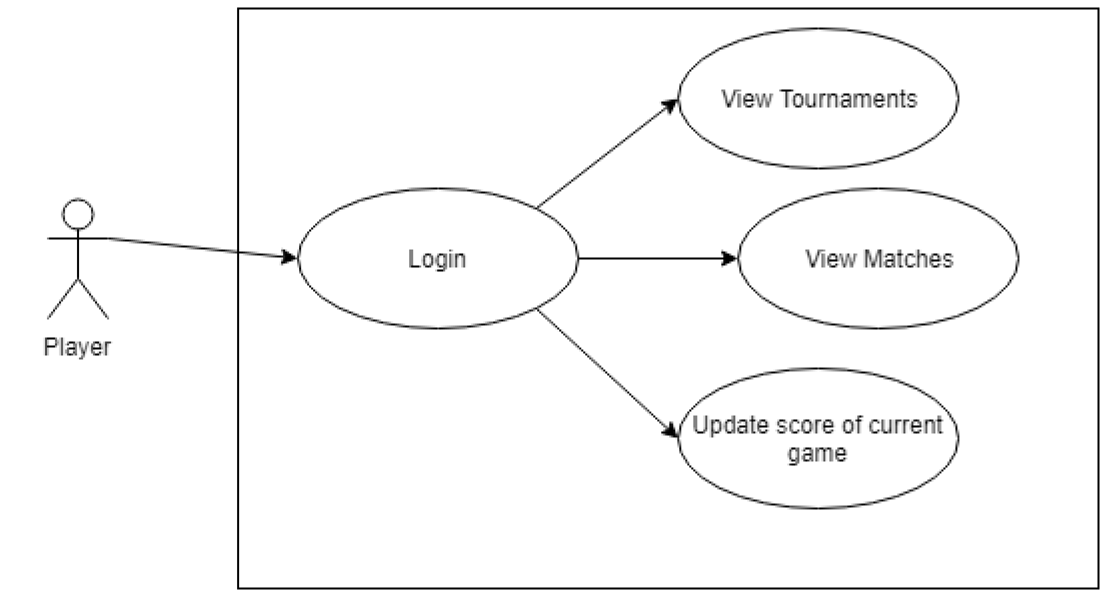
5. System displays "Wrong input data" window
6. Return to step 3

