PULP - Interactive Ordering System using Python Language

Programming Languages LAB
BSCS - J4A

Members:
Agustin, Jerome Loyd
Bermundo, Nicole
Depoo, Zynnon Kyle
Torres, Junell

Instructor:
Grachel Eliza Ching

November 2025

Version	Date	Description of Change
1.0	2025-11-22	Initial SmartTaskManager documentation
1.0	2025-11-22	Added system overview & testing sections

Executive Summary

The Smart Task & Reminder Manager System is a console-based C# application designed to help users efficiently manage tasks, organize schedules, and receive automated reminders.

The system demonstrates advanced programming concepts including modular programming, clean architecture, improved error handling, and concurrency through background reminder services. Unlike the basic midterm version, this final iteration introduces a multi-threaded environment where reminders are checked continuously without blocking the user interface.

System Overview

Background of the Project Idea Students and professionals often struggle with keeping track of tasks and deadlines in a fast-paced environment. While simple lists exist, they often lack the automation required to actively prompt users when a deadline is approaching.

Problem Addressed

- **Disorganization:** Users often forget deadlines due to a lack of centralized organization.
- **Passive Tools:** Traditional to-do lists do not actively notify the user.
- **Lack of Structure:** The previous version handled basic operations but lacked architectural structure and automation.

Objectives

- **Modular Design:** Implement a clear separation of concerns using Models, Modules, and Services.
- **Concurrency:** utilize a background thread to handle reminders independently of user input.
- **Reliability:** Improve system maintainability, readability, and input validation.

Development Environment and Tools Used

- **Programming Language:** C# (utilizing async/await and System.Text.Json).
- **IDE:** Visual Studio / Visual Studio Code (Recommended for C# development).
- **Version Control:** Git and GitHub – Used for version history and collaboration.
- **Platform:** Console Application (Windows/Cross-platform .NET).

Software design & Architecture

Diagram of program structure or module relationships.

