Assignment 1

Student: Bucur Diana

**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

## 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.

# 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.

## 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

# Non-functional Requirements

## Availability

The system is available any time there is a competition and it can be used 5 days a week, Sundays being reserved for maintenance. The system shall be operational 24 hours in the working days.

## Performance

The information is refreshed at regular intervals depending upon whether some updates have occurred or not. The system shall respond to the member in not less than two-five seconds from the time of the request submittal. The system shall be allowed to take more time when doing large processing jobs.

Responses to view information shall take no longer than a couple of seconds to appear on the screen.

## Security

Depending upon the category of user the access rights are decided. It means if the user is an administrator then he is able execute CRUD operations of any kind. All the other users, meaning the players can only retrieve some of the information from the database that concern them and their account and update the score for the match they are playing at the time.

## Testability

Junit tests were applied in different scenarios and for different functionalities in order to ensure the good working of the program and the accuracy of the information provided to the user.

## Usability

When running the program, on the screen will appear a window just like the one presented below which represents the log in. The interface is user friendly, the functionalities of buttons and other components are self explanatory, so no special training is required.

2. Use-Case Model

Use case 1: User log in

Level: user-goal level

Primary actor: Admin/Player actors

Main success scenario:

- user enters his/her e-mail

- user enters the password

- the data entered is correct, successfully found in the database and the user can use the application

Extensions: in the scenario presented above, some possible failures would be:

- the user is not registered and therefore the e-mail cannot be found in the database

- the e-mail is entered correct, but the password is not

Use case 2: Add Player

Level: user-goal level

Primary actor: Admin actors

Main success scenario:

- user enters his/her e-mail

- user enters the password

- the data entered is correct, successfully found in the database and the user can use the application

- select the “Manage user” option

- enter new player’s e-mail, password, name, age and gender

- push the “Add User” button in order to store the information

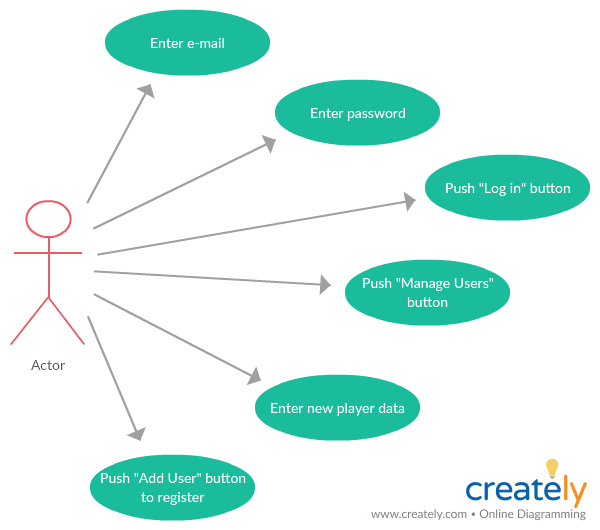
Extensions: in the scenario presented above, some possible failures would be:

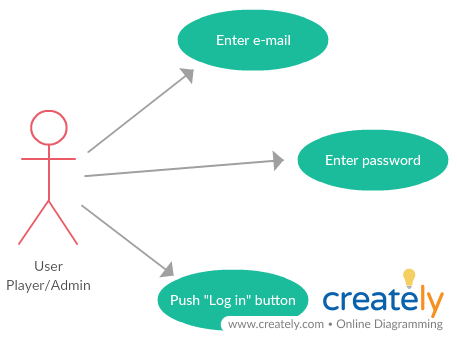
- the user is not registered and therefore the e-mail cannot be found in the database

- the e-mail is entered correct, but the password is not

- the user wanted to be introduced is already registered in the database with the given e-mail

Use case 1 Use case 2





3. System Architectural Design

**3.1 Architectural Pattern Description**

A three-tier architecture is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules. Three-tier architecture allows any one of the three tiers to be upgraded or replaced independently.

