LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

Python Project

On

Task Manager

Submitted by: K Karthik Kumar
Roll No.: RK23UPA30

Reg.no.:12324726
Course Code: CSM216
Section: K23UP

Submitted to:

Mr Aman Kumar

ACKNOWLEDMENT

I would like to express my sincere gratitude to my mentor, Aman Kumar, for their invaluable support and guidance throughout this project. Their expertise in data handling and python based Graphic user Interface was instrumental in helping me understand the principles needed to build an effective task manager GUI

I am also thankful to my friends for their constructive feedback, which helped refine the functionality and design of the program, ensuring a user-friendly experience.

Lastly, I am grateful to **Lovely Professional University** and **Upgrad** for providing a conducive environment and access to learning resources, which made this project possible.

INDEX

s.no.	content	Pg.no.
1.	introduction	04
2.	Objectives and Scope	05
3.	Application Tools	06
4.	Project Design	09
6.	Project Implementation	10
7.	Testing and Validation	17
8.	Conclusion	20
9.	Reference	21

INTRODUCTION

A **Graphical User Interface (GUI)** is a type of interface that employs graphical elements, such as icons and images, to enable users to interact seamlessly with an application. Unlike command-line interfaces, which are primarily built for functionality, GUIs focus on delivering an intuitive and user-friendly experience. This design approach ensures that users can navigate applications with ease, regardless of their technical expertise. A task manager, in particular, benefits significantly from a well-designed GUI as it helps users organize and manage their daily tasks effectively by providing a structured overview of events and responsibilities.

A **task manager** is an essential utility for both personal productivity and system performance management. Beyond personal task tracking, it plays a pivotal role in operating system maintenance. It can diagnose and resolve issues like unresponsive programs, excessive resource usage, or problematic processes that slow down the system. A thoughtfully designed Task Manager GUI incorporates visual elements such as charts, lists, and graphs, which provide users with an instant overview of system performance and allow them to identify and address bottlenecks quickly. For individuals, a task manager is indispensable for organizing and prioritizing daily activities. By streamlining tasks and allocating time effectively, users can maintain productivity throughout the day. Moreover, balanced task distribution prevents overwork, allowing users to meet deadlines while ensuring adequate time for rest and relaxation. By integrating a comprehensive view of all running tasks and resources, a task manager not only optimizes workflow but also enhances overall system efficiency, making it an invaluable tool for troubleshooting and performance monitoring.

Benefits of a GUI-Based Task Manager

- **Usability:** Simplifies user interaction by eliminating complex command-line instructions.
- **Scalability:** Adapts to handle increasing tasks or features seamlessly.
- **Ease of Navigation:** Enables quick and intuitive access to task management features.
- **Visual Representation:** Provides clear and digitized visualization of tasks and system processes.
- **Multitasking:** Facilitates the management of multiple tasks simultaneously.
- **Reduced Learning Curve:** Makes the application accessible even for non-technical users.
- **Error Handling and Specification:** Ensures robust mechanisms to handle errors gracefully, improving user experience.

OBJECTIVES & SCOPE OF PROJECT

Objectives:

The main objectives the program aims to achieve are as follows:

- **Develop a GUI:** Create a comprehensive user interface using Tkinter, prioritizing user experience and ease of use.
- **Build a Task Manager:** Develop an application that enables users to manage and store tasks efficiently using the GUI.
- **Implement Well-Defined Functions:** Provide optimized functions to support key features such as adding, removing, and editing tasks.
- **Enable Efficient Command Display:** Facilitate intuitive methods to display user commands and results clearly.
- **Integrate Web Crawling Features:** Allow the program to retrieve external data relevant to user tasks using basic crawling techniques.
- **Focus on Error Handling:** Implement robust mechanisms to handle errors gracefully, ensuring seamless functionality.
- **Ensure Easy Maintenance:** Design the program for simplified modifications and future updates.

Scope:

1. **User Interface Design**
   Create a user-friendly interface using Tkinter, emphasizing intuitive navigation and simplicity.

