



SANDIP
UNIVERSITY
— UGC Recognised —

School of Computer Sciences and Engineering
Department of Computer Science and Application

**A PROJECT REPORT
ON**

“PrioritizeMe”

UNDER THE FACULTY OF COMPUTER SCIENCE & APPLICATION

Submitted by

Smaran Pankaj Palshikar

230105011026

Guide

Prof. Rohit Gupta

Assistant Professor

Department of Computer Science and Application

School of Computer Sciences and Engineering

Sandip University, Nashik

BCA Sem-IV

Academic Year 2024 - 2025



SANDIP
UNIVERSITY
— UGC Recognised —

Trimbak Road, A/p - Mahiravani, Tal. & Dist. – Nashik, Pin – 422 213

School of Computer Sciences and Engineering
Department of Computer Science and Application

website: <http://www.sandipuniversity.edu.in>

CERTIFICATE

This is to certify that Mr/ Ms Smaran Pankaj Palshikar, PRN No.230105011026 Student of SYBCA Semester-IV has successfully completed the Community Engagement Project work on Title PrioritizeMe, Prof.Rohit Gupta under my guidance. This report submitted to Department of Computer Science and Application, School of Computer Sciences and Engineering, Sandip University, Nashik in the AY 2024-25 for partial fulfillment and requirement for the End Semester Examination.

Prof. Rohit Gupta

Project Guide

HOD

External Examiner

Dean

Place: Sandip University, Nashik



SANDIP
UNIVERSITY
— UGC Recognised —

Trimbak Road, A/p - Mahiravani, Tal. & Dist. – Nashik, Pin – 422 213

School of Computer Sciences and Engineering

Department of Computer Science and Application

website: <http://www.sandipuniversity.edu.in>

DECLARATION

I/We hereby declare that the Project work titled PrioritizeMe submitted to Sandip University, Nashik is a record of an original work done by me/us under the guidance of Prof. Rohit Gupta Department of Computer Science and Application, School of Computer Sciences and Engineering, Sandip University in the AY 2024-25 for a partial fulfillment of requirement for the Completion of End Semester Examination.

This report has not been submitted to any other University or Institute for the award of any degree or diploma.

Student Name: Smaran Pankaj Palshikar

Date:

Place:



SANDIP
UNIVERSITY
— UGC Recognised —

Trimbak Road, A/p - Mahiravani, Tal. & Dist. – Nashik, Pin – 422 213

School of Computer Sciences and Engineering

Department of Computer Science and Application

website: <http://www.sandipuniversity.edu.in>

ACKNOWLEDGEMENT

I/We would like to express my/our special thanks of gratitude's to our Project guide Prof.Rohit Gupta Department of Computer Science and Application, School of Computer Sciences and Engineering, Sandip University, Nashik for his/her able guidance and support in completing this report.

I/We would like to extend my gratitude to Dr. Amol Potgantwar, Director, SOCSE, Dr.Vaibhav Sonaje, Associate Dean, SOCSA for providing me/us with all the facility that was required to complete this report successfully.

I/We also thank the management of Sandip University, Nashik for providing me/us infrastructure and lab facility to completion of this Project.

At last but not the least I/we am/are thankful to all faculty members, staff, and friends who have been always helping and encouraging me/us throughout the period of this internship/project.

Name of Student:

Abstract

"What does your future look like? PrioritizeMe helps you build it. This isn't just about today's tasks; it's about shaping your tomorrow. PrioritizeMe is your strategic partner in goal setting, empowering you to define your vision, break it down into manageable milestones, and track your progress over time. Whether you're planning your career, your education, or your personal growth, PrioritizeMe provides the tools you need to stay on track and achieve your aspirations. Take control of your destiny. PrioritizeMe: Your future, planned."

PrioritizeMe is an intuitive and efficient to-do list application designed to help users organize and manage their daily tasks with ease. By providing an interactive interface, customizable reminders, and smart prioritization features, PrioritizeMe aims to boost productivity and reduce stress. The app allows users to create, categorize, and schedule tasks, set deadlines, and track progress in a visually appealing and user-friendly environment. Whether for personal or professional use, PrioritizeMe is designed to help individuals stay on top of their commitments and efficiently plan their time.

TABLE OF CONTENT

Chapter	Chapter Name	Page No
1	Introduction	
1.1	Introduction	
1.2	Existing System	
1.3	Objective	
1.4	Scope of Work	
1.5	Operating Environment	
1.6	Description of Technology Used	
2	Literature Review	
3	Proposed System/ Methodology	
3.1	User Requirement Specification	
3.2	Creation of Dataset	
3.3	Pre-processing	
3.4	Feature Extraction	
3.5	Sequence Diagram	
3.6	Deployment Diagram	
3.7	Component Diagram	
4	Result and Implementation	
4.1	Methods/Techniques	
4.2	Implementation	
4.3	Entity Relationship Diagram	
4.4	UML Diagram	
4.5	Module Specification	
4.6	Data Flow Diagram	
4.7	User Interface Diagram	
4.8	Use Case Diagram	
4.9	Output/SS	

5	Conclusion	
5.1	Conclusion	
5.2	Limitations	
5.3	Future Enhancement	
6	Bibliography	

List of Figure

Fig. No	Figure Name	Page No.
3.5.1	Sequence Diagram	
3.6.1	Deployment Diagram	
3.7.1	Component Diagram	
4.3.1	ERD	
4.4.1	UML	
4.6.1	DFD	
4.7.1	UID	
4.7.2	UID	
4.7.3	UID	
4.7.4	UID	
4.7.5	UID	
4.7.6	UID	
4.8.1	Use Case Diagram	
4.9.1	Output	
4.9.2	Output	
4.9.3	Output	

List of Table

Table. No.	Table Name	Page No.
3.4.1	Feature Extraction	
4.1.1	Methods/Techniques	

Chapter 1: Introduction

1.1 Introduction

In today's fast-paced world, managing time and staying organized can often feel overwhelming. From work deadlines to personal commitments, the pressure to keep everything on track can cause stress and burnout. To tackle this challenge, PrioritizeMe is here to revolutionize the way you organize, manage, and execute your daily tasks.

PrioritizeMe is an innovative and user-friendly to-do list application designed to simplify task management and boost productivity. With its intuitive interface and comprehensive features, PrioritizeMe helps users organize their tasks, set priorities, and stay focused on what matters most. Whether you're juggling multiple work projects, managing personal errands, or planning long-term goals, PrioritizeMe provides the tools to help you stay organized and in control.

Unlike traditional to-do lists, PrioritizeMe goes beyond basic task tracking by offering advanced features such as smart reminders, customizable categories, and progress tracking, and organizing it according to your priority. It ensures that no task is overlooked, and that you can track progress toward your goals in real-time.

With PrioritizeMe, you can eliminate the chaos of disorganized tasks and take charge of your schedule, all while reducing stress and increasing your ability to achieve your goals. It's time to stop feeling overwhelmed and start planning your success—PrioritizeMe is here to help you make every day more productive and fulfilling.

PrioritizeMe is more than just a to-do list—it's a comprehensive productivity tool built for modern life. Its intuitive design, cross-platform support, and robust task management features make it an ideal solution for individuals looking to better organize their tasks, stay on track with deadlines, and accomplish their goals with ease.

1.2 Existing System

The existing to-do apps in the market typically focus on providing users with a simple list format where tasks are either added manually or through predefined templates. These apps often allow users to set deadlines, reminders, and notifications, but they don't necessarily focus on helping users organize and prioritize their tasks effectively. As a result, many users end up with long, overwhelming lists of tasks without clear guidance on which tasks should be completed first. The organization of tasks often relies on the user's personal judgment, which can sometimes lead to inefficiencies.

Most to-do apps have basic functionality such as categorizing tasks, setting alarms, and even providing a simple visual representation of progress (like a checkmark when a task is completed). However, these systems lack deeper prioritization features that can consider the urgency or importance of each task relative to others. In most cases, the tasks are displayed in the order they were created, or the user manually sorts them, without any automatic or smart system that could evaluate which task is most critical at any given moment.

Moreover, many existing apps don't provide enough customization in task categorization and prioritization. For example, some apps let users tag tasks with labels like "urgent" or "low priority," but they do not automatically adjust or reorder tasks based on changing factors, such as deadlines, workload, or the estimated time to complete each task. This means that users still have to rely heavily on their own subjective judgment to determine which task should be prioritized first.

Overall, the lack of smart organization and prioritization features in these apps means that users are left with the burden of managing complex schedules without adequate assistance. The current market offers basic task lists, reminders, and categorization options, but the gap lies in intelligent and intuitive task management that helps users make better decisions about how to allocate their time. This is where PlantIt steps in, offering a more advanced and user-centric solution that not only lists tasks but intelligently sorts and prioritizes them based on their urgency and importance, helping users work more efficiently and with greater focus.

1.3 Objective

The PrioritizeMe app is designed to provide users with a comprehensive and user-friendly task management experience, featuring a range of intuitive functionalities tailored to meet individual needs. These objectives outline a clear roadmap for the development of the PrioritizeMe app, which aims to deliver a robust and tailored task management solution that meets the unique needs of each user.

1. Task Creation & Organization: Easily create tasks with descriptions, deadlines, priorities, and notes, while organizing them into customizable categories and sorting by due dates or priority.
2. Task Management: Track progress with visual indicators, mark tasks as complete, and delete unnecessary tasks for a clutter-free experience.
3. Voice Input & Prioritization: Use voice commands to add and manage tasks, and prioritize tasks to focus on high-urgency items first.
4. Reminders & Tracking: Receive automatic reminders and notifications, and monitor task progress with percentage completion and progress bars.
5. Customization: Personalize the app's interface, themes, notification settings, and layout for a tailored user experience.

