Task Manager Documentation

1. Overview

The Task Manager is built around two main classes:

- Task: Represents individual tasks.
- TaskManager: Manages operations on tasks like adding, viewing, deleting, and marking tasks as completed. It handles reading from and writing to a JSON file to store tasks.

The main() function handles the user interaction via a simple command-line interface.

2. Classes

2.1 Task Class

The Task class is used to create and manage individual tasks. Each task has the following attributes:
- id: Unique identifier for the task.
- title: The title of the task.
- description: A detailed description of the task.
- completed: Boolean flag indicating whether the task is completed.
- timestamp: The time and date when the task was created.

Constructor:

def __init__(self, task_id, title, description)
- Parameters:
  - task_id: Integer, the unique ID of the task.
  - title: String, the title of the task.
  - description: String, the description of the task.
- Description: Initializes the task with a unique ID, title, description, sets the completed status to False, and stores the current timestamp.

mark_completed()

def mark_completed(self)

- Description: Marks the task as completed by setting the completed attribute to True.

to_dict()

def to_dict(self)
- Description: Converts the task into a dictionary format for easy JSON serialization.
- Returns: A dictionary with the task's attributes.

from_dict()

@classmethod
def from_dict(cls, data)
- Parameters:
  - data: Dictionary, the dictionary representation of a task.
- Description: Creates a Task object from a dictionary. This is used when loading tasks from a JSON file.
- Returns: A new Task object.

2.2 TaskManager Class

The TaskManager class handles all task-related operations, including saving and loading tasks to and from a JSON file. It manages a list of Task objects.

Constructor:

def __init__(self, filename="tasks.json")
- Parameters:
  - filename: Optional string, the name of the JSON file used for task persistence (default is tasks.json).
- Description: Initializes the TaskManager by loading tasks from the specified JSON file (if it exists), or creates an empty list of tasks.

add_task()

def add_task(self, title, description)
- Parameters:
  - title: String, the title of the task.
  - description: String, the description of the task.
- Description: Creates a new Task object with a unique ID and adds it to the list of tasks. The

tasks are then saved to the JSON file.

view_tasks()

def view_tasks(self, completed=None)
- Parameters:
  - completed: Optional boolean, if specified, filters the tasks to show only completed or incomplete tasks.
- Description: Prints a list of tasks, optionally filtering by completed/incomplete status. If no tasks are available, it prints a message.

delete_task()

def delete_task(self, task_id)
- Parameters:
  - task_id: Integer, the ID of the task to be deleted.
- Description: Deletes the task with the specified ID from the list and updates the JSON file. If the task ID is not found, it prints a message.

mark_task_completed()

def mark_task_completed(self, task_id)
- Parameters:
  - task_id: Integer, the ID of the task to be marked as completed.
- Description: Marks the task with the specified ID as completed and updates the JSON file. If the task ID is not found, it prints a message.

save_tasks()

def save_tasks(self)
- Description: Saves the list of tasks to the specified JSON file by serializing each task into a dictionary format.

load_tasks()

def load_tasks(self)
- Description: Loads tasks from the JSON file if it exists. Converts each task from a dictionary back into a Task object. If the file does not exist, initializes an empty task list.

3. Main Program Flow

The main() function runs the command-line interface for the Task Manager. It presents users with a menu of options and performs the selected task operation.

4. Input Validation

The program includes the following input validations:
- Task Title: When adding a task, the title must not be empty (checked with title.strip()).
- Task ID: For deletion and marking tasks as completed, the task ID must be a valid integer. If not, an error message is printed.

5. JSON File Storage

- File Format: The tasks are stored in a JSON file (default: tasks.json). Each task is saved as a dictionary, with the following keys:

```
 {
   "id": 1,
   "title": "Example Task",
   "description": "This is an example task description.",
   "completed": false,
   "timestamp": "2024-01-01T12:00:00"
 }
```

- Persistence: The tasks are automatically saved after adding, deleting, or marking a task as completed. The tasks are loaded from the file when the program starts, allowing the user to continue where they left off.

6. Error Handling

- Missing File: If the JSON file does not exist when the program starts, an empty list of tasks is created.
- Invalid Task ID: When deleting or marking tasks as completed, if the task ID does not exist or is not a valid number, an error message is shown to the user.

7. How to Use

1. Run the script.
2. Select an option from the menu by typing the corresponding number.
3. Follow the on-screen prompts to add tasks, view tasks, delete tasks, or mark them as completed.
4. Exit the program by choosing option 7. Tasks will be saved automatically.

8.Project & Documentation done by:

Daniyal Haider