2. **Task Management System**
   Develop features for adding, editing, deleting, and storing tasks to streamline task tracking.

3. **Optimized Functional Implementation**
   Build efficient and modular functions to ensure smooth and reliable application performance.

4. **Command Display and Execution**
   Provide interactive methods to display and execute user commands through the GUI.

5. **Web Crawling Integration**
   Enhance the application by adding basic web crawling features to gather useful external data for users.

6. **Error Handling**
   Implement robust systems to manage unexpected inputs and issues, ensuring application stability.

APPLICATION TOOLS

Tools and Technologies Used:

- **Python Programming Language**
  - The core language used for building the application due to its simplicity, versatility, and extensive libraries.
- **Tkinter Library**
  - Python library used to develop the graphical user interfaces .
- **Custom Tkinter Library**
  - An optimized version of Tkinter library that offer more versatile and visually appealing widgets.
- **Object-Oriented Programming (OOP)**
  - Use if class and methods to define objects enabling for easy maintenance and modularity
- **Data Persistence Tools (e.g., JSON or CSV)**
  - JSON or CSV files are used for saving and loading task data, allowing persistent storage across sessions.
- **Error Handling and Debugging Tools**
  - Using try and catch blocks of code to promptly handle errors that might arise.
- **IDE (Integrated Development Environment)**
  - Development is done using Python-friendly IDEs like **PyCharm**, **VS Code**, or **IDLE**, offering code suggestions and debugging support.
- **Version control :**

  Version control of the project is done use git repositories

These tools and technologies are chosen for their efficiency, ease of use, and ability to support the requirements of a comprehensive task manager application.

The provided Python scripts offer two distinct GUI-based applications built using Tkinter: a user management system and a task management tool. Both serve as introductory examples for understanding GUI programming in Python and demonstrate key concepts like event handling, user input, and interface design. However, the applications currently use in-memory data storage, limiting their utility for persistent use, and could benefit from further enhancements.

User Management System

The user management application implements basic features such as login, signup, password update, and password retrieval. The user data is stored in a temporary dictionary (`user_db`), which is sufficient for demonstration and small-scale testing but lacks data persistence. This limitation means that once the program is terminated, all stored user information is lost, highlighting the need for potential integration with a database for more practical, real-world use.

Key functionalities include:

- on_login(): This function handles user authentication by verifying credentials against the stored data, providing feedback on the success or failure of the login attempt.
- on_signup(): Facilitates user registration by adding new accounts, while ensuring that usernames are not duplicated.
- on_set_password(): Allows existing users to update their password, offering a basic mechanism for credential management.
- on_forgot_password(): Retrieves and displays the current password for users who cannot remember their credentials.
- on_create_account(): Provides an alternative method for user registration, similar in functionality to the signup feature.

The user interface includes input fields for username and password, along with buttons for each action (e.g., login, signup, password update). The interface is visually enhanced with a gradient background created using Tkinter's `Canvas` widget, making the application more engaging. The layout prioritizes simplicity and ease of use, allowing users to navigate the various features

without difficulty. Future improvements could focus on adding persistent storage, such as a database, and implementing secure password handling techniques like hashing.

Task Management Tool

The task management tool provides a simple yet effective interface for organizing and manipulating tasks. It consists of two main components: the `TaskManager` class, which handles the core task logic, and the `UpcomingTasksPage` class, which manages the graphical user interface. The application allows users to view, add, delete, postpone, and reorder tasks, offering a flexible approach to task management.

The main features include:

- view_tasks(): Presents a dialog listing all current tasks, giving users a comprehensive view of their upcoming responsibilities.

- add_task(): Enables users to add new tasks, facilitating dynamic updates to their task list.

- delete_task(): Allows users to remove tasks, keeping the task list relevant and clutter-free.

- update_task_listbox(): Refreshes the displayed list of tasks in the GUI whenever changes are made, ensuring an accurate and up-to-date view.