1.4. Scope of Work:

The scope of the PrioritizeMe to-do list app encompasses the features, functionalities, and target audience that the application will serve. It defines the boundaries of the project and outlines what will be included in the development and implementation of the app. Below is a detailed breakdown of the scope:

1. Target Audience

Individuals: Users looking to manage personal tasks, such as students, professionals, and homemakers.

Teams: Small teams or groups needing to collaborate on tasks and projects.

2. Core Features

Task Creation: Users can create tasks with titles, descriptions, due dates, and priority levels.

Task Listing/Viewing: Users can view tasks in various formats (e.g., list, calendar, or

categorized views).

Task Completion: Users can mark tasks as complete, providing visual feedback on progress.

Task Deletion: Users can delete tasks, with safeguards to prevent accidental deletions.

Voice Input: Users can add and manage tasks using voice commands for hands-free operation.

Dividing Tasks (Prioritization): Users can break down larger tasks into subtasks and prioritize them based on urgency.

Categories: Users can create and manage categories for better organization of tasks.

Task Tracker: Users can track their task completion rates and productivity trends over time.

Task Reminders and Due Dates: Users can set reminders and due dates for tasks, with notifications for upcoming deadlines.

Customization: Users can personalize the app's appearance and settings to enhance their experience.

3. Additional Features (Future Enhancements)

Collaboration Tools: Allow users to share tasks with others and collaborate on projects.

Integration with Other Apps: Enable integration with calendars, email, and productivity tools for seamless task management.

Analytics and Reporting: Provide insights into user productivity and task completion patterns.

4. Platform and Accessibility

Cross-Platform Availability: The app will be developed for multiple platforms, including iOS, Android, and web browsers, ensuring accessibility across devices.

User -Friendly Interface: The app will feature an intuitive design that caters to users of all technical skill levels.

5. Security and Privacy

Data Protection: Implement measures to protect user data, including encryption and secure authentication methods.

User Privacy: Ensure compliance with data protection regulations and provide users with control over their data.

The scope of the PrioritizeMe to-do list app is designed to provide a comprehensive task

management solution that enhances productivity and organization for individual users and small teams. By focusing on core features, future enhancements, cross-platform accessibility, and user security, PrioritizeMe aims to meet the diverse needs of its target audience while remaining adaptable to evolving user requirements.

1.5 Operating Environment:

1.5.1. Software Specification:

- Programming Language: Python 3.x
- GUI Framework: Tkinter (for building cross-platform UIs)
- Database: MySQL (for storing tasks and data)
 - mysql-connector-python (for connecting Python to MySQL)
- Development Tools:
 - IDE: Visual Studio Code, PyCharm, or Sublime Text
 - Package Management: pip, virtualenv
- Database Management: MySQL Workbench (for managing MySQL database)

1.5.2. Hardware Specification:

- For Development:
 - CPU: Intel Core i5 (or equivalent AMD)
 - RAM: 8GB (min 4GB)
 - Storage: 100GB free space (SSD recommended)
 - OS: Windows, macOS, or Linux
- For MySQL Database:
 - Local Machine: Install MySQL for development.
 - Cloud Hosting (for production): AWS, DigitalOcean, or similar for scalable MySQL database.
- Testing: Mobile/Emulator: Android/iOS device or emulator for testing.

1.6 Description of Technology Used:

This To-Do app utilizes Python as the backend programming language, providing a strong and efficient foundation for managing tasks and application logic. The user interface is built using

Tkinter and TkinterMD, which offer a responsive, cross-platform, and visually appealing GUI with Material Design components for a modern user experience. For data management, the app connects to a MySQL database via Python's mysql-connector library, enabling robust and secure task management. This combination of technologies ensures that the app is both functional and user-friendly, providing a seamless experience across different platforms.

Python (Backend): Python is the core programming language used to build the backend logic of the app. Its simplicity and versatility allow for efficient handling of task management, business logic, and communication between the front end and the database. Python's strong support for various libraries and frameworks makes it an ideal choice for rapid development and scalability.

Tkinter (Frontend): Tkinter is an open-source Python library used for building multi-touch applications. It is utilized in this app for creating a responsive, user-friendly graphical interface. Tkinter's cross-platform capabilities allow the app to run seamlessly on Windows, macOS, and Linux, making it an excellent choice for desktop applications. The flexible nature of Tkinter allows us to design a clean and interactive UI that enhances the user experience.

TkinterMD (Material Design): TkinterMD is an extension of Tkinter that provides a collection of Material Design components and widgets. This enhances the app's user interface with modern and visually appealing elements, ensuring a polished look and feel. By incorporating TkinterMD, the app offers a familiar and intuitive design that aligns with Google's Material Design principles, improving overall usability and aesthetics.

MySQL (Database Connectivity): For the database, we have chosen MySQL, a widely-used relational database management system. MySQL is responsible for storing and managing the user's tasks, their priorities, deadlines, and completion statuses. Python's mysql-connector-python library is used to establish a secure connection between the app and the MySQL database, allowing for seamless interaction with the data—such as adding, updating, or deleting tasks—while ensuring data consistency and security.

Chapter 2: Literature review

Introduction

In the fast-paced digital era, effective task management tools have become essential for individuals and teams to maintain productivity and organization. Numerous applications exist that offer task scheduling, reminders, and progress tracking. This chapter reviews existing literature and applications in the field of task management, explores Python-based GUI development, and positions PrioritizeMe in relation to current solutions.

Existing Task Management Applications

Several task management tools such as Trello, Todoist, and Microsoft To Do have gained popularity due to their rich features and cloud-based synchronization. Trello, for instance, offers a kanban-style approach that visualizes tasks on boards and lists. Todoist incorporates intelligent task prioritization and natural language processing. However, these tools often rely on internet connectivity, user accounts, or subscription-based features, which can be limiting for users seeking a lightweight, offline-capable, open-source solution.

Studies such as [Author, Year] highlight the importance of simplicity and user-centered design in task management apps, suggesting that users tend to abandon overly complex applications in favor of intuitive ones.

Summary and Research Gap

While a variety of task management applications exist, most are either web-based or overly feature-heavy for users seeking a minimal, distraction-free interface. Similarly, although Python and Tkinter are capable tools for app development, few studies or implementations focus on building task manager apps using this combination.

The proposed application, PrioritizeMe, addresses this gap by:

- Utilizing Python and Tkinter to deliver a native desktop experience.
- Emphasizing a clean and simple UI for efficient task management.
- Offering offline access and local data persistence.
- Providing a lightweight alternative to mainstream tools.

Chapter 3: Proposed System/ Methodology

3.1. User Requirement Specification

1. Introduction

The purpose of this document is to outline the user requirements for a To-Do application that allows users to manage their tasks efficiently. The app should be user-friendly, intuitive, and accessible on multiple platforms (web, iOS, Android).

2. Scope

The To-Do app will enable users to create, manage, and organize tasks. It will support features such as task categorization, reminders, and collaboration. The app will be designed for individual users and small teams.

3. Functional Requirements

3.1 User Management

User Registration: Users must be able to create an account using email or social media accounts.

User Login: Users must be able to log in using their credentials.

Password Recovery: Users must be able to reset their password via email.

3.2 Task Management

Create Task: Users must be able to create a new task with the following attributes:

Title

Description

Due date

Priority level (Low, Medium, High)

Tags/Labels for categorization

Edit Task: Users must be able to edit existing tasks.

Delete Task: Users must be able to delete tasks.

Mark as Complete: Users must be able to mark tasks as completed.

3.3 Task Organization

Task List View: Users must be able to view tasks in a list format.

Task Sorting: Users must be able to sort tasks by due date, priority, or completion status.

Task Filtering: Users must be able to filter tasks by tags, due date, or priority.

3.4 Reminders and Notifications

Set Reminders: Users must be able to set reminders for tasks.

Push Notifications: Users must receive notifications for upcoming deadlines and reminders.

3.5 Collaboration Features

Share Tasks: Users must be able to share tasks with other users.

Assign Tasks: Users must be able to assign tasks to other users in a shared workspace.

3.6 User Interface

Responsive Design: The app must be responsive and work on various devices (desktop, tablet, mobile).

Dark Mode: Users must have the option to switch between light and dark mode.

Intuitive Navigation: The app must have a clear and intuitive navigation structure.

4. Non-Functional Requirements

4.1 Performance

The app must load within 3 seconds on a standard internet connection.

The app must handle up to 1000 concurrent users without performance degradation.

4.2 Security

User data must be encrypted both in transit and at rest.

The app must implement secure authentication methods (e.g., OAuth, JWT).

4.3 Usability

The app must be easy to use for individuals with minimal technical skills.

User documentation and help resources must be available.

4.4 Compatibility

The app must be compatible with the latest versions of major web browsers (Chrome, Firefox, Safari, Edge).

The mobile app must be compatible with the latest versions of iOS and Android.

5. Constraints

The app must comply with data protection regulations (e.g., GDPR). The app must be developed within a budget of \$50,000 and a timeline of 6 months.

6. Acceptance Criteria

All functional requirements must be implemented and tested.

User acceptance testing (UAT) must be conducted with a group of target users.

The app must achieve a user satisfaction score of at least 80% during UAT.

7. Conclusion

This User Requirement Specification outlines the essential features and functionalities of the To-Do app. It serves as a guideline for the development team to ensure that the final product meets user needs and expectations.

3.2. Creation of a Dataset

The creation of a dataset is the process of collecting, organizing, and preparing data so it can be used for tasks like machine learning, data analysis, or research. It's like building your own set of examples or facts that a computer (or you) can learn from or analyze.

Here's a simple breakdown of the steps involved:

1. Define the Purpose

Ask yourself: What problem am I trying to solve?