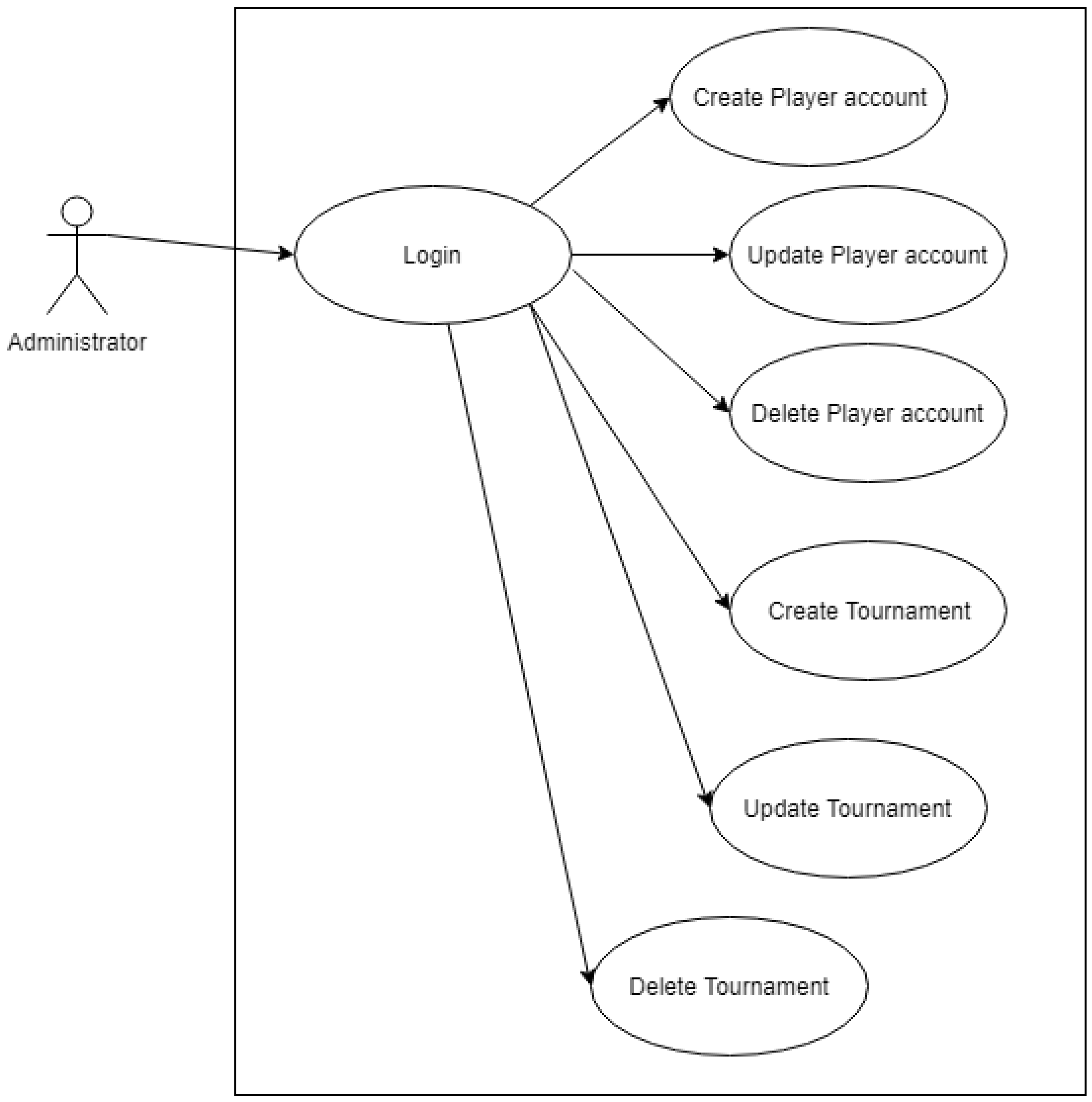
Use case: Create a new tournament
Level: User-goal level
Primary actor: System administrator
Main success scenario:

1. Administrator successfully logs in
2. System displays administrator view with possibilities to register/retrieve/update/delete player data on the left, and for tournaments on the right
3. Administrator inputs tournament name into the field on the right
4. Administrator clicks "Create tournament" button
5. System displays available registered players on the right
6. Administrator picks 8 players one by one
7. After each selected player, system removes the player from the list
8. When reached 8 selected players, system saves the data and redisplays the entire list

Extensions:
In case administrator proceeds to another operation before finishing the selection of 8 players

8. System forbids the operation, displays warning window
9. Return to step 6


Player
Login
View Tournaments
View Matches
Update score of current game


Administrator
Login
Create Player account
Update Player account
Delete Player account
Create Tournament
Update Tournament
Delete Tournament

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.

Apart from the usual advantages of modular software with well-defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. For example, a change of operating system in the presentation tier would only affect the user interface code.

Typically, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface, functional process logic that may consist of one or more separate modules running on a workstation or application server, and an RDBMS on a database server or mainframe that contains the computer data storage logic. The middle tier may be multitiered itself (in which case the overall architecture is called an "n-tier architecture").

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)**

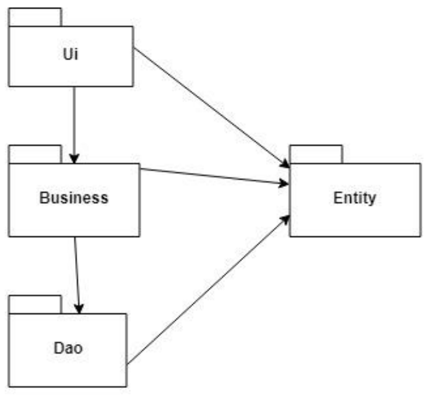
The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application’s functionality by performing detailed processing.