- free_time(): Introduces a simple placeholder feature for managing free time, offering users a chance to consider breaks in their schedule.

The user interface leverages Tkinter widgets such as `Listbox` for task display and `Button` for interaction, all organized within a central frame. This design promotes usability, providing a clear, intuitive layout that simplifies task management. Each task is accompanied by interactive buttons that enable users to postpone, prepend, or remove tasks directly from the interface, making task handling straightforward and efficient.

4. Project Design

The Task Manager application is designed with a modular architecture to promote clarity , maintainability, and scalability. Below is a detailed breakdown of its design and workflow:

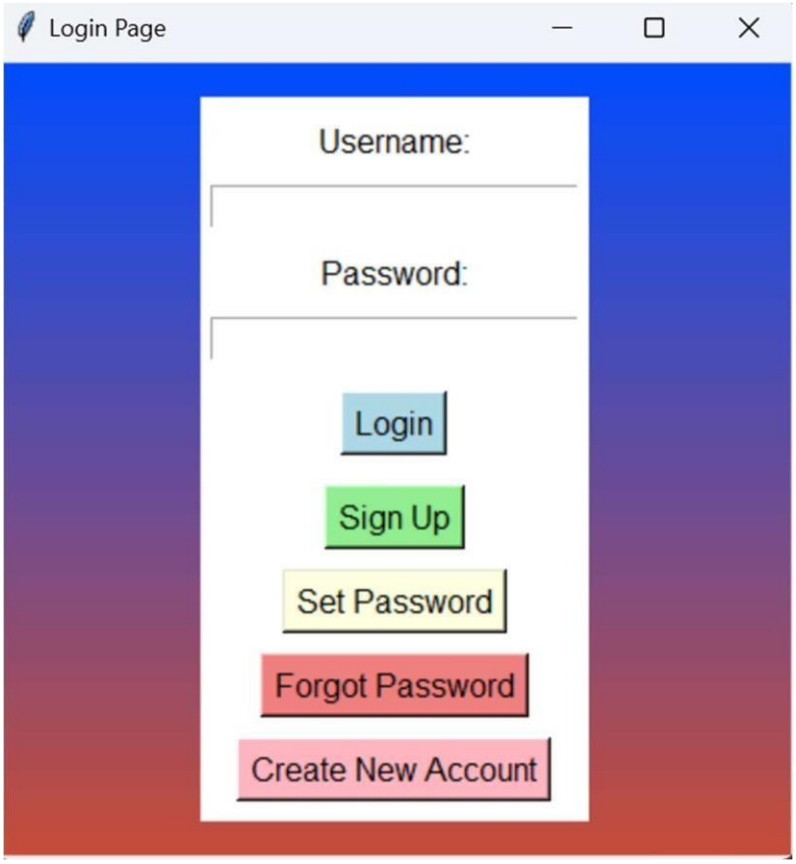
Key Components:

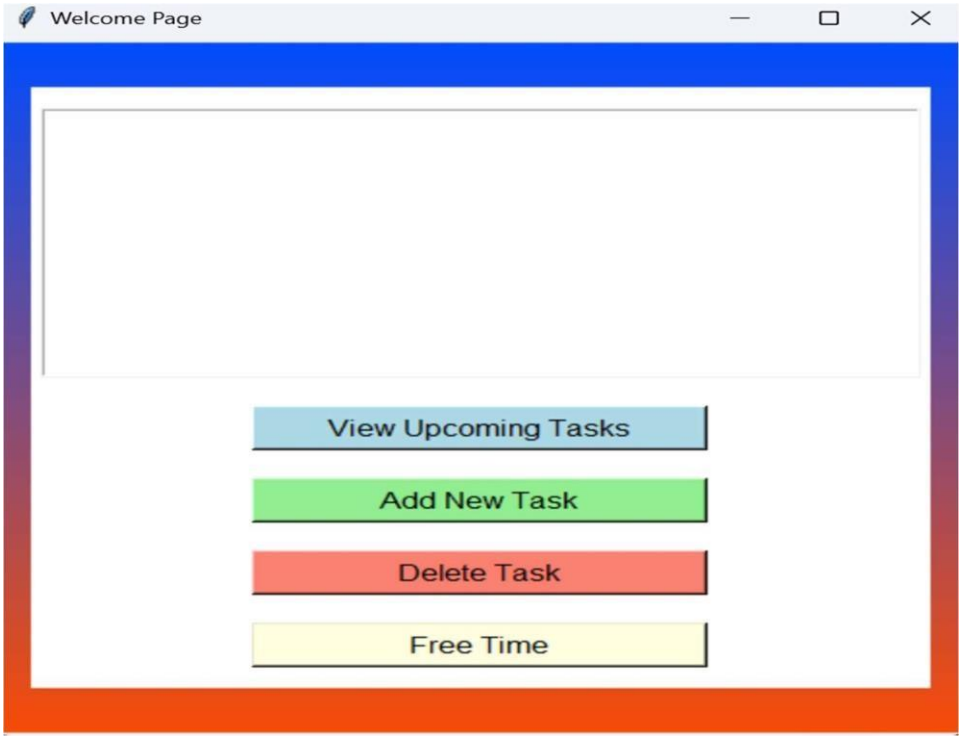
1. **Graphical User Interface (GUI):**
   - Acts as the front-end layer, enabling users to interact with the system through intuitive widgets like buttons, text fields, and list boxes.
   - Provides real-time feedback to users by updating the displayed task list dynamically.
2. **Core Logic and Functionalities:**
   - The TaskManager class encapsulates core operations such as task addition, deletion, and modification.
   - Implements methods for validating inputs, such as checking deadlines and avoiding duplicate task entries.
3. **Data Storage Layer:**
   - Handles saving and retrieving task data. The use of JSON ensures quick read/write operations, while SQLite provides a more robust solution for scalability.
4. **Error Handling Module:**
   - Aims to catch and manage all possible runtime exceptions, such as invalid user inputs or file access errors.
5. **Task Analytics and Reporting:**
   - Includes potential future features like reporting overdue tasks or visualizing upcoming deadlines.

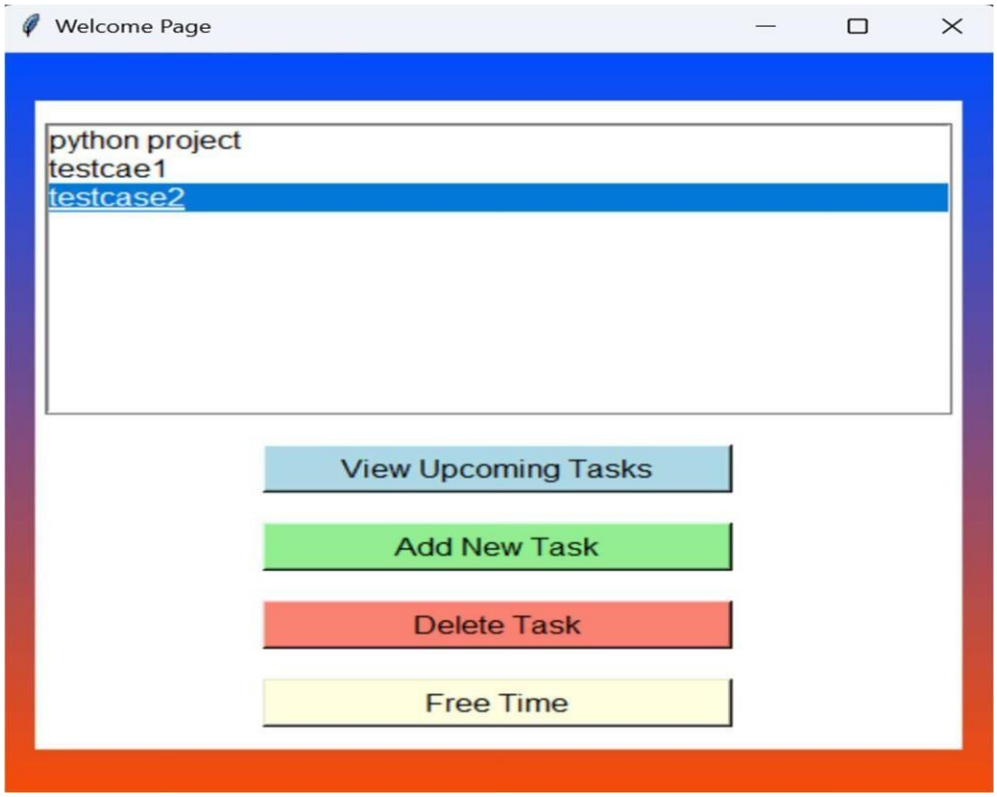
Workflow Overview:

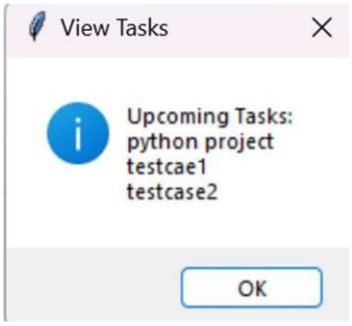
1. **Startup Phase:**
   - The application initializes by setting up the Tkinter interface and loading saved task data from storage.
2. **Task Operations:**
   - Users interact with the GUI to perform actions like adding, editing, or deleting tasks. These commands are processed by the TaskManager class.
3. **Data Validation:**
   - Inputs are validated to ensure accuracy and prevent logical errors.
4. **Display Update:**
   - The GUI reflects changes instantly, keeping users informed about the current state of their task list.
5. **Save and Exit:**
   - Tasks are saved in the storage system when the application is closed, ensuring persistence.

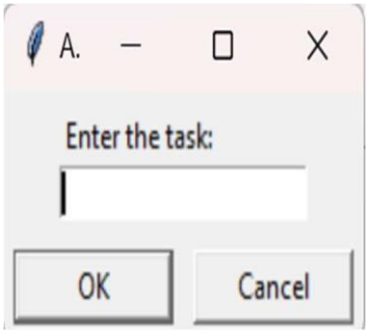
Sample images of the GUI






Welcome Page
python project
testcae1
testcase2
View Upcoming Tasks
Add New Task
Delete Task
Free Time


