2

STUDENT: ȘINCA MĂDALINA
Group: 30433

Table of Contents

1. Requirements Analysis

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. Paid tournaments are available for users to participate in. They require an enrollment fee and offer a cash prize to the winner. From the beginning of the tournament until a player winds 1st place, the prize money is kept in the account of the Tournament.

1.2 Functional Requirements

The application should have two types of users: a regular user represented by the player and an administrator user. Both kinds of users must provide an email address 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)
- 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 operations:

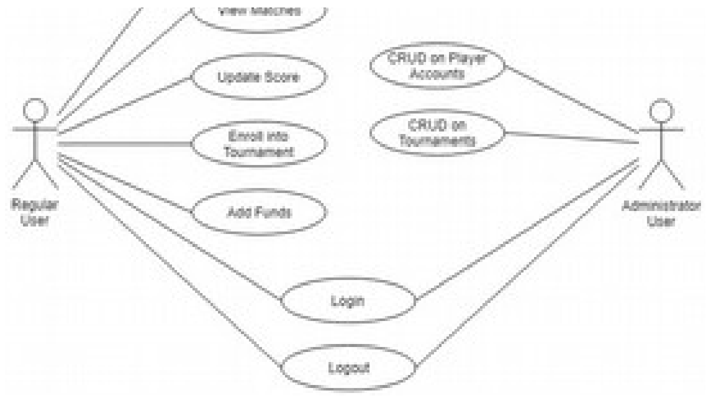
- CRUD on player accounts
- CRUD on tournaments: He creates the tournaments.
- Add money to any player's account
- Withdraw money from any player's account

Every new upcoming tournament must be created with at least one month before its start date and must have an enrollment fee specified at creation time.

1.3 Non-functional Requirements

- The application will be written in Java.
- The data will be stored in a database. The Layers architectural pattern will be used to organize the application.
- All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.
- The Model View Controller Pattern will be used for all the 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)
- 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: View Tournaments

Level: summary level

Primary actor: regular user

Main success scenario:

- The user successfully logs in
- The user chooses to view tournaments
- The application displays a list of tournaments to the user

Extensions:

- There are no available tournaments, so the application displays a message stating so.

3. System Architectural Design

3.1 Architectural Pattern Description

The application is to be designed and implemented using the MVC architecture. The layers of application will consist of the visualization (user interface) layer, the business layer, where the logic is implemented and the data access layer, which will realize the connection to the database. It can be observed that higher layers only communicate with the layers directly below them. The layers do not communicate with higher layers.

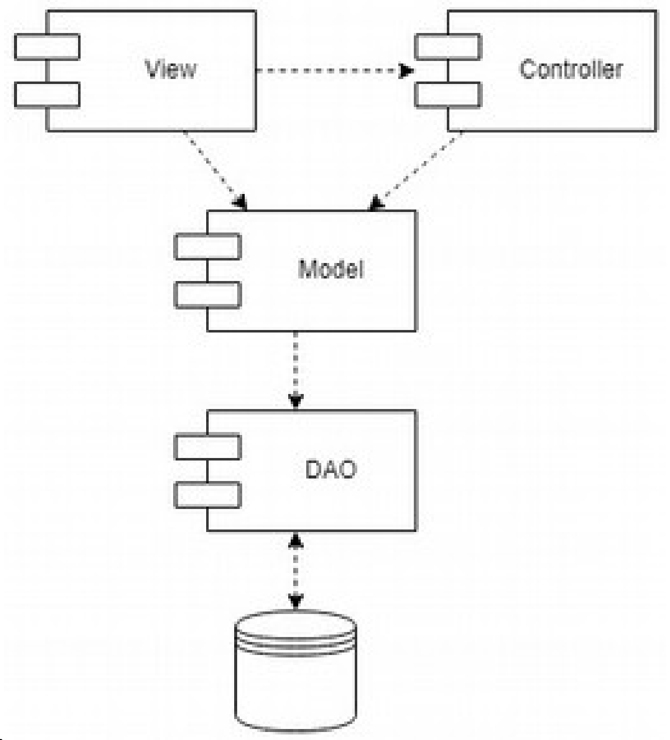
The presentation layer is the topmost level of the application. The tier displays information related to the services provided by the application. It realizes the interface between the user of the system and the application.

The business layer realizes the main logic and control of the system. It is also responsible of validating data.

The data access layer stores and retrieves information from a database of the system. The information is passed to the business layer which will perform the processing of it.

