Major Project

Personal To-Do List Application

Github Link:

https://github.com/UnnamDedeepya/Major-Project

Source code:

```
import tkinter as tk
from tkinter import messagebox
import json

# Task class definition
class Task:
    def __init__(self, title, description, category):
        self.title = title
        self.description = description
        self.category = category
        self.completed = False

    def mark_completed(self):
        self.completed = True

    def to_dict(self):
        return {
            'title': self.title,
            'description': self.description,
            'category': self.category,
            'completed': self.completed
        }

    @classmethod
    def from_dict(cls, data):
        # Validate data before creating a Task instance
        if 'title' not in data or 'description' not in data or
'category' not in data:
            # If any key is missing, return None to indicate invalid
data
            return None
        task = cls(data['title'], data['description'], data['category'])
        task.completed = data.get('completed', False)
        return task

# File handling functions
```

```
def save_tasks(tasks, filename='tasks.json'):
    with open(filename, 'w') as f:
        json.dump([task.to_dict() for task in tasks], f, indent=4)

def load_tasks(filename='tasks.json'):
    try:
        with open(filename, 'r') as f:
            # Filter out invalid tasks (i.e., those that return None)
            return [task for task in (Task.from_dict(data) for data in
json.load(f)) if task]
    except (FileNotFoundError, json.JSONDecodeError):
        # Return an empty list if the file is not found or JSON is
invalid
        return []

# Main application class
class TodoApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Personal To-Do List")
        self.tasks = load_tasks()

        # UI Elements
        self.task_listbox = tk.Listbox(root, selectmode=tk.SINGLE,
width=50, height=15)
        self.task_listbox.pack(padx=10, pady=10)

        # Buttons
        tk.Button(root, text="Add Task",
command=self.add_task).pack(pady=5)
        tk.Button(root, text="Edit Task",
command=self.edit_task).pack(pady=5)
        tk.Button(root, text="Mark Completed",
command=self.mark_task_completed).pack(pady=5)
        tk.Button(root, text="Delete Task",
command=self.delete_task).pack(pady=5)
        tk.Button(root, text="Save & Exit",
command=self.save_and_exit).pack(pady=5)

        # Load tasks into the listbox
        self.refresh_task_list()

    def refresh_task_list(self):
        self.task_listbox.delete(0, tk.END)
        for index, task in enumerate(self.tasks):
            status = " " if task.completed else "❌"
            self.task_listbox.insert(tk.END, f"{index + 1}. {task.title}
[{task.category}] - {status}")
```

```
    def add_task(self):
        title = simple_input("Enter task title:")
        if not title:
            return
        description = simple_input("Enter task description:")
        category = simple_input("Enter task category (e.g., Work,
Personal, Urgent):")
        self.tasks1.append(Task(title, description, category))
        self.refresh_task_list()
        messagebox.showinfo("Success", "Task added successfully.")

    def edit_task(self):
        selected_task_index = self.task_listbox.curselection()
        if not selected_task_index:
            messagebox.showwarning("Warning", "Please select a task to
edit.")
            return
        index = selected_task_index[0]
        task = self.tasks[index]

        task.title = simple_input("Edit task title:", task.title)
        task.description = simple_input("Edit task description:",
task.description)
        task.category = simple_input("Edit task category:",
task.category)
        self.refresh_task_list()
        messagebox.showinfo("Success", "Task edited successfully.")

    def mark_task_completed(self):
        selected_task_index = self.task_listbox.curselection()
        if not selected_task_index:
            messagebox.showwarning("Warning", "Please select a task to
mark as completed.")
            return
        index = selected_task_index[0]
        self.tasks1[index].mark_completed()
        self.refresh_task_list()
        messagebox.showinfo("Success", "Task marked as completed.")
```

```
    def delete_task(self):
        selected_task_index = self.task_listbox.curselection()
        if not selected_task_index:
            messagebox.showwarning("Warning", "Please select a task to
delete.")
            return
        index = selected_task_index[0]
        del self.tasks[index]
        self.refresh_task_list()
        messagebox.showinfo("Success", "Task deleted successfully.")

    def save_and_exit(self):
        save_tasks(self.tasks)
        self.root.quit()

def simple_input(prompt, default=""):
    input_window = tk.Toplevel()
    input_window.title(prompt)
    tk.Label(input_window, text=prompt).pack(padx=20, pady=10)
    entry = tk.Entry(input_window, width=50)
    entry.insert(0, default)
    entry.pack(padx=20, pady=5)

    def on_confirm():
        global user_input
        user_input = entry.get()
        input_window.destroy()

    tk.Button(input_window, text="OK", command=on_confirm).pack(pady=10)
    input_window.grab_set()
    input_window.wait_window()
    return user_input

# Running the application
if __name__ == '__main__':
    root = tk.Tk()
    app = TodoApp(root)
    root.mainloop()
```

Output Screens: