Student: Bogdan Rogoz
Group: 30433

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

Since the Ping-Pong Tournaments application is such a big success, the owners of the application wish to add a new feature to it: Paid Tournaments. They differ from the free tournaments available until now by the fact that they require an enrollment fee and offer a cash prize to the winner. For the moment, the player that finishes on the 1st position receives all the money from the enrollment fees of the tournament. From the beginning of the tournament until a player wins 1st place, the prize money is kept in the account of the Ping-Pong Association or in the account of the Tournament itself.

1.2 Functional Requirements

The regular user can perform the following additional operations:

- Enroll into upcoming Tournaments, by paying the enrollment fee out of their account.
- View Tournaments by category: Finished, Ongoing, Enrolled, Upcoming.
- Search Tournaments by name and by type (free / paid)

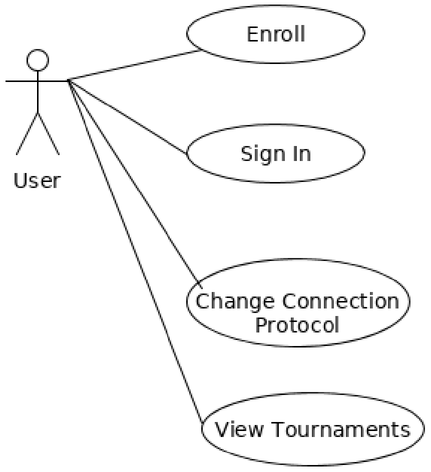
The administrator user can perform the following additional operations:

- CRUD paid tournaments.
- Add money to any player's account.
- Withdraw money from any player's account

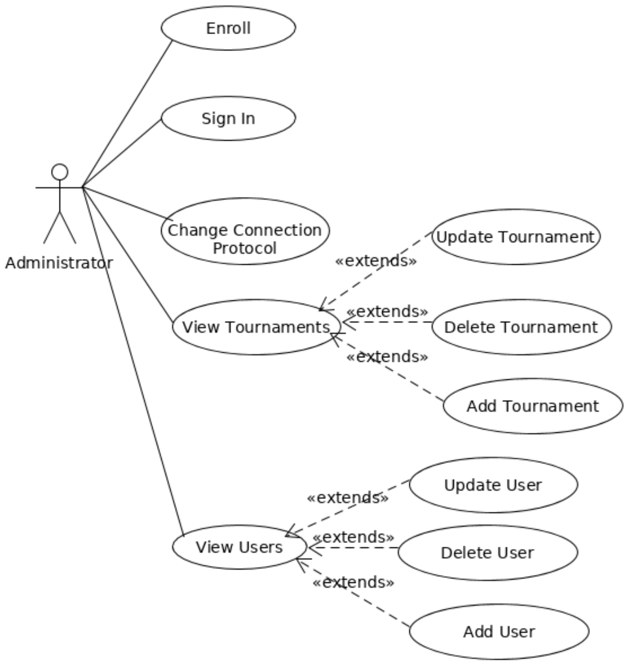
1.3 Non-functional Requirements

- Use the Model View Controller Pattern for all views.
- The Data Access Layer (DAL) will be re-implemented using an ORM framework.
- The application will use a config file from which the system administrator can set which DAL implementation will be used to read and write data to the database.
- Use the Abstract Factory design pattern to switch between the new and the old DAL implementation (implemented with JDBC or equivalent)
- All the inputs of the application will be validated against invalid data before submitting the data and saving it. (e.g. Tournament start date, Mandatory fields, The player has enough money in his account, etc.)
- Write at least one Unit Test for each new method in the business layer (e.g. Test that you can transfer money from the Player's account to the account of the Ping-Pong association).

2. Use-Case Model



Use case for User Actor



Use case for Administrator Actor

Use case : User Sign In

Lebel : user goal

Primary actor : Player

Main success scenario :

1. The user inputs his e-mail and password into the text fields
2. The user clicks “Sign in”
3. The user is redirected to a screen where all the tournaments are displayed

Extensions :

1. The user inputs his e-mail and password into the text fields
2. The user clicks “Sign in”
3. Under the “Sign in” button, a message appears, saying that the e-mail and password combination do not match

