Abbottabad University of Science & Technology

Department Of Computer Science

Assignment #1:

Submitted By:

Muhammad Abdullah 14982

Submitted To:

Ma’am Samman Shaheen

Software Requirements Specification (SRS)
Project Title: Console-Based Task Management System
Author: Muhammad Abdullah
GitHub Link: https://github.com/Dreamix-Hub/Task-Manager

---

1. Introduction

1.1 Purpose
The purpose of this document is to outline the requirements for the development of a console-based Task Management System. This system will allow users to manage tasks through the command line interface by adding, editing, deleting, listing, and marking tasks as complete.

1.2 Scope
The application is a lightweight, console-based software tool intended for individual users or developers seeking to manage their to-do items efficiently. It includes modular features such as task categorization, due dates, and status tracking. The system will follow best practices in software construction, such as modular design, documentation, and version control via Git.
1.3 Definitions, Acronyms, and Abbreviations

- **CLI**: Command-Line Interface
- **CRUD**: Create, Read, Update, Delete
- **Task**: A to-do item with a title, optional description, due date, and status

---

2. Overall Description
2.1 Product Perspective
The system is a standalone console application developed in Python. It does not require networking or database connectivity; data can be persisted using JSON files.
2.2 Product Functions
The system will provide:

- Add new task
- View all tasks
- Edit task
- Delete task
- Mark task as complete/incomplete

- Filter tasks by status or due date

2.3 User Classes and Characteristics

- **Standard User**: Interacts with the system via CLI; expected to have basic computer skills and knowledge of console navigation.

2.4 Operating Environment

- OS: Windows/Linux/MacOS
- Language: Python 3.13
- Storage: Local file system for persistence

2.5 Design and Implementation Constraints

- Must be console-based only
- Must use modular design (separate modules for data handling, logic, and UI)
- Must follow coding standards (PEP8 for Python, etc.)
- Must use Git for version control

---

3. Specific Requirements

3.1 Functional Requirements

- **FR1**: The system shall allow the user to create a new task with title, description, due date, and status.
- **FR2**: The system shall allow the user to view a list of all tasks.
- **FR3**: The system shall allow the user to update task details.
- **FR4**: The system shall allow the user to delete a task.
- **FR5**: The system shall allow the user to mark a task as complete or incomplete.
- **FR6**: The system shall allow the user to filter tasks by completion status

---

4. Non-Functional Requirements

4.1 Performance

- The system should respond to user input within 1 second.
- The system should support up to 1,000 tasks without performance degradation.

4.2 Usability

- The interface should be clear and intuitive for a command-line user.
- Help and error messages should be displayed in user-friendly language.

4.3 Reliability

- The system should not crash due to invalid user input.
- Data must be saved reliably between sessions.

4.4 Maintainability

- The code should follow consistent naming and structural conventions.
- Modules should be independently testable and replaceable.

4.5 Portability

- The application should be cross-platform and runnable on any OS with a suitable interpreter.

5 Use Case Descriptions

Use Case 1: Add Task

- Actor: User
- Description: User adds a task with a title and optional details.

Use Case 2: View Tasks

- Actor: User
- Description: User lists all existing tasks with their details.