View Tasks
Upcoming Tasks:
python project
testcae1
testcase2
OK


A.
Enter the task:
OK
Cancel

SCREENSHOTS OF MAIN FUNCTIONAL CODE

```python
import tkinter as tk
from tkinter import messagebox, simpledialog

class WelcomePage:
    def __init__(self, master):
        self.master = master
        self.master.title("Welcome Page")
        self.master.geometry("500x500")


        self.canvas = tk.Canvas(master, width=500, height=500)
        self.canvas.pack()


        for i in range(500):
            color = f'#{i//2:02x}4C{255-i//2:02x}'
            self.canvas.create_line(0, i, 500, i, fill=color)


        self.frame = tk.Frame(master, bg='white', bd=5)
        self.frame.place(relx=0.5, rely=0.5, anchor='center')


        self.task_listbox = tk.Listbox(self.frame, width=50, height=10, font=('Arial', 12))
        self.task_listbox.pack(pady=10)
```

```python
    self.button_view_tasks = tk.Button(self.frame, text="View Upcoming Tasks", command=self.view_tasks,
    self.button_view_tasks.pack(pady=10)


    self.button_add_task = tk.Button(self.frame, text="Add New Task", command=self.add_task, bg='lightg
    self.button_add_task.pack(pady=10)


    self.button_delete_task = tk.Button(self.frame, text="Delete Task", command=self.delete_task, bg='s
    self.button_delete_task.pack(pady=10)


    self.button_free_time = tk.Button(self.frame, text="Free Time", command=self.free_time, bg='lightye
    self.button_free_time.pack(pady=10)


    self.tasks = []

ef view_tasks(self):
    if not self.tasks:
        messagebox.showinfo("View Tasks", "No upcoming tasks.")
    else:
        task_list = "\n".join(self.tasks)
```

```
    def view_tasks(self):
        if not self.tasks:
            messagebox.showinfo("View Tasks", "No upcoming tasks.")
        else:
            task_list = "\n".join(self.tasks)
            messagebox.showinfo("View Tasks", f"Upcoming Tasks:\n{task_list}")

    def add_task(self):
        task = simpledialog.askstring("Add Task", "Enter the task:")
        if task:
            self.tasks.append(task)
            self.update_task_listbox()
            messagebox.showinfo("Add Task", f"Task '{task}' added.")

    def delete_task(self):
        selected_task_index = self.task_listbox.curselection()
        if selected_task_index:
            task_to_delete = self.tasks[selected_task_index[0]]
            del self.tasks[selected_task_index[0]]
            self.update_task_listbox()
            messagebox.showinfo("Delete Task", f"Task '{task_to_delete}' deleted.")
        else:
            messagebox.showwarning("Delete Task", "Select a task to delete.")
```

```
    def update_task_listbox(self):
        self.task_listbox.delete(0, tk.END)
        for task in self.tasks:
            self.task_listbox.insert(tk.END, task)

    def free_time(self):
        messagebox.showinfo("Free Time", "You can manage your free time here.")

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

```python
import tkinter as tk
from tkinter import messagebox, simpledialog

class TaskManager:
    def __init__(self):
        self.tasks = []

    def add_task(self, task):
        self.tasks.append(task)

    def remove_task(self, index):
        if 0 <= index < len(self.tasks):
            del self.tasks[index]

    def postpone_task(self, index):
        if 0 <= index < len(self.tasks):
            self.tasks.append(self.tasks[index])
            self.remove_task(index)

    def prepend_task(self, index):
        if 0 <= index < len(self.tasks):
            self.tasks.insert(0, self.tasks[index])
            self.remove_task(index)
```

```python
class UpcomingTasksPage:
    def __init__(self, master, task_manager):
        self.master = master
        self.master.title("Upcoming Tasks")
        self.master.geometry("600x400")

        self.task_manager = task_manager

        # Frame for the task list
        self.task_frame = tk.Frame(master)
        self.task_frame.pack(pady=20)

        self.update_task_listbox()

    def update_task_listbox(self):
        # Clear the existing task frame
        for widget in self.task_frame.winfo_children():
            widget.destroy()  # Destroy existing widgets in the frame

        for index, task in enumerate(self.task_manager.tasks):
            task_row = tk.Frame(self.task_frame)
            task_row.pack(fill=tk.X, padx=5, pady=5)
```

```
    def postpone_task(self, index):
        self.task_manager.postpone_task(index)
        self.update_task_listbox()

    def prepend_task(self, index):
        self.task_manager.prepend_task(index)
        self.update_task_listbox()

    def remove_task(self, index):
        self.task_manager.remove_task(index)
        self.update_task_listbox()

if __name__ == "__main__":
    root = tk.Tk()
    task_manager = TaskManager()

    # Sample tasks
    task_manager.add_task("Task 1: Complete the report")
    task_manager.add_task("Task 2: Prepare for the meeting")
    task_manager.add_task("Task 3: Buy groceries")
```

```
if __name__ == "__main__":
    root = tk.Tk()
    task_manager = TaskManager()

    # Sample tasks
    task_manager.add_task("Task 1: Complete the report")
    task_manager.add_task("Task 2: Prepare for the meeting")
    task_manager.add_task("Task 3: Buy groceries")

    app = UpcomingTasksPage(root, task_manager)
    root.mainloop()
```

```
def on_signup():
    username = entry_username.get()
    password = entry_password.get()


    if username in user_db:
        messagebox.showerror("Signup Failed", "Username already exists.")
    elif username == "" or password == "":
        messagebox.showerror("Signup Failed", "Username and password cannot be empty.")
    else:
        user_db[username] = password
        messagebox.showinfo("Signup Successful", f"Account created for {username}.")

