UTS

UNIVERSITY OF TECHNOLOGY SYDNEY

32998 .NET APPLICATION DEVELOPMENT - SPRING 2025

Assignment 2 Report – UniPlanner: A Student Timetable Management System

Prepared by

Seoyoon Kim (25388442)

Jin Lee (25388733)

Oct 16, 2025

Contents

1 Introduction and Summary of Project

UniPlanner is a Windows desktop application developed using C# and .NET 8.0, designed to help university students efficiently manage their academic and personal responsibilities in a single integrated system. The application consolidates several essential components of student life—such as subject registration, class scheduling, assignment tracking, and personal to-do management—into one cohesive and intuitive platform.

Unlike most web-based planners that depend on continuous internet connectivity or third-party synchronization, UniPlanner is built to operate entirely offline. It leverages a local SQLite database to store all user information securely, ensuring both data persistence and privacy without exposing sensitive academic data to online servers. This makes it particularly reliable for students working across multiple environments or offline conditions.

The system adopts a modular three-layered architecture, separating:

- Forms (UI) – handle all user interactions and input validation
- Services (Logic) – perform data manipulation, validation, and domain-specific logic
- Models (Data) – represent entities and enforce consistent structures through abstract base classes and interfaces

Each module—Subjects, Schedule, Tasks, and Todos—is implemented as a dedicated Form connected to its respective Service class. The Subjects module allows color-coded subject registration with unique codes and instructor details. The Schedule module provides weekly timetable management, including time-conflict detection and subject normalization via service-level validation. The Tasks module manages assignments with due dates, priorities, and links to subjects for better academic tracking. Finally, the Todos module, implemented using Entity Framework Core 8.0, enables students to manage non-academic goals with persistent storage and completion tracking through ORM-based data access.

Internally, UniPlanner employs a hybrid data access approach:

- Dapper with SQLite is used for Subject, Schedule, and Tasks to allow explicit SQL control and lightweight performance.
- Entity Framework Core 8.0 (EF Core) is applied for Todos, providing ORM flexibility and simplified CRUD handling through `UniPlannerContext`.

Database initialization and migration are handled by a custom `DbBootstrap` class, which ensures all required tables exist at startup and automatically adds missing columns (`Tasks.SubjectName`, `Todos.ModifiedDate`) to maintain compatibility across versions.

From a software engineering perspective, UniPlanner demonstrates:

- Event-driven GUI development using Windows Forms
- Data persistence and runtime schema management with SQLite
- Application of object-oriented principles (Separation of Concerns, Single Responsibility Principle, and interface abstraction)
- LINQ queries and generic collections for filtering and sorting
- Real-time input validation, error handling, and user feedback mechanisms
- Localized date/time formatting for Australian users (culture: `en-AU`)

Overall, UniPlanner represents a well-structured, maintainable, and privacy-conscious desktop application that mirrors real-world software engineering practices. Its modular design, hybrid persistence strategy, and offline capability make it both technically robust and user-centric, effectively supporting students in achieving better academic organization and productivity.

2 Development Approach and Technologies

2.1 Environment and Tools

Category	Details
Programming Language	C#
Framework	.NET 8.0 (Windows Forms)
Database	SQLite (local file-based storage)
Data Access Libraries	Dapper (Subjects, Schedule, Tasks) and Entity Framework Core 8.0 (Todos)
Integrated Development Environment (IDE)	Visual Studio 2022
External Libraries	Microsoft.Data.Sqlite, Dapper, Microsoft.EntityFrameworkCore 8.0.11, Microsoft.EntityFrameworkCore.Sqlite 8.0.11, Newtonsoft.Json
Version Control	GitHub (master-branch collaboration with in-person code integration)
Testing	Manual UI-based CRUD validation and input error handling
Localization	English (Australia) en-AU for date and time formatting

Table 1: Development Environment and Tools used in UniPlanner

2.2 Architectural Design

The UniPlanner system follows a three-layered architecture that separates the presentation, business logic, and data management responsibilities. This separation enhances maintainability, scalability, and testability.

- Presentation Layer (Forms) – manages user interface interactions, event handling, and validation feedback.
- Business Logic Layer (Services) – implements validation, CRUD operations, and domain-specific rules.
- Data Layer (Models + SQLite) – defines entity structures and handles database persistence through both Dapper and Entity Framework Core 8.0.

