31927 Application Development with .NET

Assignment Planner

Assessment Task 2: Programming Assignment - Report

Tom Golding: 14316196

24/10/2024

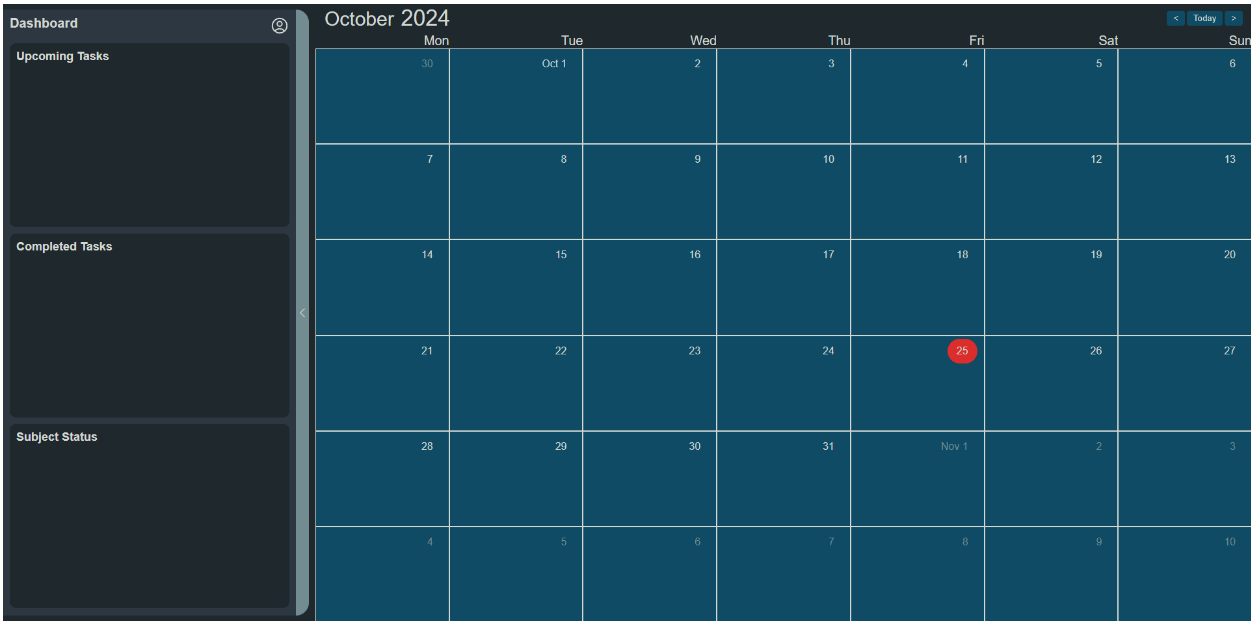
Table of Contents

Word Count: 1193 + whatever is in the images

Introduction

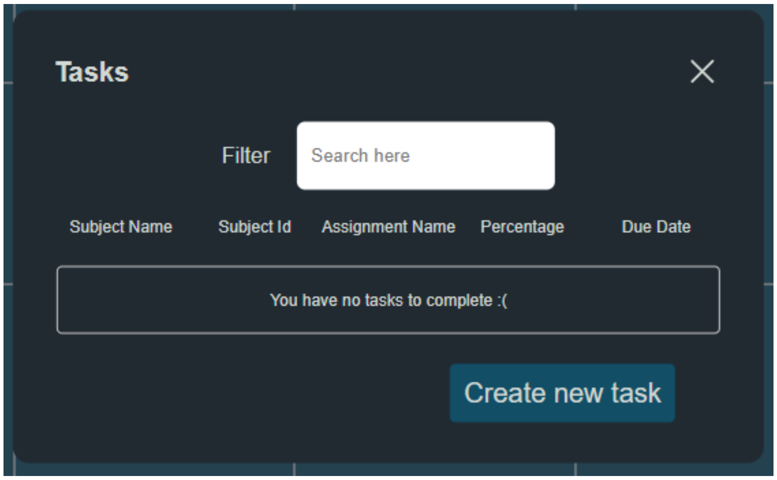
My application for Assessment Task 2 is a single-page assignment planner designed to help users manage their tasks efficiently. It features a user authentication system that allows users to register for an account and log in securely. Upon successful registration or login, users are directed to the dashboard, where the core functionalities of the application are presented in an intuitive layout.

