كلية الحاسبات والذكاء الإصطناعي
Faculty of Computers & Artificial Intelligence

Helwan University

HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Information System Department

Cinema Seat Reservation System

A graduation project dissertation by:

Mazen Mohamed Ahmed Mohamed	20220363
Mohamed Mahmoud Mohamed Nady	20220428
Ziad Maher Abdelaleem Mohamed	20220185
Beshoy Mamdouh Yaqoub	20220120
Abdullah Hassan Fathy Omar	20220282
Mohamed Hatem Ahmed El Noby	20220387

GitHub link :
https://github.com/ShinobiBoi/pl3

1-Project Overview

1.1. Objective: To develop a functional and safe console application for cinema seat booking.

1.2. Goal: Implement robust logic to manage seating inventory, prevent double-booking, and generate persistent, unique reservation tickets.

1.3. Technology: F# , which ensures data safety through immutability and simplifies state management using Discriminated Unions for clear seat status representation.

2- System Architecture and Data Model

2.1. Data Representation

We define custom types for clarity and domain safety:

A. **Seat Status :** This union explicitly defines the possible states for any single seat, ensuring impossible states are not represented.

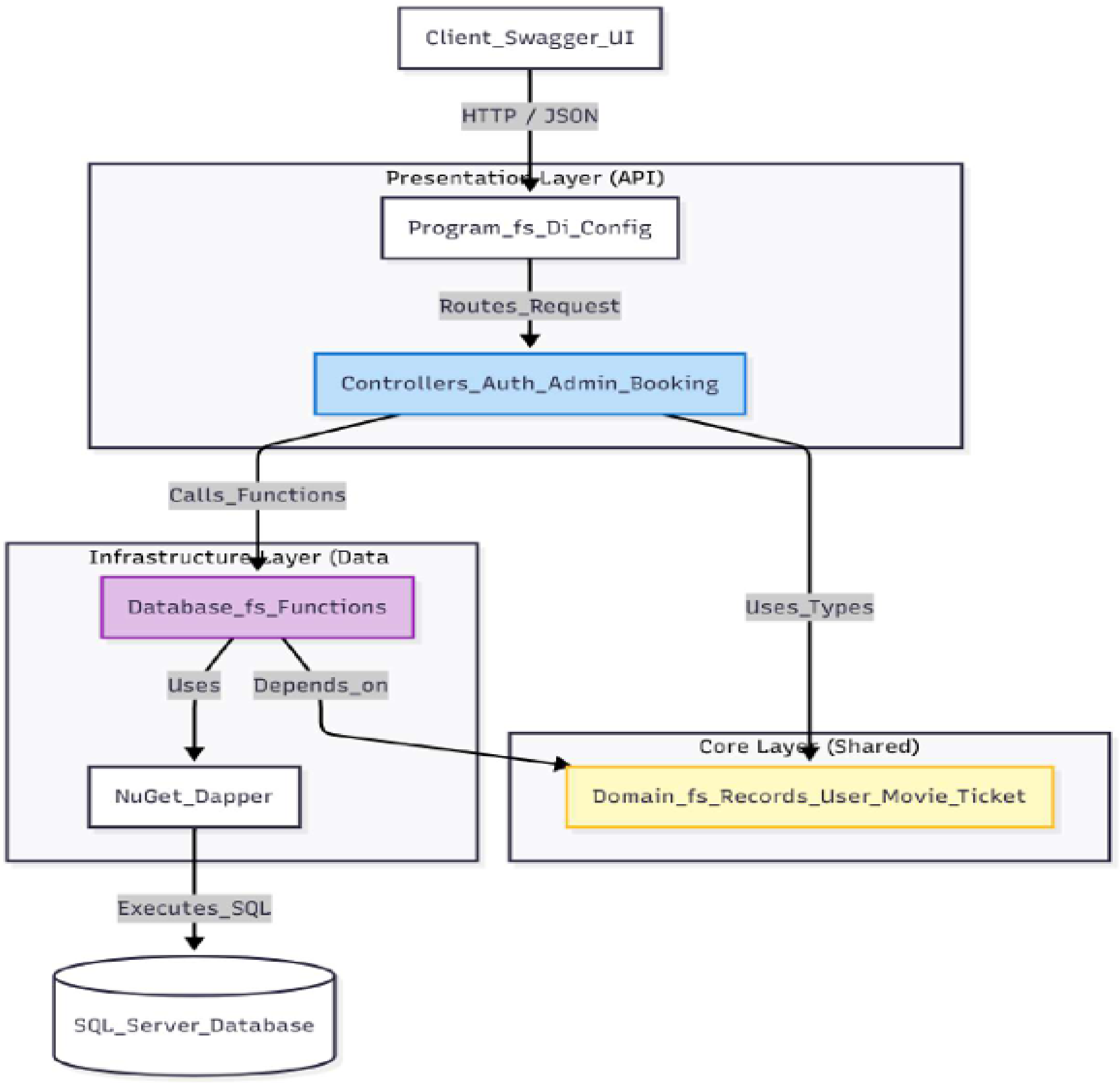
B. **Ticket Record:** A record type is used to store all essential information about a successful reservation.