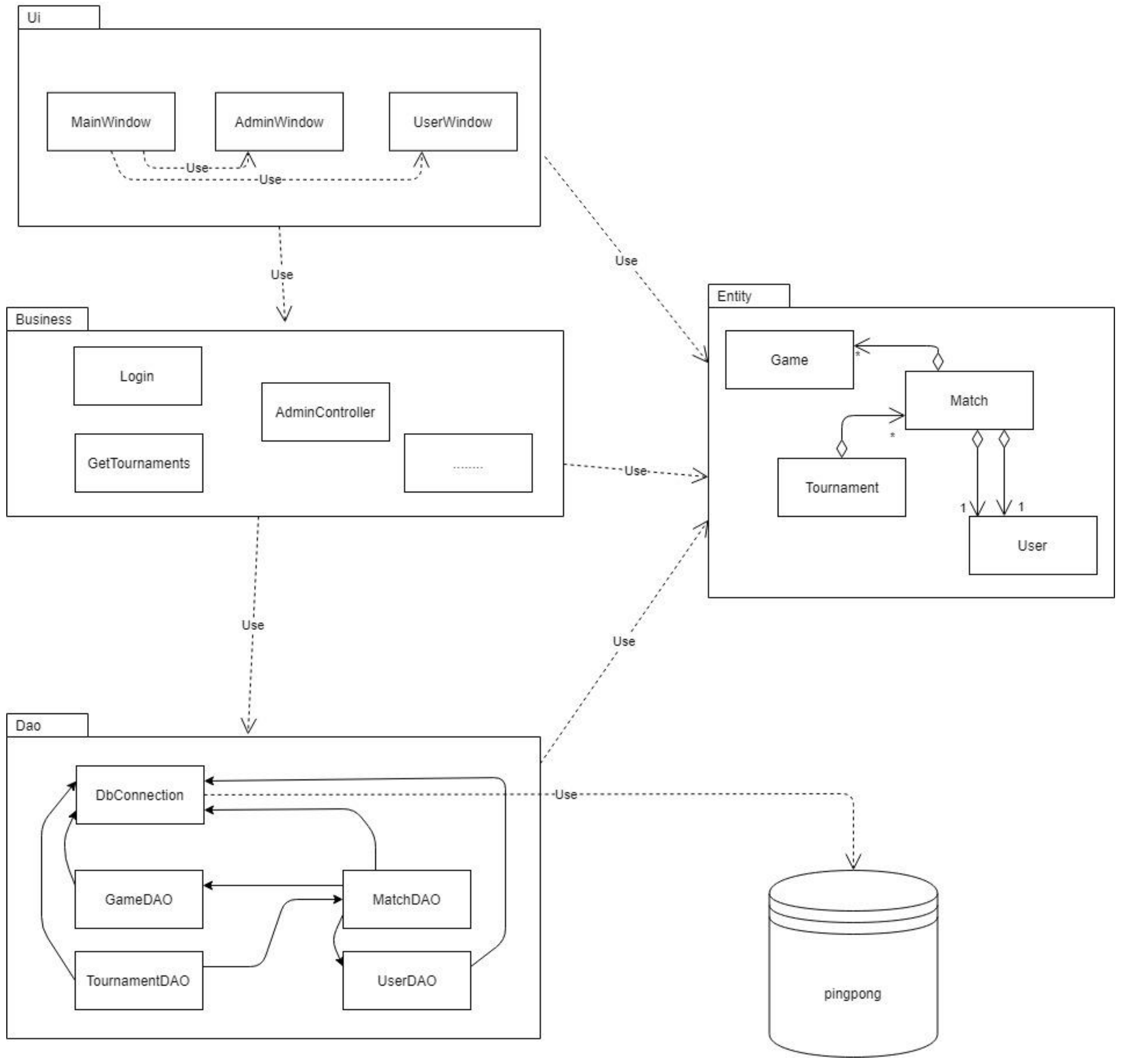
- **Data tier**

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.