Each module (Subjects, Schedule, Tasks, Todos) is managed by a dedicated Service class that encapsulates database communication and validation logic. This structure ensures that all database operations remain independent from the user interface, following the principles of separation of concerns (SoC) and single responsibility.

2.3 Hybrid Data Access Implementation

UniPlanner adopts a hybrid persistence strategy that combines direct SQL control and ORM convenience:

- **Dapper with SQLite:** used for Subjects, Schedule, and Tasks. Dapper provides explicit SQL queries and lightweight performance while allowing flexibility in data mapping.
- **Entity Framework Core 8.0:** used for the Todos module. EF Core provides modern object-relational mapping (ORM) for simplified CRUD operations through the UniPlannerContext DbContext class, offering better performance and cross-platform compatibility with .NET 8.0.

The connection string is dynamically resolved at runtime using `|DataDirectory|`, allowing the application to operate seamlessly across different environments without requiring manual configuration.

2.4 Database Initialization and Migration

The database schema is automatically managed by the `DbBootstrap` class, which ensures all necessary tables exist upon startup. It also performs lightweight runtime migrations to keep existing user data compatible with the latest schema version.

- Automatically creates four main tables: `Subjects`, `Schedule`, `Tasks`, and `Todos`.
- Adds new columns when missing, such as `Tasks.SubjectName` and `Todos.ModifiedDate`.
- Performs data migration to populate `SubjectName` in existing task records by joining with the Subjects table.
- Creates database indexes for faster query performance on frequently accessed fields.

This approach removes the need for external migration tools and ensures backward compatibility, allowing older database versions to remain usable even after feature updates.

2.5 Error Handling and Validation Strategy

All service operations are wrapped in structured **try–catch** blocks to prevent unexpected application crashes. Invalid inputs and duplicate entries trigger user-friendly messages and validation prompts. Each Form performs pre-save checks to ensure consistent data integrity and predictable application behavior.

2.6 Localization and Culture Configuration

The application explicitly sets the culture to `en-AU` at startup. This defines consistent date formats such as `dd-MM-yyyy`, localized day names, and calendar ordering based on Australian academic standards. This ensures that all user inputs, date comparisons, and schedule sorting behave consistently across systems.

2.7 Development Process

The system was developed iteratively, with each feature designed, tested, and refined before integration:

1. Database schema and table design
2. Service layer implementation using Dapper and EF6
3. UI Form development and event binding
4. Input validation and user feedback handling
5. Integration of LINQ-based search and sorting
6. Manual testing and debugging through the UI

Although GitHub branches were available, most progress was achieved through direct collaboration. Both team members met regularly to write code, perform real-time screen sharing, and merge updates together, ensuring code consistency and rapid feedback.

Overall, the development approach emphasizes modularity, data reliability, and maintainability, achieving a balance between lightweight performance and long-term flexibility.

