ASSIGNMENT 2 - Group Project

Student ID	Student Email	Student Name
24681544	maksym.bui@student.uts.edu.au	Maksym Bui
25402140	Yi.Zhang-41@student.uts.edu.au	Yi Zhang
24821466	Jerry.fu@student.uts.edu.au	Jerry Fu

Table of Contents

1. Project Idea - Introduction

The Movie Ticket System project is a comprehensive web-based application designed to streamline the process of booking movie tickets, managing cinema operations, and enhancing the customer experience through digital engagement. The system is designed to support multiple cinemas, a variety of movie screenings, and flexible ticketing options, making it suitable for real-world deployment in cinema chains or independent theaters.

At its core, the project enables users to browse available movies, view detailed screening information, select seats, and purchase tickets securely. The ticketing logic supports different ticket types, such as adult, child, and concession, with dynamic pricing and surcharges for premium or accessible seating. The system includes an active deals system and a reward mechanism. Customers can enter promo codes during checkout to receive discounts, and a deal system automatically applies discounts on eligible movies.

The Movie Ticket System has a simple and intuitive user interface that makes booking tickets easy for everyone. The design follows common standards, so users can quickly find what they need and complete their bookings without confusion.

Users can browse movies imported from The Movie Database (TMDB), filter by cinema & session, select seats, apply promo codes and complete secure bookings.

On the server-side, the minimal-API architecture exposes an endpoint for catalogues, screenings, bookings, deals, rewards, authentication and messages. Data is stored in readable JSON snapshots. Security relies on JWT tokens, PBKDF2 hashing and a semaphore-protected repo that prevents concurrent write corruption.

The environment is containerised within nixflakes so that all SDKs, Node, Electron and GTK/X11 dependencies are installed deterministically on any distro, be it Linux, WSL or macOS.

There is an imperative script (`start.sh`) that builds the .NET API, bundles the React front-end and launches the Electron desktop shell.

2. Development

2.1 Environment management - Nix & Flakes

We used Nix with Devshell and Flakes to make sure everyone on the team has the same development environment. Nix is a package manager that allows us to define all dependencies and tools in a single configuration.

To ensure each developer built and ran the project under identical conditions, the toolchain / environment was containerised with Nix Flakes. These provide the full stack so that the host computer doesn't also need it installed. This means all the tools and dependencies are set up automatically, so there are no issues with things working differently on different computers.

All toolings and dependencies are declared in a nix flake and used with devshells for identical build environments.

2.2 Electron

Initially we went with WinForms, it was a 'workable' option and worked fine for non-window OS's with Wine emulation, but the styling options were miserable. Hence the swap to something that wasn't so locked. For the desktop app, we moved to Electron instead of WinForms. Electron lets us build cross-platform desktop apps while using web technologies like HTML, CSS, and JavaScript. This means our desktop app works on Windows, macOS, and Linux without needing separate codebases and has modern styling frameworks (react.)

2.3i Backend .NET

The backend is built with ASP.NET Core (.NET 8), providing RESTful APIs for all business logic, data management, and authentication. Data is stored in JSON files, which simplifies persistence and makes the system easy to understand and maintain, especially for academic or small-scale use. TMDB provides the movie data, and the other files are pre-seeded to emulate the rest of the environment.

- BookingService - Seats, quotas, bookings
- PricingService - Surchages, discounts, rewards
- DealService - promotions
- RewardService - rewards
- Datastore - functions for structured json

2.3ii TMDB API

We query and cache the The Movie Database API[1] locally in our project to snapshot up-to-date movie information, including titles, posters, and details about each film. Utilizing the TMDB API not only streamlines the management of movie data but also significantly enhances the user experience. Our application provides a visually rich catalog featuring authentic movie images and detailed information. The inclusion of real-time data and professional-quality assets makes the application feel polished and credible, closely resembling the standards of commercial cinema websites.

2.4 Frontend React TypeScript

On the frontend, the application is developed using React 18 and TypeScript. The UI leverages Mantine, a React component library, for consistent styling and interactive elements. React Query is used for efficient data fetching and caching, enabling smooth communication with the backend APIs.