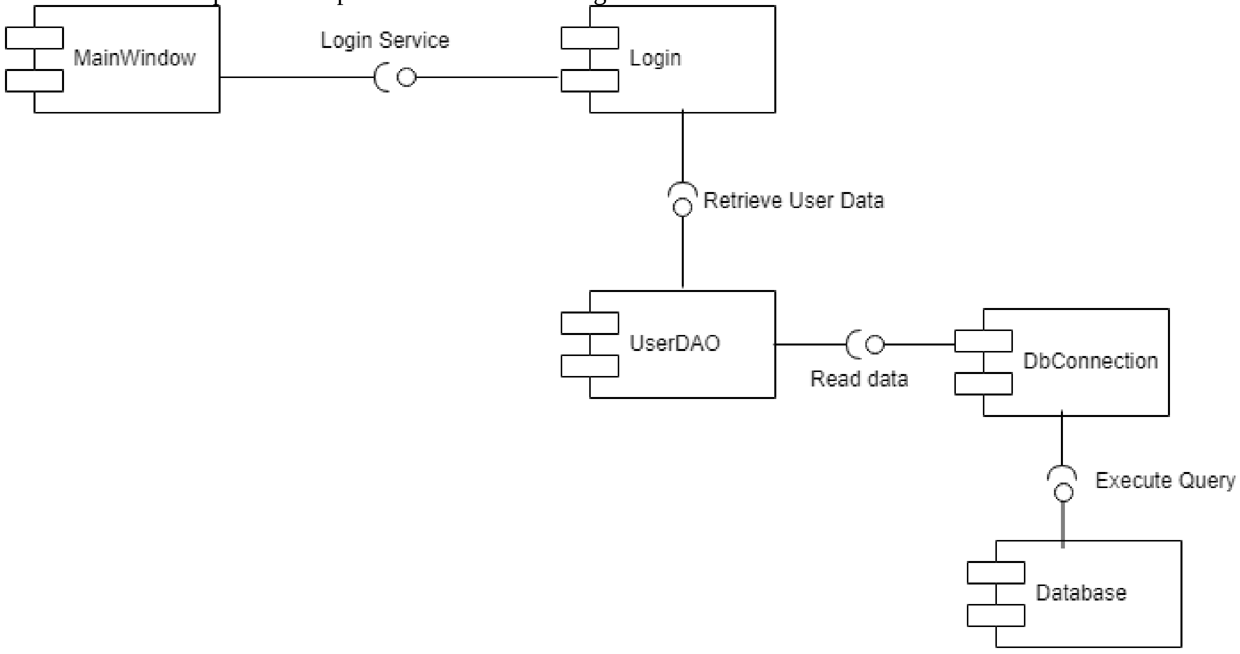
3.2 Diagrams

The next figure shows a package diagram, and a package diagram with a representation of what classes they contain. Detailed class communication is given in the chapter Class Diagram.