The application adopts a modern single-page implementation, which enhances user experience by minimising page reloads. Additional screens and functionalities are seamlessly integrated through the use of modals, allowing users to access relevant features without navigating away from the dashboard, which can be seen in the figure below. The main layout includes a sidebar containing various sections for quick access to different functionalities, alongside the calendar that serves as the primary interface for task management.



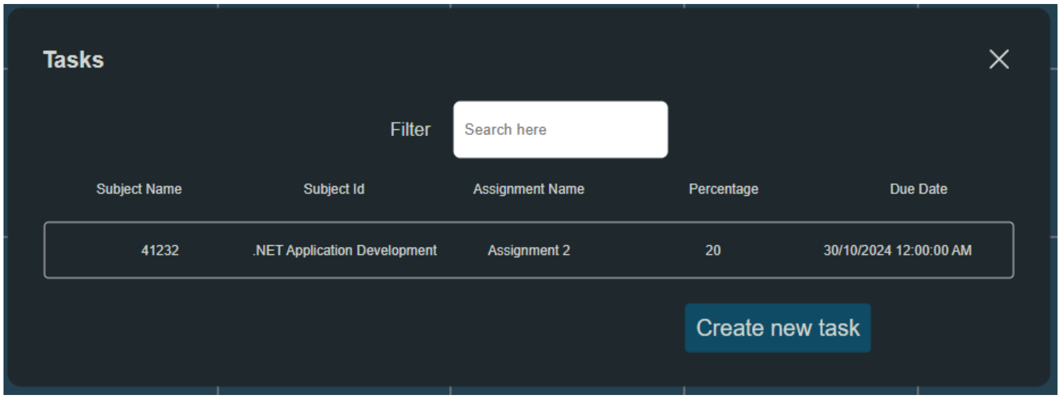
Mon	Tue	Wed	Thu	Fri	Sat	Sun
30	Oct 1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	Nov 1	2	3
4	5	6	7	8	9	10

The calendar allows users to navigate through each month effortlessly via the buttons along the calendar's top bar. Within the calendar’s main body, displays the appropriate days for the currently selected month. Clicking on a specific day opens a modal displaying all assignments scheduled for that date. This modal provides various options, including a "Create Task" button, which enables users to add new tasks directly. Additionally, selecting an individual assignment within the modal opens up detailed information, allowing users to edit or delete the task as needed. This streamlined approach to task management ensures that users can efficiently organise their assignments and stay on top of their responsibilities. Below is a set of figures displaying these UI elements that

