Student: Zavaczki Péter - Tibor

**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

This application is a database manager application, expected to be used by a table tennis association, to ease the creation and management of the tournaments organized by them.

# Functional Requirements

The application differentiates two types of users: regular users and admins (or power users), differentiated by their admin field in the ‘user’ table in the database, but both need an email address and a password to log in.

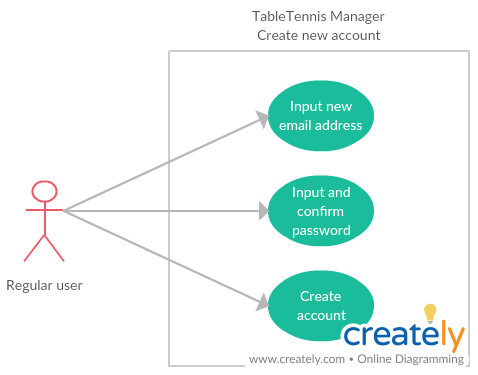
The regular user can choose to show all the data in the table or only a select row and can only access the tournaments, matches and games tables.

The admins can use every command (show all, show one, insert, delete, edit) and can access every table (user, tournament, match, game).

# Non-functional Requirements

* The application will be written in Java, with the UI written with JavaFX, using an MVC architectural model.
* The application will be layered, having a data access layer, a business layer, and a ui layer.
* The data to be managed by the application will be stored in a database.

2. Use-Case Model



Use case: Create New Account

Level: summary level

Primary actor: regular user

Main success scenario:

* The user clicks the “Create new account” button on the login screen
* The user inputs an email address, which is inexistent in the database so far
* The user inputs a password and repeats it in the next field
* The new account is successfully created with the login data being the email address and password. The created account is not an admin account.

Extensions:

* Email address format checking (<email\_username>@<domain>)
* Password constraint enforcing (longer than a given value, containing specific character types)

3. System Architectural Design

**3.1 Architectural Pattern Description**

The application will be designed and implemented using the three tier architecture. The layers are the UI (presentation) layer, the business (logic) layer and the DAO (data access) layer. The described layered architecture is set in a decreasing dependency order, meaning that the higher level layers have access to the lower level layers but not vice-versa.

The UI layer is on the top of the layer hierarchy. In this layer is the implementation of the graphical user interface, giving users a friendly environment to interact with the application.

The business layer contains the logic and control of the application, such as transmitting values from the DAO layer to the UI layer or validating the login password.

The DAO layer is responsible with communication the database, sending and retrieving information from it.

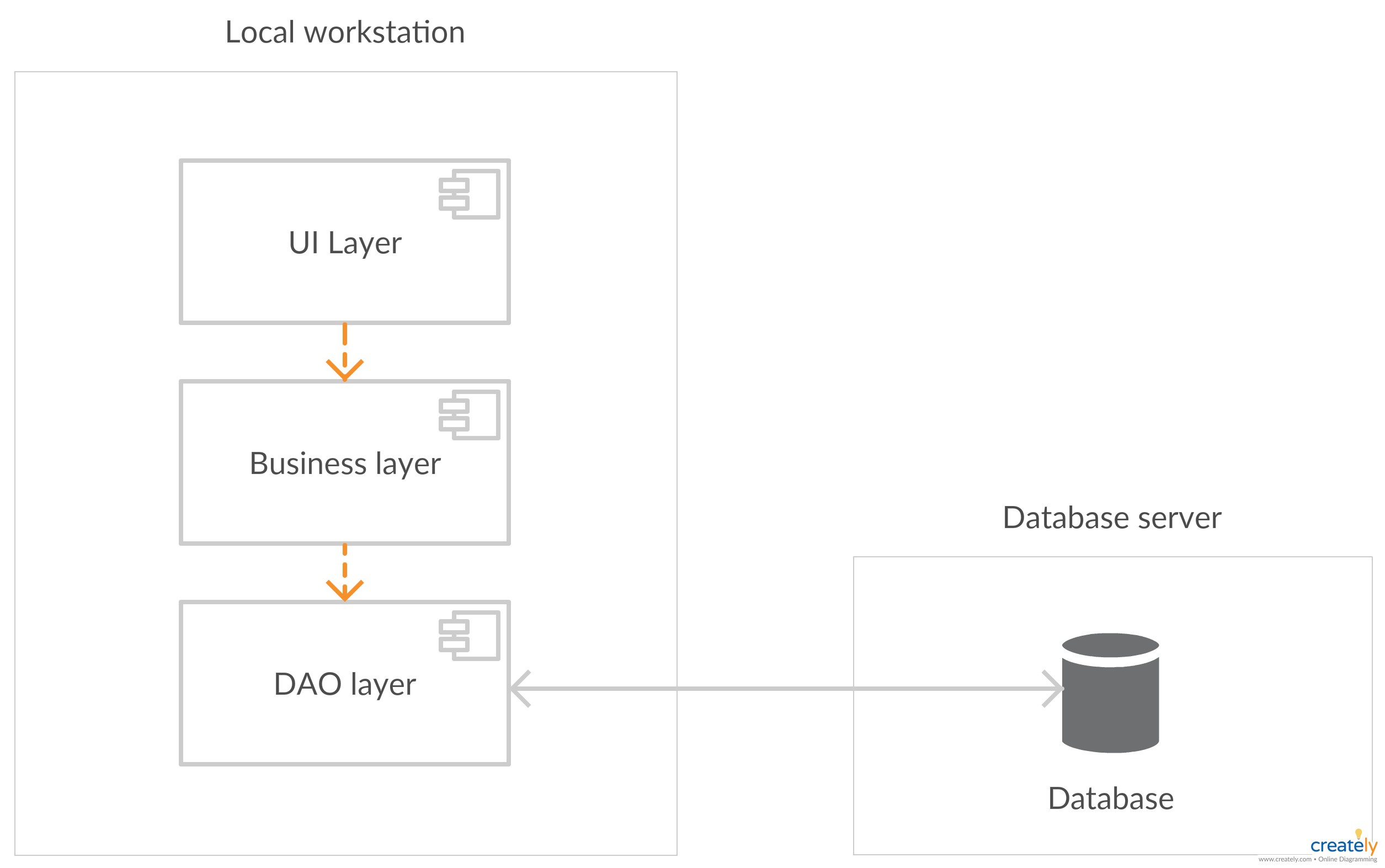
The UI layer is based on an MVC architectural pattern. The Model being defined in the data access layer, being accessed by the UI using dependency relations, the View is composed of fxml files, which are stored in the resources folder and the Controller is composed of Java classes defining the methods used on button presses and such.