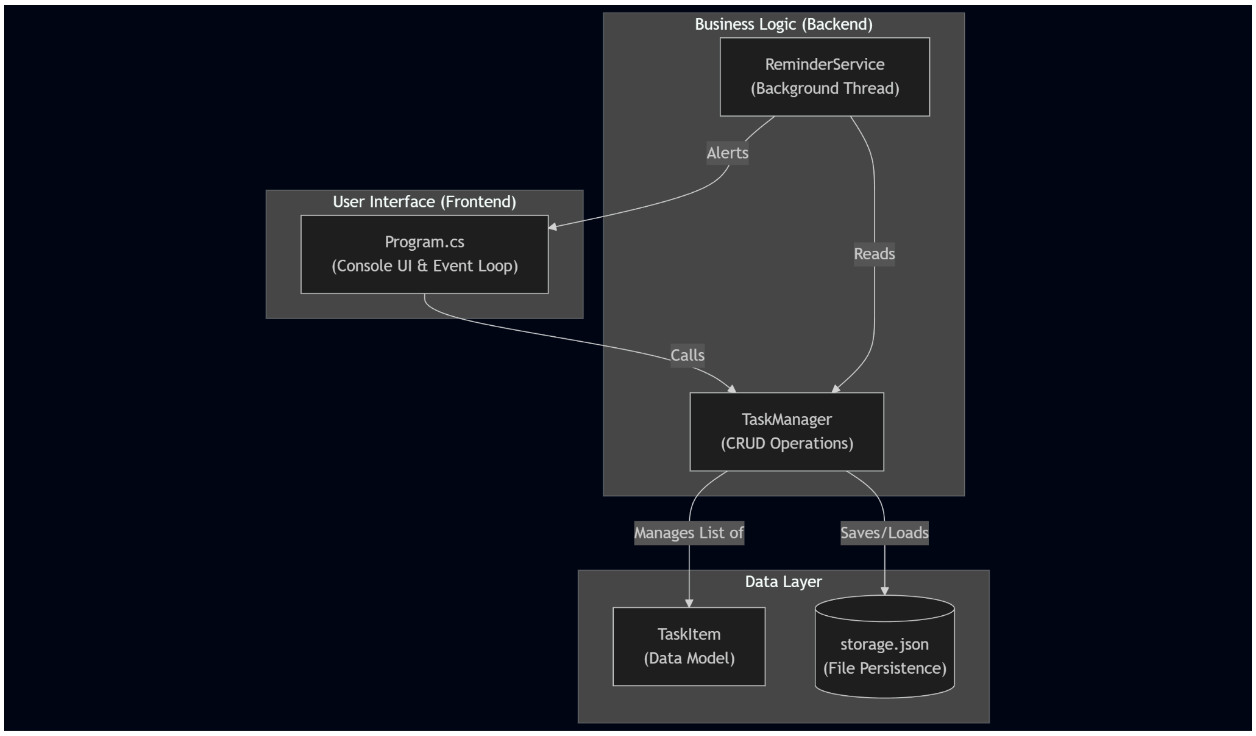

Business Logic (Backend)
ReminderService (Background Thread)
Alerts
User Interface (Frontend)
Program.cs (Console UI & Event Loop)
Reads
Calls
TaskManager (CRUD Operations)
Manages List of
Saves/Loads
Data Layer
TaskItem (Data Model)
storage.json (File Persistence)

Figure 1: Diagram of program structure or module relationships.

Explanation:

The system is divided into three major parts:

- **Frontend (Program.cs)**

  The console interface. Displays menus, handles inputs, and calls modules/services.

- **Backend (TaskManager.cs, TaskItem.cs)**

  Handles all logic: creating tasks, updating them, searching, sorting, and validation.

- **Services (ReminderService.cs)**

  A background asynchronous service that checks for tasks with upcoming deadlines and displays reminders without interrupting the user.

Explanation of Major Functions / Subprograms

Backend Functions

TaskManager.cs

- Add(...) – Creates a new task
- Delete(id) – Removes a task
- Search(query) – Finds tasks by keyword
- MarkDone(id) – Updates task state to completed
- GetAll() – Returns sorted list of tasks
- Update(updatedTask) – Saves changes to an existing task

TaskItem.cs

- Defines task fields: title, description, category, deadline, reminder settings
- Generates unique ID

Service Functions

ReminderService.cs

- Start() – Begins background reminder thread
- LoopAsync() – Checks tasks every few seconds
- Notify() – Displays reminder in console
- Stop() – Ends background task safely

Runs on a separate thread using:

```
Task.Run(() => LoopAsync(ct));
```

User Interface Functions (Program.cs)

- ShowMenu() – Displays choices
- HandleAddTask() – Collects user inputs
- HandleSearch() – Displays filtered tasks
- HandleDelete() – Deletes based on ID
- RenderTasks() – Shows formatted task list

Flowchart or pseudocode of main processes.

1.Main Program Flow

```
START
Load tasks
Start ReminderService (background)
LOOP
   Display menu
   Get user choice
   IF Add → Add Task
   IF View → Show Tasks
   IF Search → Search Tasks
   IF Delete → Delete Task
   IF Done → Mark Task Completed
END LOOP
Stop ReminderService
END
```

2. Reminder Service Flow

```
WHILE program is running:
   CHECK all tasks
   IF task has reminder enabled AND due date approaching:
      DISPLAY reminder
   WAIT 10 seconds
END WHILE
```

Concurrency / Multi-Threading Model Explanation

SmartTaskManager uses **asynchronous concurrency** to run the reminder service independently of the main user interface.