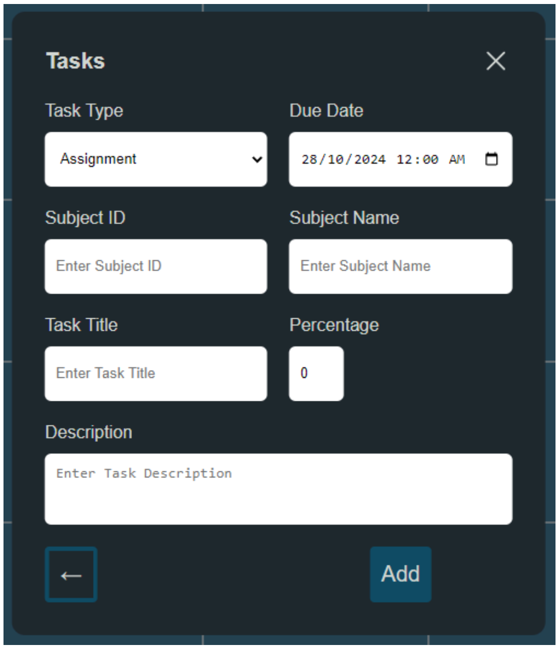


Subject Name	Subject Id	Assignment Name	Percentage	Due Date

You have no tasks to complete :(

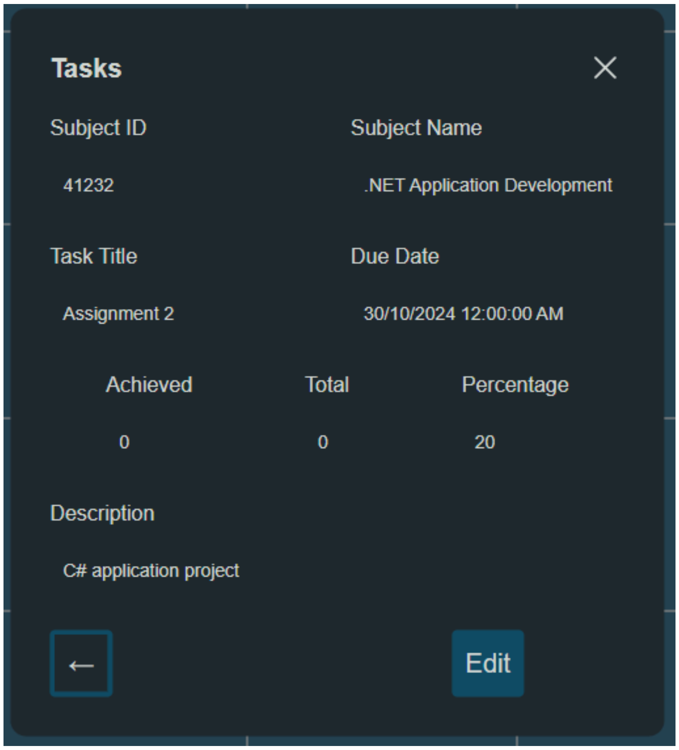


Subject Name	Subject Id	Assignment Name	Percentage	Due Date
41232	.NET Application Development	Assignment 2	20	30/10/2024 12:00:00 AM



Task Type: Assignment	Due Date: 28/10/2024 12:00 AM
Subject ID: Enter Subject ID	Subject Name: Enter Subject Name
Task Title: Enter Task Title	Percentage: 0

Description: Enter Task Description


Tasks

Subject ID: 41232

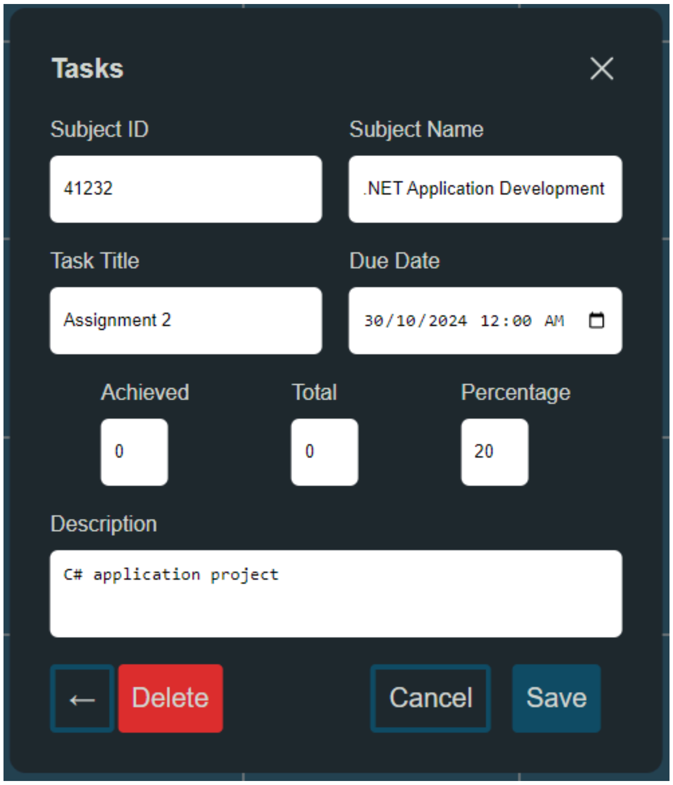
Subject Name: .NET Application Development