Hibernate is a high-performance Object/Relational persistence and query service, which is licensed under the open source GNU Lesser General Public License (LGPL) and is free to download. Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities. This tutorial will teach you how to use Hibernate to develop your database based web applications in simple and easy steps.

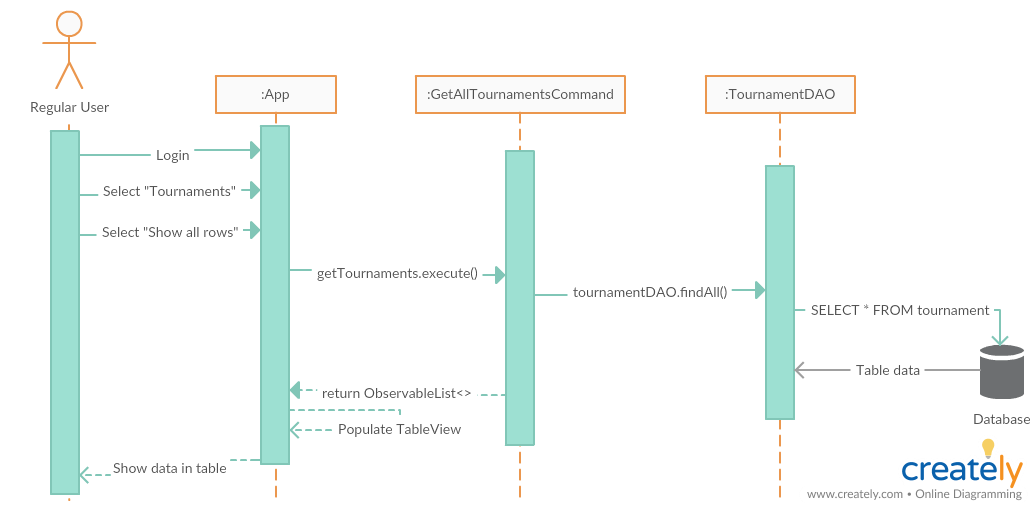
**3.2 Diagrams**

The application is built according to a three tier layered architecture, this pattern separates the UI, business and DAO layers. The upper layers have access to the lower one, but this is not true vice-versa. Additionally, the UI layer uses the MVC architectural pattern.

The layers form the whole of the application, thus they run locally, while the database may be deployed on a different system or on a network.



4. UML Sequence Diagrams



The above diagram is a sequence diagram representing the “Show all rows” action on the Tournament table.

5. Class Design

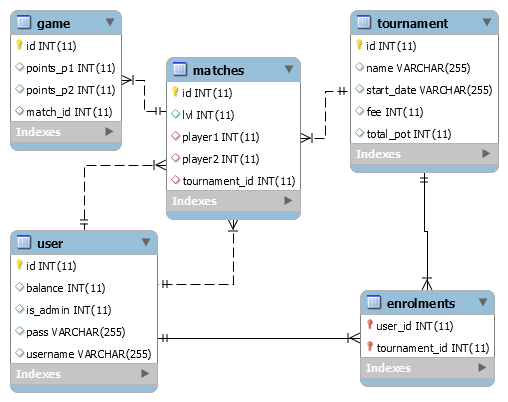
**5.1 Design Patterns Description**

The command pattern is used to process the input data from the UI, such as filtering the tournaments’ list according to the user’s preference.

**5.2 UML Class Diagram**

*TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO*

6. Data Model

**

7. System Testing

No tests are implemented at the creation of this document.

8. Bibliography

<https://github.com/buzea/SoftwareDesign2018>

<https://martinfowler.com/eaaCatalog/transactionScript.html>

<https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBzfXLWLSYVy8EbTdpGbUIG>