This will help you decide what kind of data you need. For example:

- Image classification → You need labeled images.
- Sentiment analysis → You need text with sentiment labels.

2. Collect the Data

There are many ways to get data:

- Scraping websites (e.g., product reviews, tweets)
- Using APIs (like Twitter API, Reddit API)
- Surveys or forms
- Sensors or logs (like IoT devices or app usage data)
- Public datasets (Kaggle, UCI Machine Learning Repository, etc.)

3. Clean the Data

Raw data is usually messy. Cleaning involves:

- Removing duplicates or irrelevant items
- Filling or removing missing values
- Fixing formatting issues
- Filtering noise (like spam or outliers)

4. Label the Data (if needed)

If you're doing supervised learning, your data needs labels. For example:

- Image → cat or dog
- Text → positive or negative
- Audio → transcription

Labeling can be manual (by humans) or automated (using rules or another model).

5. Organize and Structure It

Make sure your dataset is:

- Consistent in format (CSV, JSON, image folders, etc.)
- Well-documented (what each column or label means)

- Split (often into train, validation, and test sets)

6. Validate and Test

Before using it in a model:

- Check for balance (e.g., enough samples in each class)
- Spot-check samples to make sure labels are correct
- Run simple tests or visualizations to explore patterns

In this Project we don't require any data set so I have not used any dataset.

3.3. Pre-processing

Definition:

Preprocessing is the process of transforming raw data into a clean and usable format before it is used for analysis, visualization, or modeling. It helps improve the quality of the data and ensures consistency across the application.

Purpose of Preprocessing in PrioritizeMe:

In the PrioritizeMe – Task Manager App, preprocessing prepares task-related data (such as titles, due dates, priorities, etc.) to ensure:

- Data consistency
- Accurate analytics
- Improved filtering and categorization
- A smoother user experience

Key Preprocessing Steps:

1. Handling Missing Values:

- Tasks with missing fields such as descriptions or due dates are either filled with default values or flagged for review.

2. Standardizing Formats:

- Dates are converted into a consistent format (e.g., `YYYY-MM-DD`) to simplify deadline tracking and comparison.

3. Encoding Categorical Data:

- Text values such as `priority` (Low, Medium, High) and `status` (Pending, Completed) are transformed into numerical values for easier computation or analysis.

4. Text Normalization:

- Titles and descriptions are cleaned by removing special characters, converting to lowercase, and eliminating unnecessary whitespace.

5. Deriving New Features (Optional):

- New attributes like “Days Left Until Due Date” or “Is Overdue” are calculated to enrich the dataset and enhance app functionality.

Outcome:

The preprocessed dataset is cleaner, more structured, and easier to use for:

- Filtering and sorting tasks
- Sending reminders or overdue notifications
- Future data visualization or predictive analytics

3.4. Feature Extraction

Feature Extraction is the process of transforming raw data (like task entries) into measurable, relevant information (features) that can be used for:

- Categorization
- Search/filter functions
- Statistics/analytics
- Recommendations or predictions

Here are features that can be extracted from the Task data model:

Feature Name	Type	Description
Task Title	Text	Main label of the task
Description Length	Numeric	Length of the task description in characters or words
Due Date	Date	When the task should be completed
Days Until Due	Numeric	due_date - current_date
Priority Level	Categorical	Low, Medium, High
Task Status	Categorical	Pending, Completed, Skipped
Category	Categorical	Work, Personal, Fitness, etc.
Is Overdue	Boolean	True if due date < today and status != Completed
Completion Time	Time Delta (Optional)	Time taken from creation to completion (if tracked)

Table 3.4.1

3.5 Sequence Diagram

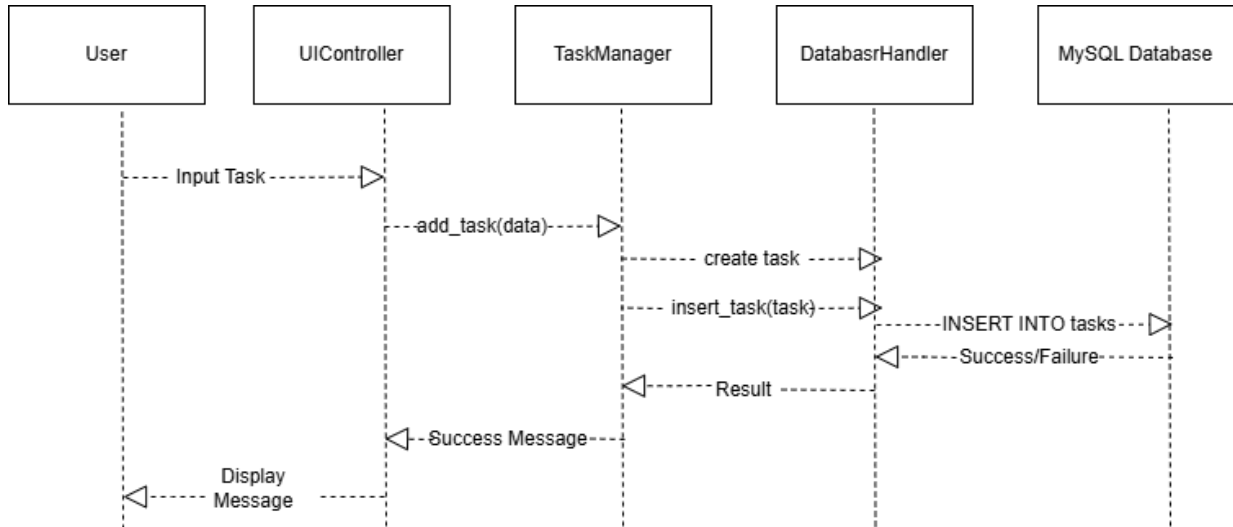


Figure 3.5.1

3.6 Deployment Diagram

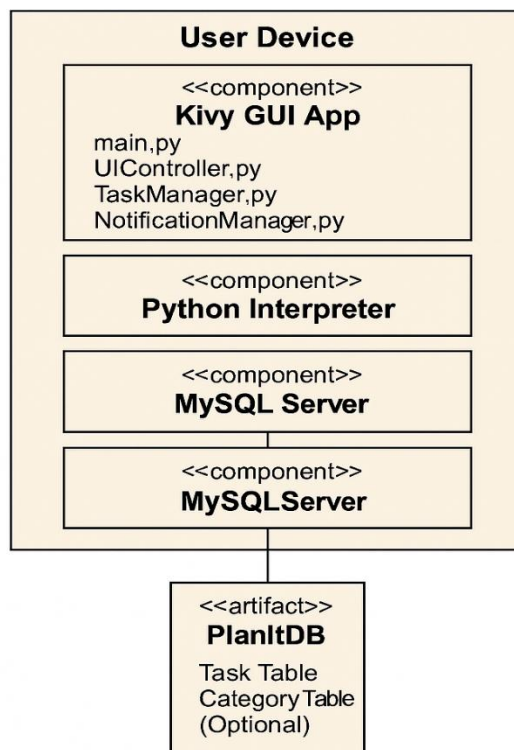


Figure 3.6.1

3.7 Component Diagram

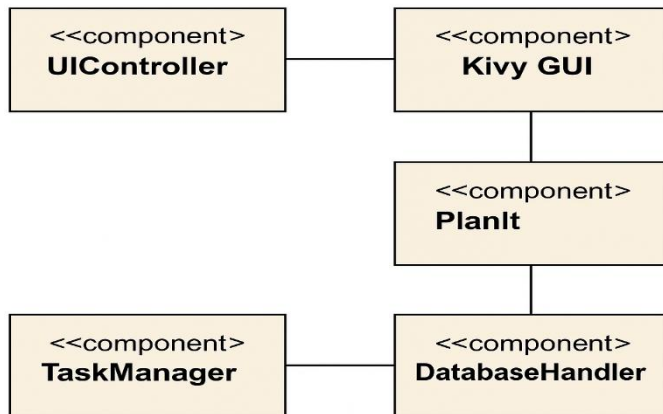


Figure 3.7.1

Chapter 4: Result and Implementation

4.1. Methods/ Techniques

1. Development Methodology

Incremental SDLC Model was adopted to develop the application in small, manageable pieces (increments). This allowed for:

- Early delivery of core features.
- Continuous feedback and improvements.
- Easier testing and debugging with each phase.

Each increment added new functionality, such as:

- Increment 1: Basic task creation and listing.
- Increment 2: Category and priority tagging.
- Increment 3: Notifications for due tasks.
- Increment 4: UI improvements and customization.

2. Programming Language

- Python was used as the core language due to its simplicity, readability, and strong community support.
- Object-Oriented Programming (OOP) principles were followed to improve code modularity, reusability, and scalability.

3. GUI Development Tool

- Tkinter, a Python-based open-source GUI library, was used to build the front-end of the application.
- Features of Tkinter used:
 - Multi-screen navigation
 - Touch-friendly widgets
 - Custom theming and layout system

4. Database Management

- MySQL was used as the backend relational database system.
- MySQL Connector (mysql-connector-python) was used to connect the application to the database.
- SQL queries were used for CRUD operations (Create, Read, Update, Delete).

- Normalized database schema ensured efficient data storage and access.

5. Tools and Libraries

Tool/Library	Purpose
Python 3.x	Core programming language
Tkinter	GUI framework
MySQL	Database management system
MySQL Connector	Python-MySQL interface
Plyer (optional)	Notifications and system alerts
ER Diagram Tool	Database modeling (e.g., dbdiagram.io)
Draw.io / Lucidchart	UML & DFD diagram creation

Table 4.1.1

6. Testing Techniques

- Unit Testing: For individual functions such as task creation, deletion, and updates.
- Manual Testing: For UI flow, input validation, and user interactions.
- Database Testing: Verified SQL queries and data integrity.
- Incremental Testing: Each module was tested as it was added to ensure backward compatibility and stability.