2.2. Seating Layout

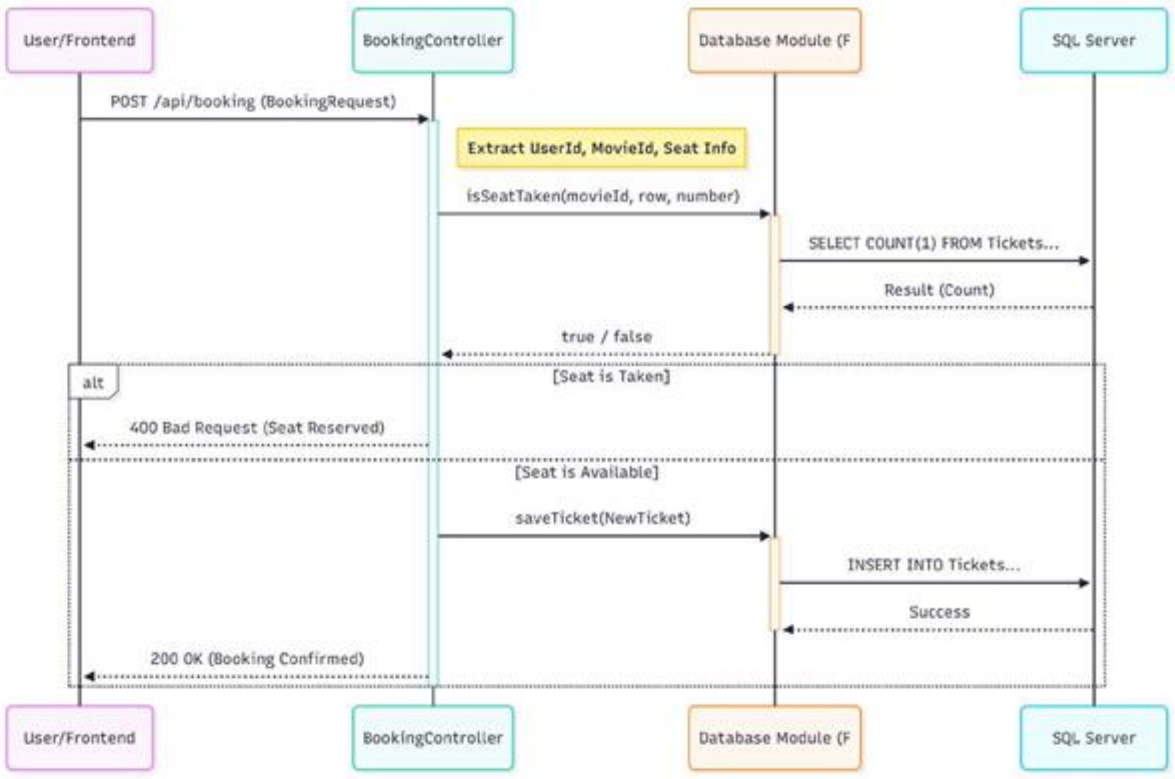
The cinema will be modeled as an immutable 2D Array where each element holds a Seat Status value.