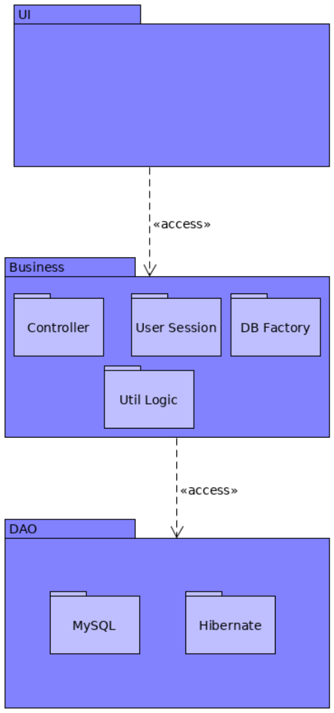
3. System Architectural Design

3.1 Architectural Pattern Description

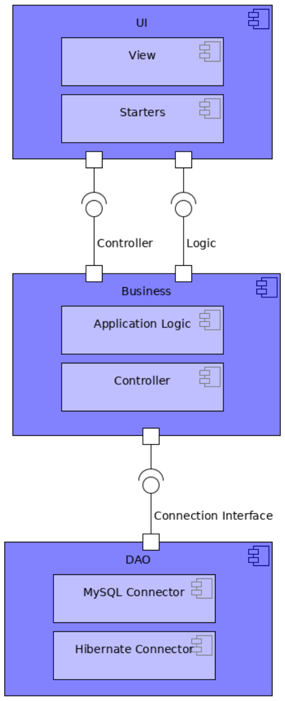
The architectural patterns used for the implementation are:

- Model – View – Controller : it is made of 3 main components:
    - Model : observable objects that hold the data represented by the View component. If the model changes, the View is updated, an vice-versa.
    - View : the visual part of the application. It contains stages, scenes, text fields etc.
    - Controller : handles user input and sets the properties of the visual components.
- 3 – Tier : used for separating other logical aspects of the application:
    - DAO : contains classes for database access and management
    - Business : contains application - specific logic and uses functionalities provided by the DAO layer
    - UI : contains the classes that form the visual aspect of the application; it makes use of the functionalities provided by the Business layer