Concurrency features include:

- Task.Run()
  async/await
- CancellationTokenSource

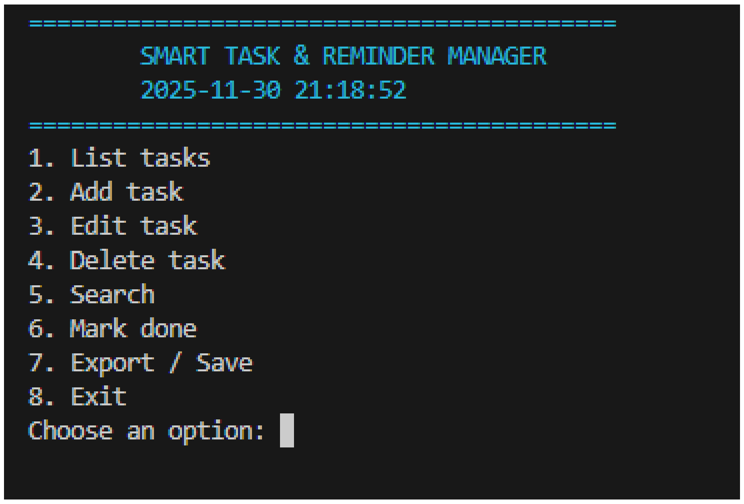
This approach ensures:

- The UI never freezes
- Reminders trigger automatically
- Background processing does not interrupt task management

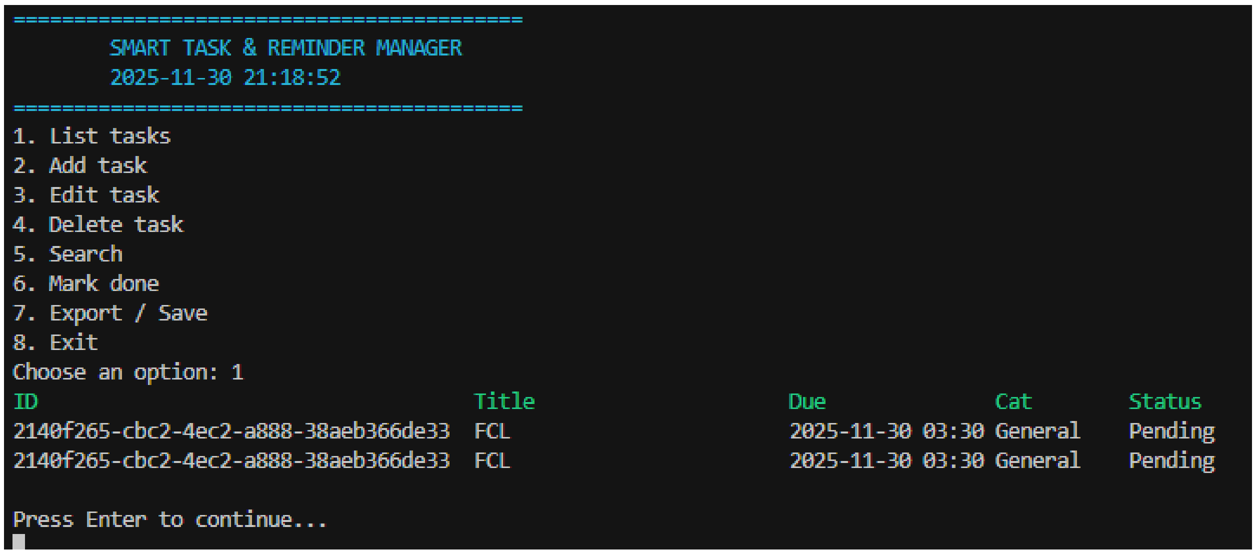
The system demonstrates proper concurrency principles required for the final project.

Implementation

Sample Input/Output:



```
==========================================
        SMART TASK & REMINDER MANAGER
        2025-11-30 21:18:52
==========================================
1. List tasks
2. Add task
3. Edit task
4. Delete task
5. Search
6. Mark done
7. Export / Save
8. Exit
Choose an option:
```



```
==========================================
        SMART TASK & REMINDER MANAGER
        2025-11-30 21:18:52
==========================================
1. List tasks
2. Add task
3. Edit task
4. Delete task
5. Search
6. Mark done
7. Export / Save
8. Exit
Choose an option: 1
ID                                    Title                     Due              Cat        Status
2140f265-cbc2-4ec2-a888-38aeb366de33  FCL                       2025-11-30 03:30 General    Pending
2140f265-cbc2-4ec2-a888-38aeb366de33  FCL                       2025-11-30 03:30 General    Pending

Press Enter to continue...
```

Highlight sections of code that demonstrate

Control Flow

Handled through menu options and conditional logic.