Task Title: Assignment 2

Due Date: 30/10/2024 12:00:00 AM

Achieved	Total	Percentage
0	0	20

Description: C# application project


Tasks

Subject ID: 41232

Subject Name: .NET Application Development

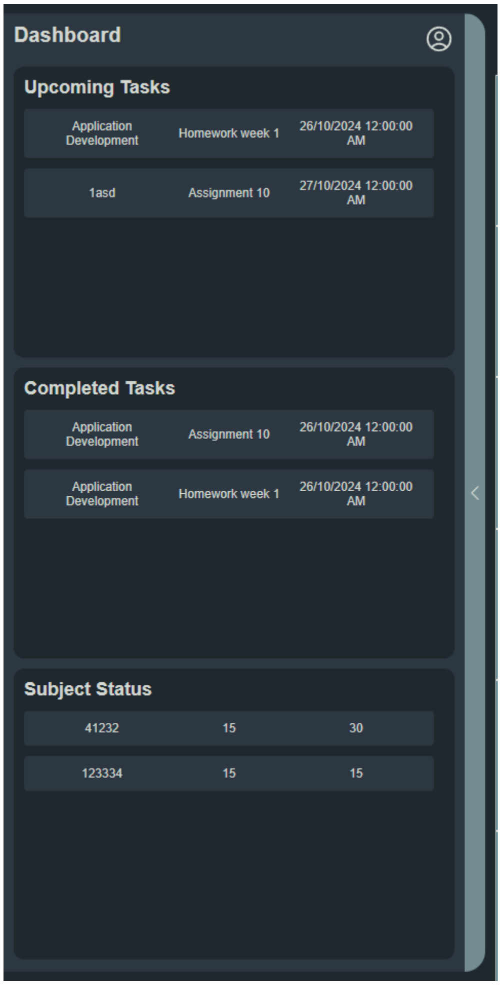
Task Title: Assignment 2

Due Date: 30/10/2024 12:00 AM

Achieved	Total	Percentage
0	0	20

Description: C# application project

The sidebar of the application is designed for easy navigation and efficient task management. It features sections for upcoming tasks and completed tasks, displaying the first ten upcoming tasks and the last ten completed tasks for quick reference. Additionally, there is a dedicated section for subject status, which organises tasks based on their subject ID and calculates the total percentage of completed assignments, providing users with a clear overview of their academic progress. At the top of the sidebar, an account icon serves as a gateway to account-related functionalities. When clicked, it opens the Account Information modal, where users can view their account details. This modal includes options for logging out, which promptly logs the user out and navigates them to the login page, and editing their account. Selecting the edit option opens another modal, allowing users to update their information or delete their account if desired. This thoughtful layout not only enhances user experience but also streamlines task and account management within the application. These UI elements can be viewed in the following figures.



Dashboard

Upcoming Tasks

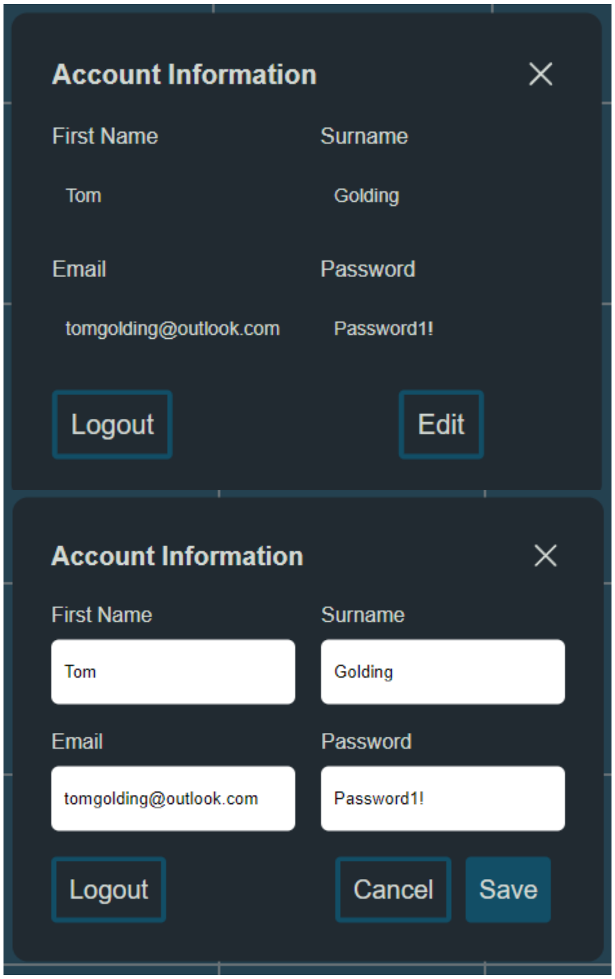
Application Development	Homework week 1	26/10/2024 12:00:00 AM
1asd	Assignment 10	27/10/2024 12:00:00 AM