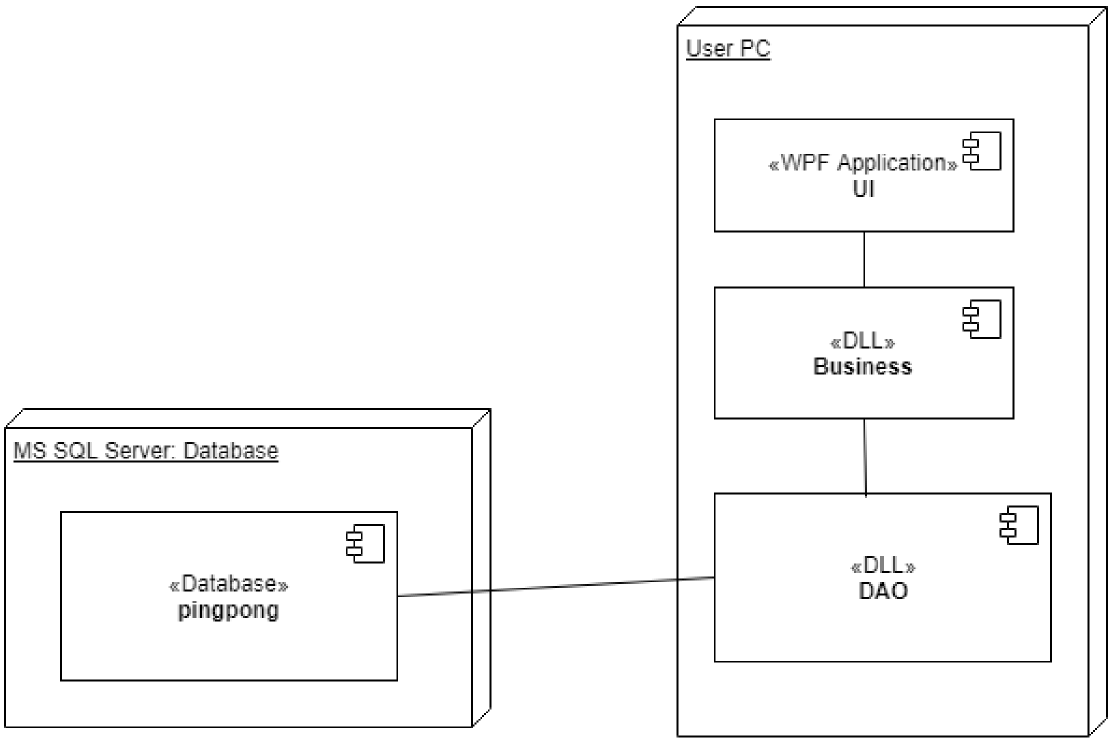

Ui
Business
Entity
Dao


Ui
MainWindow
AdminWindow
UserWindow
Use
Use
Use
Business
Login
AdminController
GetTournaments
....
Use
Use
Entity
Game
Match
Tournament
User
*
*
1
1
Use
Use
Dao
DbConnection
GameDAO
MatchDAO
TournamentDAO
UserDAO
Use
pingpong

The system would be way too complex and repetitive to display as a whole on a single component diagram so the next figure demonstrates the main idea of the interfaces between layers and how the communication takes place.
Let’s take for example the components related to the log in action.



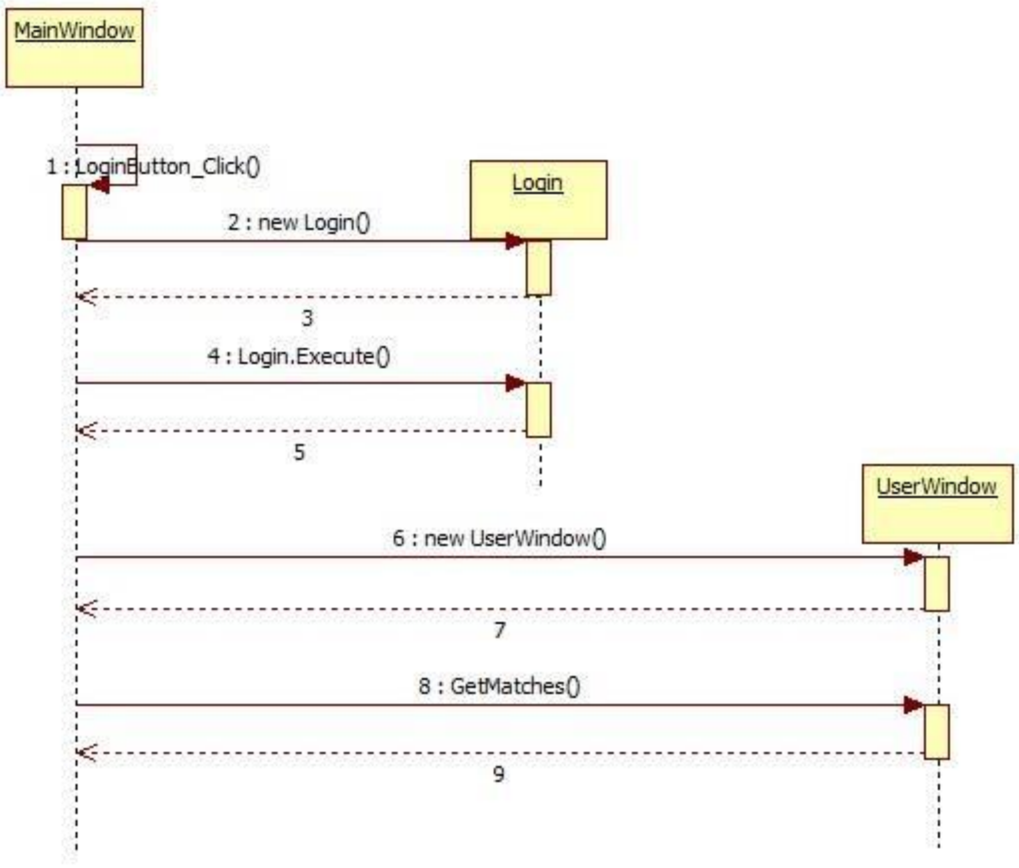
As the figure shows, we have the MainWindow component, which represents part of the user interface. This uses the interface “Log in service” provided by one of the Business layer components, in this case, Login. This latter component processes the request and by using the interface “Retrieve User Data” provided by one of the data access object components, processes it. The UserDAO accesses the Database by reading data from it, mainly by a DbConnection component. The DbConnection communicates with the Database by the “Execute Query” interface.
This is an example which can be made general on the process flow of most actions of the system: the interfaces used are provided by one of the components in the layer below.
Deployment diagram:



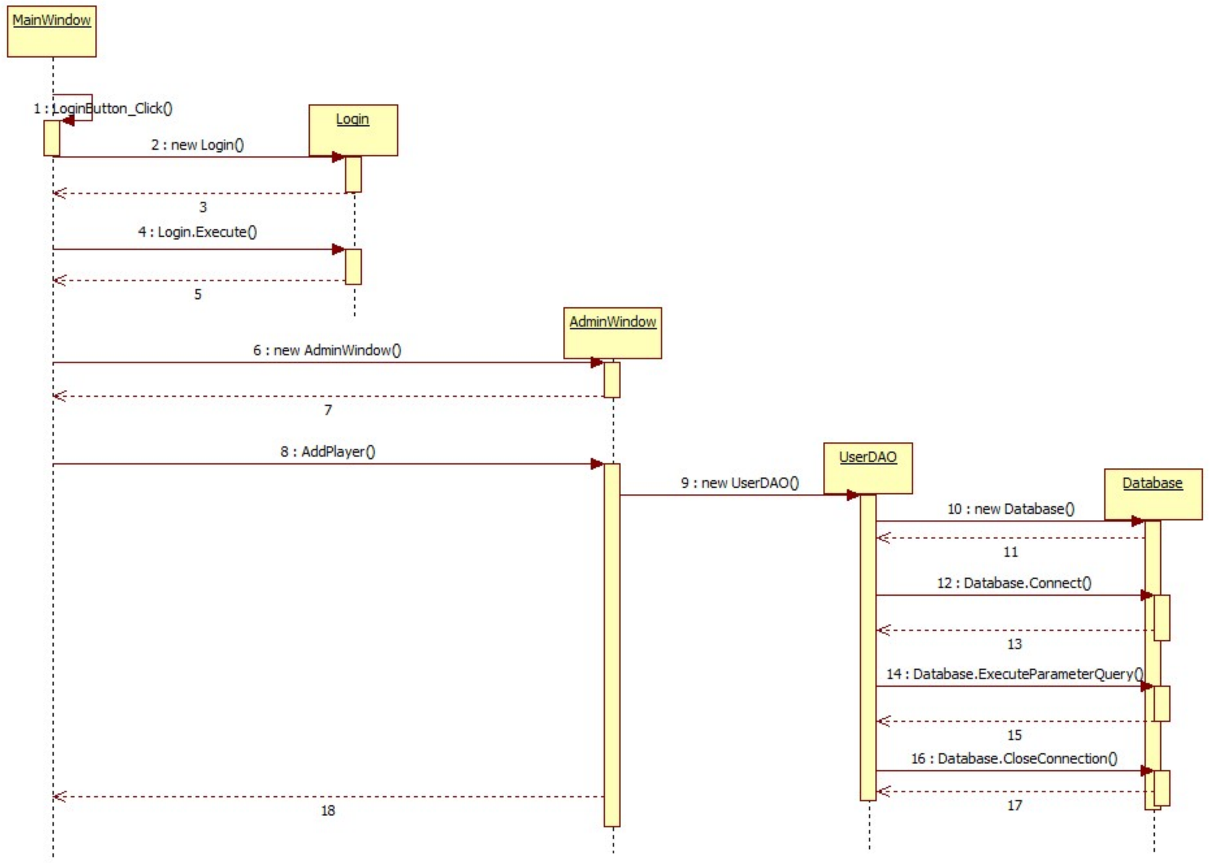
All the application except the database should be on the User's personal computer. These communicate with the served database, served by MS SQL Server. As a further improvement, this database could be hosted online.

