ASSIGNMENT COVERSHEET

UTS: ENGINEERING & INFORMATION TECHNOLOGY

SUBJECT NUMBER & NAME	NAME OF STUDENT(s) (PRINT CLEARLY)	STUDENT ID(s)
31927 Application Development with .NET	*Jianan Huang* *Raymond Huang* *SURNAME* *FIRST NAME*	25098548 24970737

STUDENT EMAIL	STUDENT CONTACT NUMBER
Jianan.Huang-3@student.uts.edu.au *Raymond.Huang-1@student.uts.edu.au*	*0430590617* *0405225371*

NAME OF TUTOR	TUTORIAL GROUP	DUE DATE
Zohair Gandhi	*03*	*20/10/2025*

ASSESSMENT ITEM NUMBER & TITLE

ASSIGNMENT-2: Group Project

- ✓ I confirm that I have read, understood and followed the guidelines for assignment submission and presentation on page 2 of this cover sheet.
- ✓ I confirm that I have read, understood and followed the advice in the Subject Outline about assessment requirements.
- ✓ I understand that if this assignment is submitted after the due date it may incur a penalty for lateness unless I have previously had an extension of time approved and have attached the written confirmation of this extension.

Declaration of originality:

We declare that the title of "ToDoListProgram" and the accompanying source code are entirely our own work, except where explicit references are made. No part of this submission has been previously published or submitted for assessment in any course or institution.

Statement of collaboration:

We declare that this project was completed collaboratively by the team. Each team member contributed substantively and reviewed the final submission.

Signature of student(s) J.H. [illegible] **Date** 20/10/25

Introduction

Although there are numerous modern task-management applications, many charge for advanced capabilities, or require an internet connection. “Smart-ToDo” is an ultra-lightweight, self-hosted Blazor-Server solution that provides users and small teams with a role-based checklist manager that has zero deployment friction. Authorised users can create, prioritise and schedule tasks, while administrators can also manage new users and inspect or edit each user’s workload. Data is persisted locally in JSON files, so there is no external database requirement in development or small scale production.

The application is intended for use by two user personas:

- ➢ End user - Students, professionals, household members and any other who would like a fast, responsive experience for planning each day.
- ➢ Administrator staff - Supervisors or IT staff who must onboard new team members, reset passwords, and audit outstanding work.

The main differentiators are real-time two-way data binding (Blazor Server), adaptive UI that behaves correctly in desktop, tablet, and phone view-ports, and a privilege matrix that is enforced at both the UI and service layer. Task records include title, category, priority (high / medium / low), due-date / time, and completion flag. In addition to sorts by priority or temporal proximity, which is instant, the application assists in focusing on the next most important thing whether that is due presently, sometime soon or overdue. The complete code base is developed using C# 13 and .NET 9 so as to utilise the component-oriented Blazor model for the most re-use while preserving type safety.

From an educational perspective, the project does address/display many of the core object sensed principles (encapsulation, high cohesion, low coupling), interface-based design, polymorphic behaviours, and modern asynchronous patterns. The core extensibility hooks (generic repositories, interface segregation) are in place to make the next version capable of swapping out the JSON storage for SQLite or SQL server; however, neither modification nor testing will impact the presentation layer. Overall, the project delivers a practical tool that meaningfully streamlines day-to-day task management.

System Design

Technology

- .NET 9.0, C# 13
- Blazor, Razor components

Layers

- UI Layer:
  - Pages:
    - **Home.razor:** Display a welcome message when not logged in; after logging in, show tasks divided into 'about to expire' and 'completed', and support checking tasks as completed.
    - **Login.razor / Register.razor:** Login / Register; Successful registration redirects to login; Successful login redirects to homepage.
    - **Tasks.razor:** Task CRUD and sorting (by Priority / Due date); supports Admin to view and edit tasks of specified users.
    - **Admin.razor:** User Management (Add / Delete / Update / View, Reset Password, Manage Tasks)
    - **Error.razor:** Error handler.
  - Layout:
    - **MainLayout.razor:** The system’s main UI layout.
    - **NavMenu.razor:** The system’s navigation menu.
- Service Layer:
  - **UserService:** For user registration, login/logout, role verification, CRUD for users, password reset; JSON data persistence
  - **TodoService:** Tasks Add/Update/Delete/GetForUser, Task Sorting, JSON Data Persistence
- Data Layer:
  - Models:
    - **User:** Id, Username, Password, Role
    - **TodoItem:** Id, UserId, Title, Category, Priority (Default M), DueDate, IsCompleted
    - **Priority**: PriorityFactory, High(), Medium(), Low(), None()
  - JSON files:
    - **Users.json:** Stores users data.
    - **Tasks.json:** Stores tasks data of each user.