Function Modularity

Project separated into:

- Models
- Modules
- Services

Concurrency

ReminderService runs in background without blocking the UI.

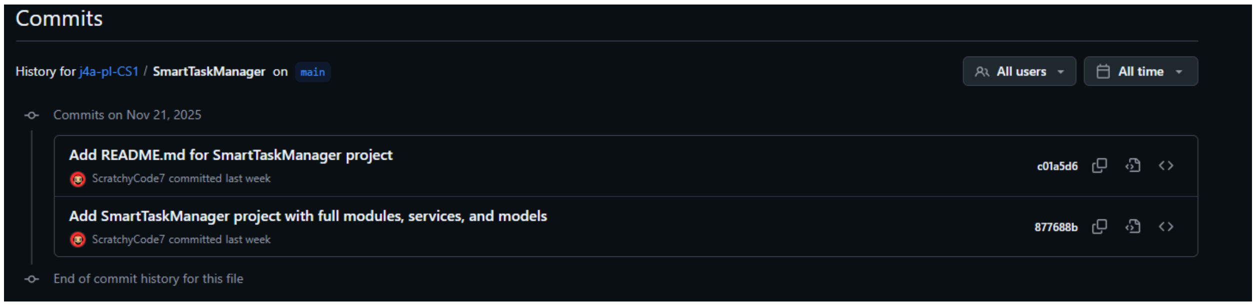
Teamwork (Git Commits)

Commit logs show:

- Feature-based commits
- Refactoring
- Bug fixes
- Parallel work contributions

Explanation of how Git commits reflect teamwork (show commit logs or history summary).

- Each member worked on separate features and modules, reflected by multiple logical commits.
- Commit messages follow the Conventional Commits style (feat:, fix:, refactor:).
- Example:
  feat: add scrollable order panel in GUI
  refactor: separate menu functions into backend module
  fix: correct image loading in frontend



Commits

History for j4a-pl-CS1 / SmartTaskManager on main

Commits on Nov 21, 2025

Add README.md for SmartTaskManager project
ScratchyCode7 committed last week — c01a5d6

Add SmartTaskManager project with full modules, services, and models
ScratchyCode7 committed last week — 877688b

