Design

Intended users:

Individuals or companies that require the ability to manage tasks for professional or personal purposes such as keeping books or organising their day etc.

User needs:

Efficient planning and resource management for timely project completion.

Automation of book indexing and reference creation to save time and minimize human errors when inputting data.

What are we trying to solve with this code?:

To do list Users can add new tasks to the list by entering a title, description, and due date. The tasks are displayed in a list view, showing the title, description, due date, and completion status of each task. Users can mark tasks as complete or delete tasks from the list. The application allows users to toggle between displaying all tasks or only incomplete tasks. Users can easily filter the task list to focus on the tasks that are yet to be completed.

When a task is selected from the task list, its details, including the title, description, due date, and completion status, are displayed in a separate section. Users can view the details of each task and mark tasks as complete if they are finished.

Some of the constraints when developing this program will be trying to implement the date and time and producing the dropdown, which will be time-consuming to create without issues occurring. Another constraint will be data structuring and how to store the to-do list data. I will be storing the data using an empty list to store tasks. Complexity in the program should be efficient enough with no addition involved, but could require some filtering within the program for the tasks, like the completed task and incomplete task function. It should be efficient by execution standards as it is only a small program, and I will aim towards simple code and try not to include irrelevant code blocks in the program.

Benefits of doing this:

This system is effective and well organized, in this example it has been used to automate book indexing, which will reduce the amount of manual data entry, the CSV file could however be changed in order to do different data sets.

The information is available almost instantly rather than having to sort through a massive dataset in order to find the data needed, information such as availability of a book, overdue status etc is readily available to the user.

Key Features:

Add Task - Users can add new tasks to the list by entering a title, description, and due date.

Mark as Complete - Users can mark tasks as complete, indicating that they have been finished.

Delete Task - Users can delete tasks from the list if they are no longer needed.

Task Filtering - Users can toggle between displaying all tasks or only incomplete tasks.

Task Details - The application provides a task details section that displays the title, description, due date, and completion status of the selected task.

Data dictionary

In my program, I will include some data structures for various variables that I will implement in my program code. The data structures I will include are:

TaskItem Class: Represents a single task item in the to-do list and includes additional details or descriptions of the given task or tasks.

List: A collection that stores the task items, and all task items added by the user are stored in this list.

Use case diagram

I have developed a use case diagram describing how the program will function when a user is operating it.

Design review

My designs for the to-do list application have different design types in looks and usability. The first UI design has some good aspects of usability with the different layout, but it looks too cluttered and would be tricky to use. The second design, which I opted for, has good usability and convenient text boxes for task information. It also has a traditional white background with black accents, making it easier to read and understand.

After receiving feedback from a peer, I decided to add captions next to the text boxes to indicate the difference between the title box and the description box.

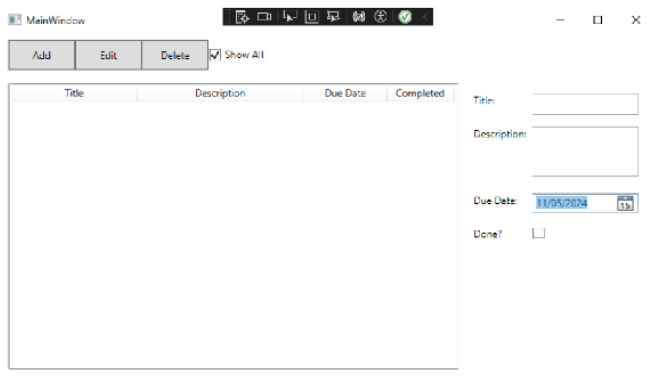
Feedback from design:

Lewis: the interface makes sense from a user standpoint, instead of messing around with lines of code, having a visual layer to deal with all the messing about for you makes a lot of sense if this were to be a popular software, because of the ease of use it brings.

