B8IT150 ADVANCED PROGRAMMING (B8IT150_2425_TME4)

Lecturer: Kanza Ali Manza

Student Id: 20036706

PDF REPORT BY Isaac Oubiyi - Advanced Programming

Higher Diploma in Science in Computing

INTRODUCTION.

For the purpose of this CA Project in Advanced Programming, I have decided to develop a Retro video game store management system. The application is a simple console-based application developed in C#. The goal of the application is to enable an administrator to manage a catalogue of games stored in the database of the video game store. I have adopted the management of this database using SQLite. The admin can essentially perform CRUD (Create,Read,Update and Delete) operation on the video game records, I have added the functionality of checking out games for customers and returning them to the store. To satisfy the requirements of this assignment, the application also demonstrates the use of sorting and searching algorithms (Bubble Sort and Binary Search) as well as a selection-based algorithm for Analytics of the game store. Which is used to find the top 5 most-checked-out games and platforms (PS1, SEGA, Game Boy, etc.). The design of the application is intentionally a console-based application. This essentially ensures the focus on object-oriented design structure, architecture layers, and algorithm implementation rather than on GUI concerns.

1. Project Design and Architectural Layer.

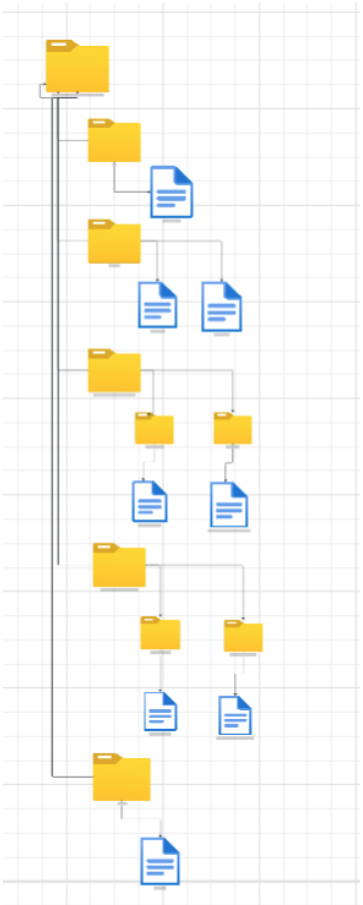
The application follows a layered architecture to help demonstrate separation of concern, i.e. at a low level following the single responsibility principle of object-oriented programming. This in turn ensures a clean design structure.

The main layers are:

- **UI** (Presentation Layer): Contains the Program class and the main menu logic that interacts with the user (Store Admin) via the console.
- **Models** (Domain Layer): Contains the core domain entities such as Game and Admin.

- **BLL** – Business Logic Layer: Contains core key service classes and algorithms that implement business rules and operations.
- **DAL** – Data Access Layer: Contains repository interfaces and concrete implementations to communicate with SQLite using Dapper. I have adopted dapper in this application with consideration of the scale of the system, speed and ease of use
- **Script**: Contains the SQL seed script (data.sql) that pre-populates the database with retro game and their associated properties (Id, Title, Developer, Platform, ReleaseYear, CopiesTotal, CopiesAvailable, TimesCheckedOut).

Folder structure of the Video Game Store Management System is as follows:



VideoGame ManagementSystem

UI/ -Program.cs

Models/ -Game.cs , -Admin.cs

BLL (Business Logic Layer)/

Algorithims/ - Algorithims.cs

Services/ - VideoGameStoreService.cs

DAL (Data Access Layer)/

IRepositories/ - IRepository.cs

SQLRepositories/ - GamesSqlRepository.cs

Script/ - data.sql

- This layout clearly separates user interaction, business logic, data access, and database seeding.

2. Classes (Entities) and Responsibilities

The key classes in the system are summarised below.

1. Game (Model)

• Represents a single video game in the store.

• Properties include:

- Id: Unique identifier in the database.
- Title: Name of the game.
- Developer: Company or team that created the game.
- Platform: The platform the game runs on (e.g., NES, Sega Genesis, Arcade, Game Boy, PS1, PC).
- ReleaseYear: Year the game was released.
- CopiesTotal: Total number of copies owned by the store.
- CopiesAvailable: Number of copies currently available for checkout.
- TimesCheckedOut: Counter used for analytics (how many times this game has been checked out).

• Overrides ToString() to provide a concise representation of a game in the console.

2. Admin (Model)

- Represents an administrator who can log into the system.
- Properties include Id, Username and Password.
- Admin credentials are stored in the admins table in the same SQLite database.