3.2 Diagrams

The system is designed around the model-view-controller architecture. This architectural pattern physically separates the view, model and controller layers. This is combines with a layered architecture, where the business layer incorporates the model, and the presentation layer contains the controller and view. The upper layers usually access only the layer directly below them. The controller handles the input from the view and updates the model. The model contains the business logic and updates the view. The conceptual architecture is presented in Figure 2

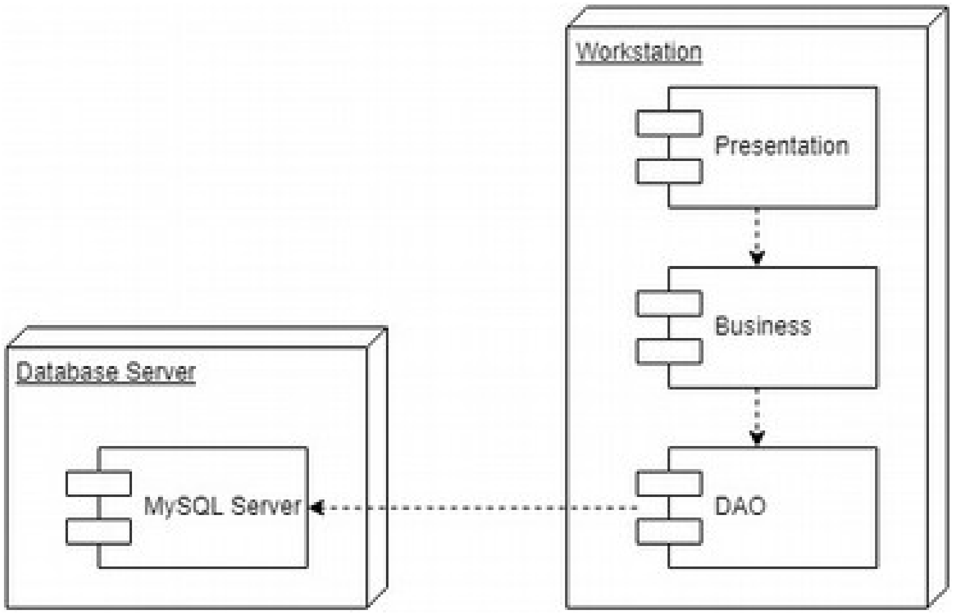


.

FIGURE 1 CONCEPTUAL ARCHITECTURE

The system will run on a single machine, while the database connection can be on a different system altogether, the connection being realized with sockets. This aspect is presented in the deployment diagram in Figure 3.

FIGURE 2 CONCEPTUAL ARCHITECTURE
FIGURE 3 DEPLOYMENT DIAGRAM



4. UML Sequence Diagrams

The sequence diagram is presented for the View Tournament scenario in Figure 4.