Ugnius: the organisation of the code is detailed fairly well, this code has some rather advanced error handling and its hard to mess up the code because of the UI system, but the GUI could use a bit of an improvement in terms of looks such as captions next to the boxes.

The gui:

For this program the user will interact with the code through a GUI, this will make it much easier to visually represent the program and make it extremely easy to read, this was a requirement for the assignment but despite being quite difficult to implement, the reward is that your program becomes much easier to use.



Here is a screenshot of what the GUI looks like when the code has been run, it should automatically open upon the running of the code.

The GUI automatically rejects invalid dates and empty titles, and will ask for you to be completely sure you wish to delete a task so that you cannot accidentally delete a task, it also asks you validate user inputs for task details, ensuring data consistency, errors should also give a message to tell you what went wrong.

Implementation

First, I will create a C# WPF application and implement the chosen UI design. I will use a grid-based layout to organize the UI elements in rows and columns. The input section will use a horizontal StackPanel to group input elements for adding new tasks. I will use TextBoxes for the task title and description, a DatePicker for the due date, and a Button to add the task.

Next, I will implement the code in the .cs file. The MainWindow class will inherit from the Window class in the XAML file and implement the INotifyPropertyChanged interface for data binding support. Private fields tasks and showIncompleteOnly will manage the task list and the display filter. The Tasks property will return the filtered list of tasks based on the showIncompleteOnly flag.

The event handler CompleteTask_Click will mark a task as completed and raise the PropertyChanged event to update the task list display. The AddTask_Click event handler will retrieve the input values and create a new task object with the provided values, adding it to the tasks list. It will also raise the PropertyChanged event to update the task list display and reset the input fields.

Finally, I will implement the event handlers for the ShowIncomplete_Checked and ShowIncomplete_Unchecked events to update the task list display based on the showIncompleteOnly flag.

Test Plan

This is my test plan for the to-do list application:

Adding a Task Successfully: Enter valid values for title, description, and due date. Click the add task button, which should add the task to the list.

Adding a Task with Missing Title or Due Date: Enter a valid description but leave the title or due date empty. Click the add task button, which should display a message box indicating the missing title or due date.

Completing a Task: Click the complete button on a task in the task list. Verify that the task is marked as complete.

Filtering Incomplete Tasks: Check the show incomplete only checkbox to verify that only incomplete tasks are listed. Completed tasks should not be shown.

Clearing the Show Incomplete Only Filter: Uncheck the show incomplete only checkbox to clear the filters and show all tasks.

Testing

Following the test plan, I will test the program's functionality and usability:

Adding a Task Successfully: Fill in the fields for the task and verify that it is added to the task list as an incomplete task.

Adding a Task with Missing Title or Due Date: Test that the program throws an error when the title or description fields are not filled in.

Completing a Task: Click the complete button and verify that the task is marked as complete.

Filtering Incomplete Tasks: Check the show incomplete only checkbox and verify that only incomplete tasks are shown.

Clearing the Show Incomplete Only Filter: Uncheck the checkbox and verify that all tasks are shown.

