Sprint# 4 Team expense tracker

- **Yamini Jonnala-00884974**
- **Nivas Reddy-00805083**
- **Sarika Burugu-00876552**

#1 overview

Our expense tracker in Python provides a comprehensive solution for users to manage their spending effectively.

By tracking daily expenses and providing detailed summaries, users can gain insights into their spending habits and make informed financial decisions.

- Expenses entry.
- Error when expenses exceed the monthly budget.

orStock®
VectorStock.com/39050843

Sprint#2 overview

For sprint#2 we have taken an input and an output file.

After running the code and entering the expense data, this data will be saved and viewed in the output file.

The data will be saved in the output file named ‘users.json’.

Sprint#3 overview

Designing a login interface, structuring the Expense Tracker GUI layout effectively, and integrating notification features are crucial steps in creating a user-friendly and functional expense tracking application. A well-designed interface coupled with intuitive navigation and helpful features enhances user engagement and promotes better financial management.

Sprint#4 Deliverables:

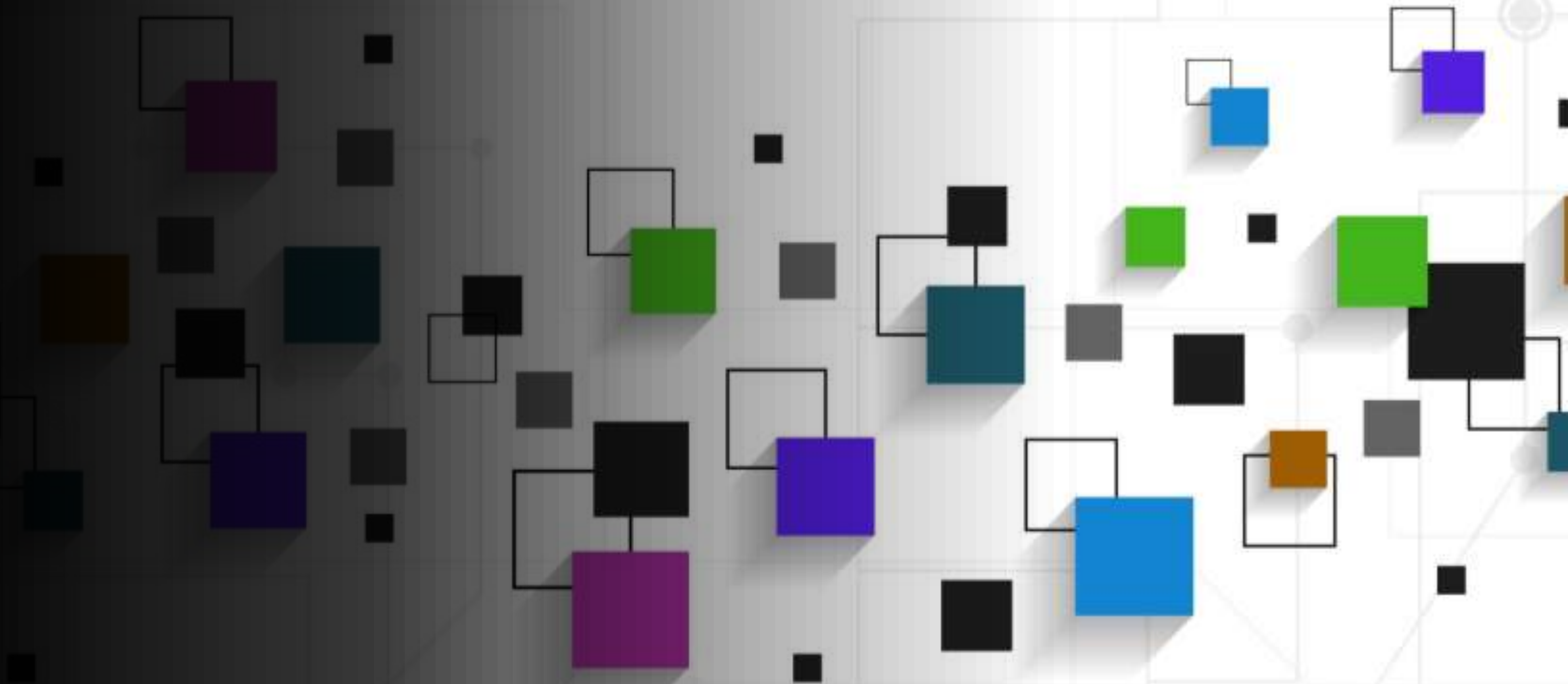
- User Interface Refinement: Polishing the appearance of the application.
- Ensuring consistency in layout, colors, and Fonts. Adjust widget sizes and positions for better Usability.
- Use simple icons and labels to guide users through the application.
- Exception Handling

Exception Handling:

Some basic error handling is implemented using try-except blocks and message boxes to inform the user about errors.