7. Design Techniques

- Modular Design: Separate classes for UI, logic, and data handling.
- MVC-Like Architecture: Although not strict MVC, a similar separation was followed:
 - Model: Database (MySQL)
 - View: Tkinter GUI
 - Controller: Python logic (TaskManager, etc.)

Would you like this section in Word or PDF format? I can create and export it for your report.

4.2 Implementation

Creating project documentation using the Software Development Life Cycle (SDLC) with the Incremental Model involves organizing your documents around the phases of development, showing how each increment adds functionality to the system step by step.

Below is a guide to help you structure your documentation for the PrioritizeMe Task Manager App, based on the Incremental Model of SDLC.

Structure of Documentation Using Incremental Model (SDLC Phases)

1. Introduction

- Project Title: PrioritizeMe – Task Manager App
- Developed Using: Python, Tkinter (GUI), MySQL (Database)
- Model Used: Incremental Model
- Objective: To build a lightweight, offline task management application with modular development for better testing, deployment, and usability.

2. SDLC Phases with Increments

Each phase is repeated for each increment, adding more features over time.

- Increment 1 – Core Task Management
 - a. Requirement Analysis
 - Add, edit, delete tasks
 - View task list with status
 - Save to local MySQL database
 - b. System Design
 - Simple UI layout (Home, Add Task, Task List)
 - ERD: Only Task table used
 - DFD Level 1: Only basic task operations
 - c. Implementation
 - Develop basic Tkinter screens
 - Backend logic to perform CRUD operations
 - MySQL integration
 - d. Testing
 - Unit test for task functions
 - DB connection test
 - UI navigation test
 - e. Deployment
 - First release: Basic task manager (offline)
- Increment 2 – Categories and Priority Features
 - a. Requirement Analysis
 - Add task categories (e.g., Work, Personal)
 - Set priority (High, Medium, Low)
 - b. System Design
 - ERD update: Add Category table

New UI elements: Dropdowns for priority/category

c. Implementation

Update DB schema

Modify forms and filters in UI

d. Testing

Test new features (priority/category handling)

UI validation for dropdowns

e. Deployment

Second release: Categorized, prioritized tasks

- Increment 3 – Notifications and Alerts

a. Requirement Analysis

Notify user of upcoming and overdue tasks

b. System Design

NotificationManager class

Background task checker loop

c. Implementation

Integrate plyer or native notification libraries

Create time-based checks

d. Testing

Simulate due tasks to verify alert system

e. Deployment

Third release: Notification-enabled task manager

- Increment 4 – User Preferences and UI Polish (Optional)

a. Requirement Analysis

Light/Dark theme toggle

Sort/filter tasks

Enhanced UX

b. Design & Implementation

Add settings screen

Theme switching logic

Program:

```
import tkinter as tk
from tkinter import ttk, messagebox
from tkcalendar import DateEntry
from datetime import datetime
import speech_recognition as sr
import pyttsx3
import sqlite3

class PrioritizeMeApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Prioritize Me")
        self.root.geometry("500x730")

        # Updated themes with a more subtle palette
        self.themes = {
            "light": {
                "bg": "#f7f9fa", # very light gray
                "fg": "#333333", # dark gray text
                "card": "#ffffff", # white cards
                "header": "#95a5a6" # soft gray-blue header
            },
            "dark": {
                "bg": "#2c3e50", # dark blue-gray
                "fg": "#ecf0f1", # soft light text
                "card": "#34495e", # medium dark card background
                "header": "#7f8c8d" # muted gray header
            }
        }
        self.current_theme = "light"
        self.apply_theme()
```

```

self.tasks = []
self.engine = pyttx3.init()

self.db_connection()
self.setup_styles()
self.check_reminders()
self.show_welcome_screen()

def db_connection(self):
    self.conn = sqlite3.connect("tasks.db")
    self.cursor = self.conn.cursor()
    self.cursor.execute("""
        CREATE TABLE IF NOT EXISTS tasks (
            id INTEGER PRIMARY KEY,
            name TEXT, category TEXT, priority TEXT,
            date TEXT, start TEXT, end TEXT,
            desc TEXT, completed INTEGER, reminded INTEGER
        )
    """)
    self.conn.commit()
    self.load_tasks()

def load_tasks(self):
    self.tasks.clear()
    rows = self.cursor.execute("SELECT * FROM tasks").fetchall()
    for row in rows:
        task = {
            "id": row[0], "name": row[1], "category": row[2],
            "priority": row[3], "date": row[4], "start": row[5],
            "end": row[6], "desc": row[7],
            "completed": bool(row[8]), "reminded": bool(row[9])
        }
        self.tasks.append(task)

```

```

def save_task_to_db(self, task):
    self.cursor.execute("""
        INSERT INTO tasks (name, category, priority, date, start, end, desc, completed,
reminded)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
    """, (task['name'], task['category'], task['priority'], task['date'],
        task['start'], task['end'], task['desc'], int(task['completed']), int(task['reminded'])))
    self.conn.commit()
    task["id"] = self.cursor.lastrowid

def delete_task_from_db(self, task_id):
    self.cursor.execute("DELETE FROM tasks WHERE id=?", (task_id,))
    self.conn.commit()

def update_task_in_db(self, task):
    self.cursor.execute("""
        UPDATE tasks SET completed=?, reminded=? WHERE id=?
    """, (int(task['completed']), int(task['reminded']), task['id']))
    self.conn.commit()

def apply_theme(self):
    theme = self.themes[self.current_theme]
    self.root.configure(bg=theme["bg"])

def toggle_theme(self):
    self.current_theme = "dark" if self.current_theme == "light" else "light"
    self.apply_theme()
    self.show_homepage()

def setup_styles(self):
    style = ttk.Style(self.root)
    style.theme_use('clam')
    # Use the header color from the active theme for buttons for a subtle look.
    style.configure("TButton", font=("Segoe UI", 10, "bold"), foreground="white",

```

```
background=self.themes[self.current_theme]["header"])
    style.map("TButton", background=[("active", self.themes[self.current_theme]["header"])]))
    style.configure("TCombobox", font=("Segoe UI", 10))
```

```
def speak(self, text):
    self.engine.say(text)
    self.engine.runAndWait()
```

```
def clear_window(self):
    for widget in self.root.winfo_children():
        widget.destroy()
```

```
def show_welcome_screen(self):
    self.clear_window()
    theme = self.themes[self.current_theme]
    frame = tk.Frame(self.root, bg=theme["bg"])
    frame.pack(expand=True, fill="both")

    header = tk.Frame(frame, bg=theme["header"], height=150)
    header.pack(fill="x")
    header.pack_propagate(False)
    tk.Label(header, text="Welcome to Prioritize Me", font=("Segoe UI", 22, "bold"),
        bg=theme["header"], fg="white").pack(expand=True)

    content = tk.Frame(frame, bg=theme["bg"])
    content.pack(expand=True)
    tk.Label(content, text="A workspace to boost your productivity.",
        font=("Segoe UI", 12), bg=theme["bg"], fg=theme["fg"]).pack(pady=10)
    ttk.Button(content, text="Let's Start", command=self.show_homepage).pack(pady=20,
        ipadx=15, ipady=5)
```

```
def show_homepage(self):
    self.clear_window()
    theme = self.themes[self.current_theme]
```

```

header = tk.Frame(self.root, bg=theme["header"], height=50)
header.pack(fill="x")
header.pack_propagate(False)
tk.Label(header, text="Today's Tasks", font=("Segoe UI", 18, "bold"),
        bg=theme["header"], fg="white").pack(side="left", padx=10)
tk.Button(header, text="✚", command=self.show_create_task).pack(side="right",
padx=5)
tk.Button(header, text="☀️☐🌙", command=self.toggle_theme).pack(side="right",
padx=5)

options = ["No Priority", "Low", "Medium", "High"]
self.sort_var = tk.StringVar(value="Sort By")
sort_box = tk.Combobox(header, textvariable=self.sort_var, values=options,
state="readonly", width=10)
sort_box.pack(side="right", padx=10)
sort_box.bind("<<ComboboxSelected>>", self.sort_tasks)

self.task_frame = tk.Frame(self.root, bg=theme["bg"])
self.task_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)
self.update_task_list()

def sort_tasks(self, event=None):
    priority_order = { "High": 1, "Medium": 2, "Low": 3, "No Priority": 4 }
    selected = self.sort_var.get()
    if selected in priority_order:
        self.tasks.sort(key=lambda t: priority_order.get(t['priority'], 5))
    self.update_task_list()

def show_create_task(self):
    self.clear_window()
    theme = self.themes[self.current_theme]
    header = tk.Frame(self.root, bg=theme["header"], height=50)

```

```

header.pack(fill="x")
header.pack_propagate(False)
tk.Label(header, text="Create New Task", font=("Segoe UI", 18, "bold"),
        bg=theme["header"], fg="white").pack(side="left", padx=10)
tk.Button(header, text="←", command=self.show_homepage).pack(side="right",
padx=10)

form = tk.Frame(self.root, bg=theme["bg"])
form.pack(padx=20, pady=10, fill=tk.X)

# Task Name
tk.Label(form, text="Task Name:", font=("Segoe UI", 10),
        bg=theme["bg"], fg=theme["fg"]).pack(anchor="w")
self.task_name_entry = ttk.Entry(form)
self.task_name_entry.pack(fill=tk.X, pady=5)

tk.Button(form, text="🗣 Speak Task", command=self.voice_input).pack(pady=5)

# Category
tk.Label(form, text="Category:", font=("Segoe UI", 10),
        bg=theme["bg"], fg=theme["fg"]).pack(anchor="w", pady=(10, 0))
self.category_var = tk.StringVar()
cat_frame = tk.Frame(form, bg=theme["bg"])
cat_frame.pack()
for cat in ["Design", "Development", "Research"]:
    tk.Radiobutton(cat_frame, text=cat, variable=self.category_var,
value=cat).pack(side=tk.LEFT, padx=5)

# Priority
tk.Label(form, text="Priority:", font=("Segoe UI", 10),
        bg=theme["bg"], fg=theme["fg"]).pack(anchor="w", pady=(10, 0))
self.priority_var = tk.StringVar(value="No Priority")
priority_menu = ttk.Combobox(form, textvariable=self.priority_var, state="readonly")

