Technical Specification for Prioritized To-Do List App

Project Overview

The Prioritized To-Do List App is an application meant to make life easier when managing tasks from different responsibilities, both personal and professional. This dynamic, user-friendly tool has been planned with great organization in mind by letting users create tasks, assign them to a specific priority level like high, medium, or low, and label them under categories like "Work", "Personal", or "School." Tasks can also include due dates that will notify you of their soon-to-expire dates so important responsibilities will not be forgotten.

The users will be able to filter tasks by priority, due date, or category in order to focus on the most important ones included in their to-do list. Moreover, the history feature in the application is available for users to review the tasks completed over time and develop a feeling of being productive and accomplished. For example, a busy professional like Sarah can add tasks in managing her workload efficiently: "Complete project report". Notifications will pop up to remind her of impending deadlines, and she can mark all the tasks easily as completed.

First of all, it will be implemented in Python, and the desktop implementation will be done using Tkinter. The tasks will be stored in a CSV file, though it is intended to migrate to SQLite for handling data more effectively. Application design will aim at simplicity: easy flow and resolving issues like the notification system and the organization of the tasks effectively.

Moving forward, the road map is to have this application evolve into a web-based application using Flask as the back end and Bootstrap for the front-end. This would provide an even more interactive and highly accessible user interface while retaining all the core functionalities that make this application very helpful in task management at daily levels.

1. Architecture

Frontend: Tkinter (Python GUI toolkit)
Backend: Pure Python
Data Storage: Initially using CSV files, with a plan to transition to SQLite for version 2.

2. Data Structure

The main data structure for tasks will be a list of dictionaries, where each dictionary represents a task with the following attributes:
- title: String - The name of the task.
- category: String - The category of the task (e.g., Work, Personal, School).
- priority: String - The priority level (High, Medium, Low).

- deadline: Date - The due date for the task.
- status: Boolean - Indicates if the task is completed (True/False).

3. Functional Requirements

- Task Creation: Users can create a new task by entering the title, category, priority, and deadline.
- Task Deletion: Users can delete a task from the list.
- Mark as Done/Not Done: Users can toggle the status of a task between completed and not completed.
- Notifications: The app will provide notifications for upcoming deadlines.
- Filtering: Users can filter tasks by priority, due date, or category.
- Display Methods: A clear and intuitive display of tasks, showing all relevant details, and a history of completed tasks.

4. User Interface Components

- Main Window: Display current tasks and options for creating or deleting tasks.
- Input Fields: For entering task details (title, category, priority, deadline).
- Buttons:
  - Add Task
  - Delete Task
  - Mark as Done
  - Filter Tasks
- Notification System: A mechanism to alert users of approaching deadlines, potentially using Tkinter's messagebox.

5. Technical Implementation Details

- CSV File Handling: Use Python's built-in csv module for reading and writing tasks to a CSV file.
- Tkinter Layout: Utilize Frame, Label, Entry, and Button widgets to create the user interface.
- Date Handling: Use Python's datetime module to manage and compare deadlines.
- Notification System: Implement a simple polling mechanism to check for upcoming deadlines periodically.

6. Data Flow

- Input: User inputs task details via the UI.
- Processing: The app processes inputs to create, modify, or delete tasks and checks for notifications.
- Output: Display updated task lists and notifications in the GUI.