3 System Flowchart

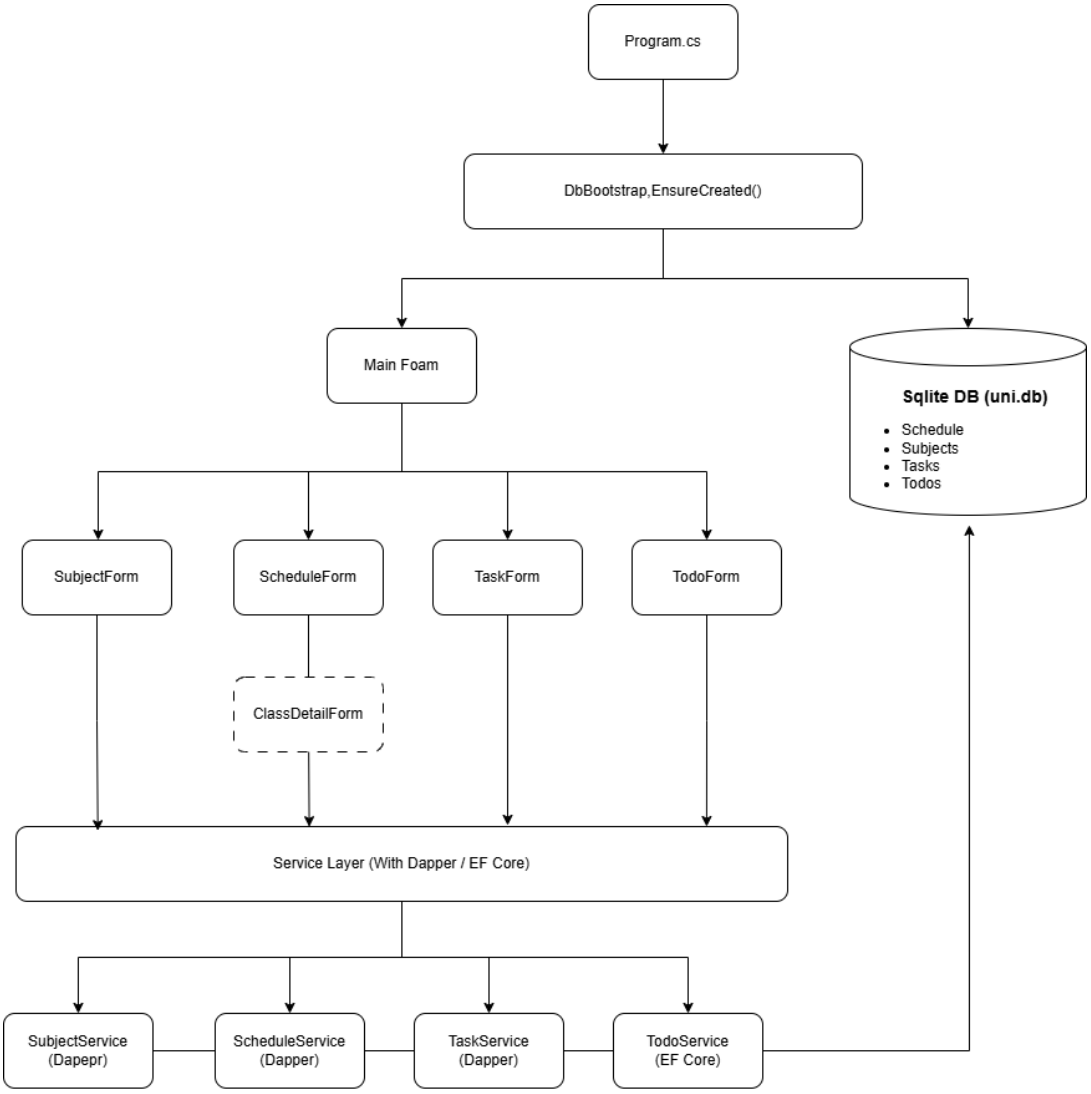

Figure 1: System Flowchart of UniPlanner

Each feature within UniPlanner is implemented as an independent module to ensure modularity, maintainability, and clear separation of responsibilities.

The MainForm serves as the central hub of the application, connecting four main functional components:

- SubjectForm
- ScheduleForm
- TaskForm
- TodoForm

Each Form is responsible for user interaction within its feature scope and directly communicates with its corresponding Service class:

- SubjectForm → SubjectService
- ScheduleForm → ScheduleService
- TaskForm → TaskService
- TodoForm → TodoService

These Service classes handle all business logic, input validation, and database communication. While the SubjectService, ScheduleService, and TaskService rely on the Dapper library for lightweight SQL operations, the TodoService leverages Entity Framework Core 8.0 (EF Core) for object-relational mapping (ORM). This hybrid approach balances performance efficiency and development simplicity—Dapper is used where speed and explicit control are important, while EF Core simplifies CRUD management for smaller modules.

Each Service interacts with a shared SQLite database (`uni.db`), but manages its own dedicated table:

- `Subjects` for subject details
- `Schedule` for weekly timetables
- `Tasks` for assignments and deadlines
- `Todos` for personal task tracking

At application startup, the DbBootstrap class initializes the SQLite database if it does not already exist. It automatically creates required tables, verifies schema integrity, and performs light migrations such as:

- Adding missing columns (e.g., `Tasks.SubjectName`, `Todos.ModifiedDate`)
- Automatically migrates existing task records by back-filling SubjectName from the Subjects table via SQL JOIN operations to maintain data consistency when the schema is updated

This ensures that UniPlanner can be executed immediately without any manual setup, while maintaining backward compatibility if the database structure evolves.