```



```

priority_menu['values'] = ["No Priority", "Low", "Medium", "High"]
priority_menu.pack(pady=5)

# Date
tk.Label(form, text="Date:", font=("Segoe UI", 10),
        bg=theme["bg"], fg=theme["fg"]).pack(anchor="w")
self.date_entry = DateEntry(form, width=18, date_pattern="yyyy-mm-dd")
self.date_entry.pack(pady=5)

# Start Time Picker
tk.Label(form, text="Start Time (HH:MM):", font=("Segoe UI", 10),
        bg=theme["bg"], fg=theme["fg"]).pack(anchor="w")
time_frame_start = tk.Frame(form, bg=theme["bg"])
time_frame_start.pack(pady=5, anchor="w")
self.start_time_hour = tk.Spinbox(time_frame_start, from_=0, to=23, width=3,
format="%02.0f")
self.start_time_hour.pack(side=tk.LEFT)
tk.Label(time_frame_start, text=":", font=("Segoe UI", 10),
        bg=theme["bg"], fg=theme["fg"]).pack(side=tk.LEFT)
self.start_time_min = tk.Spinbox(time_frame_start, from_=0, to=59, width=3,
format="%02.0f")
self.start_time_min.pack(side=tk.LEFT)

# End Time Picker
tk.Label(form, text="End Time (HH:MM):", font=("Segoe UI", 10),
        bg=theme["bg"], fg=theme["fg"]).pack(anchor="w")
time_frame_end = tk.Frame(form, bg=theme["bg"])
time_frame_end.pack(pady=5, anchor="w")
self.end_time_hour = tk.Spinbox(time_frame_end, from_=0, to=23, width=3,
format="%02.0f")
self.end_time_hour.pack(side=tk.LEFT)
tk.Label(time_frame_end, text=":", font=("Segoe UI", 10),
        bg=theme["bg"], fg=theme["fg"]).pack(side=tk.LEFT)

```

```
self.end_time_min = tk.Spinbox(time_frame_end, from_=0, to=59, width=3,  
format="%02.0f")
```

```
self.end_time_min.pack(side=tk.LEFT)
```

```
# Description
```

```
tk.Label(form, text="Description:", font=("Segoe UI", 10),  
bg=theme["bg"], fg=theme["fg"]).pack(anchor="w", pady=(10, 0))
```

```
self.desc_entry = tk.Text(form, height=4, wrap=tk.WORD)
```

```
self.desc_entry.pack(fill=tk.X, pady=5)
```

```
tk.Button(self.root, text="Create Task", command=self.save_task).pack(pady=15,  
ipadx=15, ipady=5)
```

```
def voice_input(self):
```

```
recognizer = sr.Recognizer()
```

```
with sr.Microphone() as source:
```

```
    messagebox.showinfo("Voice Input", "Listening... Please say your task name.")
```

```
    try:
```

```
        audio = recognizer.listen(source, timeout=5)
```

```
        text = recognizer.recognize_google(audio)
```

```
        self.task_name_entry.delete(0, tk.END)
```

```
        self.task_name_entry.insert(0, text)
```

```
    except:
```

```
        messagebox.showerror("Error", "Could not process voice input.")
```

```
def save_task(self):
```

```
    # Combine hour and minute spinbox values into HH:MM strings.
```

```
    start_time = f"{int(self.start_time_hour.get()):02d}:{int(self.start_time_min.get()):02d}"
```

```
    end_time = f"{int(self.end_time_hour.get()):02d}:{int(self.end_time_min.get()):02d}"
```

```
    task = {
```

```
        "name": self.task_name_entry.get().strip(),
```

```
        "category": self.category_var.get(),
```

```

        "priority": self.priority_var.get(),
        "date": self.date_entry.get(),
        "start": start_time,
        "end": end_time,
        "desc": self.desc_entry.get("1.0", tk.END).strip(),
        "completed": False,
        "reminded": False
    }

    if not task["name"] or not task["category"]:
        messagebox.showwarning("Missing Info", "Please fill out all required fields.")
        return

    self.save_task_to_db(task)
    self.tasks.append(task)
    self.speak("Task created successfully.")
    messagebox.showinfo("Task Created", "Your task has been added.")
    self.show_homepage()

def update_task_list(self):
    theme = self.themes[self.current_theme]
    for widget in self.task_frame.winfo_children():
        widget.destroy()
    if not self.tasks:
        tk.Label(self.task_frame, text="No tasks today.", bg=theme["bg"],
fg=theme["fg"]).pack()
        return

    for task in self.tasks:
        card = tk.Frame(self.task_frame, bg=theme["card"], bd=1, relief=tk.SOLID, padx=10,
pady=8)
        card.pack(fill=tk.X, pady=5)
        title = f"{task['name']} ({task['start']} - {task['end']}) [{task['priority']}]"
        if task['completed']:

```

```

        title = f"✓☐ {title}"

        tk.Label(card, text=title, font=("Segoe UI", 12, "bold"), bg=theme["card"],
fg=theme["fg"]).pack(anchor="w")

        tk.Label(card, text=f"Category: {task['category']} | Date: {task['date']}", font=("Segoe
UI", 9), bg=theme["card"], fg="gray").pack(anchor="w")

        tk.Label(card, text=task['desc'], font=("Segoe UI", 9), bg=theme["card"],
fg=theme["fg"],
                wraplength=400, justify="left").pack(anchor="w")


        btn_frame = tk.Frame(card, bg=theme["card"])
        btn_frame.pack(anchor="w", pady=5)

        if not task['completed']:
            ttk.Button(btn_frame, text="Complete", command=lambda t=task:
self.mark_as_complete(t)).pack(side=tk.LEFT, padx=5)

            ttk.Button(btn_frame, text="Delete", command=lambda t=task:
self.delete_task(t)).pack(side=tk.LEFT, padx=5)


        def mark_as_complete(self, task):
            task["completed"] = True
            self.update_task_in_db(task)
            self.speak(f"Task '{task['name']}' marked as completed.")
            self.update_task_list()


        def delete_task(self, task):
            self.delete_task_from_db(task["id"])
            self.tasks.remove(task)
            self.speak(f"Task '{task['name']}' has been deleted.")
            self.update_task_list()


        def check_reminders(self):
            now = datetime.now()

            for task in self.tasks:
                if not task["completed"] and not task.get("reminded", False):

```

```

try:
    task_dt_str = f"{task['date']} {task['end']}"
    task_dt = datetime.strptime(task_dt_str, "%Y-%m-%d %H:%M")
    if now > task_dt:
        self.speak(f"Time out for the task {task['name']}")
        task["reminded"] = True
        self.update_task_in_db(task)
except Exception as e:
    print(f"Error parsing time for '{task['name']}': {e}")
self.root.after(60000, self.check_reminders)

if __name__ == "__main__":
    root = tk.Tk()
    app = PrioritizeMeApp(root)
    root.mainloop()

