MINI PROJECT

TITLE : To solve a daily life problem

TOPIC : Time Management

NAME : Kurapati Venkata Poojitha

- **DESCRIPTION:**

This program allows you to manage your schedule by adding tasks, viewing your schedule, and marking tasks as completed. It uses a JSON file to store the schedule data.

- **Input:**

- Task name
- Due date (optional)
- Completed status (optional)

- **Output:**

- Schedule with task names, due dates, and completed status
- Confirmation messages for adding, viewing, and marking tasks as completed

- **JSON File:**

The schedule data is stored in a JSON file named schedule.json. The file contains an array of tasks, each with the following properties:

- name: Task name

- due_date: Due date (optional)
- completed: Completed status (optional)

➢ **Background:**

Arnavi is a freelance writer who works on multiple projects simultaneously. She needs a system to manage her schedule, deadlines, and progress.

➢ **Requirements:**

1. Add tasks with due dates
2. View schedule (all tasks with due dates and status)
3. Mark tasks as completed
4. Store data locally (no online storage)

➢ **Functional Requirements**:

1. User can add tasks with due dates
2. User can view schedule (all tasks with due dates and status)
3. User can mark tasks as completed
4. System stores data locally

- **Non-Functional Requirements:**

1. User-friendly interface
2. Fast data retrieval
3. Data security (local storage)

- **Use Cases:**
- **Use Case 1:** Add Task

1. User opens the application
2. User clicks "Add Task"
3. User enters task name and due date
4. System saves task and displays confirmation message

- **Use Case 2**: View Schedule

1. User opens the application
2. User clicks "View Schedule"
3. System displays all tasks with due dates and status

- **Implementation:**

1. Create a Python script to manage tasks
2. Use a JSON file for local storage
3. Implement add task, view schedule, and mark task as completed functionality
4. Test the application

- **Testing:**

1.Unit testing(individual funcyions)
2. Integration testing (entire application)
3. User acceptance testing (UAT)

- **Deployment:**

1. Deploy the application on Emily's local machine
2. Provide user documentation and support

- **Maintenance:**

1. Regularly update the application with new features
2. Fix bugs and issues
3. Provide ongoing support

This case study outlines the requirements, implementation, testing, deployment, and maintenance of a schedule management application for Arnavi

➢ MINI PROJECT CODE

TIME.PY - C:/Users/shail/OneDrive/Desktop/python/TIME.PY (3.12.5)

File Edit Format Run Options Window Help

```
import json

def load_schedule():
    try:
        with open("schedule.json", "r") as f:
            return json.load(f)
    except FileNotFoundError:
        return []

def save_schedule(schedule):
    with open("schedule.json", "w") as f:
        json.dump(schedule, f, indent=4)

def add_task():
    task_name = input("Enter task name: ")
    due_date = input("Enter due date (optional): ")
    schedule = load_schedule()
    schedule.append({"name": task_name, "due_date": due_date, "completed": False})
    save_schedule(schedule)
    print("Task added!")

def view_schedule():
    schedule = load_schedule()
    for task in schedule:
        print(f"{task['name']}: {task['due_date']}: {task['completed']}")

def mark_task_completed():
    task_name = input("Enter task name: ")
    schedule = load_schedule()
    for task in schedule:
        if task["name"] == task_name:
            task["completed"] = True
            save_schedule(schedule)
            print("Task marked as completed!")
            return
    print("Task not found!")

while True:
    print("1. Add Task")
    print("2. View Schedule")
    print("3. Mark Task as Completed")
    print("4. Exit")

    choice = input("Choose an option: ")

    if choice == "1":
        add_task()
```

Ln: 55 Col: 0

untitled

File Edit Format Run Options Window Help

```
elif choice == "2":
        view_schedule()
    elif choice == "3":
        mark_task_completed()
    elif choice == "4":
        break
```

Ln: 7 Col: 0

➢ OUTPUT:


IDLE Shell 3.12.5
File Edit Shell Debug Options Window Help
Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug 6 2024, 20:45:27) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/shail/OneDrive/Desktop/python/TIME.PY ==========
1. Add Task
2. View Schedule
3. Mark Task as Completed
4. Exit
Choose an option: 1
Enter task name: TASK 1
Enter due date (optional): 2024-09-25
Task added!
1. Add Task
2. View Schedule
3. Mark Task as Completed
4. Exit
Choose an option: 2
TASK 1: 2024-09-25: False
1. Add Task
2. View Schedule
3. Mark Task as Completed
4. Exit
Choose an option: 3
Enter task name: TASK 1
Task marked as completed!
1. Add Task
2. View Schedule
3. Mark Task as Completed
4. Exit
Choose an option: 4
>>>
Ln: 31 Col: 0

- **Advantages:**

- Easy to use and navigate
- Fast data retrieval
- Local storage (no online storage required)
- User-friendly interface
- Ideal for individuals with multiple tasks and deadlines

➢ **Disadvantages:**

- Limited features compared to commercial scheduling software
- No reminders or notifications
- No collaboration features

➢ **Conclusion:**

In conclusion, the Schedule Manager program is a simple yet effective tool for managing tasks and deadlines. It allows users to add tasks with due dates, view their schedule, and mark tasks as completed. The program uses a JSON file for local storage, making it easy to use and maintain.

The program's features and functionality make it an ideal solution for individuals who need to manage multiple tasks and deadlines, such as freelancers, students, and professionals.

Its user-friendly interface and fast data retrieval capabilities make it easy to use and navigate.

Overall, the Schedule Manager program is a valuable tool for anyone looking to improve their productivity and stay organized. Its simplicity, effectiveness, and ease of use make it a great solution for managing tasks and deadlines.