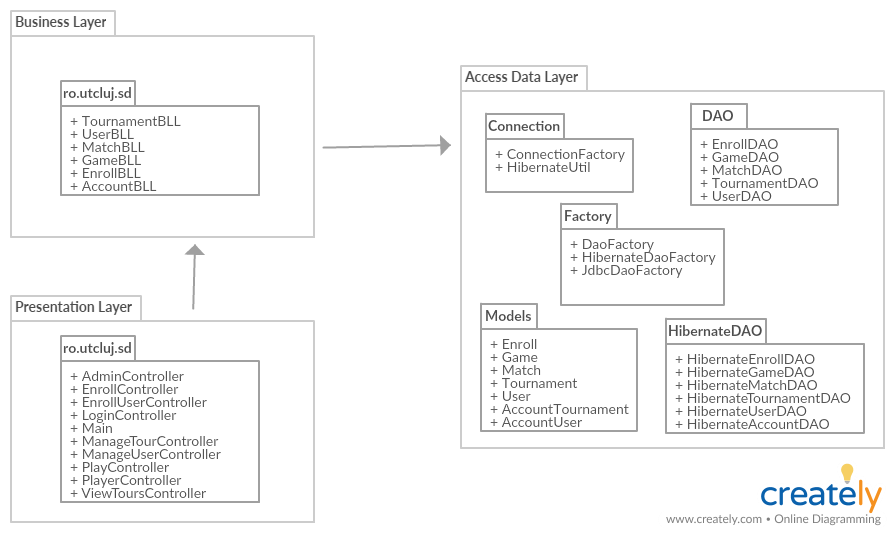
The three tiers in a three-tier architecture are:

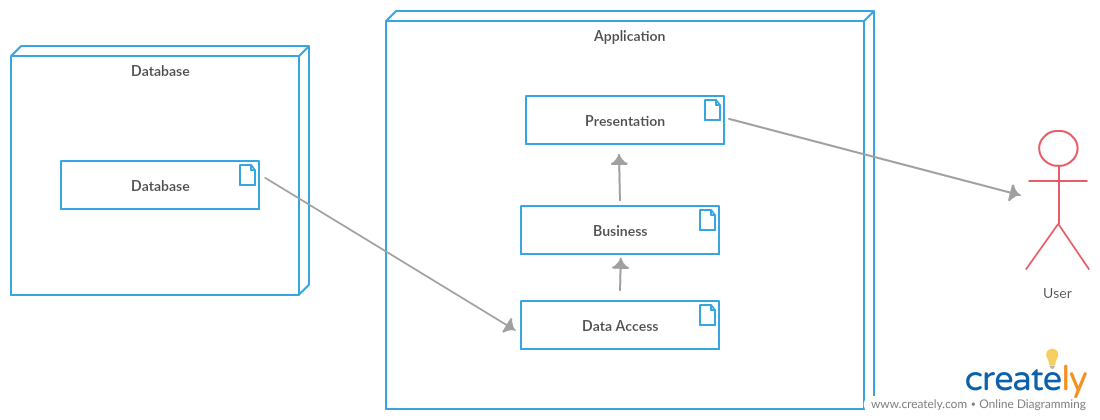
1. Presentation Tier: Occupies the top level and displays information related to services available on a website. This tier communicates with other tiers by sending results to the browser and other tiers in the network.
2. Application Tier: Also called the middle tier, logic tier, business logic or logic tier, this tier is pulled from the presentation tier. It controls application functionality by performing detailed processing.
3. Data Tier: Houses database servers where information is stored and retrieved. Data in this tier is kept independent of application servers or business logic.

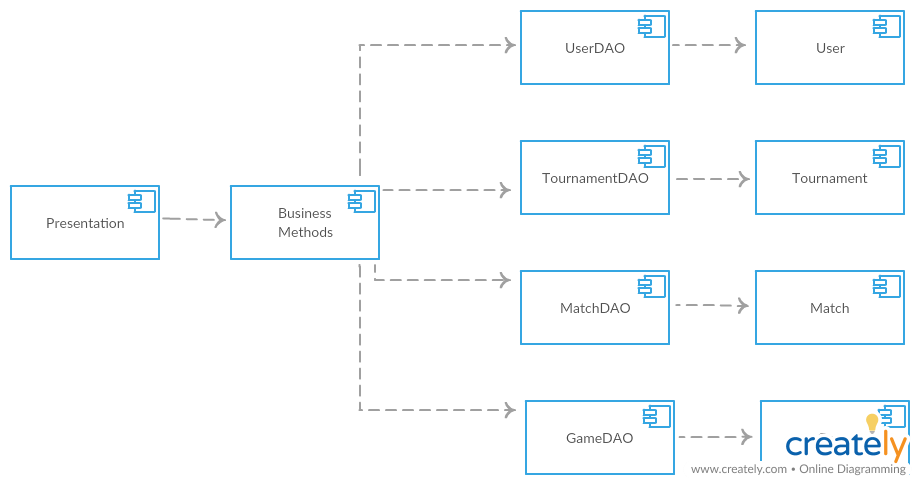
Model–view–controller (MVC) is an [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) commonly used for developing [user interfaces](https://en.wikipedia.org/wiki/User_interface) that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and parallel development.