```

4.3 Entity Relationship Diagram

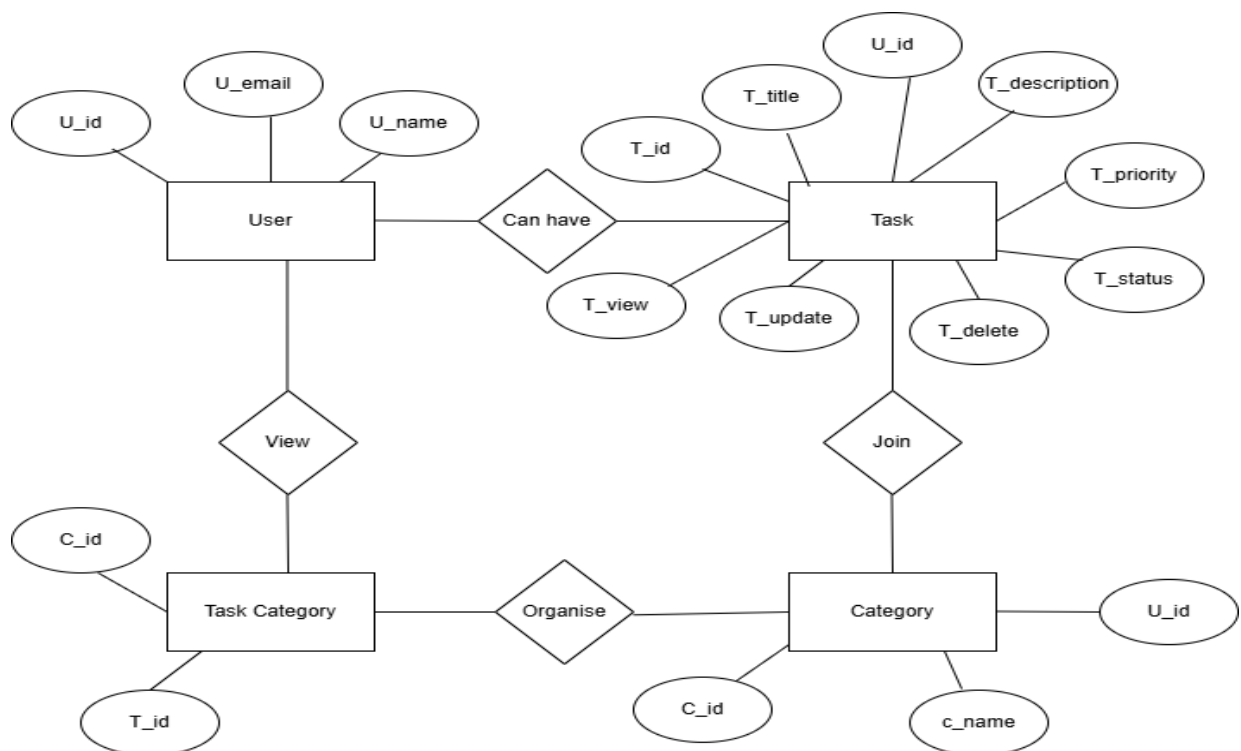


Figure 4.3.1

4.4 UML diagram

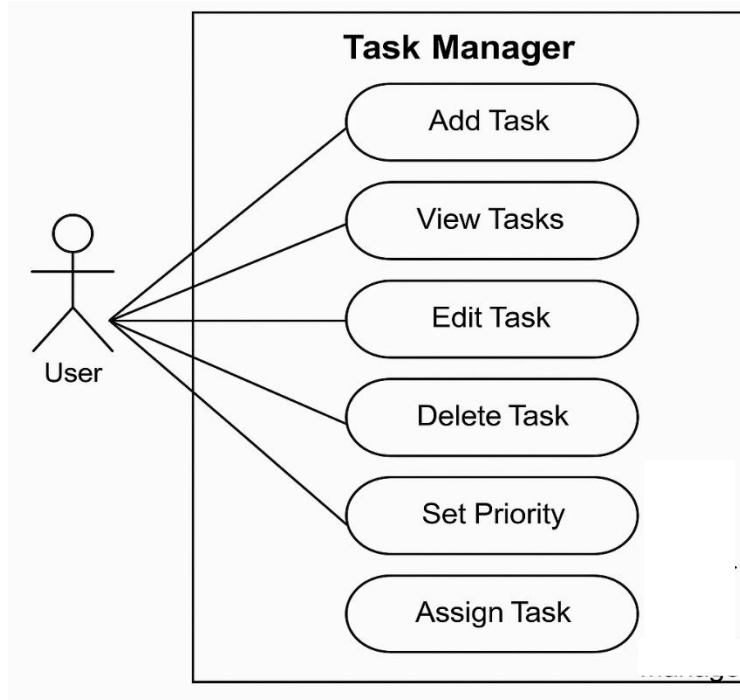


Figure 4.4.1

4.5 Module specification

The PrioritizeMe task manager is divided into modular components to promote clean architecture, maintainability, and scalability. Each module handles a specific responsibility, ensuring a separation of concerns between the interface, logic, and data layers.

1. User Interface (UI) Module

Technology: Tkinter (Python Library)

Purpose: Handles all user interactions, screen navigation, and display elements.

Key Components:

- Home Screen: Dashboard showing today's tasks, pending items, and quick actions.
- Task List Screen: Displays categorized task lists (e.g., "Today", "Upcoming", "Completed").
- Add/Edit Task Form: Input fields for task name, description, due date, priority, and category.
- Settings Screen: App preferences and theme toggles.

Functions:

- show_task_list()
- open_task_form()
- display_alert(message)
- switch_screen(screen_name)

2. Task Management Module

Technology: Python

Purpose: Core logic for creating, updating, retrieving, and deleting tasks.

Functions:

- add_task(task_data)
- edit_task(task_id, updated_data)
- delete_task(task_id)
- mark_complete(task_id)
- filter_tasks(by_date=None, by_priority=None)

Features:

- Task priority levels (High, Medium, Low)
- Due date tracking and overdue highlighting
- Optional tags or categories

3. Database Module

Technology: MySQL (with mysql-connector-python or SQLAlchemy)

Purpose: Handles storage, retrieval, and persistence of data.

Tables:

- users (optional, if login is implemented)
- tasks
- task_id (INT, Primary Key)
- title (VARCHAR)
- description (TEXT)
- due_date (DATE)
- priority (ENUM)
- status (ENUM: pending/completed)
- created_at (DATETIME)
- updated_at (DATETIME)

Functions:

- connect_db()
- insert_task(task_data)
- update_task(task_id, new_data)
- get_tasks(filter=None)
- delete_task(task_id)

4. Authentication Module (Optional)

Technology: Python + MySQL

Purpose: Manage user login, session, and data isolation.

Features:

- User registration/login
- Password hashing (using bcrypt or similar)
- Session tracking (for multi-user systems)

5. Notification & Alerts Module

Technology: Python (Optional: plyer or custom alerts)

Purpose: Sends reminders or alerts for due/overdue tasks.

Functions:

- check_due_tasks()
- trigger_alert(task_id)

4.6 Data flow diagram

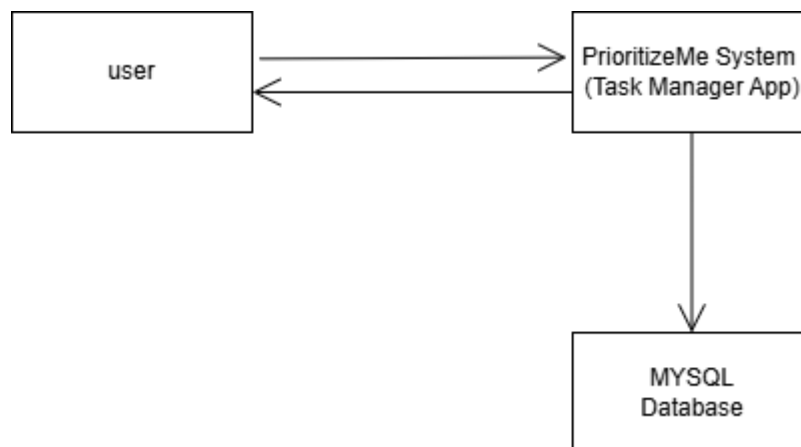


Figure 4.6.1

4.7 User Interface Design

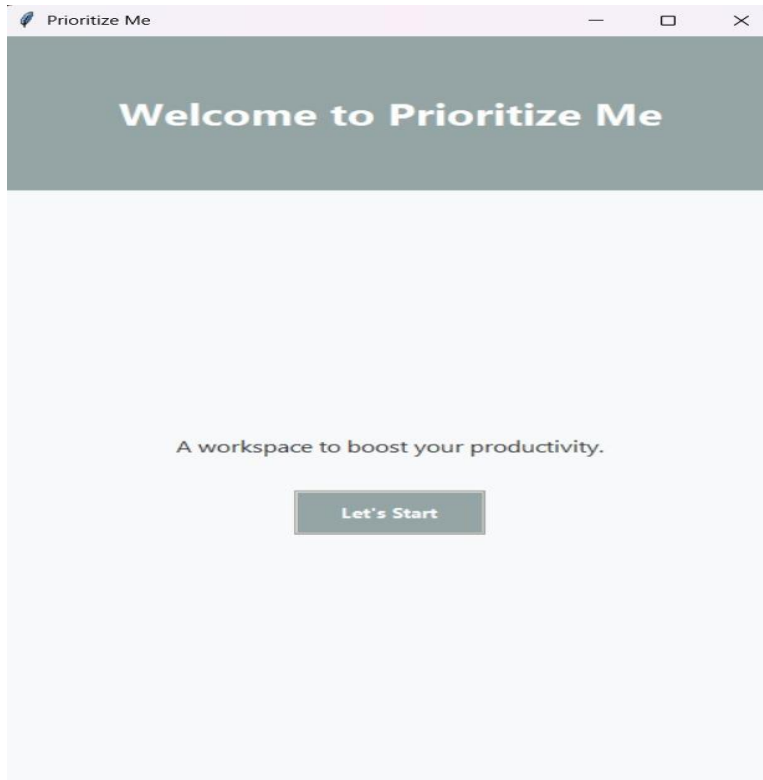


Figure 4.7.1

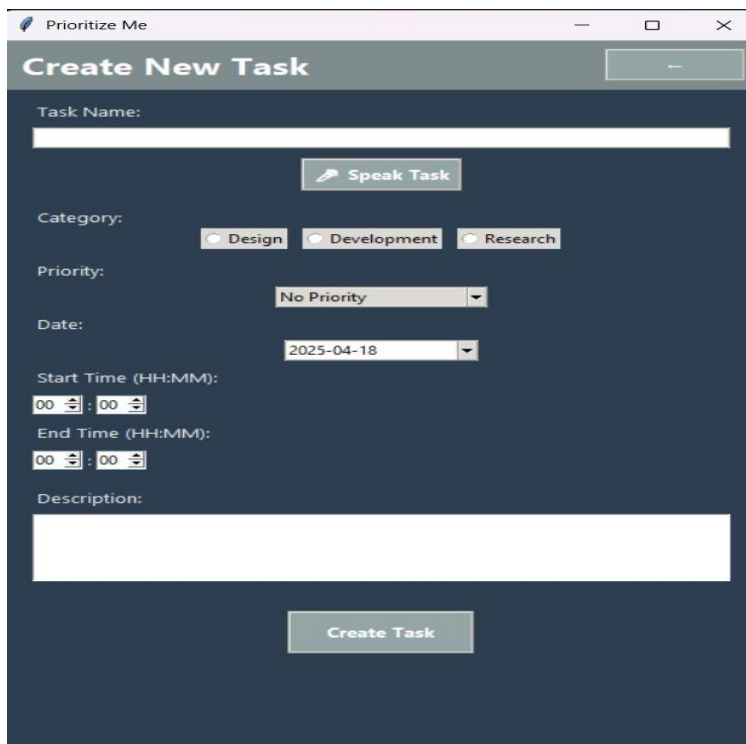
A screenshot of a web browser window titled "Prioritize Me" showing a "Create New Task" form. The form is set against a dark blue background. At the top, there is a header "Create New Task" with a close button. Below the header, the form contains several fields: "Task Name:" with a text input field; a "Speak Task" button with a microphone icon; "Category:" with three radio buttons labeled "Design", "Development", and "Research"; "Priority:" with a dropdown menu showing "No Priority"; "Date:" with a date picker showing "2025-04-18"; "Start Time (HH:MM):" with two time pickers both set to "00 : 00"; "End Time (HH:MM):" with two time pickers both set to "00 : 00"; and "Description:" with a large text area. At the bottom of the form is a "Create Task" button.

