```
# importing necessary libraries
import os
import tkinter as tk
from tkinter import ttk, messagebox, simpledialog
import json
from datetime import datetime

# defining the main application class
class TodoList:
    def __init__(self, master):
        # setting up the main window
        self.master = master
        self.master.title("To-Do List")
        self.master.geometry("400x500")
        self.master.configure(bg="#BD5F12")

        # setting up styles
        style = ttk.Style()
        style.theme_use('clam')
        style.configure("TFrame", background="#BD5F12")
        style.configure("TButton", padding=10, font=('Times New Roman', 10))
        style.configure("TLabel", background="#BD5F12", font=('Times New Roman', 10))
        style.configure("TEntry", padding=10, font=('Times New Roman', 10))
        style.configure("Treeview", font=('Times New Roman', 10), rowheight=25)
        style.configure("Treeview.Heading", font=('Times New Roman', 10, 'bold'))
        style.map('TButton', background=[('active', '#FFA500')])
```

```
        # setting up the main frame
        self.frame = ttk.Frame(self.master, padding=10, style="TFrame")
        self.frame.pack(fill=tk.BOTH, expand=True)

        # setting up widgets
        self.task_var = tk.StringVar()
        self.task_entry = ttk.Entry(self.frame, textvariable=self.task_var, width=30, style="TEntry")
        self.task_entry.grid(row=0, column=0, padx=5, pady=5, sticky="ew")

        # Add Task button
        self.add_button = ttk.Button(self.frame, text="Add Task", command=self.add_task)
        self.add_button.grid(row=0, column=1, padx=5, pady=10, sticky="ew")

        # Task list display
        self.task_tree = ttk.Treeview(self.frame, columns=("Task","Priority","Created At"), show="headings", style="Treeview")
        self.task_tree.heading("Task", text="Task")
        self.task_tree.heading("Priority", text="Priority")
        self.task_tree.heading("Created At", text="Created At")
        self.task_tree.grid(row=1, column=0, columnspan=2, padx=5, pady=5, sticky="nsew")

        # Color coding based on priority
        self.task_tree.tag_configure("High", background="#FF8783")
        self.task_tree.tag_configure("Mid", background="#F7E26D")
        self.task_tree.tag_configure("Low", background="#63C763")
```

```
        # Adding scrollbar
        scrollbar = ttk.Scrollbar(self.frame, orient=tk.VERTICAL,
command=self.task_tree.yview)
        scrollbar.grid(row=1, column=2, sticky="ns")
        self.task_tree.configure(yscrollcommand=scrollbar.set)

        # Control buttons for delete, edit, save, and sort
        self.delete_button = ttk.Button(self.frame, text="Delete Task",
command=self.delete_task)
        self.delete_button.grid(row=2, column=0, padx=5, pady=5, sticky="ew")

        self.edit_button = ttk.Button(self.frame, text="Edit Task", command=self.edit_task)
        self.edit_button.grid(row=2, column=1, padx=5, pady=5, sticky="ew")

        self.save_button = ttk.Button(self.frame, text="Save Tasks", command=self.save_tasks)
        self.save_button.grid(row=3, column=0, columnspan=2, padx=5, pady=5, sticky="ew")

        self.sort_button = ttk.Button(self.frame, text="Sort by Priority",
command=self.sort_by_priority)
        self.sort_button.grid(row=4, column=0, columnspan=2, padx=5, pady=5, sticky="ew")

        # configuring grid weights
        self.frame.rowconfigure(1, weight=1)
        self.frame.columnconfigure(0, weight=1)
        self.frame.columnconfigure(1, weight=1)

        self.load_tasks()
```

 # defining methods for task operations

```
  def add_task(self):
    task = self.task_var.get().strip()
    if task:
      priority = simpledialog.askstring("Priority", "Enter priority (High,Mid,Low):",
parent=self.master)
      if priority and priority.lower() in ["high", "mid", "low"]:
        created_at = datetime.now().strftime("%Y-%m-%d %H:%M:%S")


        self.task_tree.insert("", tk.END, values=(task, priority.capitalize(), created_at))
        self.task_var.set("")
      else:
        messagebox.showwarning("Invalid Input", "Please enter High, Mid, or Low.")
    else:
      messagebox.showwarning("Warning", "Please enter a task.")
```

 # method to delete a selected task

```
  def delete_task(self):
    selected_items = self.task_tree.selection()
    if selected_items:
      for item in selected_items:
        self.task_tree.delete(item)
    else:
      messagebox.showwarning("Warning", "Please select a task to delete.")
```

 # method to edit a selected task

```
  def edit_task(self):
    selected_items = self.task_tree.selection()
    if selected_items:
      item = selected_items[0]
      current_task, current_priority, current_created = self.task_tree.item(item, "values")
      new_task = simpledialog.askstring("Edit Task", "Update the task:",
initialvalue=current_task)
      new_priority = simpledialog.askstring("Edit Priority", "Update priority (High, Mid,
Low):", initialvalue=current_priority)
      if new_task and new_priority and new_priority.lower() in ["high", "mid", "low"]:
        self.task_tree.item(item, values=(new_task, current_created,
new_priority.capitalize()))
      else:
        messagebox.showwarning("Invalid Input", "Please enter valid task and priority.")
    else:
      messagebox.showwarning("Warning", "Please select a task to edit.")
```

 # method to save tasks to a JSON file

```
  def save_tasks(self):
    tasks = [self.task_tree.item(child)["values"] for child in self.task_tree.get_children()]
    with open("tasks.json", "w") as f:
      json.dump({"tasks": tasks}, f, indent=2)
    messagebox.showinfo("Success", "Tasks saved successfully.")
```

 # method to load tasks from a JSON file

```
  def load_tasks(self):
```

```
    if os.path.exists("tasks.json"):
        try:
            with open("tasks.json", "r") as f:
                data = json.load(f)
                tasks = data.get("tasks", [])
                for task in tasks:
                    if isinstance(task, list) and len(task) == 3:
                        self.task_tree.insert("", tk.END, values=(task[0], task[1], task[2]),
tags=(task[1],))
                    elif isinstance(task, list) and len(task) == 2:
                        self.task_tree.insert("", tk.END, values=(task[0], task[1], ""), tags=(task[1],))
                    elif isinstance(task, str):  # fallback for old format
                        self.task_tree.insert("", tk.END, values=(task, "Mid", ""), tags=("Mid",))
        except (json.JSONDecodeError, KeyError):
            messagebox.showerror("Error", "Failed to load tasks. File may be corrupted.")
    else:
        messagebox.showinfo("Info", "No saved tasks found.")
```

 # method to sort tasks by priority

```
  def sort_by_priority(self):
    priority_order = {"High": 1, "Mid": 2, "Low": 3}
    tasks = [
        (priority_order[item[1]], item[0], item[1], item[2])
        for item in [self.task_tree.item(i)["values"] for i in self.task_tree.get_children()]
        if len(item) == 3
    ]
    tasks.sort()
```

```
        for i in self.task_tree.get_children():
            self.task_tree.delete(i)
        for _, task, priority, created_at in tasks:
            self.task_tree.insert("", tk.END, values=(task, priority, created_at), tags=(priority,))

# running the application
if __name__ == "__main__":
    root = tk.Tk()
    app = TodoList(root)
    root.mainloop()
```