Ping-Pong Tournament Layers

Analysis and Design Document

Mureșian Dan-Viorel
30433

Table of Contents

1. Requirements Analysis 2
1.1. Assignment Specification 2
1.2. Functional Requirements 2
1.3. Non-Functional Requirements 2
1.3.1. Availability 2
1.3.2. Performance 2
1.3.3. Testability 2
1.3.4. Usability 2
2. Use-Case Model 3
3. System Architectural Design 4
3.1. Architectural Pattern Description 4
3.2. Diagrams 5
4. UML Sequence Diagrams 5
5. Class Design 6
5.1. Design Pattern Description 6
5.2. UML Class Diagram 6
6. Data Model 7
7. System Testing 7
8. Bibliography 7

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.

1.3.3. Testability

- Unit testing;

1.3.4. Usability

- User-friendly GUI;

2. Use-Case Model

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

- Start application;
- Login with email and password;
- Select the tournament you are taking part in;
- Select the current match you are playing;
- Update the current game you are playing;

Extensions: -.

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

- Start application;
- Login with an administrator account;
- Select the Tournaments panel;
- Select “Create Tournament”;
- Enter tournament data and enroll players.

Extensions: -.

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

- Start application;
- Login with a player account;
- Select the Tournaments panel;
- Select an ongoing tournament;
- Select a match from the tournament.

Extensions: -.

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

- Start application;
- Login with a player account;
- Select the Tournaments panel;
- Select a tournament;

Extensions: -.

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).

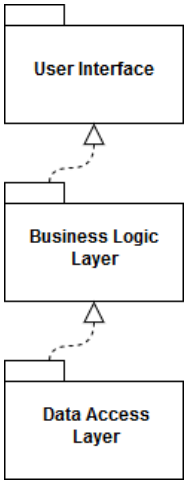
Application tier (business logic, logic tier, or middle tier):

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.

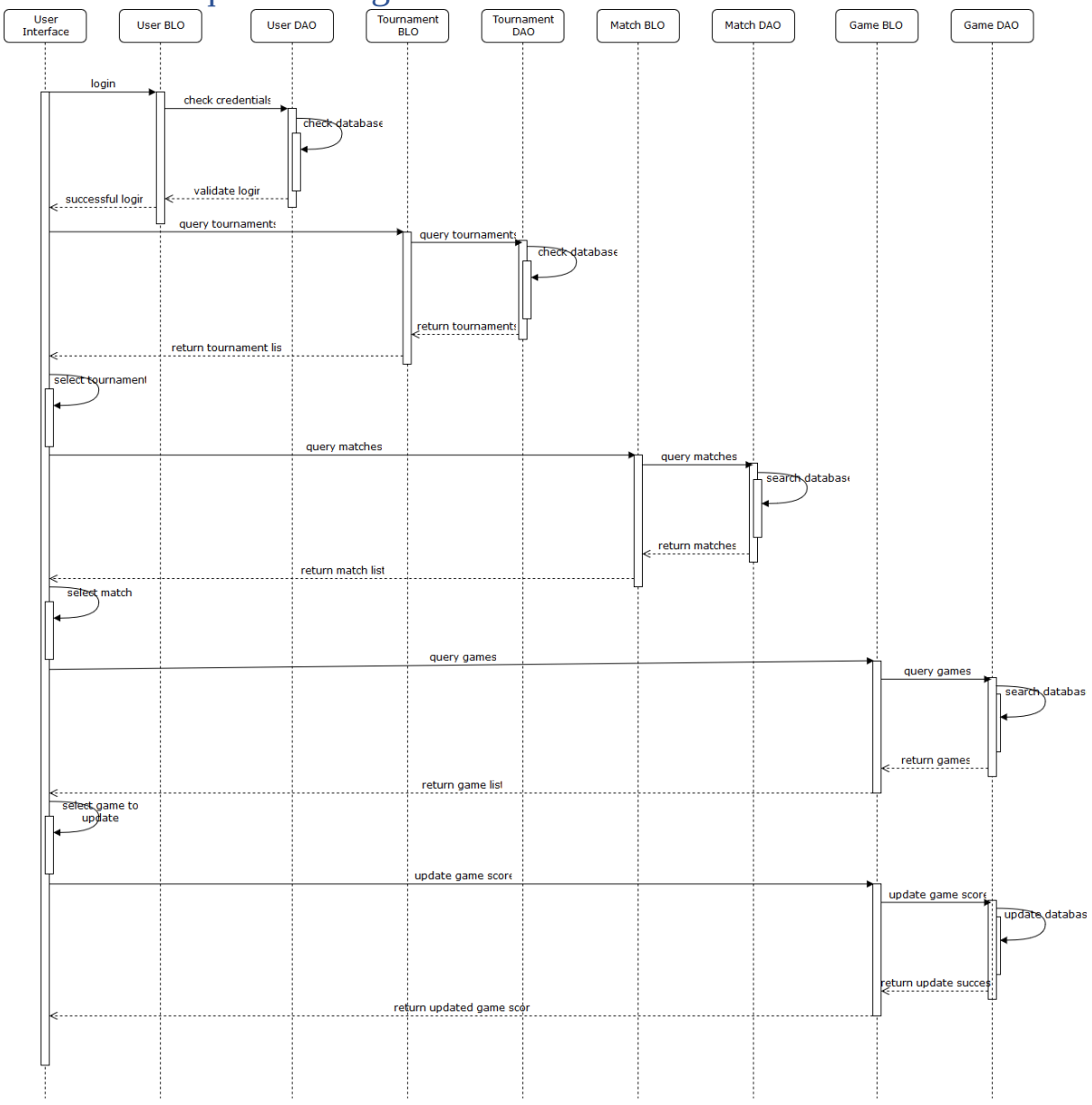
Data tier:

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

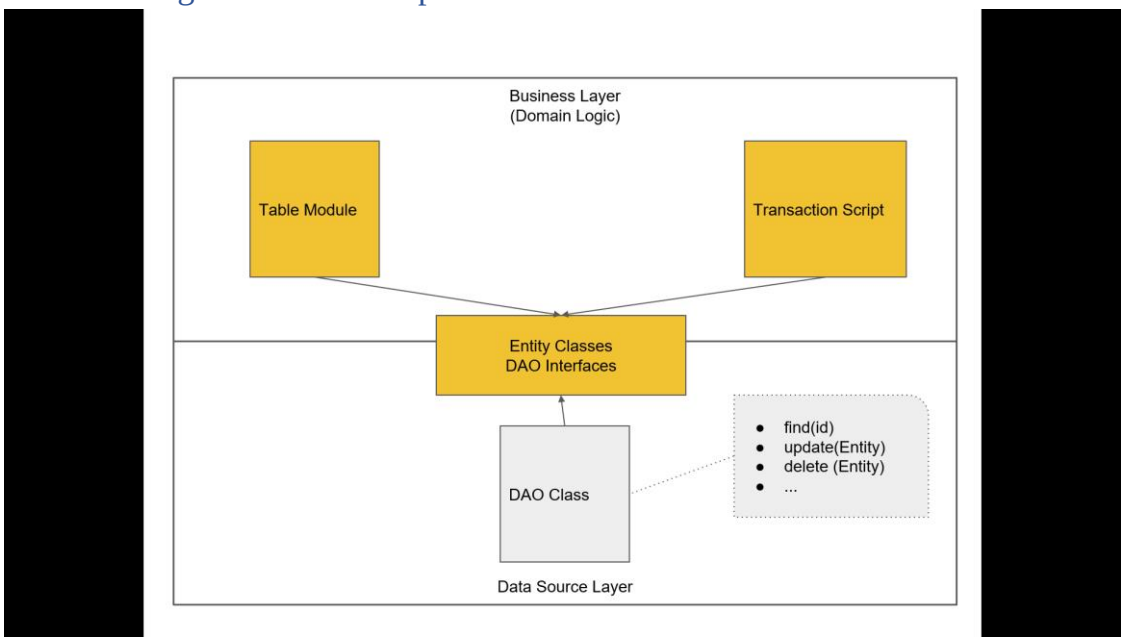


4. UML Sequence Diagrams

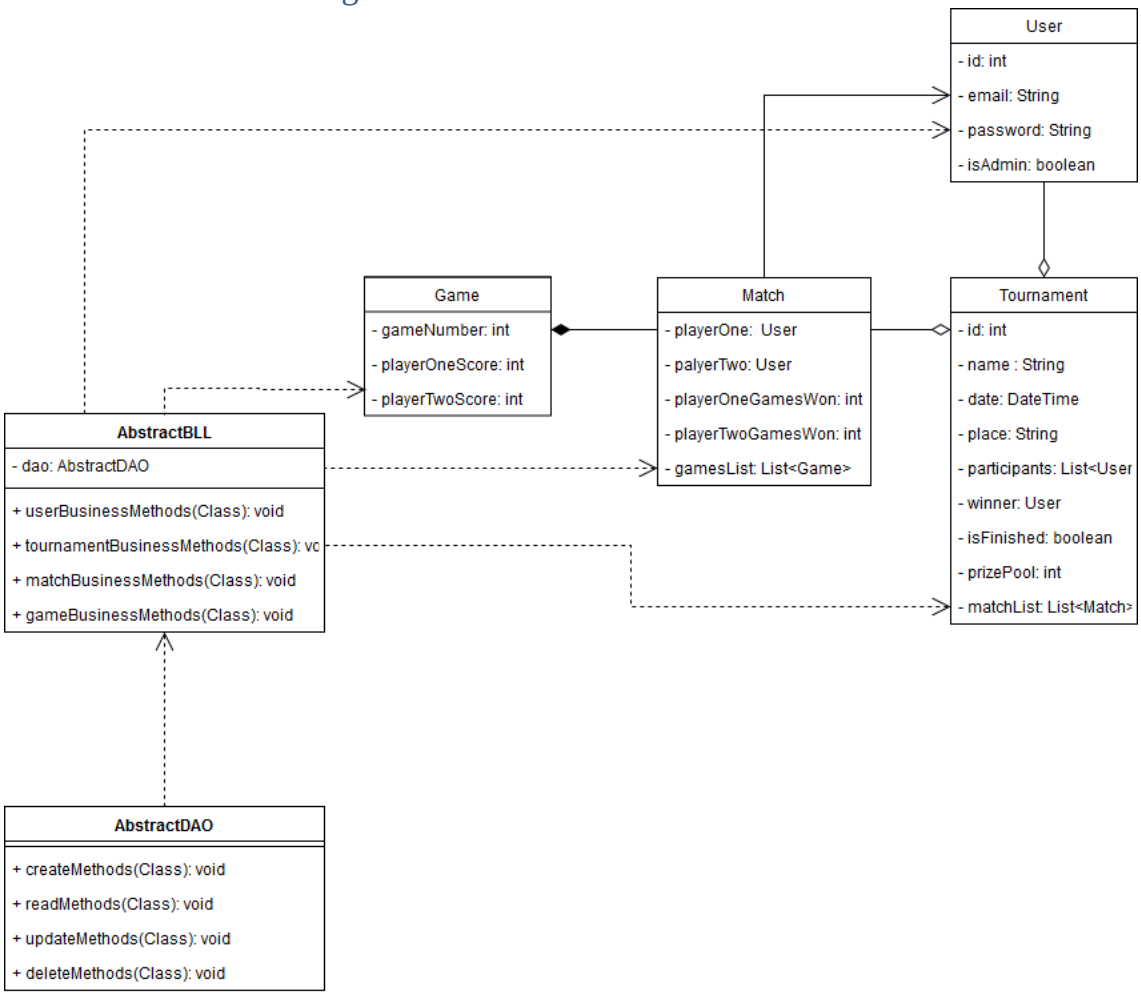


5. Class Design

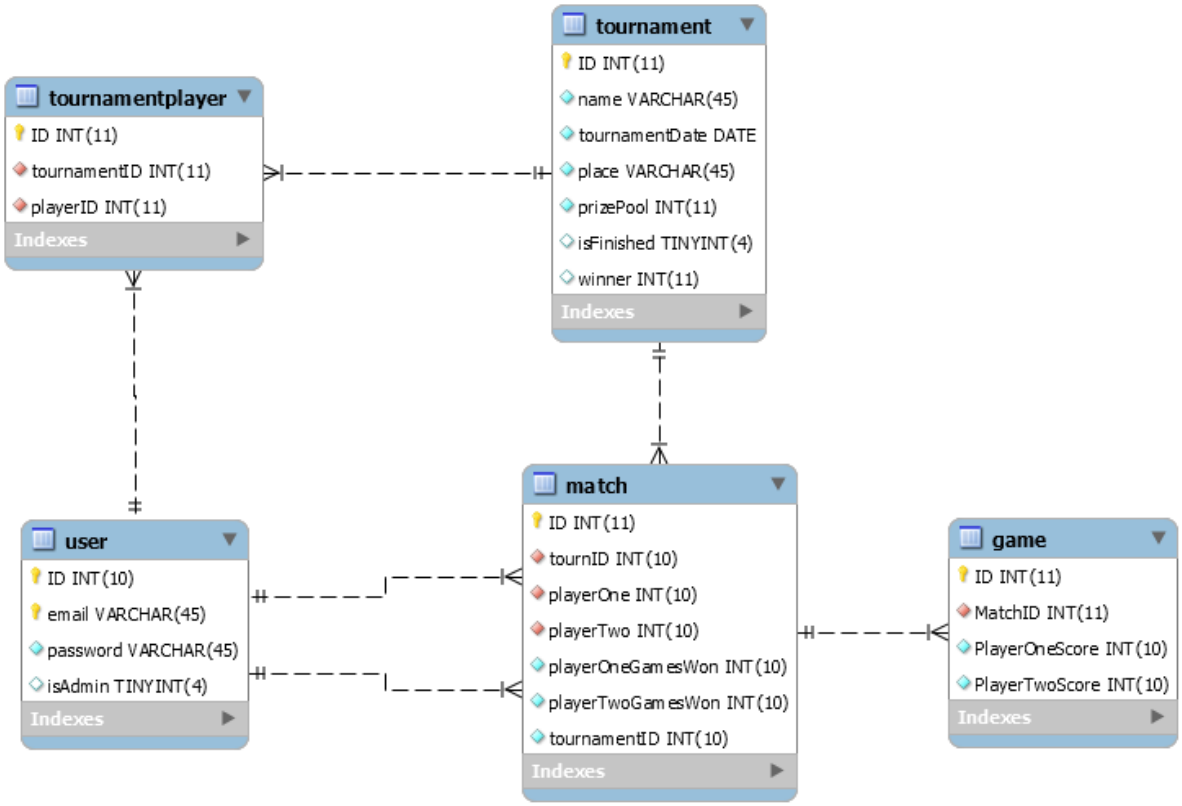
5.1. Design Pattern Description



5.2. UML Class Diagram



6. Data Model



7. System Testing

See reference number 6.

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. Domain Logic and Data Source Patterns presentation;
6. L1_Testing_Techniques.pdf.