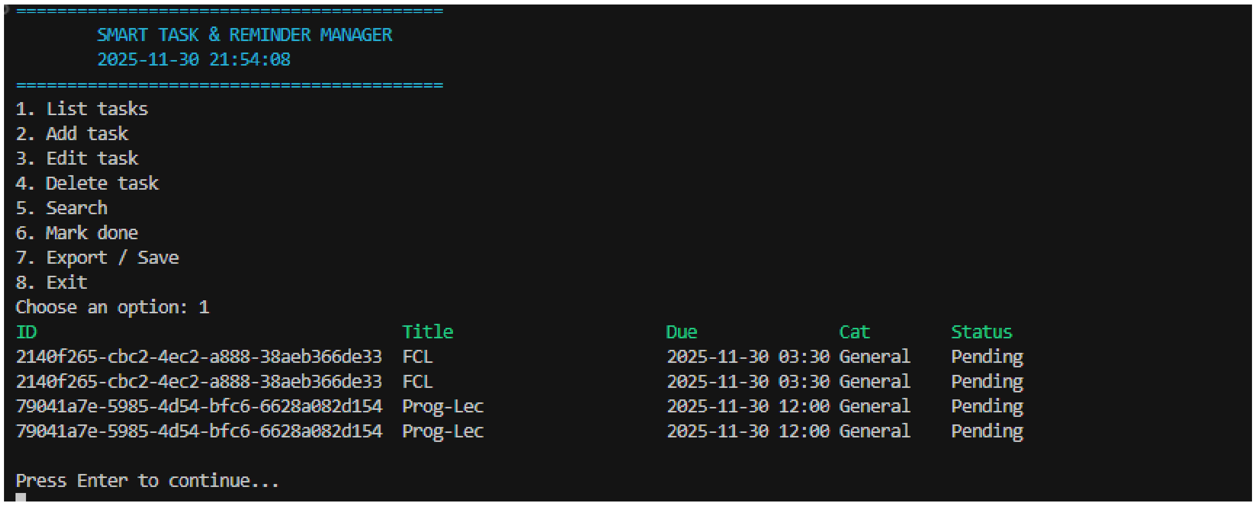
End of commit history for this file

Testing Evaluation

Test cases used (inputs, expected outputs, actual outputs).

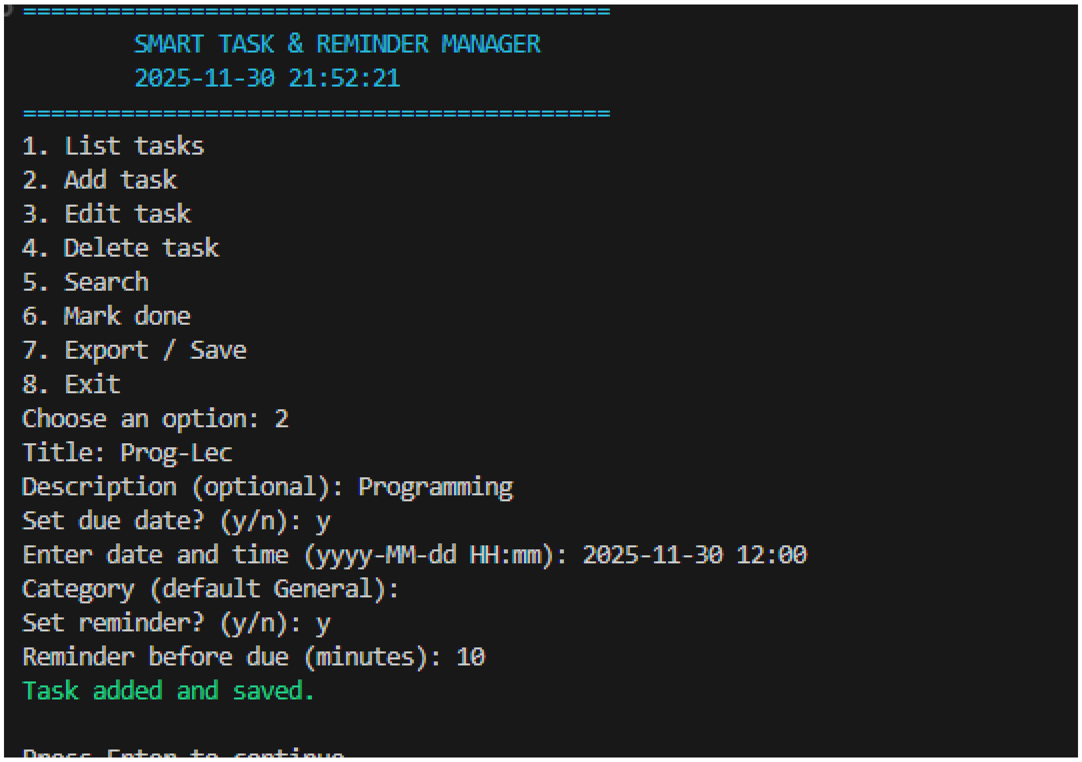
Test Case	Input	Expected Output	Actual Output	Result
Add Task	Title + Date	Task saved	Task saved	Pass
Invalid Date	“32/13/2025”	Error	Error	Pass
Reminder Trigger	Task due soon	Reminder appears	Reminder appears	Pass
Delete Task	Existing ID	Task removed	Task removed	Pass
Search	Keyword	Filtered list	Correct	Pass