- **Set Budget Function (set_budget):**

When setting a new budget, the code attempts to convert the entered value to a float. If the entered value cannot be converted to a float (e.g., if the user enters non-numeric characters), a ValueError will occur. This error is caught using a try-except block, and a message box is displayed to inform the user that the budget must be a valid number.



- **Add Expense Function (add_expense):**

When adding a new expense, the code attempts to convert the entered amount to a float. If the entered amount cannot be converted to a float (e.g., if the user enters non-numeric characters), a ValueError will occur. This error is caught using a try-except block, and a message box is displayed to inform the user that the amount must be a valid number.

- **Date Parsing in View Graph Function (view_graph):**

When parsing the date of each expense to calculate total expenses for each month, there is a try-except block to handle ValueError. If an expense's date is in an invalid format, such as not conforming to the expected format ("%Y-%m-%d"), the code prints a message to indicate that there's an invalid date format for that expense.

Authentication Page:

- Username Entry:
- Background: Light gray (#F0F0F0)
- Text Color: Black

- Password Entry:
- Background: Light gray (#F0F0F0)
- Text Color: Black

- Sign Up Button:
- Color: Green (#4CAF50)

- Log In Button:
- Color: Blue (#008CBA)

Expense Entry Page:

Expense Entry: - Background: Light gray (#F0F0F0) - Text Color: Black- Date Entry:

 Background: Light gray (#F0F0F0) - Text Color: Black - Description Entry:

 Background: Light gray (#F0F0F0) - Text Color: Black- Category Entry: - Background: Light gray (#F0F0F0) - Text Color: Black- Amount Entry: - Background: Light gray (#F0F0F0) - Text Color: Black- Monthly Budget Entry: - Background: Light gray (#F0F0F0) - Text Color: Black- Set Budget Button: - Color: Green (#4CAF50)- Add Expense Button: - Color: Blue (#008CBA)- View Expenses Button: - Color: Red (#f44336)- View Graph Button: - Color: Orange (#FF9800)

Graph Colors:

Graph Colors:- Total Expenses by Month Graph: - Bar Color: Green (#4CAF50) - Title: "Total Expenses by Month" - X-Axis Label: "Month" - Y-Axis Label: "Total Expenses ($)“