Completed Tasks

Application Development	Assignment 10	26/10/2024 12:00:00 AM
Application Development	Homework week 1	26/10/2024 12:00:00 AM

Subject Status

41232	15	30
123334	15	15



Account Information

First Name: Tom
Surname: Golding
Email: tomgolding@outlook.com
Password: Password1!

Account Information

First Name: Tom
Surname: Golding
Email: tomgolding@outlook.com
Password: Password1!

Development approach

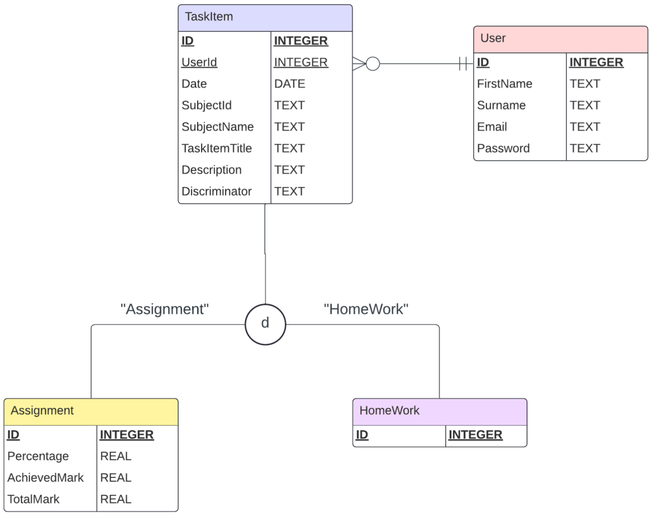
Frameworks

In developing my application, I employed the Blazor framework, which allowed me to build a modern and interactive web application with C# and .NET. Blazor's client-server model enabled a smooth user experience by handling much of the application logic on the client side while maintaining server-side interaction for data management. I leveraged several key packages, including Microsoft.EntityFrameworkCore (version 8.0.10), which provided robust data access using Entity Framework Core. This included EntityFrameworkCore.SqlServer for SQL Server integration and EntityFrameworkCore.Sqlite for SQLite support, both crucial for managing databases in different environments. Additionally, tools such as EntityFrameworkCore.Tools and EntityFrameworkCore.Design supported database migrations and schema updates.

For the client-server interaction, I used Microsoft.AspNetCore.Components.WebAssembly.Server, ensuring seamless communication between the Blazor WebAssembly front end and the backend services.

Before choosing Blazor, I considered using Windows Forms, which is well-known for desktop application development. However, upon reviewing its capabilities, I found that it was outdated for my specific needs in building a web-based, cross-platform application. Windows Forms lacked the flexibility and modern web-focused features that Blazor provides, leading me to conclude that Blazor was the superior choice for a scalable, interactive web application.

Development



After choosing Blazor as the framework, I began by sketching out an idea for my application, focusing primarily on the data models that would be used. This step helped solidify the structure of the data and the relationships between entities, ultimately guiding the development of the following Entity Relationship Diagram (ERD). The ERD provided a clear visual representation of how different components of the application would interact with the database, ensuring that the design was scalable and aligned with the application's needs.