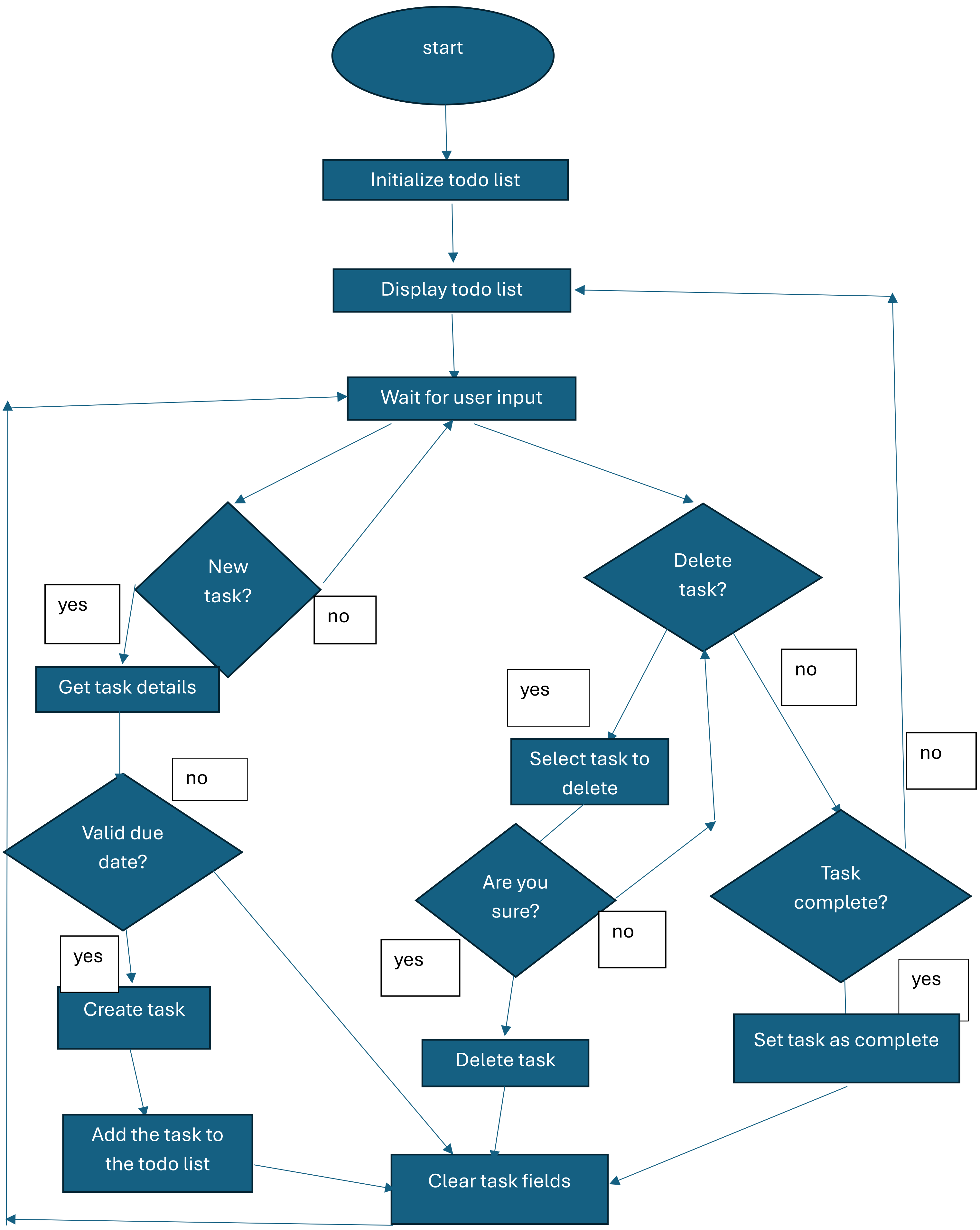
Final Review

In my WPF application, I feel like I have included good namespace organization in my code, which helps improve readability and maintainability. The style and formatting of the code are well-formatted. However, I could enhance code clarity by using braces for single-line if statements and using Pascal casing for method names.

For better validation, I could consider using WPF validation mechanisms, such as data annotations or implementing IDataErrorInfo in the Task class, to provide visual feedback to the user directly within the UI. The error handling in my code includes a validation check to ensure that the user enters a title and due date before adding a task. displaying a message box to alert the user is suitable for a simple application like this.

Overall, my code demonstrates a good understanding of WPF concepts and the basics of event handling and data binding, and the adherence to user requirements.

Flow chart:


start
Initialize todo list
Display todo list
Wait for user input
New task?
yes
no
Get task details
Valid due date?
no
yes
Create task
Add the task to the todo list
Delete task?
yes
no
Select task to delete
Are you sure?
yes
no
Delete task
Task complete?
no
yes
Set task as complete
Clear task fields

Problem 2

Design

For the design of my program, I started with making a class diagram to show what classes, methods, and objects I should use. The class diagram provides a small indication of the structure I should use in the program.