Authentication is handled via JWT (JSON Web Tokens), supporting secure login and role-based access for both users and administrators. The system also includes unit tests written with NUnit to validate backend logic and ensure reliability.

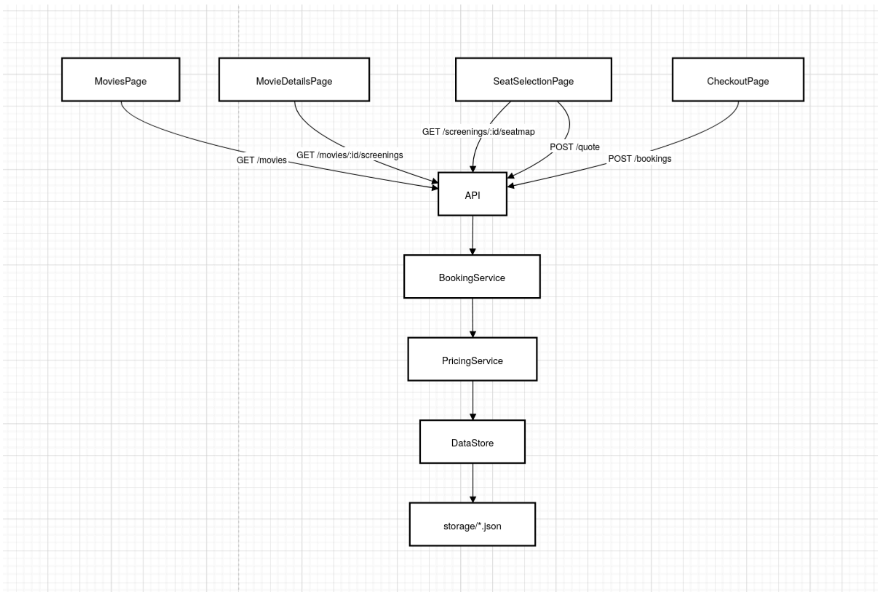
- MoviesPage - catalogue page
- MovieDetailsPage - session listing
- SeatSelectionpage - seat maps and ticket counter
- CheckoutPage - customer details, promo code, validation
- DealsPage, ManageDealsPage - Admin deal management
- MailBoxPage - message inbox
- LoginPage, RegisterPage, MyOrdersPage - Auth and order history

2.5 Git

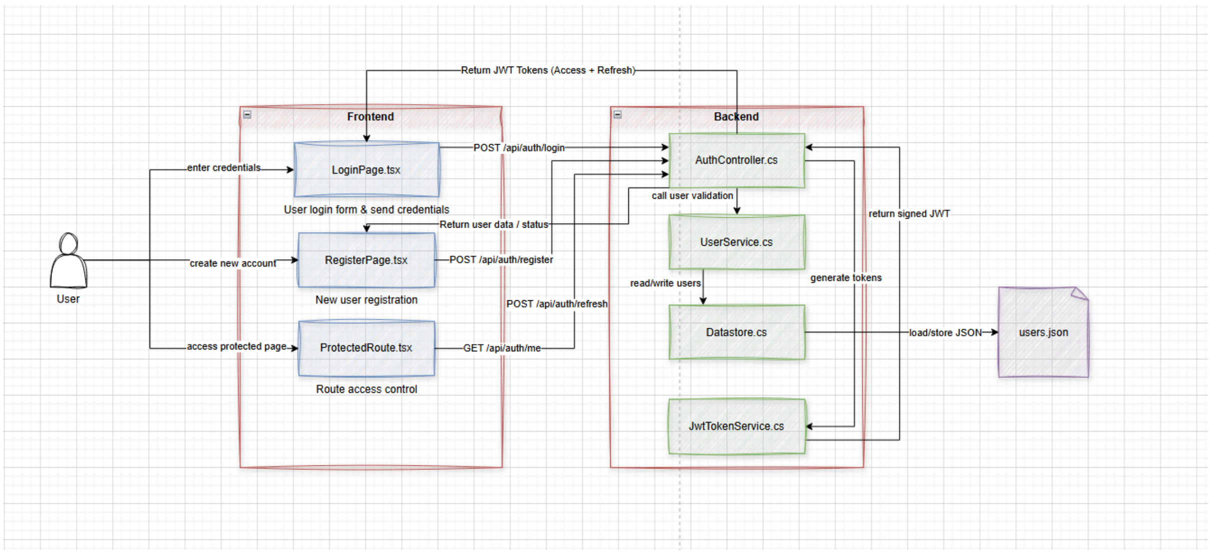
We used Git for version control throughout the development of our project. Git allowed our team to collaborate efficiently, track changes, and manage different features and bug fixes in separate branches. This made it easy to review code, resolve conflicts, and maintain a clear history of our work.

