Task Manager Application

NHL Stenden | Final Assignment

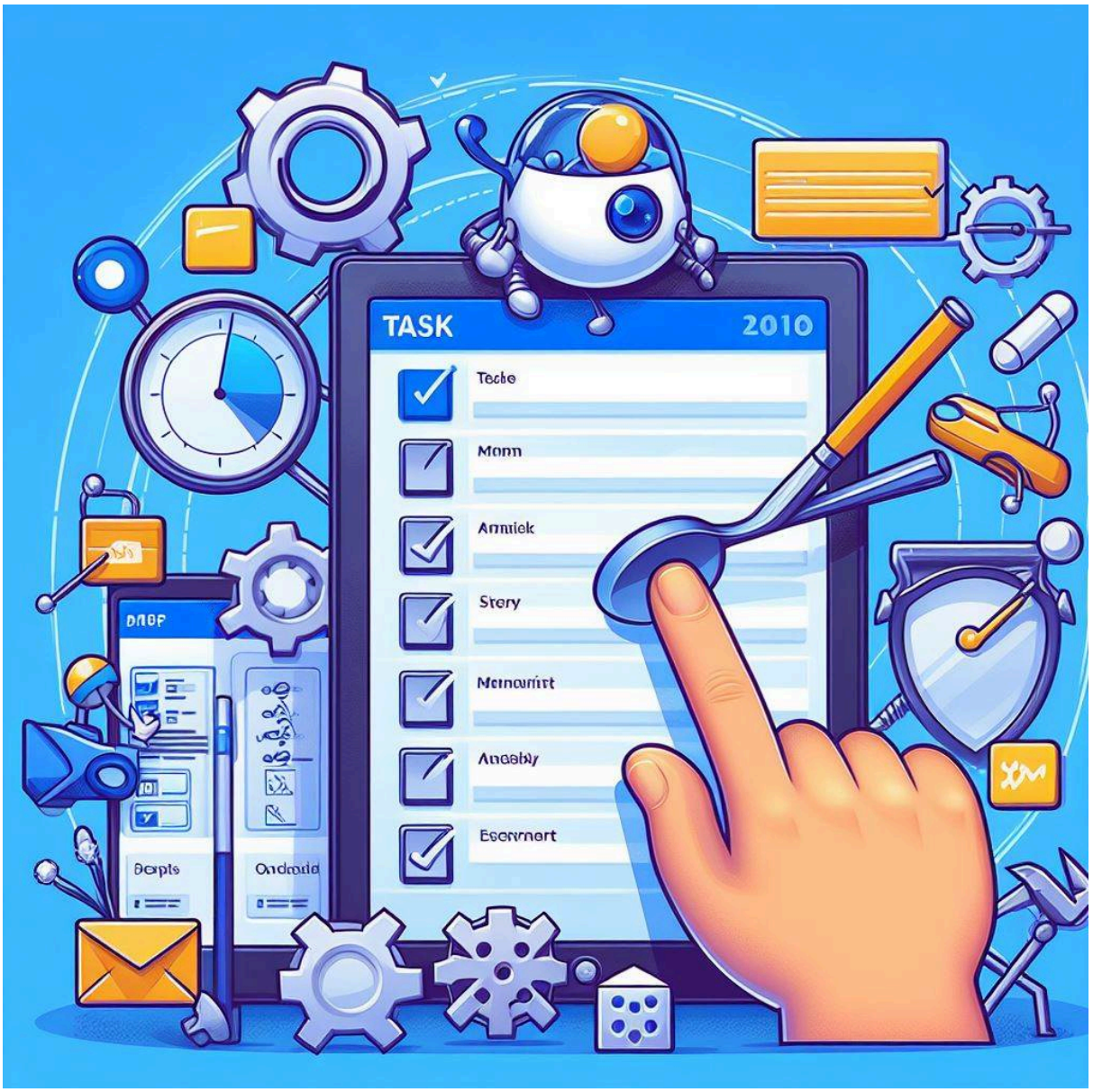
Virag Szabo (4727444)

Information Technology

Content

Objective

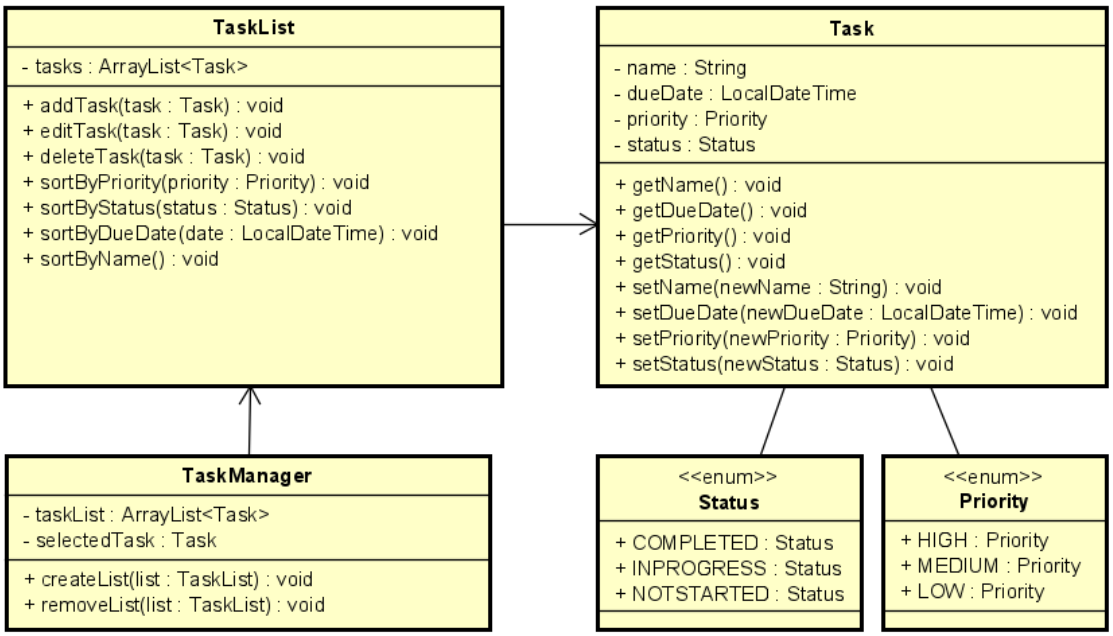
Develop a task manager application using C# that falls under the category of applications with a standard Windows GUI. The application should allow users to manage their tasks efficiently.