Figure 4.7.2

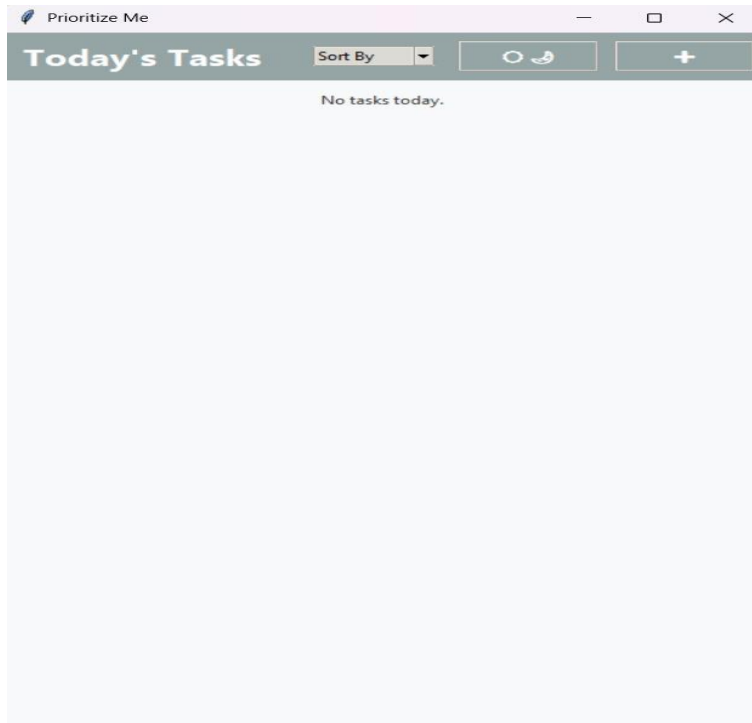


Figure 4.7.3

A screenshot of a web application window titled "Prioritize Me". The main heading is "Create New Task". The form contains the following fields and controls:

- Task Name:** A text input field containing "Complete the project".
- Speak Task:** A button with a microphone icon.
- Category:** Three radio buttons labeled "Design", "Development" (selected), and "Research".
- Priority:** A dropdown menu showing "Medium".
- Date:** A dropdown menu showing "2025-04-18".
- Start Time (HH:MM):** A time picker showing "11 : 00".
- End Time (HH:MM):** A time picker showing "12 : 00".
- Description:** A text area containing "Complete the project as soon as possible".
- Create Task:** A button at the bottom of the form.

Figure 4.7.4

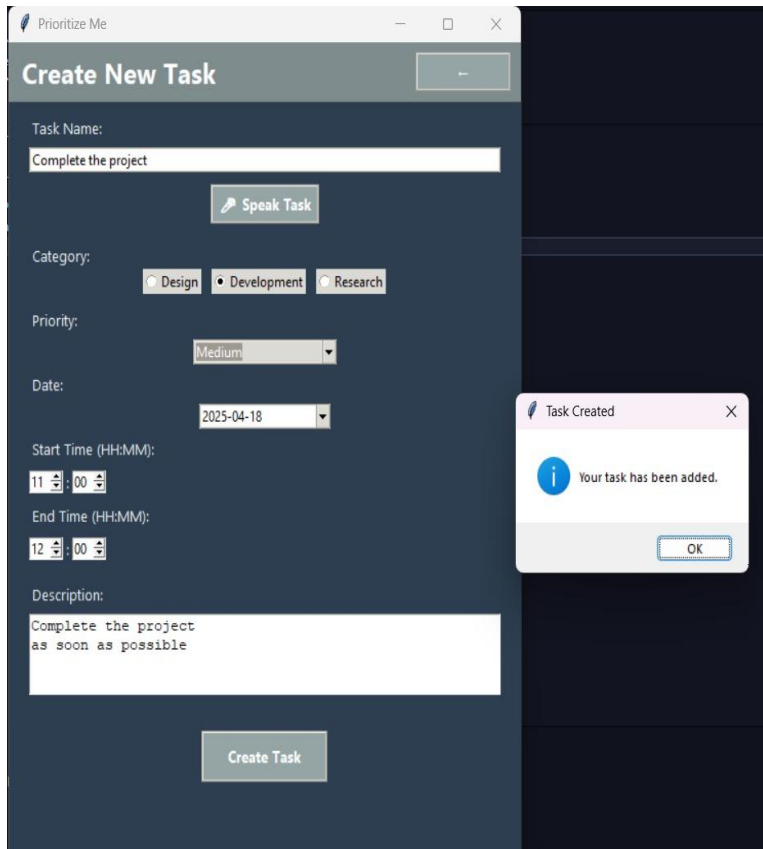


Figure 4.7.5

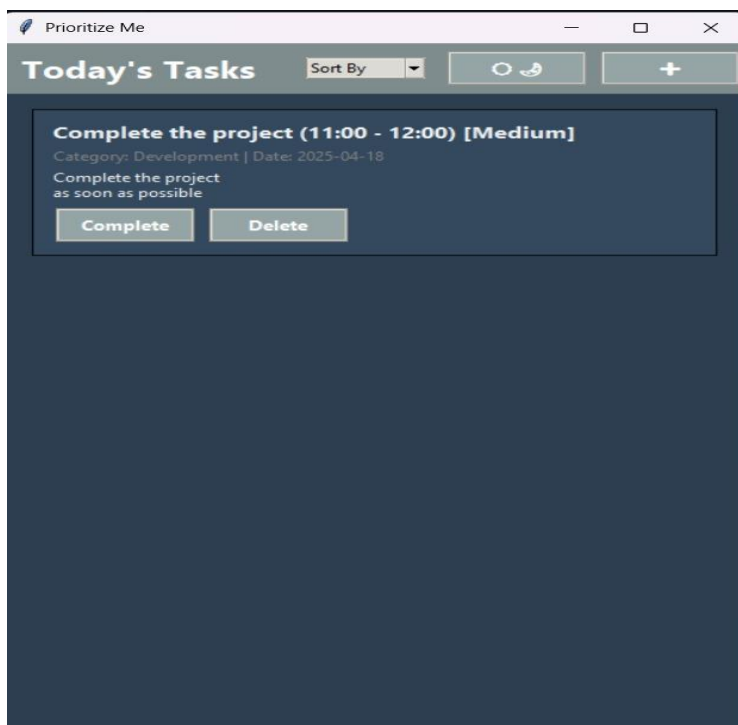


Figure 4.7.6

4.8 Use case diagram

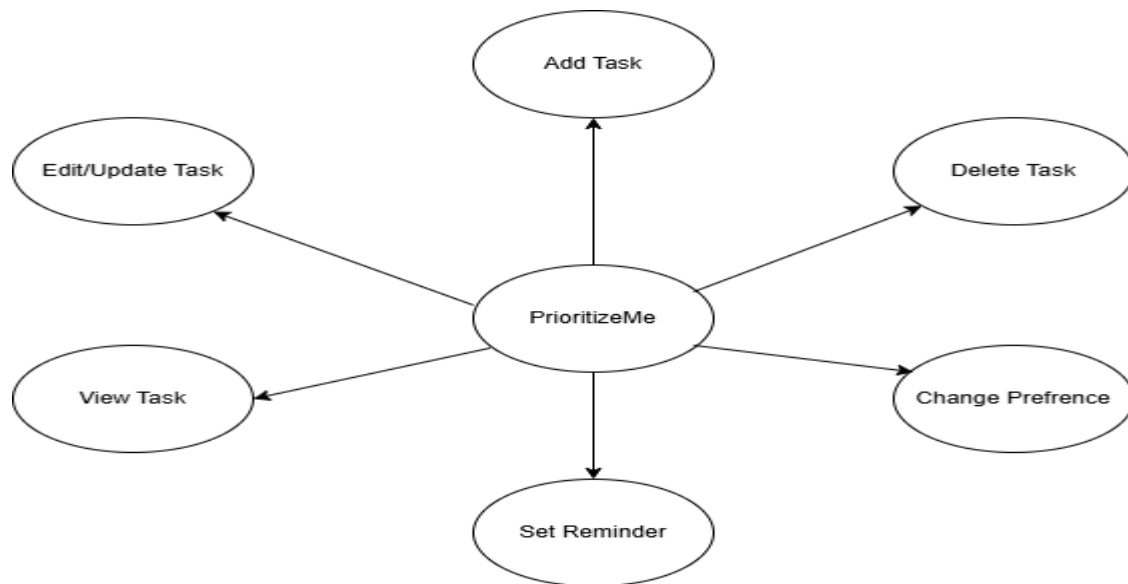


Figure 4.8.1

4.9 Output/ Screenshot

The screenshot shows a web browser window titled 'Prioritize Me'. The main heading is 'Create New Task'. Below the heading is a text input field for 'Task Name:'. To the right of this field is a 'Speak Task' button with a microphone icon. Below the 'Task Name' field are three radio buttons for 'Category:': 'Design', 'Development', and 'Research'. Below the category buttons is a 'Priority:' dropdown menu currently set to 'No Priority'. Below the priority dropdown is a 'Date:' dropdown menu currently set to '2025-04-18'. Below the date dropdown are two time input fields: 'Start Time (HH:MM):' and 'End Time (HH:MM):', both currently set to '00 : 00'. Below the time fields is a 'Description:' text area. At the bottom of the form is a 'Create Task' button.