Use Case 3: Update Task

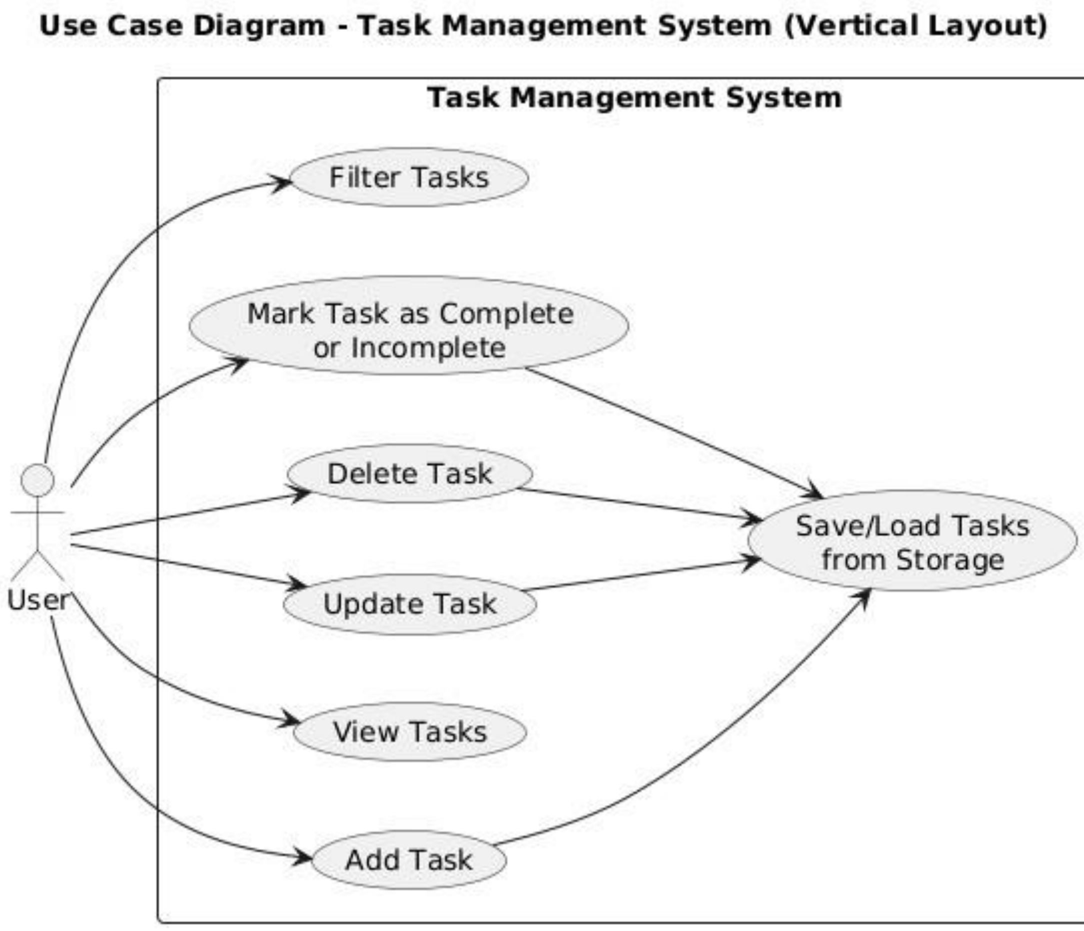
- Actor: User
- Description: User selects a task and modifies its details.

Use Case 4: Delete Task

- Actor: User
- Description: User deletes a task permanently.

Use Case 5: Mark Task Complete/Incomplete

- Actor: User
- Description: User updates the task status.


Use Case Diagram - Task Management System (Vertical Layout)
Task Management System
Filter Tasks
Mark Task as Complete or Incomplete
Delete Task
Save/Load Tasks from Storage
Update Task
View Tasks
Add Task
User

6 Class Diagram Description – Task Management System

The **class diagram** illustrates the static structure of the system, highlighting the core classes, their attributes, operations (methods), and relationships. It follows **object-oriented design principles** to promote modularity, separation of concerns, and maintainability.

---

1. UIManager Class
Purpose: Manages the user interface through a command-line menu, handling all input and output operations.

- **Methods:**
    - display_menu(): Displays the available options in the task manager.
    - get_user_input(): Collects user input from the console.
    - show_tasks(tasks: List<Task>): Displays a list of task objects.
    - show_message(msg: string): Outputs a custom message to the user.

Interacts with: TaskManager for performing operations based on user input.

---

2. TaskManager Class
Purpose: Acts as the central controller that maintains a list of tasks and provides core business logic.

- **Attributes:**
    - tasks: List<Task> – A list storing all current tasks.
- **Methods:**
    - add_task(task: Task): Adds a new task to the list.
    - update_task(task_id: int): Updates task details by ID.
    - delete_task(task_id: int): Removes a task by ID.
    - filter_tasks(by_status: bool, by_date: Date): Filters tasks by completion status or due date.
    - list_tasks(): Returns the current list of tasks.

Associates with:

- Task (composition) — Manages instances of Task.
- StorageManager — For persistence (loading/saving tasks).
- Indirectly used by UIManager.

---

3. Task Class
Purpose: Represents a task entity with key properties and behavior.

- **Attributes:**
    - id: int – Unique identifier for the task.
    - title: string – Title or name of the task.
    - description: string – Detailed information about the task.
    - due_date: Date – The due date of the task.
    - completed: boolean – Status of the task (True if completed).
- **Methods:**
    - mark_complete(): Marks the task as completed.
    - mark_incomplete(): Marks the task as not completed.