All database connections are configured dynamically through the `App.config` file using the `|DataDirectory|` parameter. This allows the data directory to adapt automatically to the

runtime environment—whether the application is launched in Visual Studio or distributed as a standalone executable.

Finally, the flow of data within UniPlanner follows a clear top-down architecture:

1. The user interacts with the Form (UI layer)
2. The Form delegates logic and persistence to its Service (business layer)
3. The Service executes read/write operations on the SQLite database (data layer)
4. The DbBootstrap utility ensures schema reliability at every run

This design enforces a strong separation of concerns, minimizes coupling, and keeps the system extensible for future enhancements such as analytics or cloud synchronization.

4 Role of Team Members

Team Overview

The UniPlanner project was collaboratively developed by two team members: Jin Lee and Seoyoon Kim. Both members contributed to all stages of the project, including planning, design, coding, testing, and documentation. While GitHub was used for version control and tracking commits on the master branch, the majority of development occurred through in-person collaboration sessions. During these sessions, both members worked side by side to design, implement, and test code, frequently sharing their screens and editing the same files in real time.

Roles and Responsibilities

Team Member	Main Role	Contributions
Jin Lee	Backend and Database Logic	Designed and implemented the database schema and the `DbBootstrap` initializer. Developed the `SubjectService`, `ScheduleService`, and `TaskService` classes. Integrated Dapper with SQLite, implemented TodoService using Entity Framework Core 8.0 for ORM-based persistence, structured LINQ queries for filtering and sorting, and managed data validation and exception handling. Contributed to the organization of project architecture and documentation consistency.
Seoyoon Kim	Frontend and Testing	Designed and implemented the Windows Forms interface (`MainForm`, `ScheduleForm`, `TodoForm`, and related dialog forms). Connected UI components to backend logic using event-driven programming. Performed functional and integration testing, validated user inputs, and refined interface interactions. Assisted in preparing presentation materials and the final report.

Table 2: Team roles and responsibilities in UniPlanner development

Individual Responsibilities

Jin Lee (Backend and Structure Design)

Jin focused on establishing the application's internal structure and backend logic. He implemented the automatic database initialization process and defined a consistent data access layer using Dapper. He also ensured that LINQ queries and lambda expressions were optimized for data manipulation. Jin contributed to the creation of reusable components and wrote technical documentation describing database workflows and schema evolution.

Seoyoon Kim (UI Design and Testing)

Seoyoon took the lead in designing an intuitive and cohesive user interface. She ensured that the Windows Forms layout was consistent across modules and provided smooth user interaction. Her testing efforts included verifying CRUD operations, resolving UI-event bugs, and improving form responsiveness. She also coordinated visual design elements, ensuring the application's usability and clarity.

Collaboration Process

Throughout the project, both members maintained close collaboration. Although GitHub was used for commit tracking and version merging on the master branch, most coding and debugging were performed collaboratively during in-person sessions. The team shared screens, edited files together, and verified changes in real time. This workflow allowed for fast iteration, consistent coding conventions, and immediate resolution of design or logic conflicts. The result was a coherent and well-integrated application that reflects both members' strengths in software design and teamwork.

5 Acknowledgments and AI Usage

We would like to thank our tutor and the UTS teaching staff for their continuous support and valuable feedback. Additionally, **ChatGPT (GPT-5)** was used in a limited capacity for:

- Project ideation and feature brainstorming
- Folder structure and architecture planning
- Documentation refinement and grammar clarity

All source code, database logic, and user interface development were completed manually by the team.

References

- Microsoft Docs (2025). *Entity Framework 6 using System.Data.SQLite*. Retrieved from https://learn.microsoft.com/en-us/dotnet/
- Dapper Official Documentation (2025). *Dapper – a simple object mapper for .NET*. Retrieved from https://github.com/DapperLib/Dapper
- SQLite Documentation (2025). *SQLite Documentation*. Retrieved from https://www.sqlite.org/docs.html
- Microsoft Learn (2025). *Visual Studio 2022 Documentation*. Retrieved from https://learn.microsoft.com/visualstudio/