Development Approach

Tasks were organised into brief, test-led iterations where this week we completed a working vertical slice: first login/register, then task create/read/update/delete (CRUD), and then management of admin users. Before any production code was written, a failing NUnit test was created; the minimum implementation was added to make it pass, and then the implementation was refactored as long as the tests remained green. This cycle kept any regressions visible, while producing a Git history that resembled the order of feature development.

Architecture and Technologies

The fundamental architecture of the system behaves in .NET 9 with Razor Components (Blazor Server) as the primary framework. This enables server logic written in C# to be combined with web-based visualisation. A standard Dependency Injection was provided to register core classes such as UserService and TodoService, which are injected as singletons to facilitate testing and scalability. This provides a clean separation of concerns between C# components. For example, the user and task services adhere to their designated responsibilities and transparently implement the persistence mechanism via injected file stores.

Implementation Details

For reasonable ease of deployment and configuration, the data persistence implementation is file-based utilising System.Text.Json. User and task data are contained within two JSON files stored in the data directory (users.json and tasks.json) which are automatically generated. The files are loaded on application startup. Records (tasks) are written to the appropriate file upon user addition, modification, or removal (or task). This approach mitigates the need for redundant data to be maintained or complicated database configuration that would not be suitable for this use case.

User Interface and Design

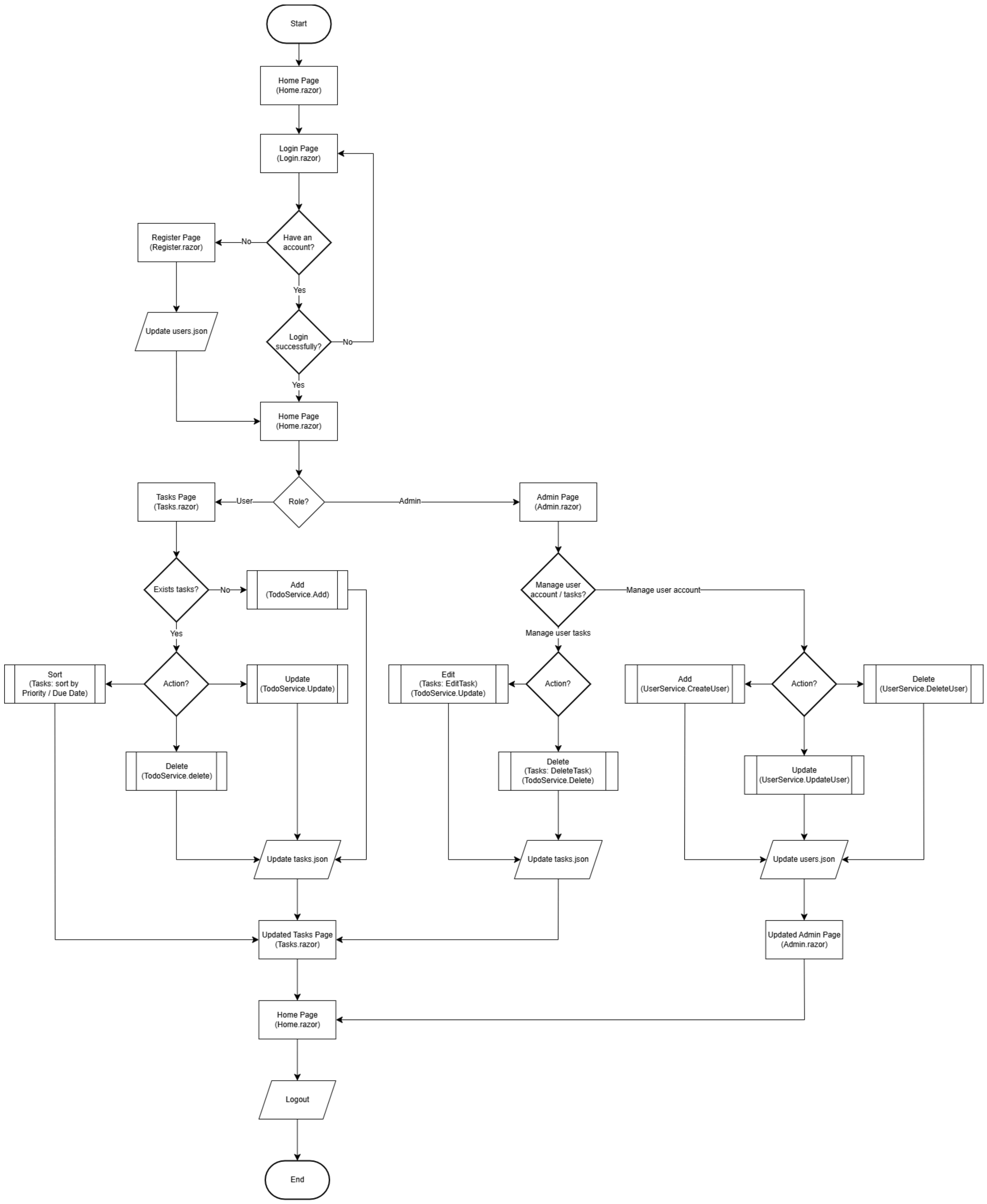
The User Interface design is fundamentally basic, making use of Bootstrap built in via Razor layouts on the front end. The main pages are Login.razor, Home.razor, and Admin.razor, and they were designed to be a basic responsive design. For user role control, the administrator can create, edit, and delete users from the application, and anyone authorised can freely add, remove, or edit their tasks.