Figure 4.9.1

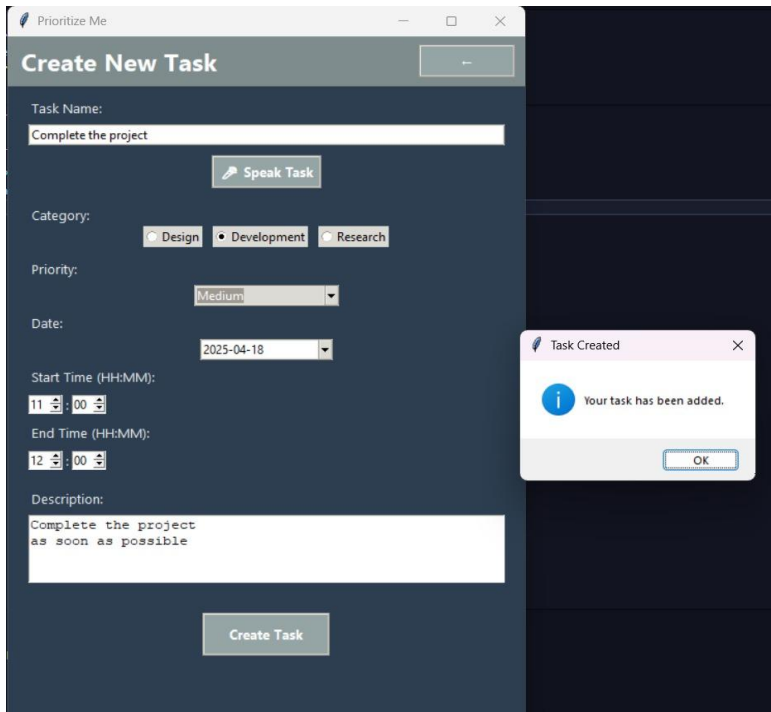


Figure 4.9.2

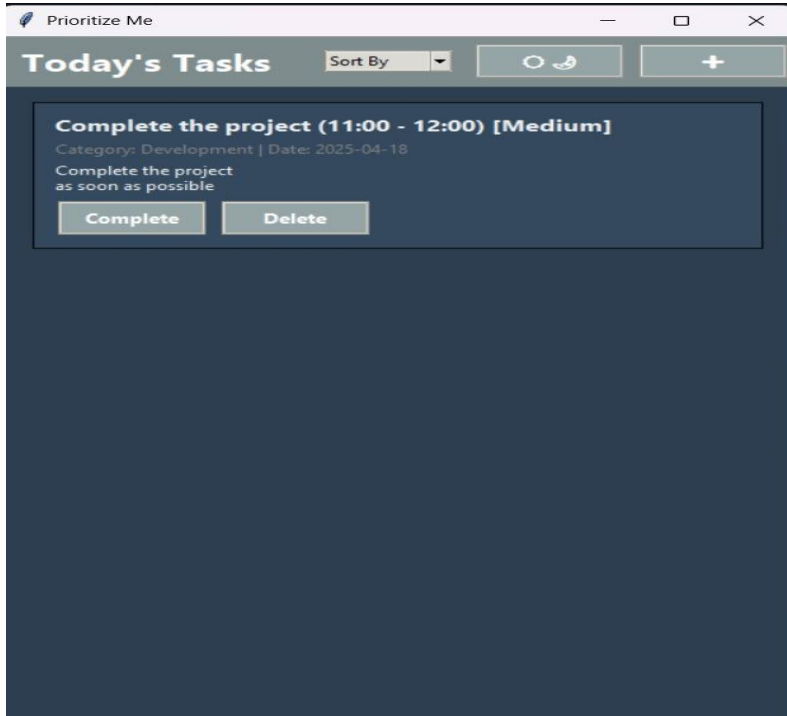


Figure 4.9.3

Chapter 5: Conclusion

5.1 Conclusion

The To-Do app, built using Python, Tkinter, TkinterMD, and MySQL, offers a powerful and user-friendly solution for task management. By leveraging Tkinter for a responsive and cross-platform interface and MySQL for reliable data storage, the app allows users to efficiently organize and prioritize their daily tasks. The integration of TkinterMD enhances the app's UI, providing a modern, Material Design-inspired experience.

While the app addresses core task management needs, such as task creation, editing, and deletion, there are areas for future growth, such as cloud syncing, mobile support, and advanced features like AI-based prioritization and collaboration. Additionally, implementing user authentication, push notifications, and offline support would make the app more robust and versatile.

Overall, this To-Do app serves as a strong foundation for task management with room for further enhancements to meet the evolving needs of users. It is a reliable tool for individuals looking to stay organized and focused, with potential for growth into a fully integrated, cross-platform productivity tool.

5.2 Limitations

Limited Cross-Platform Support:

While Tkinter allows the app to run on Windows, macOS, and Linux, deploying it to mobile platforms (iOS/Android) can be more challenging. While Tkinter does support mobile deployment, it requires additional setup and adjustments, and the performance may not be as optimized as native mobile apps built with platforms like React Native or Swift.

Lack of Cloud Syncing:

The app is designed to work locally with a MySQL database, meaning the tasks are stored on the local machine or server. There is no built-in feature for syncing tasks across multiple devices or with cloud storage, so if users switch devices or need remote access, they may face limitations.

Dependency on MySQL:

Using MySQL as the database can be a limitation in some cases. While it's robust, setting up and managing a MySQL server can be more complex compared to simpler, more lightweight

databases like SQLite for smaller apps. Additionally, for cloud-based syncing or scaling the app, you'd need to handle MySQL hosting and maintenance, which may add extra overhead.

Performance on Mobile:

While Tkinter supports mobile platforms, performance can sometimes be suboptimal compared to native mobile applications. Mobile devices have limited resources compared to desktops, and Tkinter-based apps may face challenges related to speed and responsiveness on mobile devices, especially for larger, more complex applications.

Limited User Interface Components:

TkinterMD provides many Material Design components, but it may not offer as many ready-to-use features as other more mature frontend frameworks like Flutter or React Native. This could limit customization options and require more effort to design complex UI elements.

Scalability Issues:

As the app grows, the use of MySQL might pose scalability issues, especially if it needs to handle a large amount of data or many concurrent users. For a more robust, scalable system, a NoSQL database (like MongoDB) or cloud-based solutions might be better suited for handling high traffic or large datasets.

No Offline Support:

The app requires an active internet connection if hosted on a remote server (depending on how the database is managed). If a user is offline, they may not be able to access or update tasks if the database isn't locally available.

User Authentication:

The app does not include a user authentication system (e.g., login/logout, password management). This means multiple users cannot have personalized task lists without additional features being added.

Limited Integration:

The app is relatively self-contained and does not offer integration with other services (e.g., Google Calendar, Trello, or other task management tools) or APIs for synchronization, notifications, or task sharing. Adding integrations could make the app more powerful but would increase complexity.

5.3 Future Enhancements

Cloud Syncing and Multi-Device Support:

Implement cloud-based syncing so users can access their tasks across multiple devices (e.g.,

phone, tablet, desktop).

Use cloud storage solutions like Firebase, Google Cloud, or AWS to store tasks and settings, ensuring they are accessible from anywhere.

Mobile App Deployment:

Optimize the app for Android and iOS using Tkinter or consider switching to other frameworks like React Native or Flutter for smoother, more efficient mobile performance.

Enhance performance on mobile devices to ensure smooth and responsive user experiences.

Allow users to create accounts, save their tasks, and synchronize across different devices.

Push Notifications:

Implement push notifications to remind users of upcoming tasks, deadlines, or daily/weekly summaries of pending tasks.

Firebase Cloud Messaging (FCM) or OneSignal can be used for cross-platform push notification integration.

Collaboration and Sharing:

Enable task sharing so users can share tasks or lists with others, making it useful for team projects or family tasks.

Support for real-time collaboration on tasks with commenting and progress tracking.

Advanced Reminders & Alerts:

Add location-based reminders, where tasks can trigger reminders based on the user's location (e.g., "Pick up groceries when you're near the store").

Implement recurring tasks with customized intervals for tasks that need to be repeated.

Integration with Third-Party Services:

Enable integration with popular productivity tools like Google Calendar, Trello, or Slack to import/export tasks and stay connected with other productivity systems.

API integration for integrating with other systems and services, enhancing the app's versatility.

Performance Optimization:

Optimize database queries for faster load times and better performance, especially as the user base and data grow.

Consider switching to NoSQL databases (e.g., MongoDB) if the app scales up with large data sets or needs high availability.

Bibliography

1. <https://www.blackbox.ai/chat/SKtkm3s>
2. <https://app.diagrams.net/>
3. <https://youtu.be/dNNJZapp3M4?si=YhoY2KVG-iwQtfHH>
4. <https://youtu.be/sNi7JSobgKo?si=yExEdBHAXTneuxOg>
5. <https://www.w3schools.com/sql/>
6. <https://chatgpt.com/>
7. <https://www.tpointtech.com/software-engineering-incremental-model>
8. https://www.researchgate.net/figure/NCREMENTAL-MODEL2_fig2_283507061
9. <https://www.youtube.com/watch?v=Fe3lEsd-UJw&t=1493s>
10. <https://www.youtube.com/watch?v=kRWtSkIYPFI&list=PLy5hjmUzdc0nMkzhphsqgPCX62NFhkell>