Diagram

UML

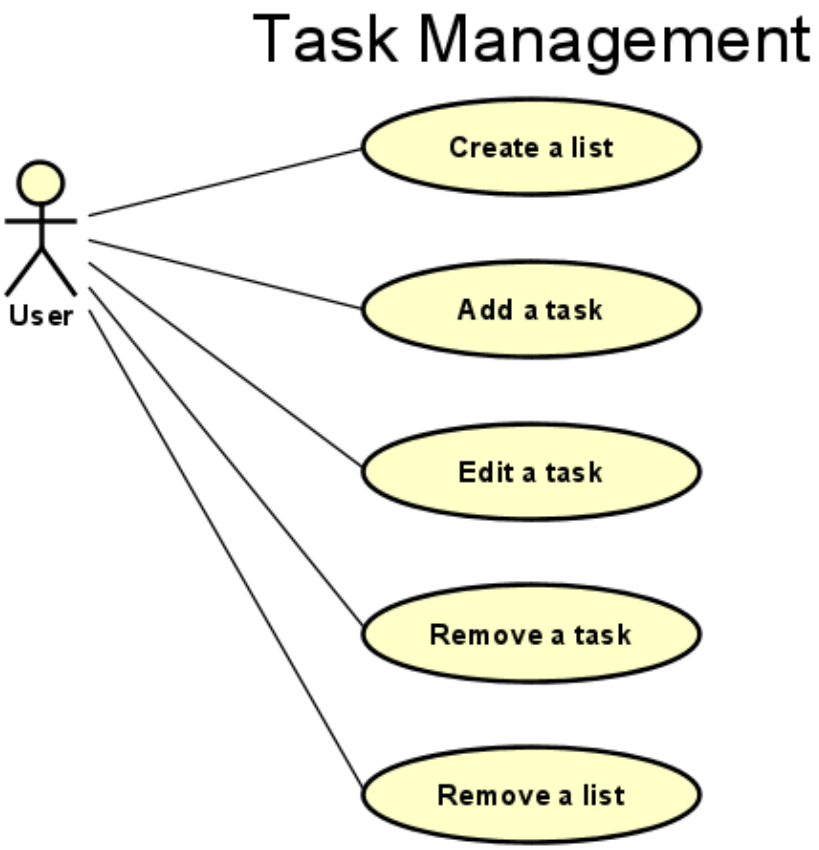
1. **TaskManager Class:**
   - Manages the overall application.
   - Contains a list of the task lists and the currently selected task.
   - Provides methods to create and remove lists.
   - Includes methods for sorting tasks.
2. **TaskList Class:**
   - Represents a task list screen.
   - Holds a list of tasks.
   - Provides methods to add, edit, and remove tasks.
   - Implements sorting and filtering functionality.
3. **Task Class** represents a task with attributes like name, due date, priority, and status.
4. **Priority Enum** represents the priority of a task (High, Medium, Low).
5. **Status Enum** represents the status of a task (Completed, In Progress, Not Started).



Use Case

The user interacts with the Task Manager App through various actions:

- **Create Task List:** The user can create a new task list.
- **Add Task:** The user can add a new task to the task list.
- **Edit Task:** The user can modify the attributes of an existing task.
- **Remove Task:** The user can delete a task from the task list.
- **Remove Task List:** The user can delete an entire task list.



Features

User Interface Design

Windows Presentation Foundation (WPF) was chosen for the project due to its maturity, widespread use in desktop application development on Windows, and compatibility with Windows 11. With a rich set of controls, layout features, and robust data binding capabilities, WPF is ideal for creating modern and visually appealing user interfaces. It fully supports MVVM architecture, ensuring a clean separation of concerns between the UI and business logic layers. Additionally, its seamless integration into Visual Studio enhances the development experience for C# developers.

MVVM Architecture

Model-View-ViewModel is a design pattern that is going to be used to separate concerns in the application.

Architecture

The **Task** class serves as the **Model**, encapsulating data and business logic (name, due date, priority, status), responsible for managing the application's data. **Views**, represented by the UI components like the **TaskList** screen, are designed using WPF's XAML for a declarative UI definition. The **ViewModel**, exemplified by the **TaskManager** class, acts as an intermediary between the View and Model, managing state and interaction. MVVM's data binding ensures responsive and maintainable UI by automatically updating associated Views.

Key Features

Data Binding: WPF’s robust data binding system allows you to connect UI elements directly to ViewModel properties, adding synchronization between the UI and underlying data.
Commands: MVVM provides the use of commands to handle user interactions. With this feature, it can bind the UI actions to ViewModel methods.
Dependency Properties: The dependency properties of WPF facilitate a powerful and efficient way to enable data flow between the View and ViewModel.

Version Control

GitHub is going to be used as the version control where I update the READ.ME file with more details about the project. The students created a [private repository] for the project.

Testing and Quality Assurance

Unit Tests are going to be implemented for testing critical functionalities of the project adding a new Test Project to the project within the Visual Studio.