**3.2 Diagrams**

Package diagram:



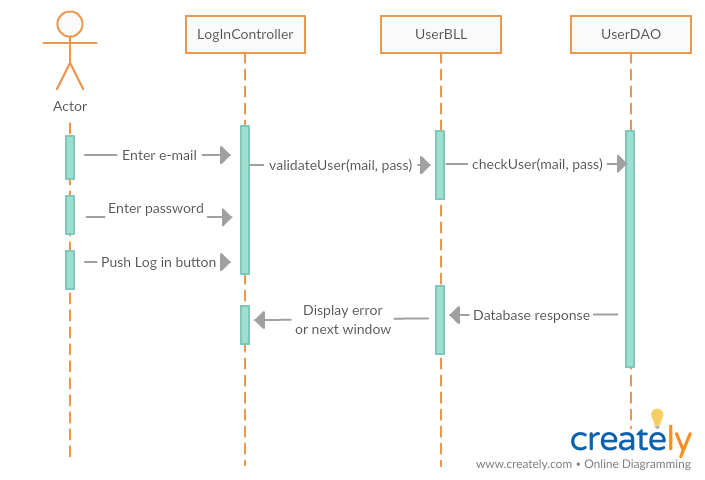
Deployment diagram:



Component diagram:

1. UML Sequence Diagrams

Sequence diagram for log in scenario:



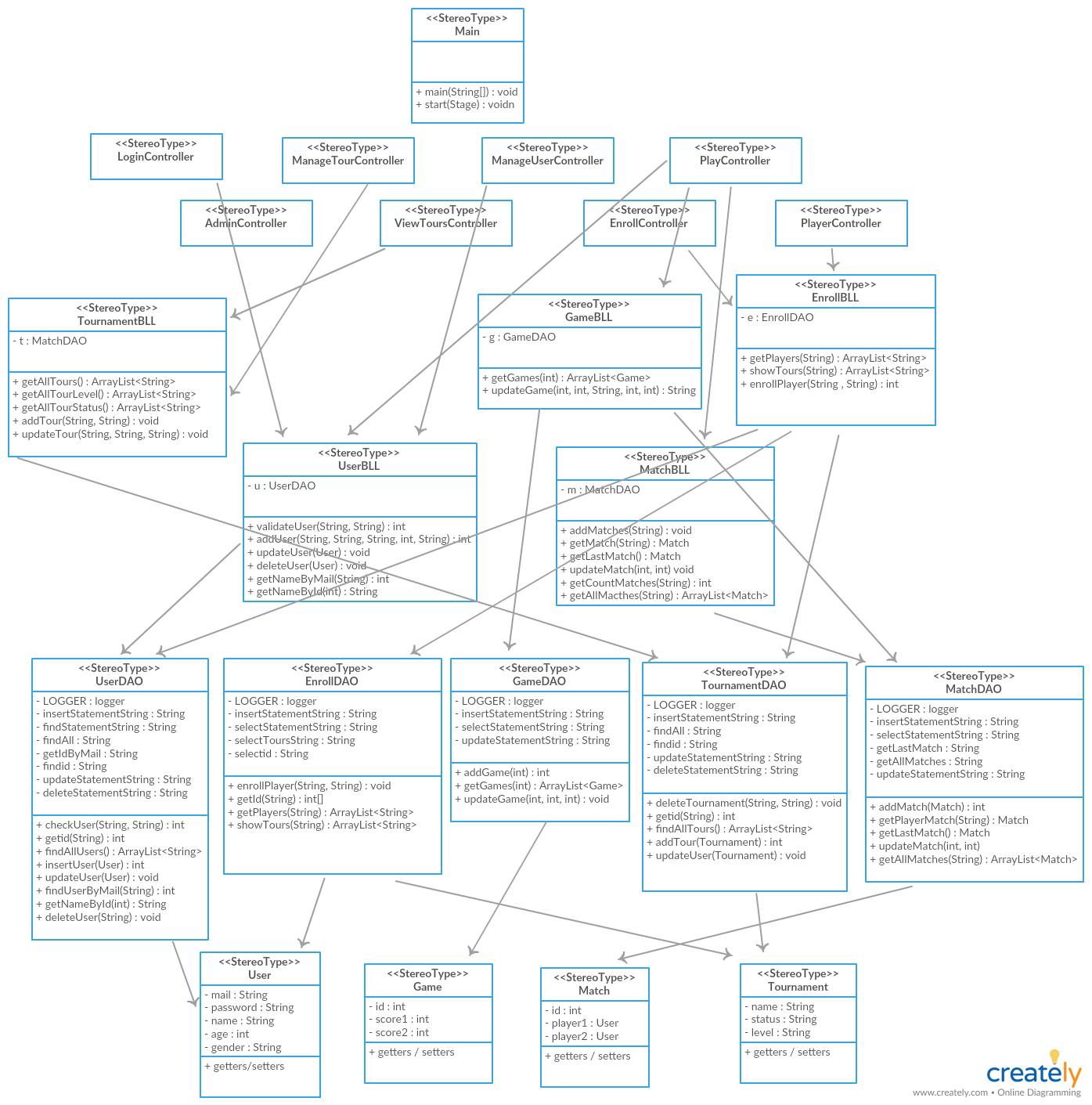
5. Class Design

**5.1 Design Patterns Description**

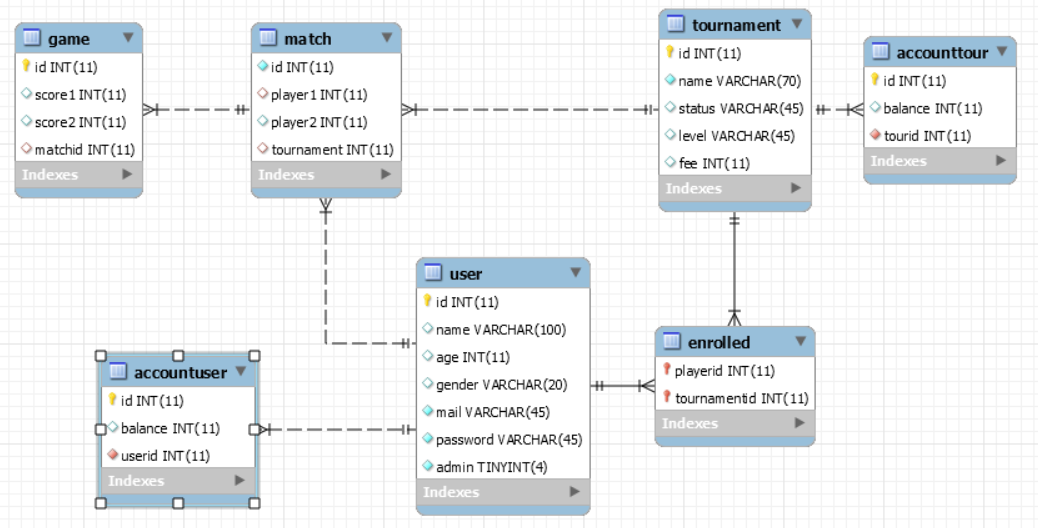
Singleton pattern involves a single class which is responsible to create an object while making sure that only single object gets created. I used this pattern in the creation of the connection to the database, to ensure the fact that a single connection is established and to maintain it.

Abstract Factory design pattern is one of the Creational pattern. Abstract factory pattern implementation provides us a framework that allows us to create objects that follow a general pattern. So at runtime, abstract factory is coupled with any desired concrete factory which can create objects of desired type.

**5.2 UML Class Diagram**



6. Data Model



7. System Testing

Unit testing is a [software testing](https://en.wikipedia.org/wiki/Software_testing) method by which individual units of [source code](https://en.wikipedia.org/wiki/Source_code), sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

Each [test case](https://en.wikipedia.org/wiki/Test_case) is independent from the others. Substitutes such as [method stubs](https://en.wikipedia.org/wiki/Method_stub), [mock objects](https://en.wikipedia.org/wiki/Mock_object), [fakes](https://en.wikipedia.org/wiki/Mock_object#Mocks.2C_fakes.2C_and_stubs), and [test harnesses](https://en.wikipedia.org/wiki/Test_harness) can be used to assist testing a module in isolation. Unit tests are typically written and run by [software developers](https://en.wikipedia.org/wiki/Software_developer) to ensure that code meets its design and behaves as intended.

To ensure that a player is not enrolled to a tournament multiple times, there is implemented a validation test that handles the situation by displaying an error message. A Junit test was made to verify this vital functionality. A similar case is for wanting to add an already existing user to the database, in which case another error message would pop up to inform you that the player is registered. The tests made for the the UserBLL class check if the methods are functioning properly.

In addition to the first assignment, as the account mechanism was introduced, a test for checking if the balance is retrieved correctly was implemented also.

8. Bibliography

# Martin Fowler et. al, Patterns of Enterprise Application Architecture, Addison Wesley, 2003 (Chapters 10 and 9)

# <https://www.techopedia.com/definition/24649/three-tier-architecture>

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

# <https://www.tutorialspoint.com/javafx/>

# <https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>

# <https://www.tutorialspoint.com/junit/junit_writing_tests.htm>

# 8.7 <https://en.wikipedia.org/wiki/Unit_testing>