3. Flow Charts

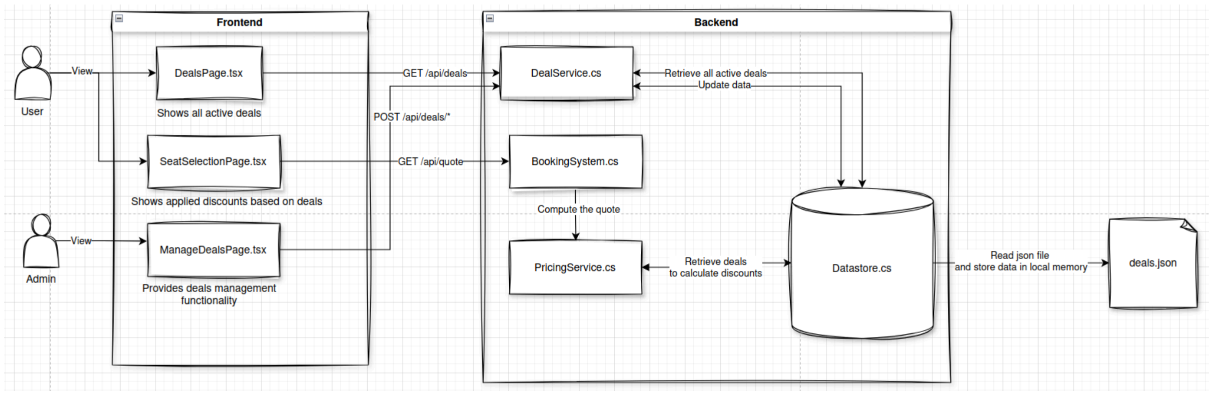
3.1 Booking System



3.2 Authentication System



3.3 Deal System



4. Responsibilities

Student	Responsibilities
Maksym Bui	- Reward System - Deal System - TMDB API integration - Notification system
Jerry Fu	- Booking System - Design full stack architecture - Setup environment for development
Yi Zhang	- Prepare Github repository for version control - Authentication System - Implementation of JWT tokens

Maksym Bui:
- Deal domain: Entity + persistence + API + pricing hook (`DealService`, API endpoints, pricing integration).
- Rewards: Earn/redeem flow + confirmation inbox (`RewardService`, `MessageService`)
- Admin interfaces: Deals management console
- Testing/Quality: NUnit `DealServiceTests`

Jerry Fu:
- Environment Stage: `flake.nix` Dotnet, node, electron `desktop/src/main.ts`, wine (legacy), runner (`start.sh`) script.
- Architecture booking pipeline: Seat-validation -> quote replay > notification (`BookingService, PricingService`
- Frontend movie sessions: Movie catalogue -> session selection -> seat selection (`MoviesPage`, `SeatSelectionPage`, `CheckoutPage`, `utils/ticketing.ts`
- Data generation: randomised pre-seeding data and snapshotting the TMDB

Yi Zhang:
- Security: JWT Token, PBKDF2 Hash and salt
- Auth API: register, login, auth/me, order history
- Frontend account flows: login/register/my-orders with hard guards forms and routing

Appendix

Project Github Repo: https://github.com/maksymbui/MovieTicketSystem

References

1. The Movie Database (TMDB). (n.d.). Getting started. Retrieved October 19, 2025, from https://developer.themoviedb.org/docs/getting-started