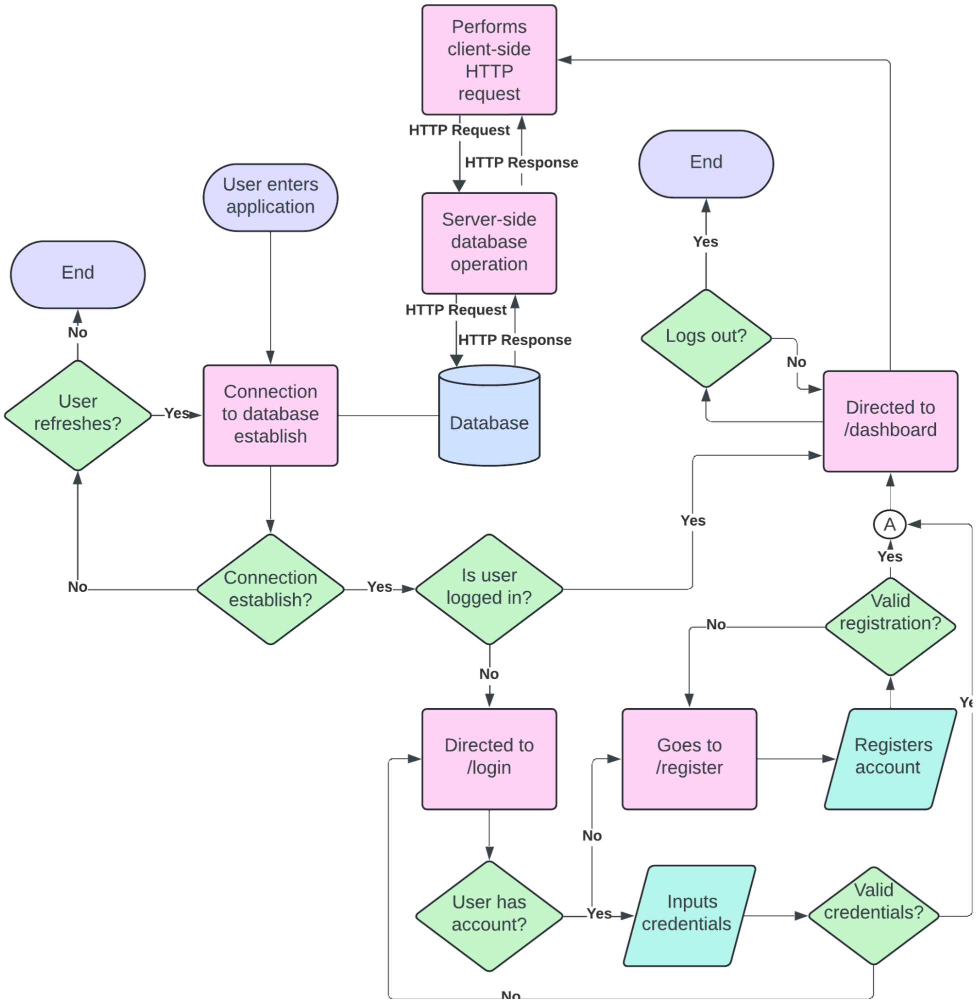
For the user interface (UI), my approach was to keep things simple and efficient. I sketched out the UI using basic HTML and CSS, which allowed for rapid prototyping and iteration. This approach enabled me to quickly test and refine the layout and functionality before committing to more complex design elements. Using simple web technologies also ensured that the UI was responsive and easily adaptable to the dynamic nature of the Blazor framework, providing a strong foundation for future enhancements.

After prototyping the UI with simple HTML and CSS, I transitioned to Blazor, where I refined the design by splitting the interface into reusable components. This component-based structure allowed for cleaner, more modular code, making the UI easier to maintain and update. Once the structure was set, I integrated the database and added functionality for data manipulation on the client side.

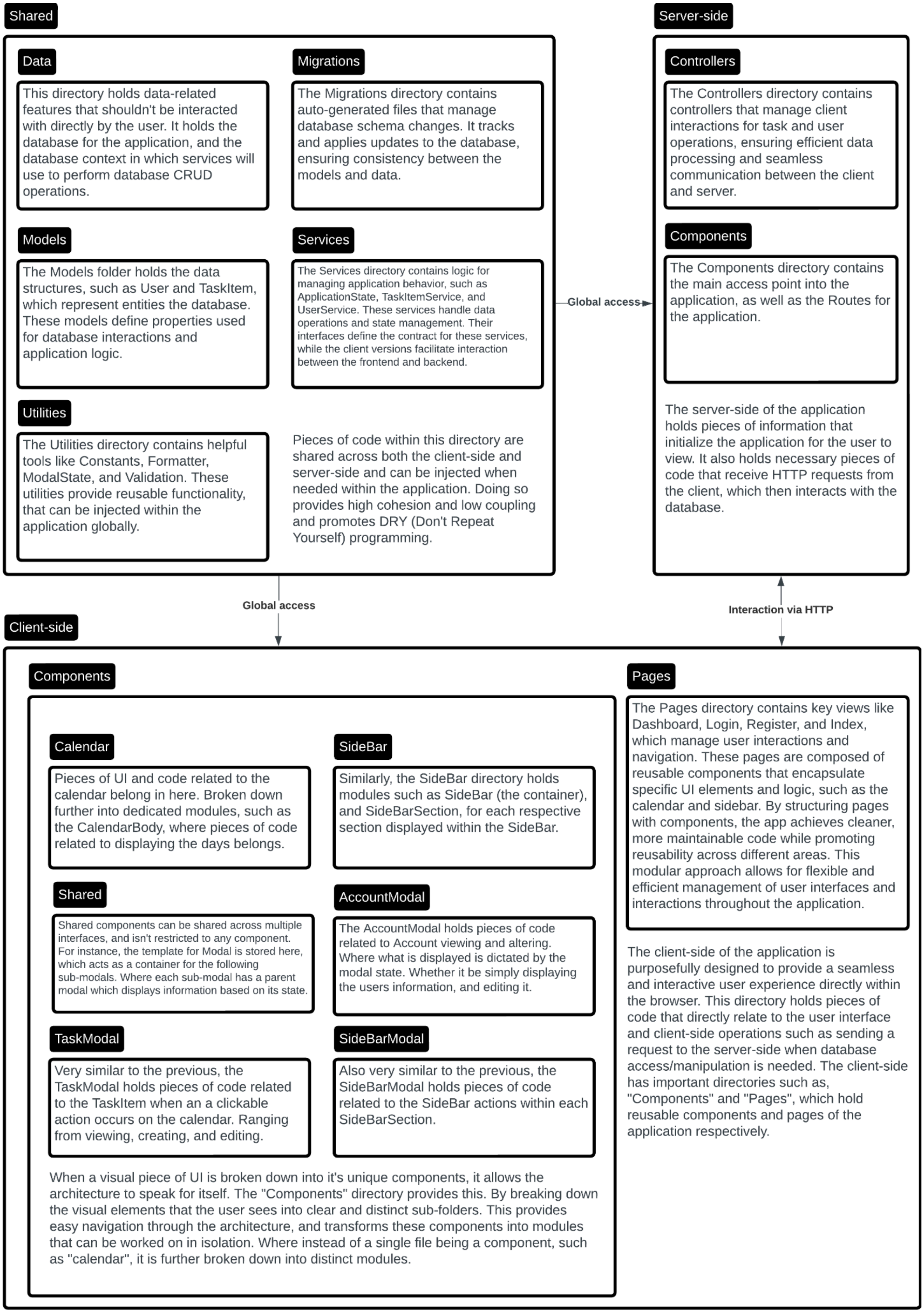
From this point, the development process primarily involved maintaining and refactoring the existing codebase, ensuring that the application remained efficient and easy to navigate. I focused on enhancing functionality by implementing features like validation, error handling, and improved data presentation. This iterative process would not only improve the user experience but also ensure that the application was scalable and adaptable to future requirements. By continually refining the components and their interactions, I was able to create a robust and flexible application that would meet the needs of my project goal while simplifying ongoing development efforts.

Flowchart

The figure below provides a high-level overview of the flowchart, capturing the main stages and decision points in the process.



The figure below illustrates the architecture of the codebase, detailing the contents and purpose of each directory.



Shared

Data

This directory holds data-related features that shouldn't be interacted with directly by the user. It holds the database for the application, and the database context in which services will use to perform database CRUD operations.

Migrations

The Migrations directory contains auto-generated files that manage database schema changes. It tracks and applies updates to the database, ensuring consistency between the models and data.

Models

The Models folder holds the data structures, such as User and TaskItem, which represent entities the database. These models define properties used for database interactions and application logic.