def on_set_password():
    username = entry_username.get()
    if username not in user_db:
        messagebox.showerror("Set Password Failed", "Username does not exist.")
        return
```

```
new_password = entry_password.get()
if new_password == "":
    messagebox.showerror("Set Password Failed", "New password cannot be empty.")
    return

user_db[username] = new_password
messagebox.showinfo("Set Password Successful", "Password updated successfully.")

new_password = entry_password.get()
if new_password == "":
    messagebox.showerror("Set Password Failed", "New password cannot be empty.")
    return

user_db[username] = new_password
messagebox.showinfo("Set Password Successful", "Password updated successfully.")

f on_forgot_password():
username = entry_username.get()
if username not in user_db:
    messagebox.showerror("Forgot Password Failed", "Username does not exist.")
    return
elif username in user_db:
    messagebox.showinfo("dont worry", f"your old password is {user_db[username]}.")
```

```python
def on_create_account():
    username = entry_username.get()
    password = entry_password.get()

    if username in user_db:
        messagebox.showinfo("Create Account", "You are already a user.")
    elif username == "":
        messagebox.showerror("Create Account Failed", "Username should not be empty.")
    else:
        user_db[username] = password
        messagebox.showinfo("Create Account Successful", f"Account created for {username}.")
```

Testing and validation

1. unit testing : to test individual functionalities

Test Description	Input	Expected Output	Actual Output	Status
Validate login credentials	Enter valid/invalid credentials	Successful login or error message	Correct login/logout behavior	Passed
Test signup functionality	Enter valid/duplicate data	Account created or error message	Signup working as expected	Passed
Test changing passwords	Enter new passwords	Messagebox of updated password	Password updated in dictioinary	Passed
Validate Create new account	Enter credentials to create new account	New account created message to be delivered	New key value pair created	Passed

Test decryption feature	Enter valid encrypted text	Original text retrieved	Decryption successful	Passed
Validate data transfer and creation	Enter valid data And check functionality	Credential credibility	Correct result	Passed
Test adding new tasks	Enter new tasks	Task added succesfully	Task added successfully	Passed
Test deleting task	Select tasks to delete	Task deleted	Working as expected	Passed
Validate view upcoming task	Check all incomdisplayed	Tasks displayed succesfully	Incomplete task displayed	Passed

Validate functionioning of prepond and postpone tasks	Select tasks and prepond or postpone tasks	Task display order changed	Task order successful	Passed
Test plan future feature	Add new tasks	Tasks added succesfulley	Task addition succesfull	Passed
Validate adding deadline	Enter valid data And add deadline	Accurate result displayed	Correct result	Passed
Test days to deadline	Enter current date	Number of days before deadline displayed	Accurate results	Passed
Enter improper dates to find error	Enter date like 31 feb	Error message displayed	Working as expected	Passed
Validate the functioning of complete and incomplete tasks	Clicking on separate tasks to check if tasks are complete	Tasks completed and incomplete displayed separately	Task displayed separetely	Passed

2. integration testing: multiple modules tested to check integration

Test Description	**Input**	**Expected Output**	**Actual Output**	**Status**
Test login and navigation to menu page	Enter valid login credentials	User is redirected to the menu page	Navigation successful	Passed
Validate menu navigation to converters	Select any converter from the menu	Converter module opens without error	Navigation smooth	Passed
Test return to menu from converter	Use “Back to Menu” button	User is returned to the main menu	Transition seamless	Passed
Verify data flow between modules	Perform conversions and navigate between modules	Results and navigation are consistent	Results retained correctly	Passed
Test logout functionality from a module	Logout from any converter page	Application returns to login page	Logout works as expected	Passed
Validate functioning of tasks	Task addition, deletion, modification tested	Original text is retrieved successfully	Proper functioning of tasks	Passed
Test view task	Fetch incomplete tasks	Incomplete tasks retrieved sucessfully	API integration successful	Passed