List Task



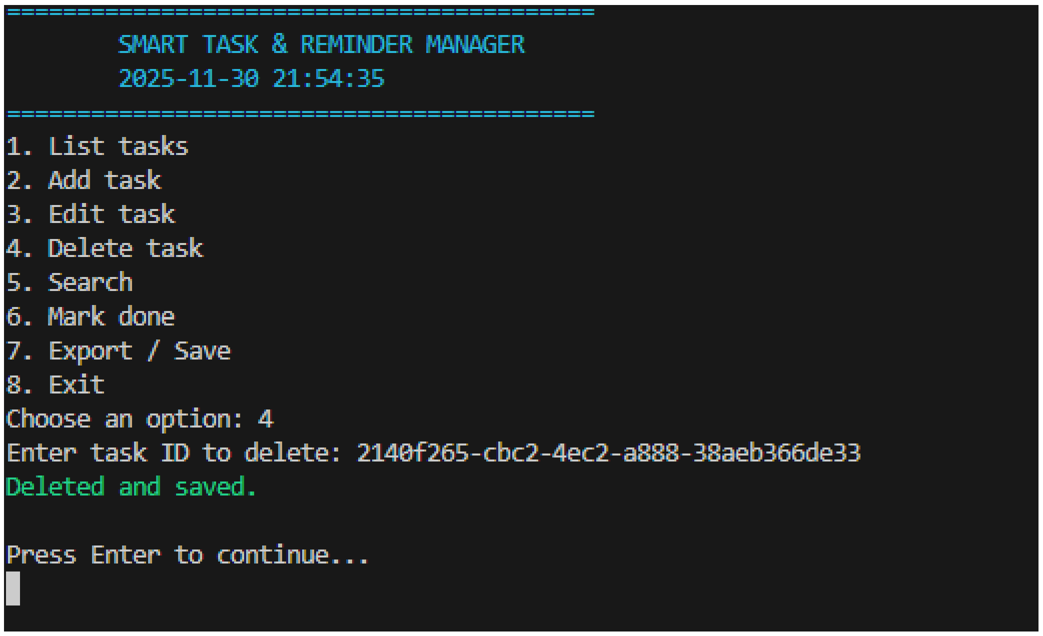
```
==========================================
        SMART TASK & REMINDER MANAGER
        2025-11-30 21:54:08
==========================================
1. List tasks
2. Add task
3. Edit task
4. Delete task
5. Search
6. Mark done
7. Export / Save
8. Exit
Choose an option: 1
ID                                    Title                     Due              Cat        Status
2140f265-cbc2-4ec2-a888-38aeb366de33  FCL                       2025-11-30 03:30 General    Pending
2140f265-cbc2-4ec2-a888-38aeb366de33  FCL                       2025-11-30 03:30 General    Pending
79041a7e-5985-4d54-bfc6-6628a082d154  Prog-Lec                  2025-11-30 12:00 General    Pending
79041a7e-5985-4d54-bfc6-6628a082d154  Prog-Lec                  2025-11-30 12:00 General    Pending

Press Enter to continue...
```

Add Task



```
==========================================
        SMART TASK & REMINDER MANAGER
        2025-11-30 21:52:21
==========================================
1. List tasks
2. Add task
3. Edit task
4. Delete task
5. Search
6. Mark done
7. Export / Save
8. Exit
Choose an option: 2
Title: Prog-Lec
Description (optional): Programming
Set due date? (y/n): y
Enter date and time (yyyy-MM-dd HH:mm): 2025-11-30 12:00
Category (default General):
Set reminder? (y/n): y
Reminder before due (minutes): 10
Task added and saved.
```

