GROUP MEMBERS

ATWIJUKIRE APOPHIA M23B23/051

WATNEN ERIC OYWAK M23B23/066

Purpose

The **Scheduling Assistant** is a console-based Python application designed to help users organize, prioritize, and visualize their schedules. It allows users to:

1. Add tasks with specified details.
2. Sort tasks based on criteria.
3. Find tasks overlapping specific times.
4. Analyze task density for specified intervals.
5. Visualize tasks using a Gantt chart.

System Design Overview

The system consists of the following:

- **Core Data Structure:** A list of tasks, where each task is represented as a tuple: (start_time, end_time, task_type, description, priority)
- **Functional Modules:** Methods for managing, analyzing, and visualizing tasks.

Functional Requirements

1. **Add Task:** Add a task to the schedule while maintaining the sorted order.
2. **Sort Tasks:** Reorganize the task list based on a specified criterion.
3. **Find Task by Time:** Identify tasks overlapping a given time.
4. **Analyze Busy Slots:** Summarize task density over time intervals.
5. **Plot Gantt Chart:** Visualize tasks using a Gantt chart grouped by type.

Pseudo-Code

1. Add Task

```
FUNCTION add_task(start, end, task_type, description, priority):
    CREATE a tuple for the task (start, end, task_type, description, priority)
    INSERT task into the sorted list using bisect.insort
    DISPLAY "Task added successfully"
```

2. **Sort Tasks**

```
FUNCTION sort_tasks(key):
    IF key == 'deadline':
        CALL merge_sort(tasks, lambda x: x[1])  # Sort by end time
    ELSE IF key == 'priority':
        CALL merge_sort(tasks, lambda x: -x[4])  # Sort by descending priority
    ELSE IF key == 'type':
        CALL merge_sort(tasks, lambda x: x[2])  # Sort alphabetically by type
    DISPLAY "Tasks sorted based on {key}"
```

3. **Find Task by Time**

```
FUNCTION find_task_by_time(query_time):
    INITIALIZE overlapping_tasks = []
    FOR each task IN tasks:
        IF task.start_time <= query_time <= task.end_time:
            ADD task to overlapping_tasks
    IF overlapping_tasks IS NOT EMPTY:
        DISPLAY overlapping_tasks
    ELSE:
        DISPLAY "No tasks found for the specified time"
```

4. **Analyze Busy Slots**

```
FUNCTION analyze_busy_slots(interval_hours):
    IF tasks IS EMPTY:
```

```
    DISPLAY "No tasks to analyze"
    RETURN

  SET min_time = minimum start_time of tasks
  SET max_time = maximum end_time of tasks
 SET interval = interval_hours as timedelta
  INITIALIZE current_time = min_time
  INITIALIZE busy_slots = []

  WHILE current_time < max_time:
    SET end_time = current_time + interval
    COUNT tasks overlapping [current_time, end_time]
    APPEND (current_time, end_time, count) to busy_slots
    SET current_time = end_time

  FOR each slot IN busy_slots:
    DISPLAY time range and task count
```

5.Plot Gantt Chart

```
  FUNCTION plot_gantt_chart():
    INITIALIZE personal_tasks = filter tasks where type = 'personal'
    INITIALIZE academic_tasks = filter tasks where type = 'academic'

    INITIALIZE figure and axis for plotting

    PLOT personal_tasks as blue bars with labels
    PLOT academic_tasks as green bars with labels

    SET axis labels and chart title
    DISPLAY the Gantt chart
```

6. Merge Sort Helper Functions

```
FUNCTION merge_sort(data, key):
  IF data has 1 or fewer elements:
    RETURN data
  SPLIT data into left and right halves
  SORT left = merge_sort(left, key)
  SORT right = merge_sort(right, key)
  RETURN merge(left, right, key)

FUNCTION merge(left, right, key):
  INITIALIZE result = []
  INITIALIZE pointers i, j = 0

  WHILE i < len(left) AND j < len(right):
    IF key(left[i]) <= key(right[j]):
      ADD left[i] to result
      INCREMENT i
    ELSE:
      ADD right[j] to result
      INCREMENT j

  APPEND remaining elements from left and right to result
  RETURN result
```

Data Flow

1. **Input:** User provides task details or queries via console inputs.
2. **Processing:** Methods manipulate the internal task list based on the user command.
3. **Output:** Results are displayed on the console or visualized using a Gantt chart.

Tools & Libraries

1. **bisect:** Efficiently insert tasks in sorted order.
2. **datetime:** Manage task start/end times and time intervals.
3. **matplotlib:** Generate Gantt charts.
4. **Custom Merge Sort:** Sort tasks with customizable keys.