4. UML Sequence Diagrams

User interaction sequence diagram of logging in:



Administrator interaction sequence diagram for logging in and connecting to the database:



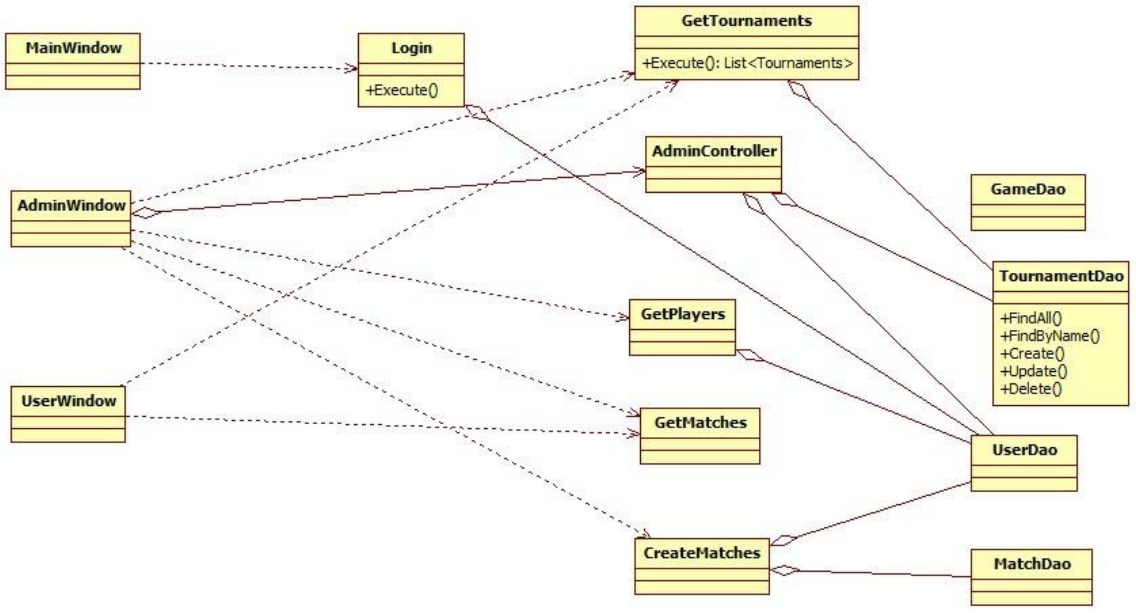
5. Class Design

5.1 Design Patterns Description

The database class implements the singleton design pattern. We need it to be a singleton in order to not have multiple instances of the database connection.