Delete Task

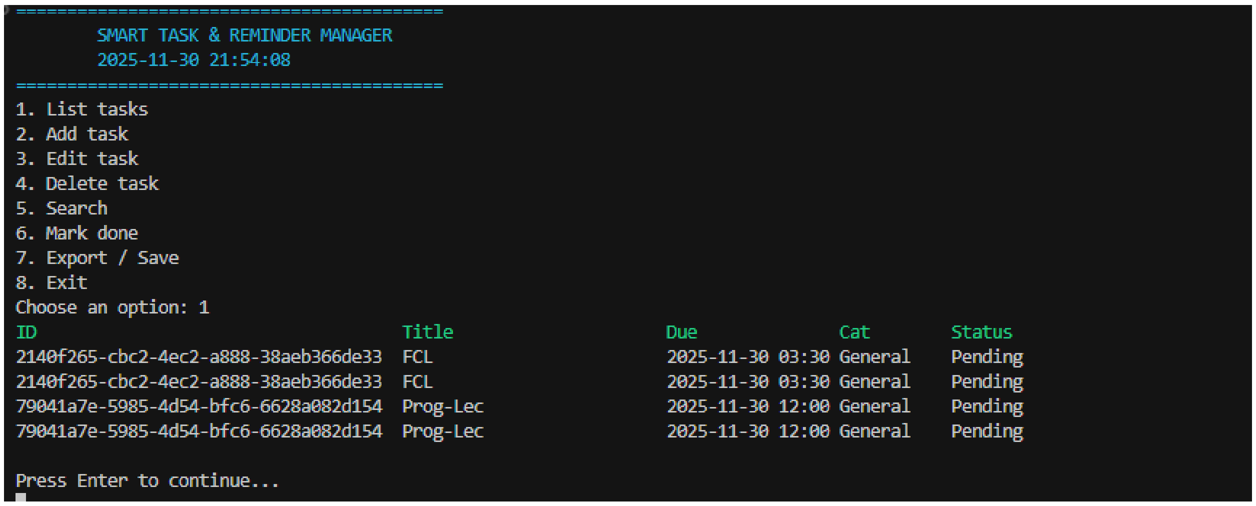


```
==========================================
        SMART TASK & REMINDER MANAGER
        2025-11-30 21:54:35
==========================================
1. List tasks
2. Add task
3. Edit task
4. Delete task
5. Search
6. Mark done
7. Export / Save
8. Exit
Choose an option: 4
Enter task ID to delete: 2140f265-cbc2-4ec2-a888-38aeb366de33
Deleted and saved.

Press Enter to continue...
```

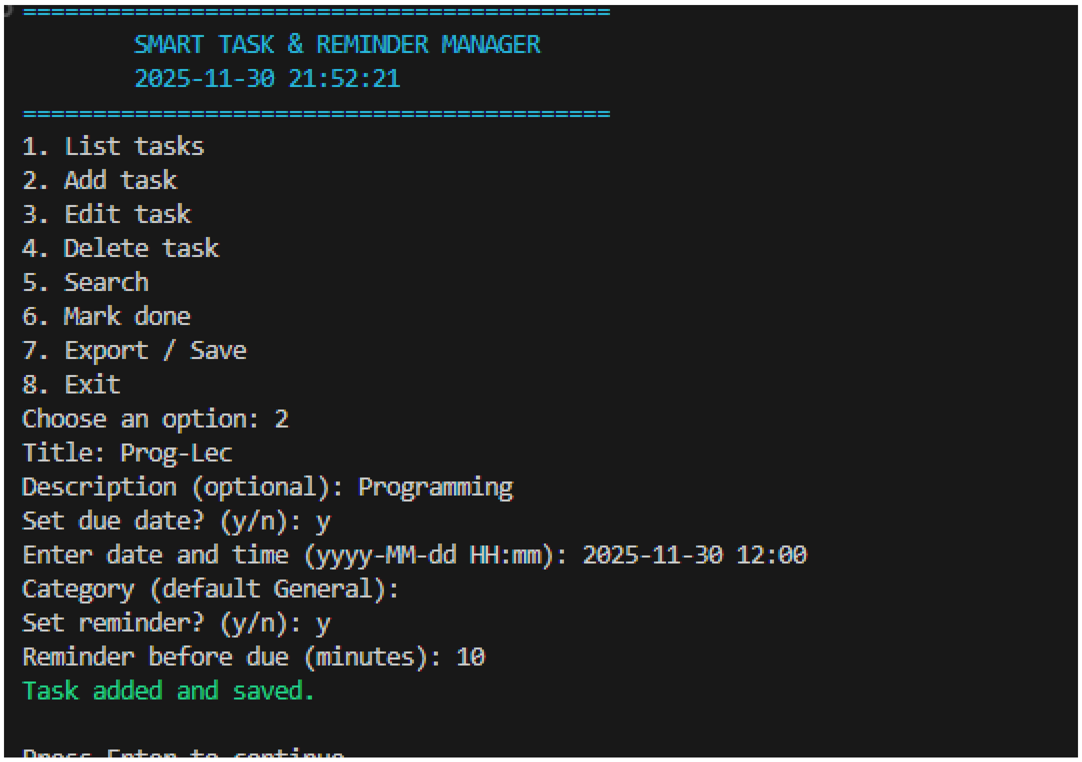
Discussion of results, issues, or limitations.
The system works consistently for all task operations.