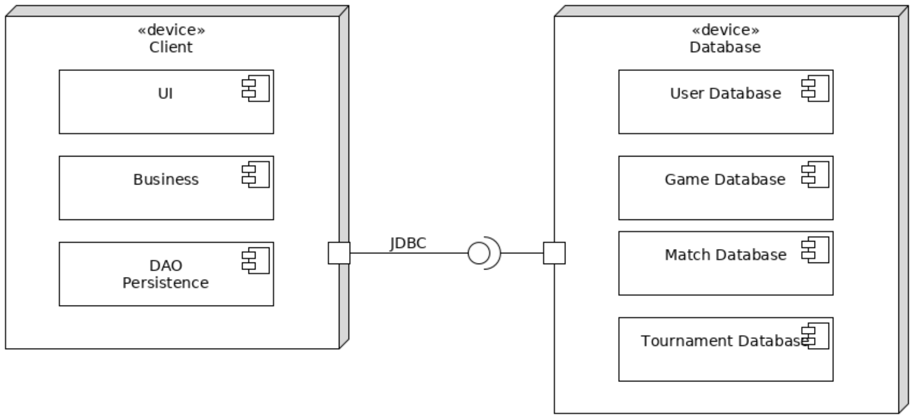
3.2 Diagrams



Package diagram

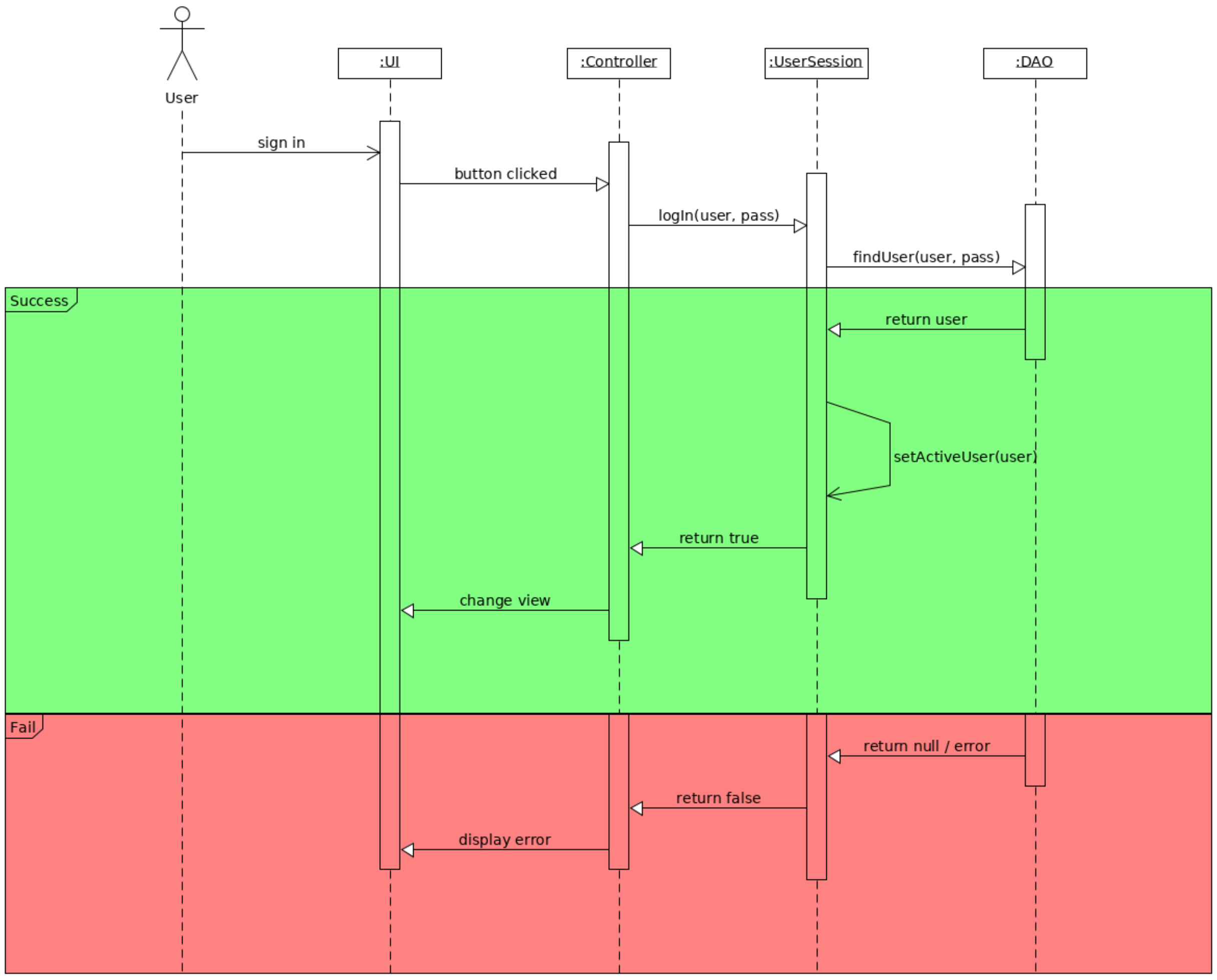


Component diagram



Deployment diagram

4. UML Sequence Diagrams



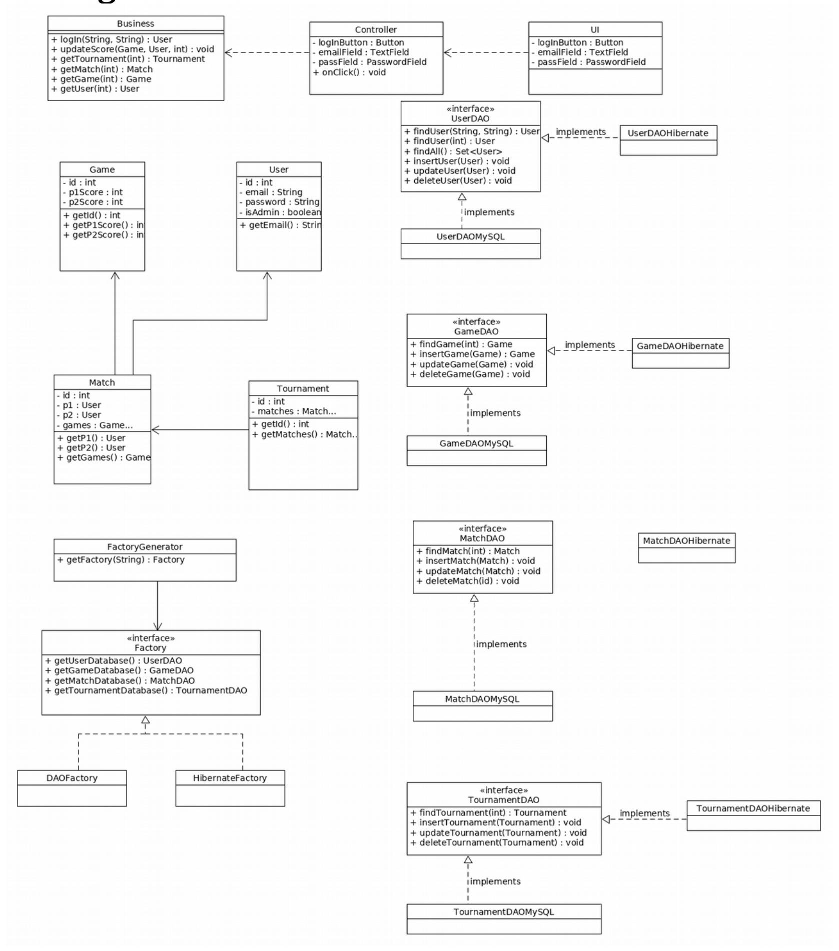
5. Class Design

5.1 Design Patterns Description

The design patterns used in this project are:

- Abstract factory : given an abstract type, the abstract factory generates instances of its subclasses, depending on the given parameter. It is used for interchanging the database communications methods (MySQL connector vs. Hibernate)
- Singleton : only a single instance of an object is allowed to exist. In order to achieve this, the constructor must be private and the instance must be retrieved through a *getInstance()* method. It is used for assuring that there aren't multiple instances of the MySQL Driver, otherwise undeterministic behaviour may occur.
- Iterator : when processing collections of data (eg. searching for an element inside a list, setting certain properties for the elements inside a set)