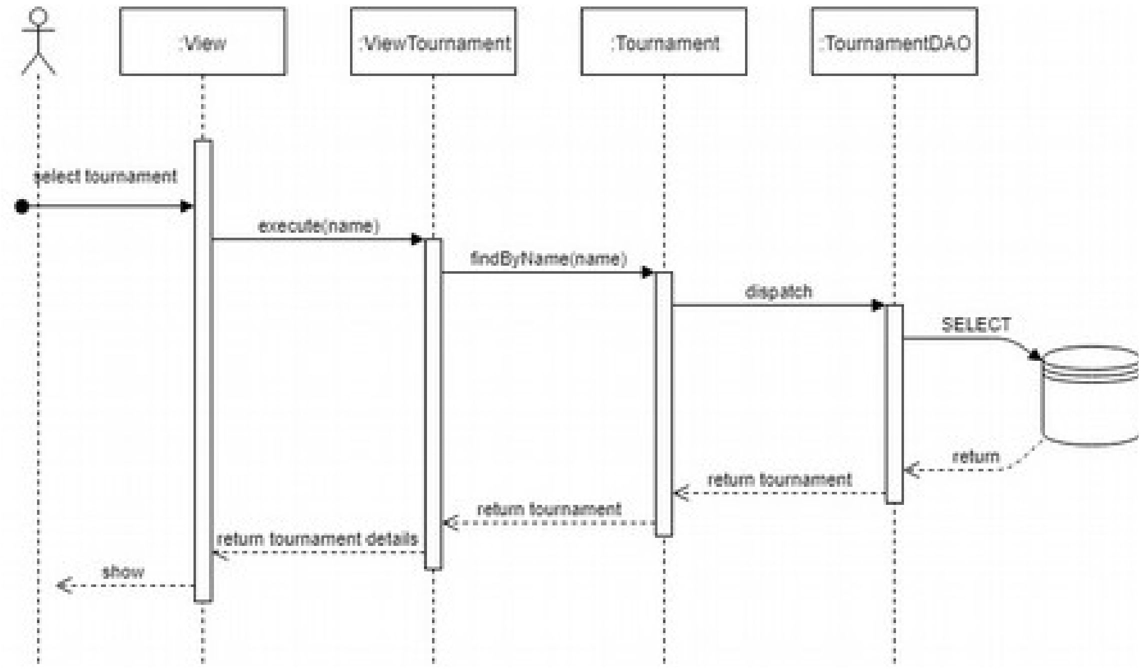


FIGURE 4 SEQUENCE DIAGRAM

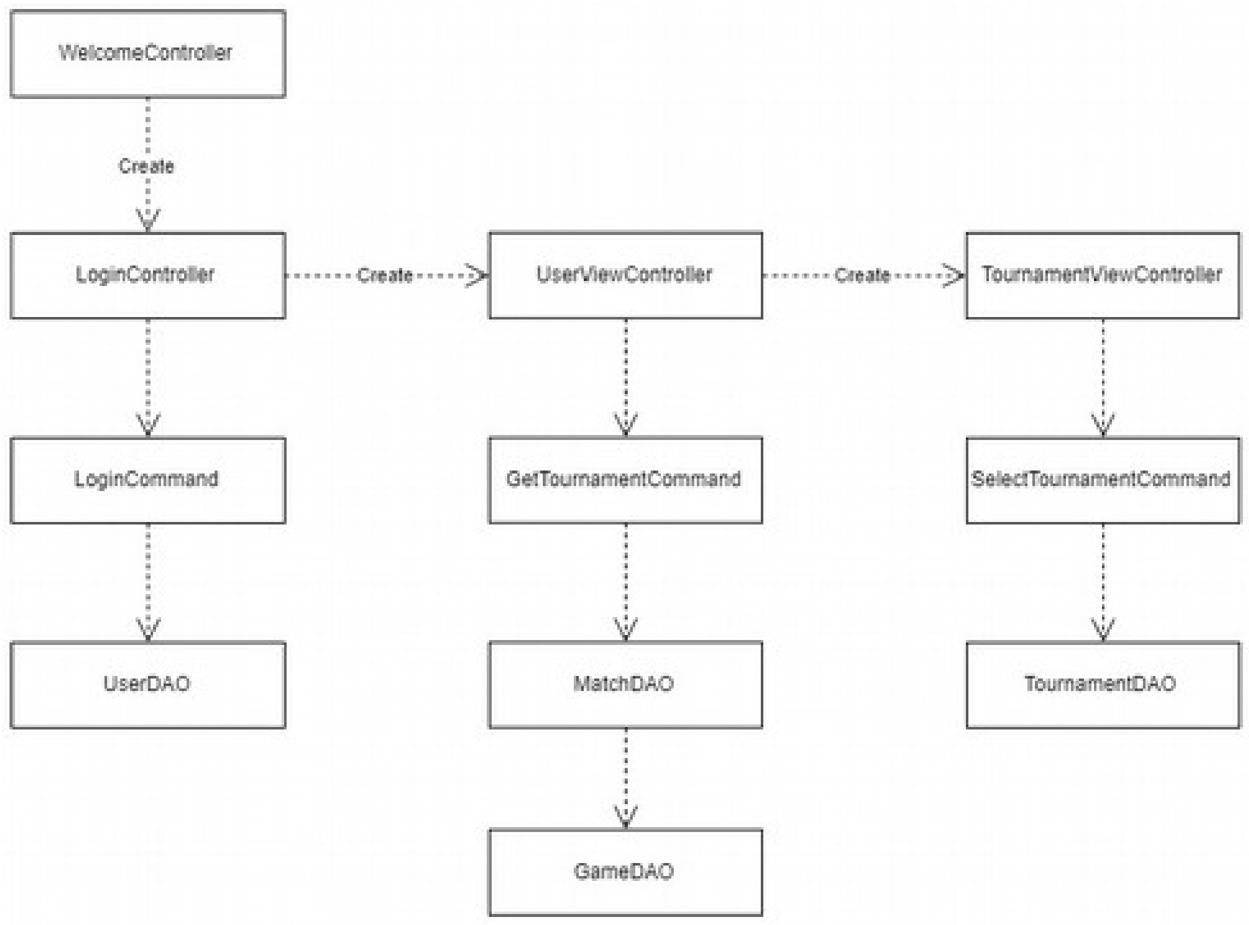
5. Class Design

5.1 Design Patterns Description

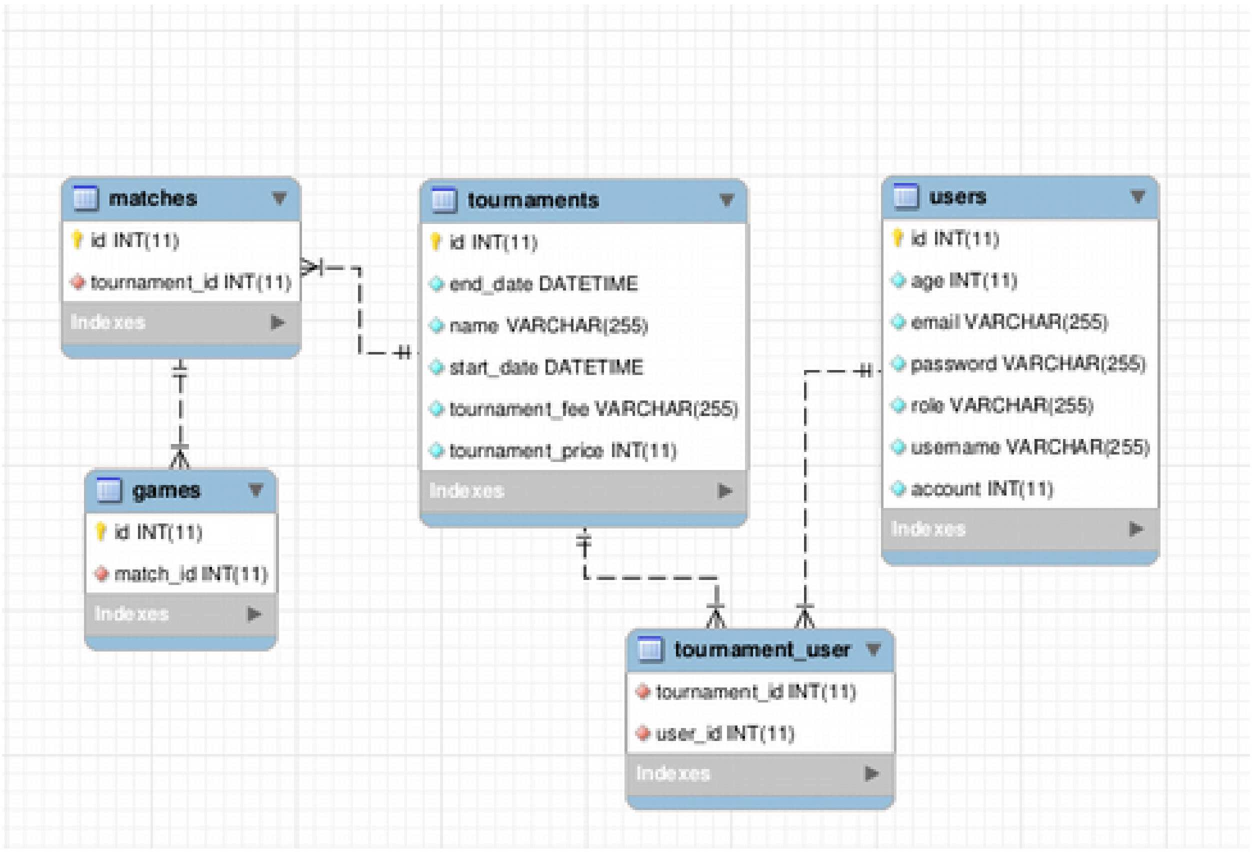
The singleton creational pattern is used to model the connection to the database. The connection is created when the system accesses the database for the first time. The connection is closed when the application is terminated. The connection is created with a Factory creational pattern via the ConnectionFactory.

A version of the command pattern is used to model the passing of information from the UI level to the business, and to the DAO layer. The commands encapsulate information about the request and the UI elements will create command objects that will perform the necessary business and data access actions.

5.2 UML Class Diagram



6. Data Model



7. System Testing

No unit tests have been implemented.

8. Bibliography

Baeldung.com/

Mkyong.com/