- The reminder service accurately detects upcoming deadlines.
- Robust error handling prevents invalid input crashes.
- Limitations:
    - No file-based saving (optional future extension).
    - Console UI limits advanced layout capabilities.
    - Single-user system only (no multi-user mode).

Screenshot of working outputs.



```
=========================================
        SMART TASK & REMINDER MANAGER
        2025-11-30 21:54:08
=========================================
1. List tasks
2. Add task
3. Edit task
4. Delete task
5. Search
6. Mark done
7. Export / Save
8. Exit
Choose an option: 1
ID                                    Title                     Due              Cat        Status
2140f265-cbc2-4ec2-a888-38aeb366de33  FCL                       2025-11-30 03:30 General    Pending
2140f265-cbc2-4ec2-a888-38aeb366de33  FCL                       2025-11-30 03:30 General    Pending
79041a7e-5985-4d54-bfc6-6628a082d154  Prog-Lec                  2025-11-30 12:00 General    Pending
79041a7e-5985-4d54-bfc6-6628a082d154  Prog-Lec                  2025-11-30 12:00 General    Pending

Press Enter to continue...
```



```
=========================================
        SMART TASK & REMINDER MANAGER
        2025-11-30 21:52:21
=========================================
1. List tasks
2. Add task
3. Edit task
4. Delete task
5. Search
6. Mark done
7. Export / Save
8. Exit
Choose an option: 2
Title: Prog-Lec
Description (optional): Programming
Set due date? (y/n): y
Enter date and time (yyyy-MM-dd HH:mm): 2025-11-30 12:00
Category (default General):
Set reminder? (y/n): y
Reminder before due (minutes): 10
Task added and saved.
```

Ethical and Professional Reflection

- **How did your team ensure ethical collaboration (no plagiarism, fair contribution)?**

All members contributed original work, coordinated through GitHub, and avoided plagiarism. AI assistance (when used) was for learning and debugging—not copying full solutions.

- **How does your system ensure data privacy (if applicable) and responsible programming?**

  The system uses safe input handling, avoids collecting personal or sensitive data, and maintains clear, maintainable code.
- **What lessons can you apply from professional practice and version control ethics?**

  Team members learned the importance of modular architecture, version control discipline, and writing documentation that meets professional standards.

Independent Learning Component

- What new concept or tool they independently learned (e.g., Git branching, threading, or modular architecture).

  Torres: I learned how asynchronous programming works in C#, specifically using Task, await, and background loops. This helped me understand how real-world applications perform non-blocking operations. I learned deeper Git branching strategies, which helped me contribute without causing merge conflicts and ensured clean collaboration. I focused on improving input validation and error handling, which taught me how to prevent crashes and improve user experience. I learned modular architecture—splitting the system into Models, Modules, and Services—which made the code easier to maintain and understand.

References

Microsoft Docs – C# async/await (2025)

GitHub Docs – Commit & Branching Guidelines (2025)

StackOverflow – Background Task Implementation (2025)

Real Python – Software Design Principles (used for structural reference)