Ping-Pong Tournament Layers

Analysis and Design Document

Mureșian Dan-Viorel
30433

Table of Contents

1. Requirements Analysis

1.1. Assignment Specification

Use JAVA/C# API to 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

- 2 types of users: player and administrator which must provide an email and a password in order to access the application;
- Regular users can: view tournaments, view matches, update the score of their current game (the system will detect when games and matches are won;
- Administrators can: perform CRUD operations on both player accounts and tournaments;
- Data stored in a database;
- Layers architectural pattern will be used to organize the 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.

1.3. Non-Functional Requirements

1.3.1. Availability

Maintenance will take place after each tournament has ended in order to update the database, check results and determine prize winners. Maintenance will last between 1 and 2 hours. Monthly maintenance will be performed once per month as well, in the Thursday before the end of the month and will last from 10 PM to 6 AM the following day.

1.3.2. Performance

- Approximately 1000 users will be able to run the application at a given time, while only 1 administrator might perform operations at any given time.

2. Use-Case Model - to be updated

Use Case: Modify Own Score
Level: player level
Primary Actor: player
Main success scenario:

- Start application;
- Click the Login button;
- Login with email and password;
- Click the View Tournaments button;
- Select the tournament you are taking part in;
- Click the View Tournament button;
- Click the Refresh button to populate the list;
- Double click on your name to get the match list;
- Select the current ongoing game;
- Click View Game;
- Use + for increasing the score, - for decreasing it;
- Click Update Score when you are done;
- Click Refresh to see updates;

Extensions:

- Once a game has finished, the score can't be modified anymore;
- Once a match has finished, the score in a game can't be modified anymore;
- Clicking on another player's match will generate an error.
- Invalid Login credentials;

Use Case: Create Tournament
Level: administrator level
Primary Actor: administrator
Main success scenario:

- Start application;
- Click the Login button;
- Login with an administrator account;
- Click the View Tournaments button;
- Click the Create Tournament button;
- Enter tournament data and click Create Tournament;

Extensions:

- Entering null data will generate an error;
- You cannot create an account without an administrator account;
- Invalid Login credentials;

Use Case: View matches
Level: player level
Primary Actor: player
Main success scenario:

- Start application;
- Click the Login button;
- Login with email and password;
- Click the View Tournaments button;
- Select the tournament you are taking part in;
- Click the View Tournament button;
- Click the Refresh button to populate the list;
- Double click on your name to get the match list;

Extensions:

- Invalid Login credentials;

Use Case: View tournaments
Level: player level
Primary Actor: player
Main success scenario:

- Start application;
- Click the Login button;
- Login with email and password;
- Click the View Tournaments button;

Extensions:

- Invalid Login credentials;

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").

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).

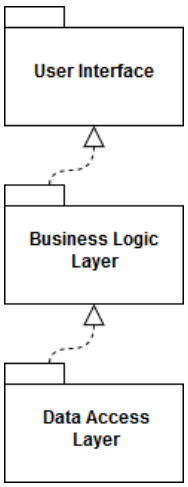
Application tier (business logic, logic tier, or middle tier):

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.

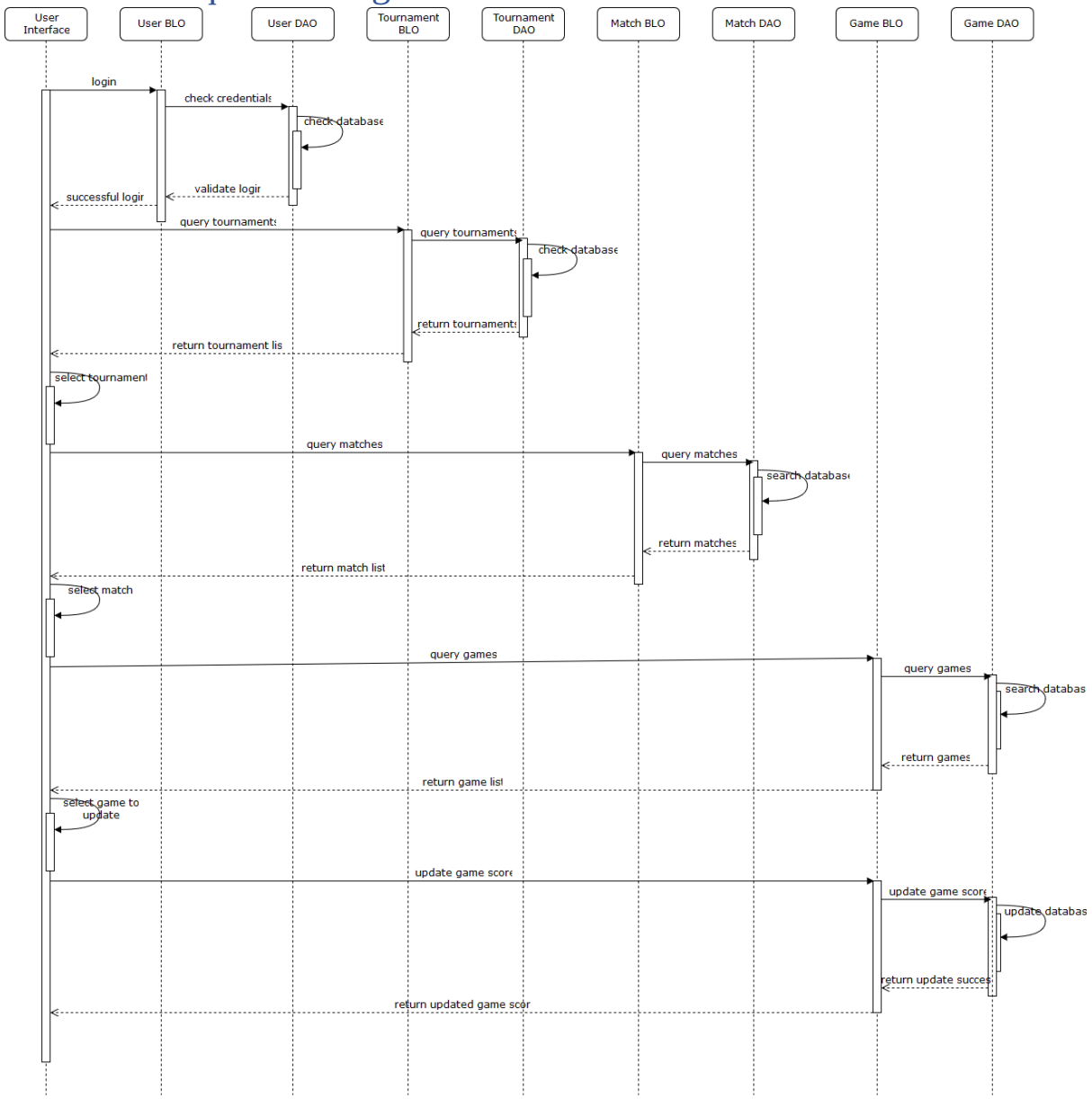
Data tier:

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



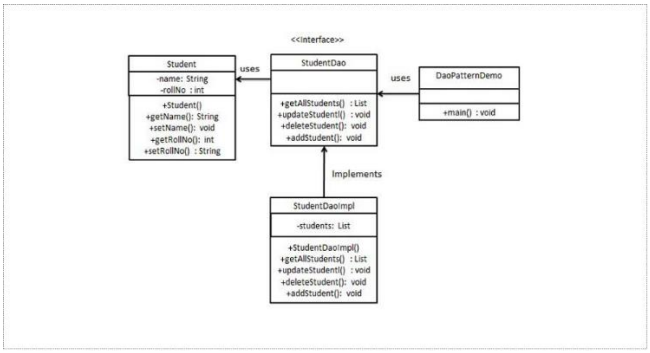
4. UML Sequence Diagrams



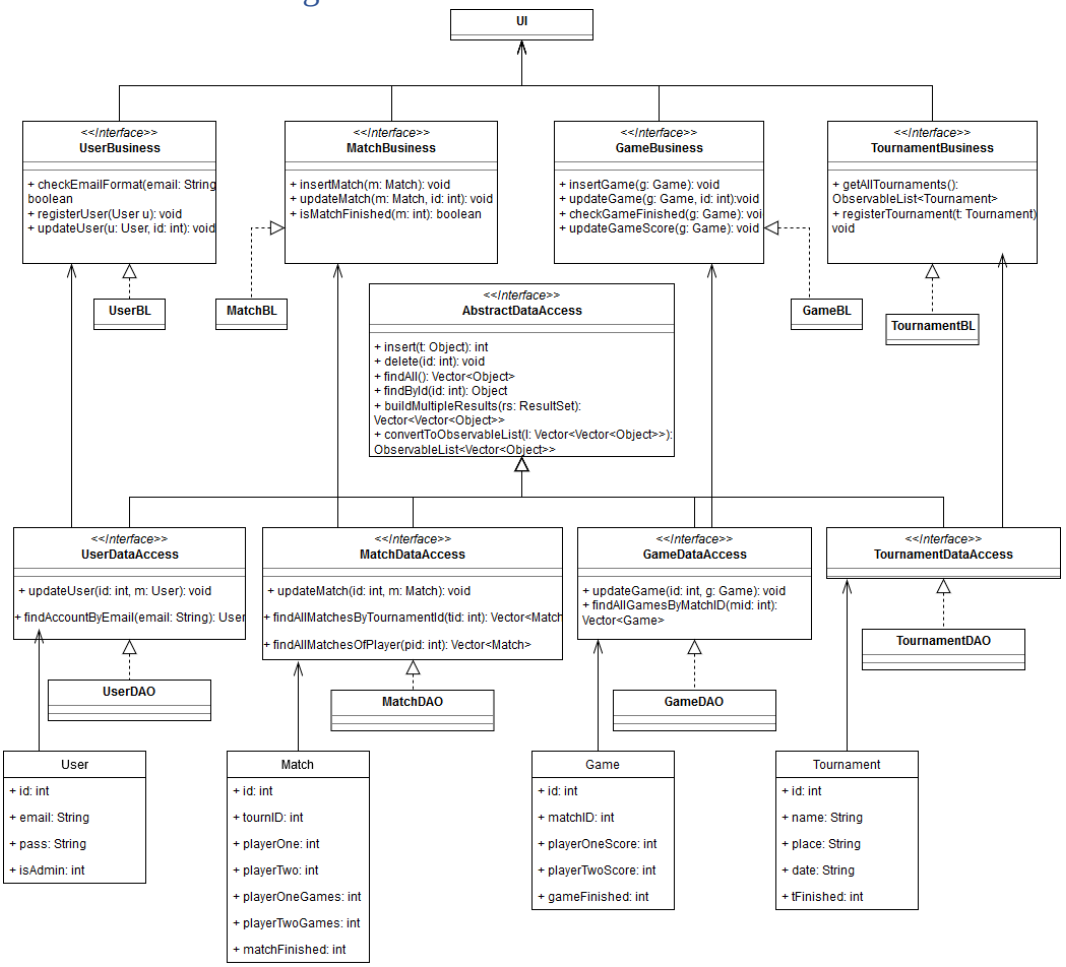
5. Class Design

5.1. Design Pattern Description transaction script, dao pattern, 3 tier

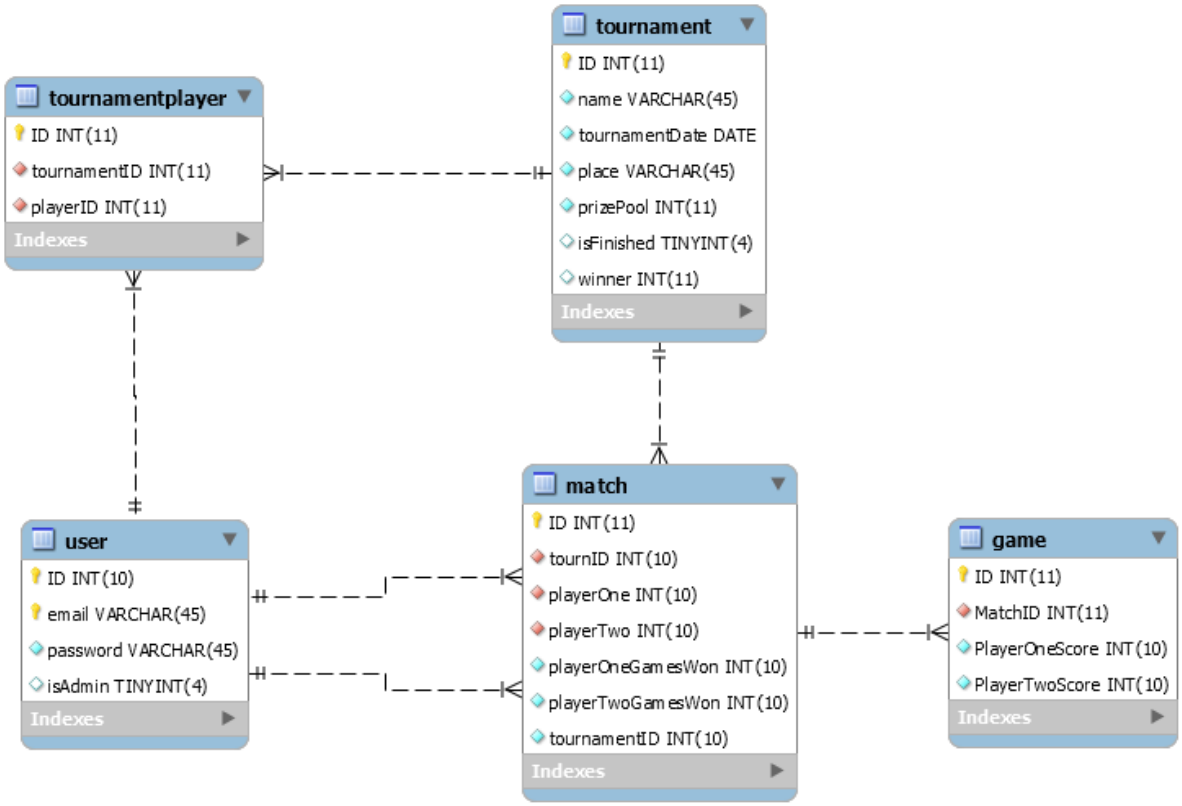
In computer software, a data access object (DAO) is an object that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, the DAO provides some specific data operations without exposing details of the database. This isolation supports the single responsibility principle. It separates what data access the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), from how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).



5.2. UML Class Diagram



6. Data Model



7. System Testing

7.1. White-box Testing

White-box testing uses the control structure of the procedural design to derive test cases that (i) guarantee that all independent paths within a module have been exercised at least once, and that (ii) exercise all logical decisions, all loops and internal data structures. Control structure testing is an example of white-box testing method which includes the following:

a) Condition testing

- Exercises the logical conditions in a program module;
- If a condition is incorrect, then at least one component of the condition is incorrect. Therefore, types of errors in a condition include the following:
  - Boolean operator error (incorrect/missing/extra Boolean operators);
  - Boolean variable error;
  - Boolean parenthesis error;
  - Relational operator error;
  - Arithmetic expression error.

- Strategies:
    - *Branch testing* - for a compound condition C, the true and false branches of C and every simple condition in C need to be executed at least once.
    - *Domain testing* - for E1 <relational-operator> E2, 3 tests are required to make the value of E1 greater than, equal to, or less than that of E2.
    - *Branch and relational testing* - detects branch and relational operator errors in a condition if all Boolean variables and relational operators in the condition occur only once and have no common variables.