3. IRepository (DAL Interface)

- A generic repository interface used to abstract data access.

- Methods include Add, Update, Delete, GetAll and GetById.
- Demonstrates abstraction

4. GameSqlRepository (DAL Implementation)

- Implements IRepository using SQLite and Dapper.
- Responsible for executing SQL statements against the VideoGames table (insert, update, delete, select).
- Uses parameterised queries to prevent SQL injection and to keep data access logic out of the BLL

5. VideoGameStoreService (Business Logic Service)

• Depends on IRepository and acts as the main business logic layer for games.

• Responsibilities include:

  – Adding, updating and deleting games.
  – Retrieving all the games and sorting them before returning.
  – Searching for a game by title using a binary search.
  – Checking out a game by title (decreasing CopiesAvailable and incrementing TimesCheckedOut).
  – Returning a game by title (increasing CopiesAvailable when appropriate).
  – Providing analytics such as the most popular game, the most popular platform, and top 5 lists.

6. Algorithms (Static Utility Class)

• Contains the algorithm implementations used by the service layer:

  – BubbleSort: Used to sort the list of games alphabetically by Title.
  – BinarySearch: Used to efficiently find a game by Title once the list is sorted.
  – GetTop5Games: Uses a selection-style sort to order games by TimesCheckedOut and return the top five.
  – GetTop5Platforms: Aggregates total checkouts per platform and uses selection-style sorting to rank them.

3. Algorithims Implemented for the Video Game Store Management System

I have intentionally implemented classic algorithims manually into the project to demonstrate my understanding of time complexity and algorithmic thinking rather than relying on the inbuilt .NET methods.

- Bubble Sort (Sorting by Game Title)
  I have implemented this sorting algorithm because of its simple comparison-based logic. It repeatedly steps through the list of games (gameTitle), comparing adjacent items, and then

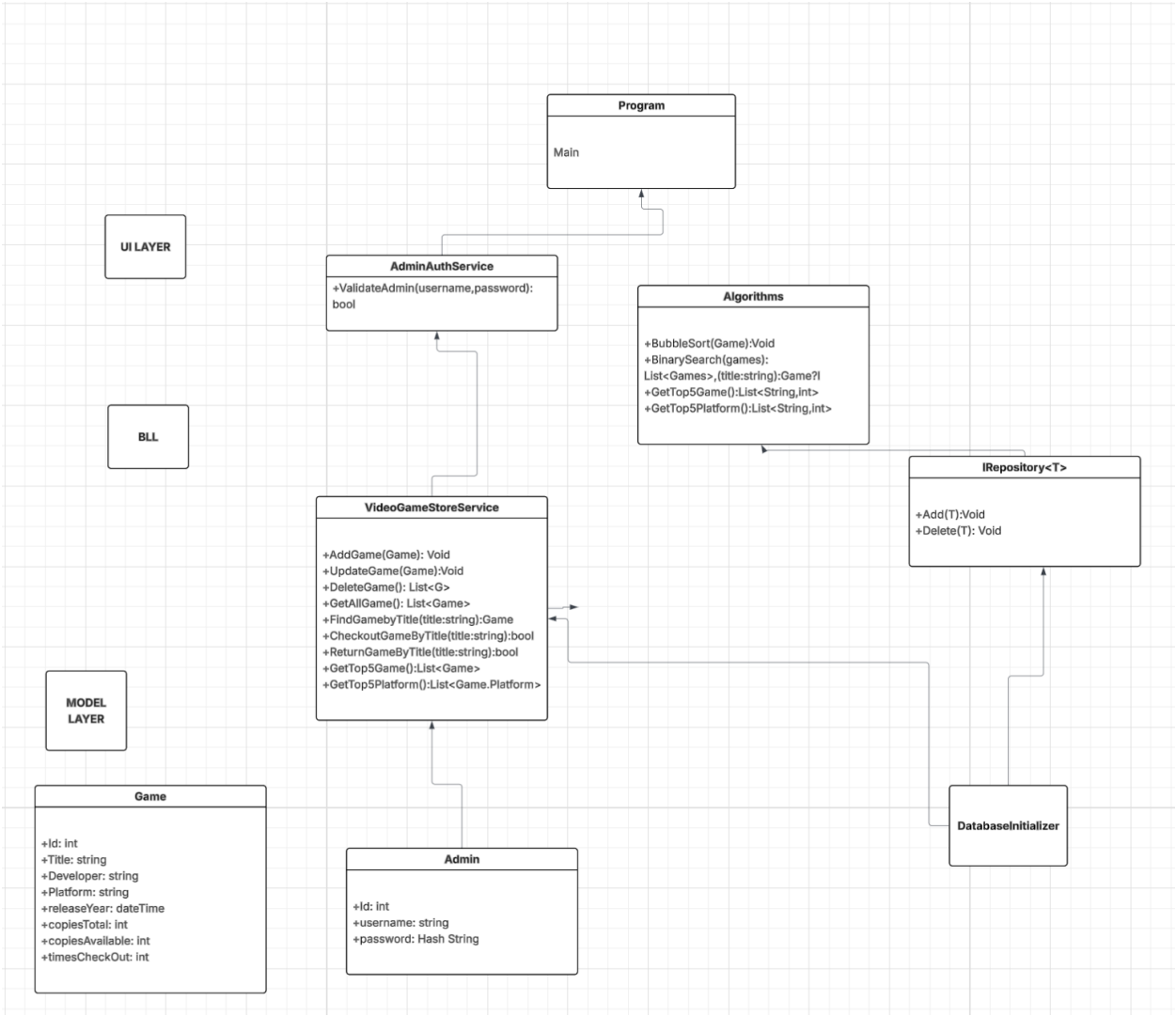
swaps them in the correct order. With the time complexity of $O(n^2)$ it is easy to implement considering the relatively small dataset in the application.