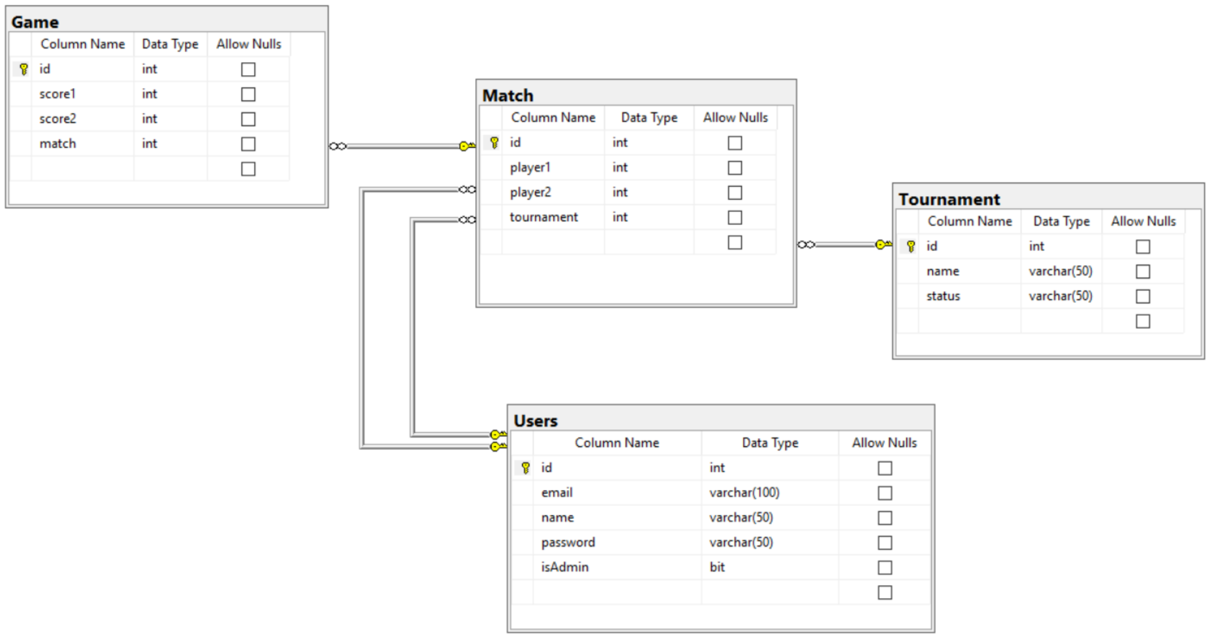
5.2 UML Class Diagram

A basic class diagram with many classes and methods missing is presented below. This is to show how the interaction takes place within the application. Classes grouped together, being in the same package can be considered of having a similar structure.



7. Data Model

The data model consists of 4 tables, conceptually the same as our Entity classes.
The database diagram looks like the following:



Game

Column Name	Data Type	Allow Nulls
id	int	☐
score1	int	☐
score2	int	☐
match	int	☐
		☐

Match

Column Name	Data Type	Allow Nulls
id	int	☐
player1	int	☐
player2	int	☐
tournament	int	☐
		☐

Tournament

Column Name	Data Type	Allow Nulls
id	int	☐
name	varchar(50)	☐
status	varchar(50)	☐
		☐

Users

Column Name	Data Type	Allow Nulls
id	int	☐
email	varchar(100)	☐
name	varchar(50)	☐
password	varchar(50)	☐
isAdmin	bit	☐
		☐

Table Users contains all the users of the systems, players and administrators as well. It contains their e-mail (with which the login is executed) a name to display
The Tournament class represents the model for the tournament and it holds the information that belongs to it, the name in this case.
The Match class is the model that represents the match and it is part of a tournament. Besides the tournament it holds information about the two players that take part in a match.
The Game table represents the game that is played. It holds information about the scores and all games belong to a match.

7. System Testing

The application code is tested via unit tests. Most methods that do database operations (DAOs) have been covered by unit tests from the beginning of development in order to have a good foundation to build the rest of the application on. These tests have been continuously ran during development to validate if changes made didn't affect the parts of the code that were already working.

7. Bibliography

1. https://en.wikipedia.org/wiki/Multitier_architecture