a) Loop testing

7.2. Black-box Testing

Black-box testing, also called behavioral testing, focuses on the functional requirements of the software and enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing is not an alternative to white-box techniques; rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods. Black-box testing attempts to find errors in the following categories: (1) incorrect or missing functions, (2) interface errors, (3) errors in data structures or external data base access, (4) behavior or performance errors, and (5) initialization and termination errors. Black-box testing includes methods such as the following:

a) Equivalence partitioning

- Divides the input domain of a program into classes of data from which test cases can be derived.
- An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition. Equivalence classes may be defined according to the following guidelines:
    - If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
    - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
    - If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
    - If an input condition is Boolean, one valid and one invalid class are defined.

b) Boundary value analysis

- Complements equivalence partitioning by exercising bounding values. Rather than selecting any element of an equivalence class, the boundary value analysis leads to the selection of test cases at the "edges" of the class.

8. Bibliography

1. https://pongworld.com/table-tennis-sport/rules;
2. https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture;
3. https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm;
4. https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html;
5. https://en.wikipedia.org/wiki/Data_access_object;
6. http://disi.unal.edu.co/dacursci/sistemasycomputacion/docs/SWEBOK/Systems%20Engineering%20-%20EAA%20-%20Patterns%20of%20Enterprise%20Application%20Architecture%20-%20Addison%20Wesley.pdf;