Verify system behavior during navigation	Perform multiple navigations (menu to converters and back)	Application remains stable	Stable and functional	Passed

Validate functionioning of prepond and postpone tasks	Select tasks and prepond or postpone tasks	Task display order changed	Task order successful	Passed
Test plan future feature	Add new tasks	Tasks added succesfulley	Task addition succesfull	Passed
Validate adding deadline	Enter valid data And add deadline	Accurate result displayed	Correct result	Passed
Test days to deadline	Enter current date	Number of days before deadline displayed	Accurate results	Passed
Enter improper dates to find error	Enter date like 31 feb	Error message displayed	Working as expected	Passed
Validate the functioning of complete and incomplete tasks	Clicking on separate tasks to check if tasks are complete	Tasks completed and incomplete displayed separately	Task displayed separetely	Passed

Test Description	**Input**	**Expected Output**	**Actual Output**	**Status**
Test login and navigation to menu page	Enter valid login credentials	User is redirected to the menu page	Navigation successful	Passed
Validate menu navigation to converters	Select any converter from the menu	Converter module opens without error	Navigation smooth	Passed
Test return to menu from converter	Use “Back to Menu” button	User is returned to the main menu	Transition seamless	Passed
Verify data flow between modules	Perform conversions and navigate between modules	Results and navigation are consistent	Results retained correctly	Passed
Test logout functionality from a module	Logout from any converter page	Application returns to login page	Logout works as expected	Passed

Validate functioning of tasks	Task addition, deletion, modification tested	Original text is retrieved successfully	Proper functioning of tasks	Passed
Test view task	Fetch incomplete tasks	Incomplete tasks retrieved sucessfully	API integration successful	Passed
Verify system behavior during navigation	Perform multiple navigations (menu to converters and back)	Application remains stable	Stable and functional	Passed

Conclusion

The Task Manager GUI project effectively demonstrates the capabilities of Python and the Tkinter library in creating a user-friendly application for organizing and managing tasks. The project showcases robust functionality by providing core features such as adding, editing, deleting, and viewing tasks, along with options for prioritization and rescheduling. The design is focused on usability, with a clean and intuitive interface leveraging interactive components such as dropdowns and buttons ensuring smooth navigation and task management. Splitting the logic into modular components such as the "Task Manager" and "Upcoming Tasks Page" classes emphasizes a well-structured and maintainable code base, making the program extensible and adaptable to future enhancements. Additionally, the design includes robust error handling mechanisms to gracefully handle unexpected inputs and system problems.

Despite its strengths, the application is currently limited by its reliance on in-memory storage, which results in lost tasks at program termination. Integrating persistent storage solutions like JSON, CSV, database systems etc. would greatly increase its practicality in real-world usage. The app also has extensibility potential with features like notifications, calendar integration, priority-based sorting, advanced scheduling algorithms etc. Additionally, optimizing the system to handle large data sets and concurrent user operations improves performance and usability. Overall, this design provides a solid foundation for task management, effectively balancing functionality, u usability, and maintainability, while providing extensive opportunities for future improvements and enhancements.

Reference

Tkinter Documentation

- Comprehensive guide for creating GUIs with Python.
  https://docs.python.org/3/library/tkinter.html

JSON Handling in Python

- Learn about data serialization and deserialization with JSON.
  https://docs.python.org/3/library/json.html

SQLite for Python Applications

- Official documentation for using SQLite with Python.
  https://sqlite.org/docs.html

CustomTkinter Repository

- Advanced library for creating modern GUIs in Python.
  https://github.com/TomSchimansky/CustomTkinter

RealPython on Error Handling

- Practical tips for implementing error management in Python.
  https://realpython.com/python-exceptions/