Technical Design Document: Personal Scheduling Assistant

Project Overview

The **Personal Scheduling Assistant** is a desktop-based task scheduling application designed to help users organize and optimize their time effectively. It provides tools for task management, schedule optimization, and visualization through a Gantt chart. The application is built using Python's `tkinter` for the GUI and leverages algorithms to optimize task schedules based on deadlines, priorities, and durations.

Functional Requirements

Core Features

1. **Task Management:**
   - Add, view, and manage tasks.
   - Attributes for each task:
     - Task ID
     - Description
     - Deadline (date and time)
     - Priority (integer value)
     - Task Type (e.g., personal, academic)
     - Duration (in minutes)
2. **Task Sorting and Searching:**
   - Sort tasks by:
     - Deadline
     - Priority
     - Task type
   - Search tasks by a specific deadline.
3. **Schedule Optimization:**
   - Select tasks that maximize priority within a given time limit using dynamic programming.
4. **Visualization:**
   - Display a Gantt chart showing scheduled tasks and their durations relative to deadlines.

Non-Functional Requirements

1. **Usability:**
   - Easy-to-use interface with clear input fields and intuitive buttons.
2. **Scalability:**
   - Efficient handling of up to 100 tasks without performance degradation.
3. **Reliability:**
   - Ensure data consistency when adding, sorting, and optimizing tasks.
4. **Portability:**
   - Cross-platform compatibility for Windows, macOS, and Linux.

System Design

Architecture Overview

The application uses a **modular design** consisting of the following components:

1. **Task Management System** (Handles task creation and storage using `Task` and `Scheduler` classes).
2. **User Interface** (Built with `tkinter` to provide GUI interaction).
3. **Optimization Algorithm** (Dynamic programming for schedule optimization).
4. **Visualization Tool** (Uses `matplotlib` to create Gantt charts).

Class Design

1. Task Class

Encapsulates task details and provides comparison logic for sorting.

- **Attributes:**
  - `task_id`: Unique identifier.
  - `description`: Task description.
  - `deadline`: Deadline as a `datetime` object.
  - `priority`: Integer representing the task's importance.
  - `task_type`: Task category (e.g., personal or academic).
  - `duration`: Time required to complete the task (in minutes).
- **Methods:**
  - `__init__(...)`: Constructor to initialize task attributes.
  - `__lt__(...)`: Enables comparison based on deadlines for sorting.

Pseudo code for Task Class:

```
 Initialize task_id, description, deadline, priority, task_type, duration

Convert deadline from string to datetime object

Method __lt__(self, other):

 Return True if self.deadline < other.deadline
```

2. Scheduler App Class

Provides the user interface for interaction with the application.

- **Attributes:**
  - `root`: Root `tkinter` window.
  - Input fields (`desc_entry`, `deadline_entry`, etc.) for task details.
  - `tree`: Tree view widget for displaying tasks.
- **Methods:**
  - `add_task()`: Captures input and adds a task to the scheduler.
  - `optimize_schedule()`: Prompts the user for available time and displays optimized tasks.
  - `show_gantt_chart()`: Calls `Scheduler.plot_schedule()` to show the Gantt chart.

Pseudo code for Task Class:

```
Initialize tasks as an empty list

  Method add_task(task):

  Method add_task(task):

  Add task to the heap using heapq

    Method get_sorted_tasks(by):

        If by == 'deadline', sort tasks by deadline

        If by == 'priority', sort tasks by priority (descending)

        If by == 'type', sort tasks alphabetically by type

        Return sorted tasks list
```

```
Method optimize_schedule(total_minutes):
    Initialize DP array dp[n+1][total_minutes+1]
    Fill DP array to find maximum priority within time limit
    Trace back to retrieve selected tasks
    Return list of selected tasks
Method plot_schedule():
    Create Gantt chart using matplotlib
```

UI Design

Key Components

1. **Task Entry Frame:**
   - Fields for entering task details.
   - "Add Task" button to save the task.
2. **Task Display:**
   - Treeview widget for displaying task attributes.
3. **Buttons:**
   - "Optimize Schedule" for optimization.
   - "Show Gantt Chart" for visualization.

Data Flow

Task Addition

1. User inputs task details in the form.
2. `add_task()` method creates a `Task` object and adds it to the scheduler.
3. Task details are displayed in the Treeview widget.

Schedule Optimization

1. User specifies available time.
2. `optimize_schedule()` uses the DP algorithm to compute the optimal schedule.
3. Optimized tasks are displayed in a popup message.

Visualization

1. User clicks "Show Gantt Chart."
2. `plot_schedule()` generates a Gantt chart using `matplotlib`.

Future Enhancements

1. **Save and Load Functionality:** Allow users to save tasks to a file and load them later.
2. **Recurring Tasks:** Add support for repeating tasks.
3. **Enhanced Visualization:** Include color coding for task types in the Gantt chart.
4. **Mobile Version:** Develop a mobile application for increased portability.

Tools and Technologies

- **Programming Language:** Python
- **Libraries:**
  - `tkinter` (GUI)
  - `heapq` (Task management)
  - `matplotlib` (Visualization)
- **Environment:** Cross-platform (Windows, macOS, Linux)