3. System Modeling and Design Diagrams

3.1. System Architecture Diagram



3.2. Sequence Diagram



4. Testing Framework Choice: xUnit.net

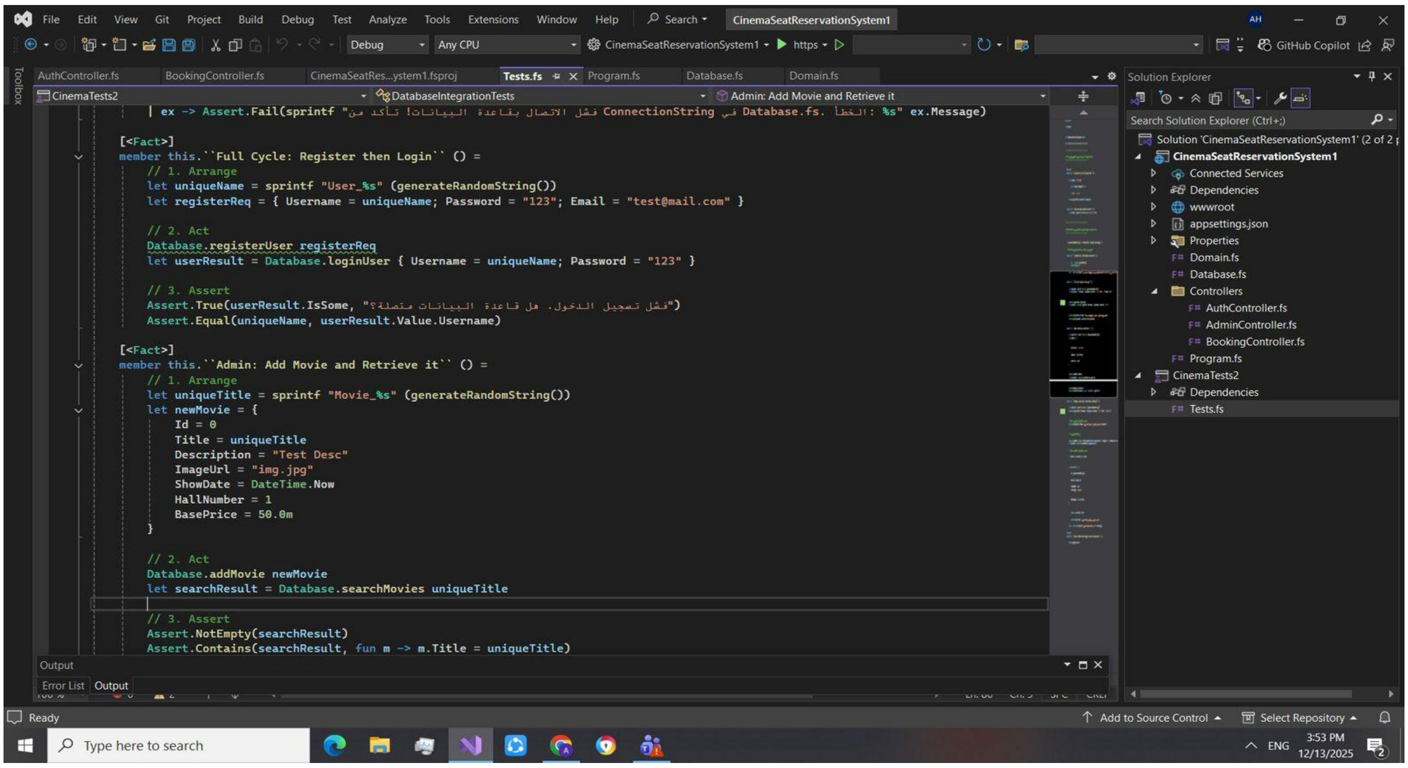
.NET Ecosystem Integration: xUnit is the industry-standard framework for .NET projects (C# and F#), ensuring seamless integration with the build environment.

Functional Focus: xUnit’s design, emphasizing Fact and Theory, perfectly aligns with the F# functional paradigm, allowing easy testing of pure functions with predictable outputs.

Immutability Testing: It facilitates the verification of functions that return new immutable data structures, which is core to the F# design philosophy.

Extensibility: It supports powerful F# testing extensions like FsCheck for advanced property-based testing.

Code Snippet:



```
            | ex -> Assert.Fail(sprintf "فشل الاتصال بقاعدة البيانات! تأكد من ConnectionString في Database.fs. الخطأ: %s" ex.Message)

    [<Fact>]
    member this.``Full Cycle: Register then Login`` () =
        // 1. Arrange
        let uniqueName = sprintf "User_%s" (generateRandomString())
        let registerReq = { Username = uniqueName; Password = "123"; Email = "test@mail.com" }

        // 2. Act
        Database.registerUser registerReq
        let userResult = Database.loginUser { Username = uniqueName; Password = "123" }

        // 3. Assert
        Assert.True(userResult.IsSome, "فشل تسجيل الدخول. هل قاعدة البيانات متصلة؟")
        Assert.Equal(uniqueName, userResult.Value.Username)

    [<Fact>]
    member this.``Admin: Add Movie and Retrieve it`` () =
        // 1. Arrange
        let uniqueTitle = sprintf "Movie_%s" (generateRandomString())
        let newMovie = {
            Id = 0
            Title = uniqueTitle
            Description = "Test Desc"
            ImageUrl = "img.jpg"
            ShowDate = DateTime.Now
            HallNumber = 1
            BasePrice = 50.0m
        }

        // 2. Act
        Database.addMovie newMovie
        let searchResult = Database.searchMovies uniqueTitle

        // 3. Assert
        Assert.NotEmpty(searchResult)
        Assert.Contains(searchResult, fun m -> m.Title = uniqueTitle)
```

Test Cases(8) :-

First of all, Our Core Strategy : The Test Pyramid

1. Business Logic Tests (Unit Tests)

- Calculate Price for Platinum Ticket: Verifies that the ticket pricing function applies the correct multiplier (2.0x) for the 'Platinum' category, ensuring accurate financial calculations within the Core Layer.

- Ticket ID Format Should Be Correct: Confirms that the resulting ticket identifier string adheres to the defined structure (e.g., starts with MOVXX-RXX-SXX-), validating the data formatting rules.

