upGrad ENTERPRISE

Assignment Case Study: To-Do List Management System

Problem Statement: To-Do List Management System

Objective: Develop a console-based To-Do List Management System to allow users to manage their tasks efficiently. The system should enable users to add new tasks, view all tasks in the order they were added, remove tasks by their position, and exit the application.

Requirements:

1. Task Addition:

- Users can add a new task by providing a task description (a string).
- Tasks should be stored in the order they are added to maintain a chronological sequence.
- Display a confirmation message when a task is successfully added.

2. View Tasks:

- Display all tasks in the order they were added, with each task numbered (starting from 1) for easy reference.
- If no tasks exist, display a message indicating the list is empty.

3. Task Removal:

- Users can remove a task by specifying its number (position in the list).
- Validate the provided task number to ensure it is within the valid range.
- Display the removed task's description or an error message for invalid inputs (e.g., non-existent task number or non-numeric input).

4. User Interface:

- Provide a menu-driven console interface with the following options:
  - Add Task
  - View Tasks
  - Remove Task
  - Exit
- Prompt users for inputs and handle invalid inputs gracefully with appropriate error messages.

5. Data Structure:

- Use a `List<string>` to store tasks, preserving the order of addition and allowing access by index for viewing and removal.

- Ensure the system is simple and suitable for beginner to intermediate C# learners.

6. Constraints:

- Task descriptions are non-empty strings.

- Task numbers for removal are 1-based (e.g., the first task is number 1) to be user-friendly.

- Handle invalid inputs, such as non-numeric input for task numbers or out-of-range indices.

- The system should avoid advanced features like file storage or complex data structures to keep the focus on `List` usage.

Expected Output Example:

To-Do List Manager

1. Add Task

2. View Tasks

3. Remove Task

4. Exit

Choose an option: 1

Enter task: Buy groceries

Task added!

To-Do List Manager

1. Add Task

2. View Tasks

3. Remove Task

4. Exit

Choose an option: 1

Enter task: Call doctor

Task added!

To-Do List Manager

1. Add Task

2. View Tasks

3. Remove Task

4. Exit

Choose an option: 2

Tasks:

1. Buy groceries

2. Call doctor

To-Do List Manager

1. Add Task

2. View Tasks

3. Remove Task

4. Exit

Choose an option: 3

Enter task number to remove: 1

Removed: Buy groceries

To-Do List Manager

1. Add Task

2. View Tasks

3. Remove Task

4. Exit

Choose an option: 2

Tasks:

1. Call doctor

To-Do List Manager

1. Add Task

2. View Tasks

3. Remove Task

4. Exit

Choose an option: 3

Enter task number to remove: 5

Invalid task number.

To-Do List Manager

1. Add Task

2. View Tasks

3. Remove Task

4. Exit

Choose an option: 4

Technical Requirements:

- Use C# with .NET (any version compatible with `List`).
- Implement error handling for invalid inputs (e.g., invalid task numbers or menu options).
- Use `List<string>` to store tasks, leveraging its ability to maintain order and support index-based operations.
- Keep the code simple, well-commented, and suitable for beginner to intermediate learners.
- Avoid advanced features like file I/O, databases, or complex data structures to focus on `List` functionality.

Learning Outcomes:

- Understand how to use `List` for ordered collections of items.
- Learn to perform basic operations like adding, viewing, and removing items from a `List`.
- Practice input validation and error handling in a console application.
- Gain experience with iteration (using `for` or `foreach`) and index-based access in C#.