These color choices aim to maintain consistency, clarity, and visual appeal throughout the application.

```
import tkinter as tk
from tkinter import messagebox
import matplotlib.pyplot as plt
from collections import defaultdict
from datetime import datetime
from PIL import Image, ImageTk
import json
from file_handler import FileHandler

class AuthenticationPage:
    def __init__(self, master):
        self.master = master
        self.master.configure(bg="#FFFFFF")  # Set background color
        self.frame = tk.Frame(self.master, bg="#FFFFFF")  # Frame background color
        self.frame.pack()

        self.user_filename = "users.json"
        self.users = FileHandler.load_users_from_file()

        self.username_label = tk.Label(self.frame, text="Username:", bg="#F0F0F0", fg="black")  # Label colors
        self.username_label.grid(row=0, column=0, sticky="e")
        self.username_entry = tk.Entry(self.frame)
        self.username_entry.grid(row=0, column=1, sticky="w")

        self.password_label = tk.Label(self.frame, text="Password:", bg="#F0F0F0", fg="black")
        self.password_label.grid(row=1, column=0, sticky="e")
        self.password_entry = tk.Entry(self.frame, show="*")
        self.password_entry.grid(row=1, column=1, sticky="w")

        self.sign_up_button = tk.Button(self.frame, text="Sign Up", command=self.sign_up, bg="#4CAF50", fg="white")  # Button colors
        self.sign_up_button.grid(row=2, columnspan=2, pady=5)

        self.login_button = tk.Button(self.frame, text="Log In", command=self.login, bg="#008CBA", fg="white")
        self.login_button.grid(row=3, columnspan=2, pady=5)
    def save_users_to_file(self):
        FileHandler.save_users_to_file(self.users)

    def sign_up(self):
        username = self.username_entry.get()
        password = self.password_entry.get()
        if username and password:
            if username in self.users:
                messagebox.showerror("Sign Up Error", "Username already exists. Please choose another one.")
            else:
                self.users[username] = {"password": password, "expenses": []}
                self.save_users_to_file()
                messagebox.showinfo("Sign Up Success", "Account created successfully. You can now log in.")
        else:
            messagebox.showerror("Sign Up Error", "Please enter both username and password.")

    def login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()
        if username in self.users and self.users[username]["password"] == password:
            self.frame.destroy()
            self.master.title("Expense Tracker - Welcome, " + username)
            ExpenseEntryPage(self.master, self.users, username)
        else:
            messagebox.showerror("Login Error", "Invalid username or password.")

  ass ExpenseEntryPage:
    def __init__(self, master, users, username):
        self.master = master
        self.master.configure(bg="#F0F0F0")
        self.frame = tk.Frame(self.master, bg="#F0F0F0")
        self.frame.pack()

        self.users = users
        self.username = username
        self.current_user = self.username
        self.monthly_budget = self.load_monthly_budget()
```

```
self.expense_label = tk.Label(self.frame, text="Expense:", bg="#F0F0F0", fg="black")
self.expense_label.grid(row=0, column=0, sticky="e")
self.expense_entry = tk.Entry(self.frame)
self.expense_entry.grid(row=0, column=1, sticky="w")

self.date_label = tk.Label(self.frame, text="Date (YYYY-MM-DD):", bg="#F0F0F0", fg="black")
self.date_label.grid(row=1, column=0, sticky="e")
self.date_entry = tk.Entry(self.frame)
self.date_entry.grid(row=1, column=1, sticky="w")

self.description_label = tk.Label(self.frame, text="Description:", bg="#F0F0F0", fg="black")
self.description_label.grid(row=2, column=0, sticky="e")
self.description_entry = tk.Entry(self.frame)
self.description_entry.grid(row=2, column=1, sticky="w")

self.category_label = tk.Label(self.frame, text="Category:", bg="#F0F0F0", fg="black")
self.category_label.grid(row=3, column=0, sticky="e")
self.category_entry = tk.Entry(self.frame)
self.category_entry.grid(row=3, column=1, sticky="w")

self.amount_label = tk.Label(self.frame, text="Amount:", bg="#F0F0F0", fg="black")
self.amount_label.grid(row=4, column=0, sticky="e")
self.amount_entry = tk.Entry(self.frame)
self.amount_entry.grid(row=4, column=1, sticky="w")

self.budget_label = tk.Label(self.frame, text="Monthly Budget:", bg="#F0F0F0", fg="black")
self.budget_label.grid(row=5, column=0, sticky="e")
self.budget_entry = tk.Entry(self.frame)
self.budget_entry.grid(row=5, column=1, sticky="w")
self.budget_entry.insert(0, self.monthly_budget)

self.set_budget_button = tk.Button(self.frame, text="Set Budget", command=self.set_budget, bg="#4CAF50", fg="white")
self.set_budget_button.grid(row=6, columnspan=2, pady=5)

self.add_expense_button = tk.Button(self.frame, text="Add Expense", command=self.add_expense, bg="#008CBA", fg="white")
self.add_expense_button.grid(row=7, columnspan=2, pady=5)

self.view_expenses_button = tk.Button(self.frame, text="View Expenses", command=self.view_expenses, bg="#f44336", fg="white")
self.view_expenses_button.grid(row=8, columnspan=2, pady=5)
```