- Seat Layout: Should Generate 8x8 Matrix Logic: Checks the integrity of the in-memory seat map creation, ensuring it initializes with the correct 8x8 dimensions and allows for the successful simulation of a seat status update (e.g., from "Free" to "Booked").

2. Database Integration Tests (Infrastructure Layer Tests)

- A_Connectivity: Check Database Connection: A critical test to confirm the application can successfully establish a connection to the underlying SQL Server database, ensuring the entire Infrastructure Layer is operational.

- Full Cycle: Register then Login: Validates the core Authentication Service by testing the end-to-end user registration and subsequent login using the newly created credentials, confirming data persistence and validation work correctly.

- Admin: Add Movie and Retrieve it: Tests the administrative data management workflow: adding a new movie record and then querying the database to confirm its persistent storage and correct retrieval.

- Booking: Create User, Create Movie, then Book: This is the flagship integration test. It executes the full transaction: creating all necessary entities (user, movie), saving a ticket, and finally querying the database to assert that the specific seat is correctly marked as 'taken', validating the entire booking flow.

- Search: Should Return Empty for Non-Existent Movie: A robust test to ensure the search functionality handles missing data gracefully, returning an empty collection when querying for a non-existent movie title.

5. Core Feature Implementation Details

User Authentication:
Goal: Restrict access to the booking and management functionalities.
Implementation: The AuthController module contains functions to validate user credentials against a hardcoded list (or a simple file store) and return a status (e.g., Authenticated or AccessDenied).

Display Seating Map: A function iterates over the 2D array and uses pattern matching on seatMatrix to render the grid.

Select Seats: User inputs (SeatRow, SeatNumber) are parsed and validated against array bounds to prevent exceptions.

Prevent Double Booking: The BookTicket function explicitly checks

6. Team Roles and Module Mapping

1. **Seat Layout Architect:** Defines the initial dimensions and state of the cinema grid

2. **Display Developer:** Translates the seatMatrix[,] array into a user-friendly console visualization.

2. **Booking Logic Developer:** Implements the core business rules for reservation (BookTicket) and manages the immutable state change.

4. **Ticket System Developer:** Handles the generation of unique IDs and formatting of the ticket data.

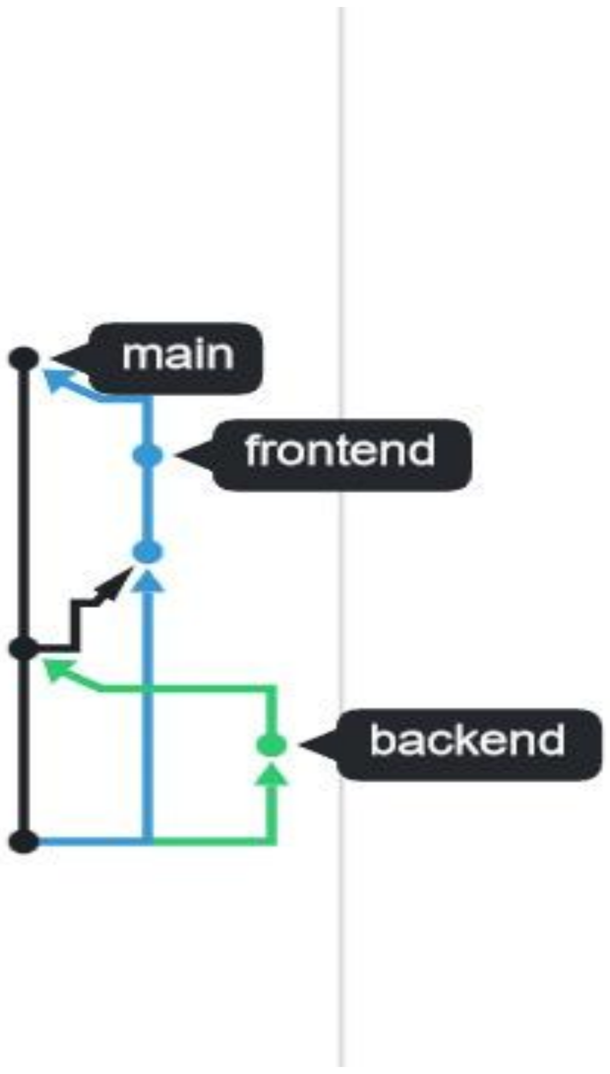
5. **File Storage Developer:** Manages the persistence of booked tickets to disk.

6. **UI Developer:** Implements the main menu, user input handling, and coordinates calls between the Auth and Booking modules.

7. **Tester:** Develops unit and integration tests using xUnit for all core functional units.

8. **Documentation Lead:** Compiles and maintains all project documentation and technical specifications.

GitHub Map


main
frontend
backend