# Project Outline: Custom To-Do List Application for Windows

#### 1. Project Goals

* **Primary Goal:** Create a user-friendly to-do list app for Windows that helps track and organize tasks, with customizable reminders to ensure tasks are completed.
* **Secondary Goal:** Enhance your Windows application development skills while learning to integrate useful features like notifications, data storage, and a clean UI.

# 2. Project Setup

* **Tech Stack**:
  + **Language**: Python (using Tkinter or PyQt), C# (WPF), or Electron.js (JavaScript)
  + **Database**: SQLite or Local JSON file for simpler data storage, or SQLite for more structured storage.
  + **Notifications**: Windows Notifications (via win10toast library for Python) or native notifications if using C#.
* **Development Environment**: Set up IDE (VS Code, PyCharm for Python, Visual Studio for C#) and version control (GitHub/Git) for project tracking.

# 3. Core Functionalities

* **Task Creation**:
  + Ability to add tasks with attributes like title, description, priority level, and due date.
* **Task Management**:
  + Mark tasks as complete, edit task details, and delete tasks.
* **Priority and Categories**:
  + Allow users to set priorities (e.g., High, Medium, Low) and assign categories (e.g., Work, School, Personal).
* **Reminder System**:
  + Set reminders for each task based on due date, with customizable notification times (e.g., 30 minutes, 1 hour before).
* **Filter and Sort**:
  + Options to filter tasks by date, priority, and completion status, and sort tasks by due date or priority.

# 4. Advanced Features (Optional)

* **Recurring Tasks**:
  + Add support for daily, weekly, or custom recurrence patterns.
* **Calendar Integration**:
  + Display tasks in a calendar view to provide a visual timeline.
* **Customizable Themes**:
  + Light and dark mode, with options for custom color schemes.
* **Progress Tracking and Analytics**:
  + Track completed tasks and time management stats (e.g., tasks completed on time vs. late).
* **Data Export**:
  + Export task data as CSV or JSON for backup or external analysis.

# 5. UI/UX Design

* **Task List View**:
  + A clear list layout with task details visible, using icons or color codes for priority.
* **Task Details Panel**:
  + Pop-up or side panel for viewing and editing full task details.
* **Notification and Reminder Alerts**:
  + Non-intrusive notifications with Windows notifications or in-app pop-ups.
* **Customization**:
  + Basic settings menu for adjusting reminder times, notification style, and theme preferences.

# 6. Project Phases & Milestones

* **Phase 1**: Setup & Basic UI
  + Set up project structure, initial UI layout for task list and add/edit task forms.
* **Phase 2**: Core Features
  + Implement task creation, deletion, editing, and task list display with filtering options.
* **Phase 3**: Reminder System & Notifications
  + Integrate reminders, notification scheduling, and Windows notifications.
* **Phase 4**: Advanced Features
  + Add recurring tasks, priority sorting, and calendar integration if desired.
* **Phase 5**: Testing & Finalizing
  + Conduct thorough testing, fix bugs, and ensure a smooth, user-friendly experience.
* **Phase 6**: Deployment
  + Package as a Windows executable and test installation and performance on multiple Windows setups.

# 7. Future Enhancements

* **Cross-Platform Compatibility**:
  + Explore options to make the app work on other OS like macOS or Linux.
* **Cloud Sync**:
  + Enable syncing of tasks across devices via cloud storage integration (e.g., Google Drive, Dropbox).
* **Voice Assistant Integration**:
  + Integrate with Cortana or other voice assistants for task reminders and quick task addition.

# 1. Setting Up the Environment

* **IDE**: Use PyCharm, VS Code, or any IDE you're comfortable with.
* **Python Libraries**:
  + **Tkinter** or **PyQt** for the GUI.
  + **SQLite3** for task data storage, or a JSON file if you prefer simpler storage.
  + **win10toast** for desktop notifications.
* **Packaging Tool**: **PyInstaller** will be used to convert the Python script into a .exe file.

# 2. Core Modules

**A. Task Data Storage**

* **SQLite Database**:
  + Use SQLite as it’s lightweight and well-suited for managing structured data.
  + Define a simple table with fields for Task\_ID, Title, Description, Priority, Due\_Date, and Status.
  + SQL queries can help add, edit, delete, and retrieve tasks.
* **Alternative**: A JSON file, if you want simpler, flat-file storage without structured query requirements.

**B. GUI Design with Tkinter or PyQt**

* **Tkinter**:
  + Easy to learn and sufficient for basic UI needs.
  + Ideal for creating entry fields, buttons, and task display lists.
* **PyQt** (if you want a more polished UI):
  + Requires the PyQt5 package, offering more customization and a modern look.
  + Use Qt Designer to design the GUI visually, then convert it into Python code.
  + **Suggested UI Layout**:
    - **Main Window**: Task list, sorting, and filtering options.
    - **Task Form Window**: For adding/editing tasks with fields for title, due date, and priority.

**C. Task Management Functions**

* **Add Task**: A form that collects details and saves to the SQLite database.
* **Edit Task**: Retrieve and modify a specific task’s details.
* **Delete Task**: Remove a task from the list and database.
* **Mark Complete**: Update the task status in the database.
* **Filter & Sort Tasks**: Sorting and filtering by priority, due date, and completion status.

**D. Reminder and Notification System**

* **Setting Reminders**: Allow users to set due dates for tasks.
* **Desktop Notifications**:
  + Use win10toast to send toast notifications as task reminders.
  + Schedule reminders to pop up based on the due date/time, using Python’s time or datetime library in combination with a loop or scheduler.
  + Example reminder code:

python

Copy code

from win10toast import ToastNotifier

toast = ToastNotifier()

toast.show\_toast("Task Reminder", "You have a task due soon!", duration=10)

# 3. Advanced Features (Optional)

* **Recurring Tasks**: Set a recurrence pattern and have the app automatically duplicate tasks based on the frequency.
* **Task Prioritization**: Color-code tasks in the GUI based on priority level.
* **Theme Customization**: Light and dark modes, using custom Tkinter themes or styling in PyQt.
* **Data Export**: Export task data to a .csv or .json file for backup.

# 4. Converting to .exe with PyInstaller

* Install PyInstaller:

bash

Copy code

pip install pyinstaller

* Navigate to your project directory and create the .exe:

bash

Copy code

pyinstaller --onefile --windowed your\_script.py

* **Options**:
  + --onefile packages everything into a single .exe file.
  + --windowed prevents a terminal window from opening with the app.
* **Output**: The .exe file will be located in the dist folder of your project directory.

# 5. Final Testing and Deployment

* **Testing**: Run the .exe on multiple Windows setups if possible to ensure compatibility.
* **Bug Fixes**: Ensure error handling, especially for database operations, to prevent crashes.
* **Installation Package**: You could use Inno Setup to create a simple installer for your .exe file, making it easier to distribute.