- Binary Search (Searching by Game Title)
  I implemented this very efficient searching algorithm with the time complexity of O(log n). The algorithm requires the inputs of the sorted list first (from the bubble sort). The Binary Search method can easily and quickly locate the game being searched by halving the search range. This method of searching outweighs a linear search and demonstrates the advantages of the combination of the sorting and searching algorithm.

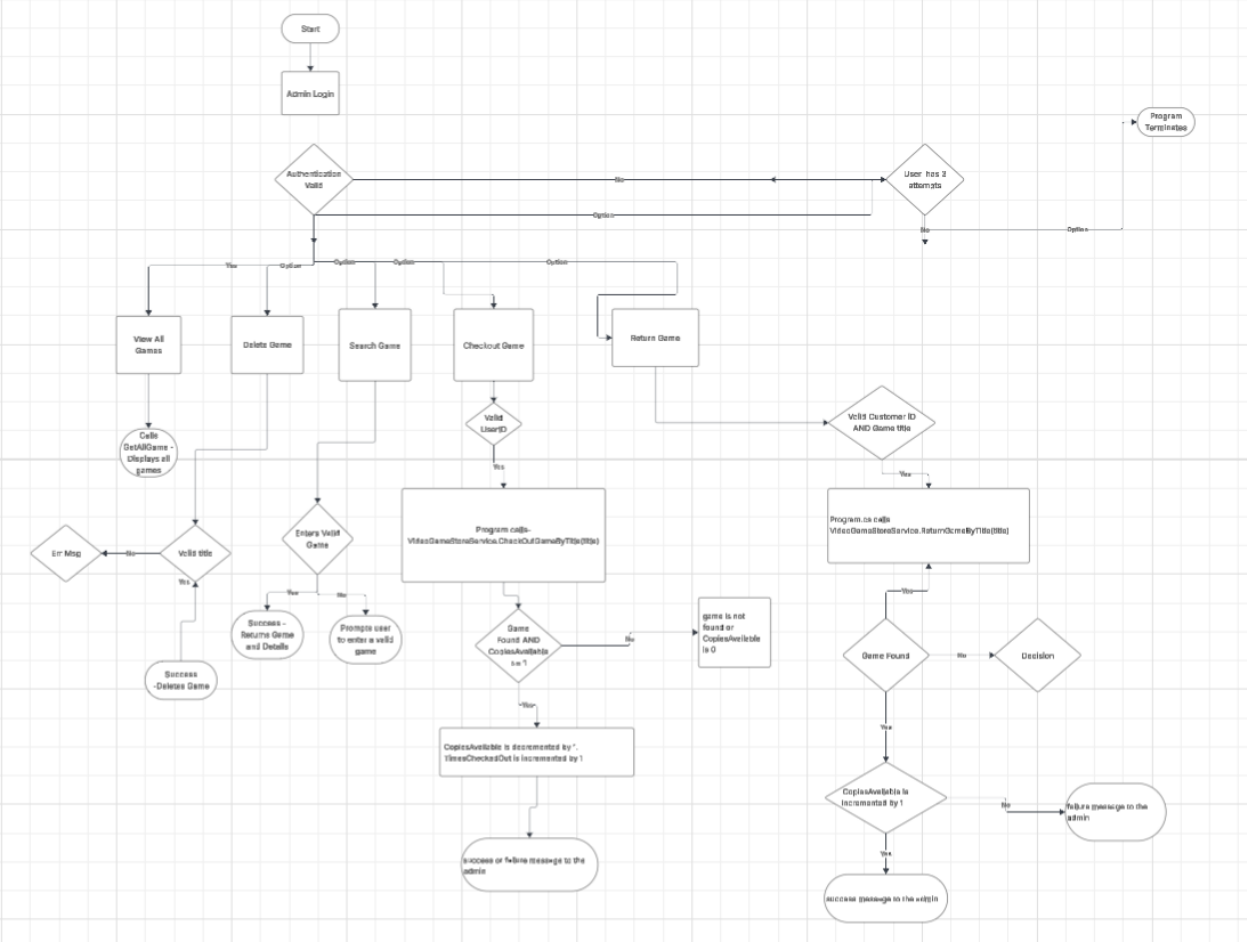
- Selection Sort (Analytics)
  The application offers the admin the ability to display the top 5 performing games in the stores. Using a selection-sort loop, iterating through a list, repeatedly selecting the highest checked out games (TimesCheckedOut). Using the TimesCheckedOut value and swapping it to the front. This ensures a list sorted in descending order of popularity. For displaying the top 5 platform (consoles), the function (GetTop5Platforms) first aggregates the TimesCheckedOut per platform, then converts into a list and applies a similar selection process to rant the top performing platforms in the store. This in turn helps the admin to identify which are the best-selling consoles amongst customers using the store.