Testing

The application was tested using NUnit as well as Moq and coverlet.collector. In short, it implements and validates critical core logic in UserService and TodoService: load/save data to/from repurpose and to trigger events within the system. Ultimately, it enables you to achieve spiralling reliable solid components and modify future development.

The development and source control took place within Visual Studio using GitHub – the reasoning is tightly based in the deep integration with debug/deploy stacks and the contributing simpler project management overall. In summary, the presented system shows good modularity, maintainability, and readability meaning perfect for university/practitioner development, or development with small businesses.

Flowchart (see page below)


Start
Home Page (Home.razor)
Login Page (Login.razor)
Have an account?
No
Register Page (Register.razor)
Update users.json
Yes
Login successfully?
No
Yes
Home Page (Home.razor)
Role?
User
Tasks Page (Tasks.razor)
Admin
Admin Page (Admin.razor)
Exists tasks?
No
Add (TodoService.Add)
Yes
Action?
Sort (Tasks: sort by Priority / Due Date)
Update (TodoService.Update)
Delete (TodoService.delete)
Update tasks.json
Manage user account / tasks?
Manage user account
Manage user tasks
Action?
Edit (Tasks: EditTask) (TodoService.Update)
Delete (Tasks: DeleteTask) (TodoService.Delete)
Update tasks.json
Action?
Add (UserService.CreateUser)
Delete (UserService.DeleteUser)
Update (UserService.UpdateUser)
Update users.json
Updated Tasks Page (Tasks.razor)
Updated Admin Page (Admin.razor)
Home Page (Home.razor)
Logout
End

Contributions

Smart-ToDo was delivered by a cohort of three, Jianan, Raymond, and Yue. Working in one-week sprints, with communications and planning through a Discord chat group. Each member crossed into neighbouring layers (UI, service, persistence, test). The table underneath summarises who did what through commits in their designated branches, merged onto the main Github repository branch.

Group Member	Contributions
Yue Song	- Created the Admin account logic and Admin.razor page. - Cleaned up the NavMenu.razor. - Started on the project flowchart.
Raymond Huang	- Developed main tasks function; CRUD operations (Create, Edit, Delete tasks). Task structure; title, categories, priority, and date/time to json. - Created Tasks.razor page. - Edited Home.razor page, able to view tasks in chronological order in separate tables for un/completed status, and included tickboxes. - In the Home.razor page, added a task search bar, filters for categories and priorities. - Modified Priority.cs to fulfil polymorphism; inheritance & overriding. - Checkboxes are now added to the Home.razor page. - Configured the NavMenu.razor for Admin accounts to be able to get to the Admin page. - Contributed on the documentation report; Introduction, Development Approach.
Jianan Huang	- Implemented Login & Register functionalities. - Implemented and reformatted all Service.cs models. - Added the function of sorting based on Priority & Due Date in Task.razor. - Implemented a new feature, allowing the admin to edit each user's tasks. - Added NUnit testing for TodoService & UserService. - Contributed to the System Design. - Improvement on the Flowchart.