Used by: TaskManager to create and manage task objects.

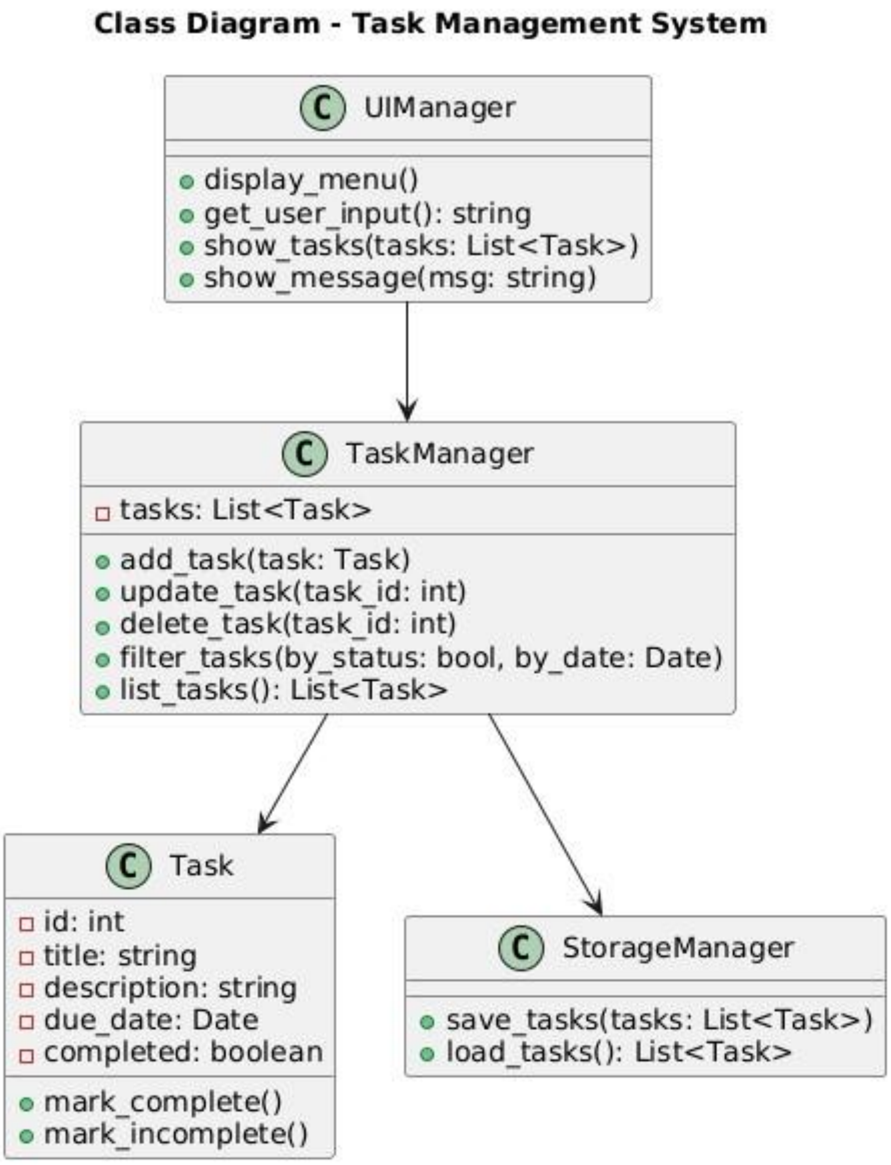
---

4. StorageManager Class

Purpose: Handles the persistence layer, managing how tasks are saved and loaded (e.g., from a file or database).

- **Methods:**
    - save_tasks(tasks: List<Task>): Persists the list of tasks.
    - load_tasks() -> List<Task>: Loads and returns a list of tasks.

Used by: TaskManager (indirectly) to store and retrieve task data.


Class Diagram - Task Management System
UIManager
display_menu()
get_user_input(): string
show_tasks(tasks: List<Task>)
show_message(msg: string)
TaskManager
tasks: List<Task>
add_task(task: Task)
update_task(task_id: int)
delete_task(task_id: int)
filter_tasks(by_status: bool, by_date: Date)
list_tasks(): List<Task>
Task
id: int
title: string
description: string
due_date: Date
completed: boolean
mark_complete()
mark_incomplete()
StorageManager
save_tasks(tasks: List<Task>)
load_tasks(): List<Task>