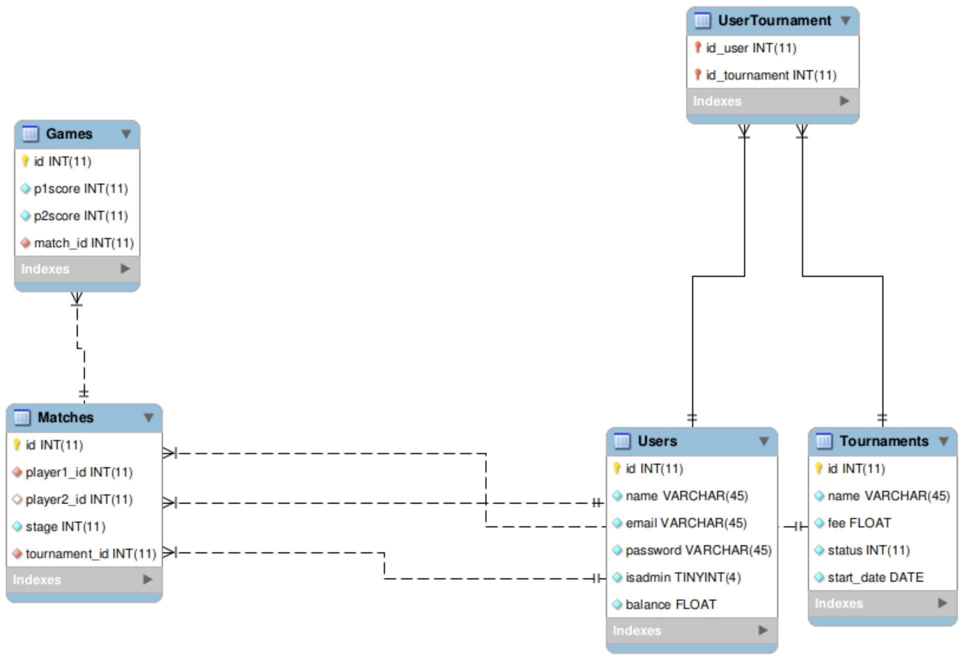
5.2 UML Class Diagram



6. Data Model

The Data models used in the application are:

- User / UserDTO : represents the ones who use the application; can be either a player (regular user) or admin. UserDTO is the MVC wrapper for the User class.
- Game / GameDTO: represents a game from a set. GameDTO is the MVC wrapper for the Game class.
- Match / MatchDTO : contains information about the games played by 2 players. MatchDTO is the MVC wrapper for the Match class.
- Tournament / TournamentDTO : contains multiple matches. TournamentDTO is the MVC wrapper for the Tournament class.



7. System Testing

Multiple unit tests can be employed, in order to test the system's sanity:

- connect to the database and check status
- check if the "match ended" logic is correct, for a couple of test inputs
- check if invalid input data is correctly processed by the application

8. Bibliography

https://www.journaldev.com/3793/hibernate-tutorial
https://sourcemaking.com/design_patterns/abstract_factory