These algorithms are essential for the demonstration of efficient algorithmic principles for the scale of this Application.

4.UML Class Diagram



5. Flowchart for Check-out / Return Process



Check-out Process:

1. Admin chooses the "Check Out a Game" option from the console menu.

2. The system optionally asks for a Customer ID (for logging purposes) and then asks for a Game Title.

3. Program.cs calls VideoGameStoreService.CheckOutGameByTitle(title).

4. The service retrieves all games from the repository and sorts them (BubbleSort).

5. The service uses BinarySearch to find the game by title.

6. If the game is not found or CopiesAvailable is 0, the service returns false.

7. If the game is found and at least one copy is available:

- CopiesAvailable is decremented by 1.
- TimesCheckedOut is incremented by 1.
- The updated Game is persistent, using the repository's Update method.

8. Program.cs displays a success or failure message to the admin.

Return Process:

1. Admin chooses the "Return a Game" option from the console menu.

2. The system asks for a Customer ID and then for a Game Title.

3. Program.cs calls VideoGameStoreService.ReturnGameByTitle(title).

4. The service retrieves and sorts the games, then locates the correct game.

5. If the game is not found, the service returns false.

6. If CopiesAvailable is already equal to CopiesTotal, the service also returns false to prevent over-returning.

7. Otherwise, CopiesAvailable is incremented by 1 and the repository Update method is called.

8. Program.cs displays a success or failure message.

6.Conclusion

The Video Game Store Management System demonstrates a two-layered console application. The Application showcases clean architecture, OO design principles, and the implementation of classic algorithms such as bubble sort and binary search. The adoption of SQLite database offers a light integration of database, Dapper-based repositories and a dedicated Business Logic Layer ensure concerns and responsibility are kept separate and the codebase integrity remains intact and remains maintainable. Working on this project further enhanced my knowledge on how simple algorithms can be used to derive useful insights into a real-world application such as the analytic function I have implemented in this system. I managed to navigate the complexity of the search and sorting algorithm to help meet the CRUD functionality of the application. Furthermore, I would like to improve the other aspect of this management in the future, perhaps building on the analytic functions of the system, implement a customer element to the check-in and check-out option of the system which in turn will enhance the security and integrity of the system.

References

https://deviq.com/principles/separation-of-concerns

https://lucid.app/lucidchart/f5a262a4-844d-41af-99d1-afd709da2265/edit?invitationId=inv_4ab26838-ee87-418c-836f-7da5de486dfa&page=0_0#

https://www.geeksforgeeks.org/software-engineering/library-management-system/

https://stackoverflow.com/questions/76340509/using-sqlite-from-a-c-sharp-console-application-built-with-the-command-line