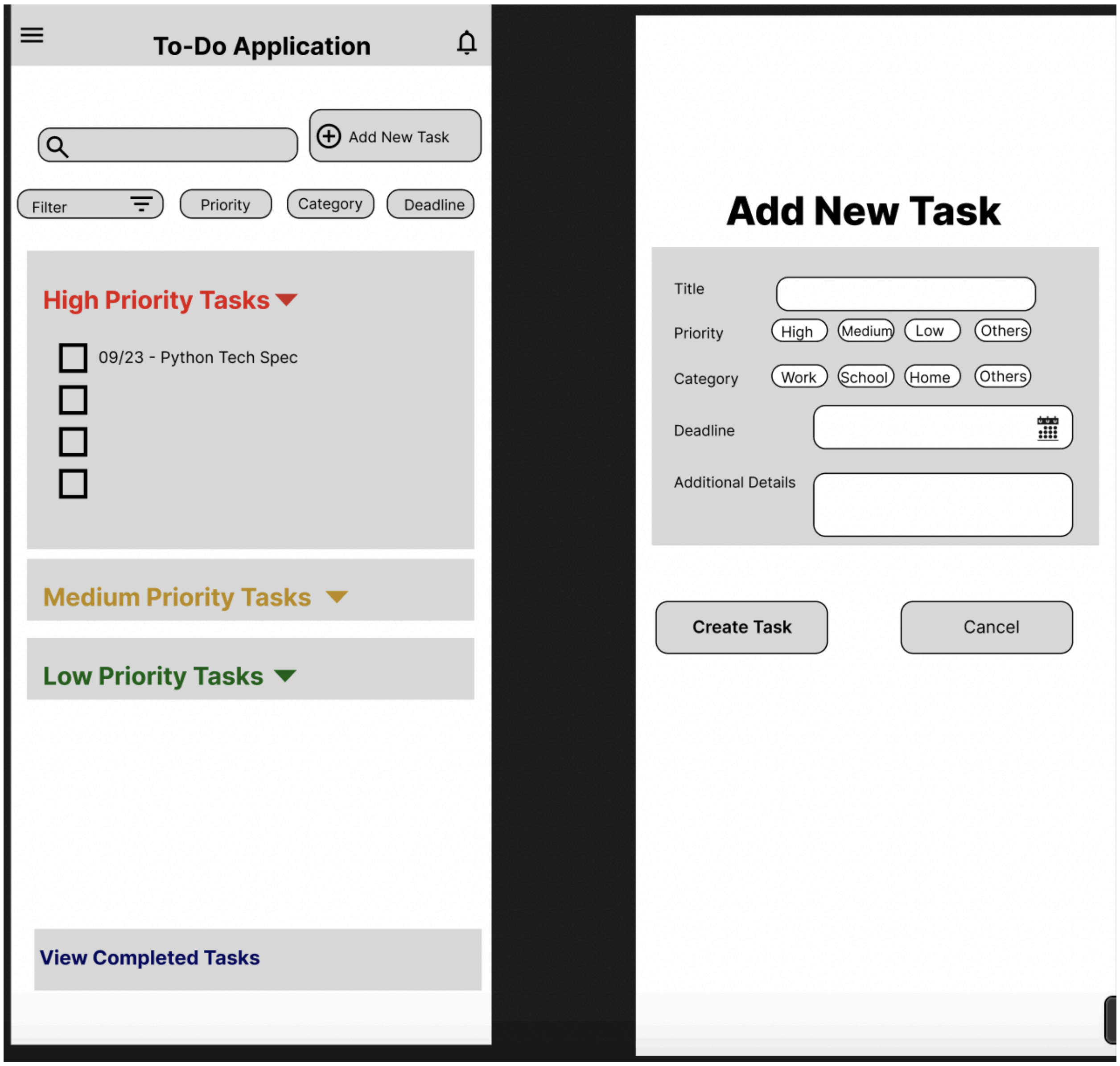
7. Future Considerations

- Transition from CSV file storage to SQLite for enhanced data handling.
- Implement bootstrap or react for front-end
- Implement a more complex notification system for better user experience.

Final Assessment

- Biggest Change: Streamlined functionality for better user interaction.
- Confidence Level: Medium confidence in implementing the core features as specified.
- Potential Problems: Ensuring the notification system works seamlessly.
- Areas Needing Help: Guidance on integrating the filter and notification features effectively.

Rough Sketch:



Github Repo Link: https://github.com/Divya-Suresh-24/to-do-application-project