Services

The Services directory contains logic for managing application behavior, such as ApplicationState, TaskItemService, and UserService. These services handle data operations and state management. Their interfaces define the contract for these services, while the client versions facilitate interaction between the frontend and backend.

Utilities

The Utilities directory contains helpful tools like Constants, Formatter, ModalState, and Validation. These utilities provide reusable functionality, that can be injected within the application globally.

Pieces of code within this directory are shared across both the client-side and server-side and can be injected when needed within the application. Doing so provides high cohesion and low coupling and promotes DRY (Don't Repeat Yourself) programming.

Server-side

Controllers

The Controllers directory contains controllers that manage client interactions for task and user operations, ensuring efficient data processing and seamless communication between the client and server.

Components

The Components directory contains the main access point into the application, as well as the Routes for the application.

The server-side of the application holds pieces of information that initialize the application for the user to view. It also holds necessary pieces of code that receive HTTP requests from the client, which then interacts with the database.

Client-side

Components

Calendar

Pieces of UI and code related to the calendar belong in here. Broken down further into dedicated modules, such as the CalendarBody, where pieces of code related to displaying the days belongs.

SideBar

Similarly, the SideBar directory holds modules such as SideBar (the container), and SideBarSection, for each respective section displayed within the SideBar.

Shared

Shared components can be shared across multiple interfaces, and isn't restricted to any component. For instance, the template for Modal is stored here, which acts as a container for the following sub-modals. Where each sub-modal has a parent modal which displays information based on its state.

AccountModal

The AccountModal holds pieces of code related to Account viewing and altering. Where what is displayed is dictated by the modal state. Whether it be simply displaying the users information, and editing it.

TaskModal

Very similar to the previous, the TaskModal holds pieces of code related to the TaskItem when an a clickable action occurs on the calendar. Ranging from viewing, creating, and editing.

SideBarModal

Also very similar to the previous, the SideBarModal holds pieces of code related to the SideBar actions within each SideBarSection.

When a visual piece of UI is broken down into it's unique components, it allows the architecture to speak for itself. The "Components" directory provides this. By breaking down the visual elements that the user sees into clear and distinct sub-folders. This provides easy navigation through the architecture, and transforms these components into modules that can be worked on in isolation. Where instead of a single file being a component, such as "calendar", it is further broken down into distinct modules.

Pages

The Pages directory contains key views like Dashboard, Login, Register, and Index, which manage user interactions and navigation. These pages are composed of reusable components that encapsulate specific UI elements and logic, such as the calendar and sidebar. By structuring pages with components, the app achieves cleaner, more maintainable code while promoting reusability across different areas. This modular approach allows for flexible and efficient management of user interfaces and interactions throughout the application.

The client-side of the application is purposefully designed to provide a seamless and interactive user experience directly within the browser. This directory holds pieces of code that directly relate to the user interface and client-side operations such as sending a request to the server-side when database access/manipulation is needed. The client-side has important directories such as, "Components" and "Pages", which hold reusable components and pages of the application respectively.

Where to find what you need

- At least one example of polymorphism which achieves a useful purpose (either through inheritance, method/constructor overloading/overriding):
    - Assignment_2.Shared/Data/ApplicationDbContext.cs
- At least two examples of Interface:
    - Assignment_2.Shared/Services/ITaskItemService.cs
    - Assignment_2.Shared/Services/IUserService.cs
- At least one example of NUnit tests:
    - Assignment_2.Tests/ModelTests.cs
- At least one example of Anonymous method with LINQ using Lambda expression:
    - Assignment_2.Shared/Utilities/ContainerUtilities.cs
- At least one example of Generics/Generic based Collection
    - Assignment_2.Shared/Utilities/ContainerUtilities.cs
- Error handling:
    - Assignment_2/Controllers
- Input validations:
    - Assignment_2.Shared/Models (Each property has an attribute for validating input)
    - Assignment_2.Shared/Utilities/Validation (Extra validation attributes were made)
    - These attributes are used in conjunction with razor components (Assignment_2.Client/pages/(login and register)) and for the database.
- C# code predominantly found in:
    - Assignment_2/Controllers
    - Assignment_2.Shared/

Role of team members

- Everything: me
- Energy needs: Monster