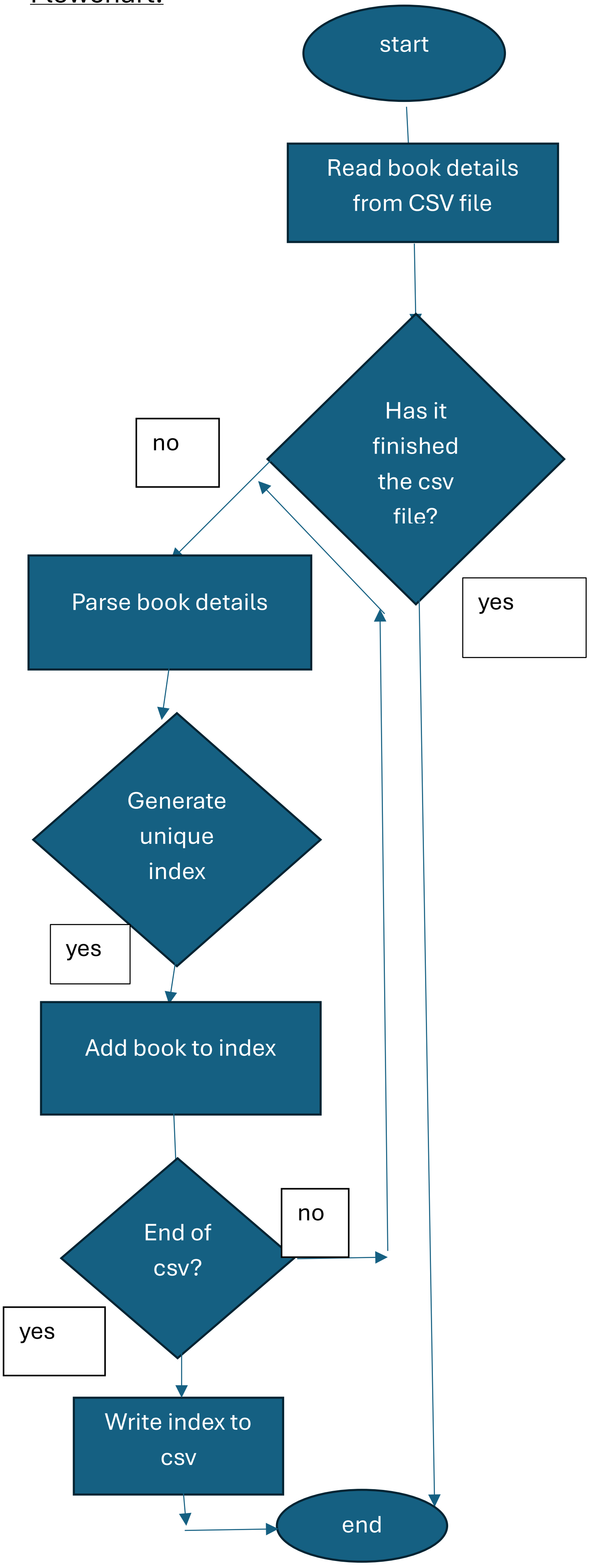
The idea is to have a program that successfully reads all the contents from the CSV file and then have it output that it has done so successfully.

When the application is launched or ran in visual studio, the code will automatically take from a predetermined input and then from that will compile all of the CSV into a set of records.

Implementation

This problem is more complex and has taken some time to develop a solution. The initial solution doesn't meet all the requirements, but I will implement all the client's requirements after reviewing my code.

Flowchart:


start
Read book details from CSV file
Has it finished the csv file?
no
yes
Parse book details
Generate unique index
yes
Add book to index
End of csv?
no
yes
Write index to csv
end