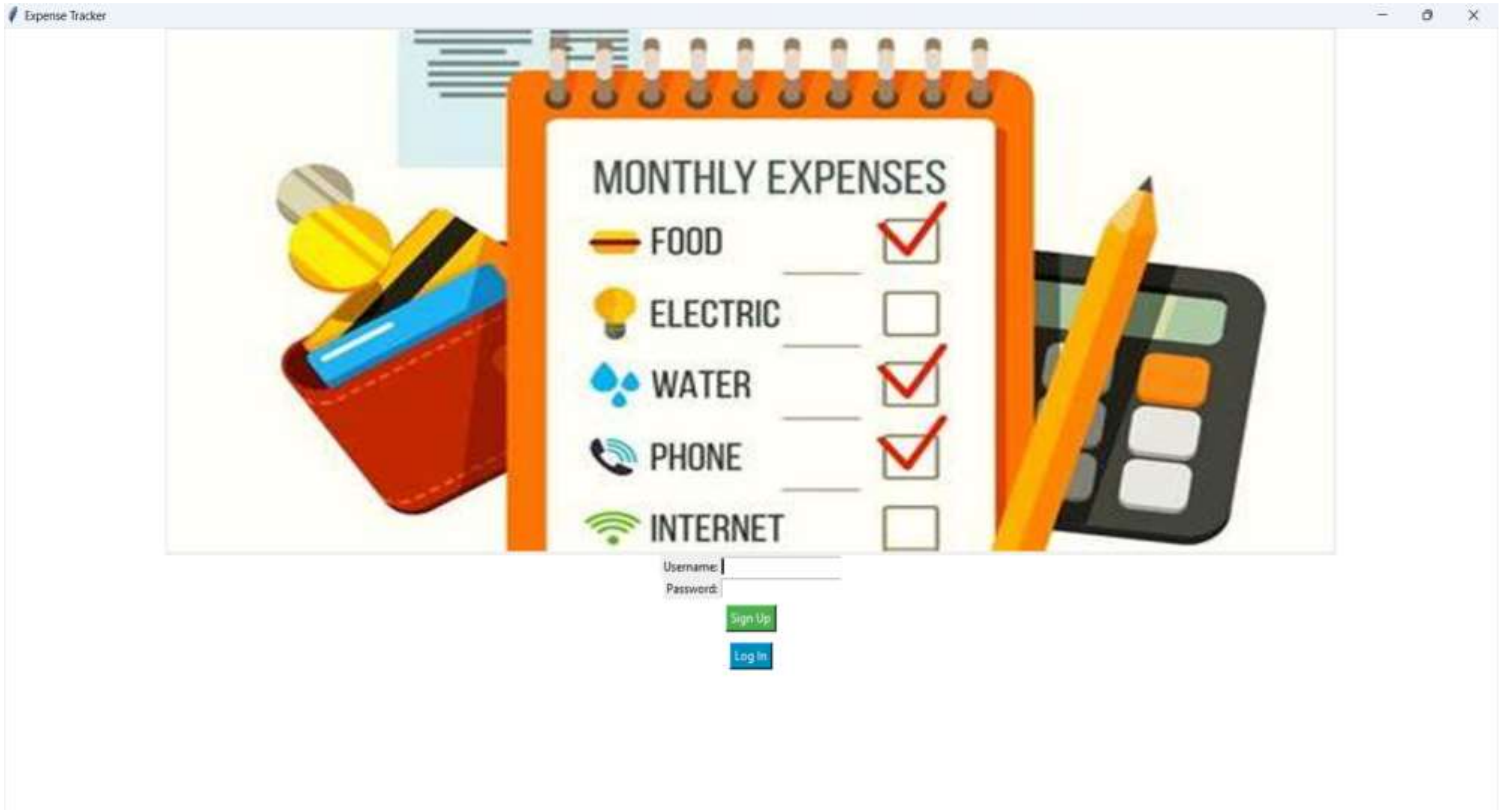
```
        self.view_graph_button = tk.Button(self.frame, text="View Graph", command=self.view_graph, bg="#FF9800", fg="white")
        self.view_graph_button.grid(row=9, columnspan=2, pady=5)

    def load_monthly_budget(self):
        return self.users[self.username].get("monthly_budget", 0)

    def save_monthly_budget(self, budget):
        self.users[self.username]["monthly_budget"] = budget
        self.save_users_to_file()

    def set_budget(self):
        new_budget = self.budget_entry.get()
        if new_budget:
            try:
                new_budget = float(new_budget)
                self.monthly_budget = new_budget
                self.budget_entry.delete(0, tk.END)
                self.budget_entry.insert(0, self.monthly_budget)
                self.save_monthly_budget(new_budget)
                messagebox.showinfo("Budget Updated", "Monthly budget updated successfully!")
            except ValueError:
                messagebox.showerror("Budget Entry Error", "Budget must be a valid number.")
        else:
            messagebox.showerror("Budget Entry Error", "Please enter a budget.")

    def save_users_to_file(self):
        FileHandler.save_users_to_file(self.users)

    def add_expense(self):
        date = self.date_entry.get()
        description = self.description_entry.get()
        category = self.category_entry.get()
        amount = self.amount_entry.get()
        if date and description and category and amount:
            try:
```

```
            amount = float(amount)
            self.users[self.username]["expenses"].append({
                "date": date,
                "description": description,
                "category": category,
                "amount": amount
            })
            messagebox.showinfo("Expense Added", "Expense added successfully!")
            self.save_users_to_file()

            # Check if monthly expenses exceed budget
            monthly_expenses = sum(expense["amount"] for expense in self.users[self.username]["expenses"])
            if monthly_expenses > self.monthly_budget:
                messagebox.showwarning("Budget Exceeded", "Total monthly expenses exceed the budget!")
        except ValueError:
            messagebox.showerror("Expense Entry Error", "Amount must be a valid number.")
    else:
        messagebox.showerror("Expense Entry Error", "Please fill in all fields.")
def view_expenses(self):
  expenses = self.users[self.username]["expenses"]
  if expenses:
      expenses_window = tk.Toplevel(self.master)
      expenses_window.title("Expenses")

      # Create a frame for displaying expenses
      expenses_frame = tk.Frame(expenses_window, bg="#FFFFFF")
      expenses_frame.pack(padx=10, pady=10)

      # Add color to even and odd rows
      for idx, expense in enumerate(expenses):
          bg_color = "#FFFFFF" if idx % 2 == 0 else "#CCCCCC"
          expense_str = f"{expense['date']}: {expense['description']} - {expense['category']}: ${expense['amount']}"
          expense_label = tk.Label(expenses_frame, text=expense_str, bg=bg_color)
          expense_label.pack(anchor="w")
  else:
      messagebox.showinfo("Expenses", "No expenses to show.")
```

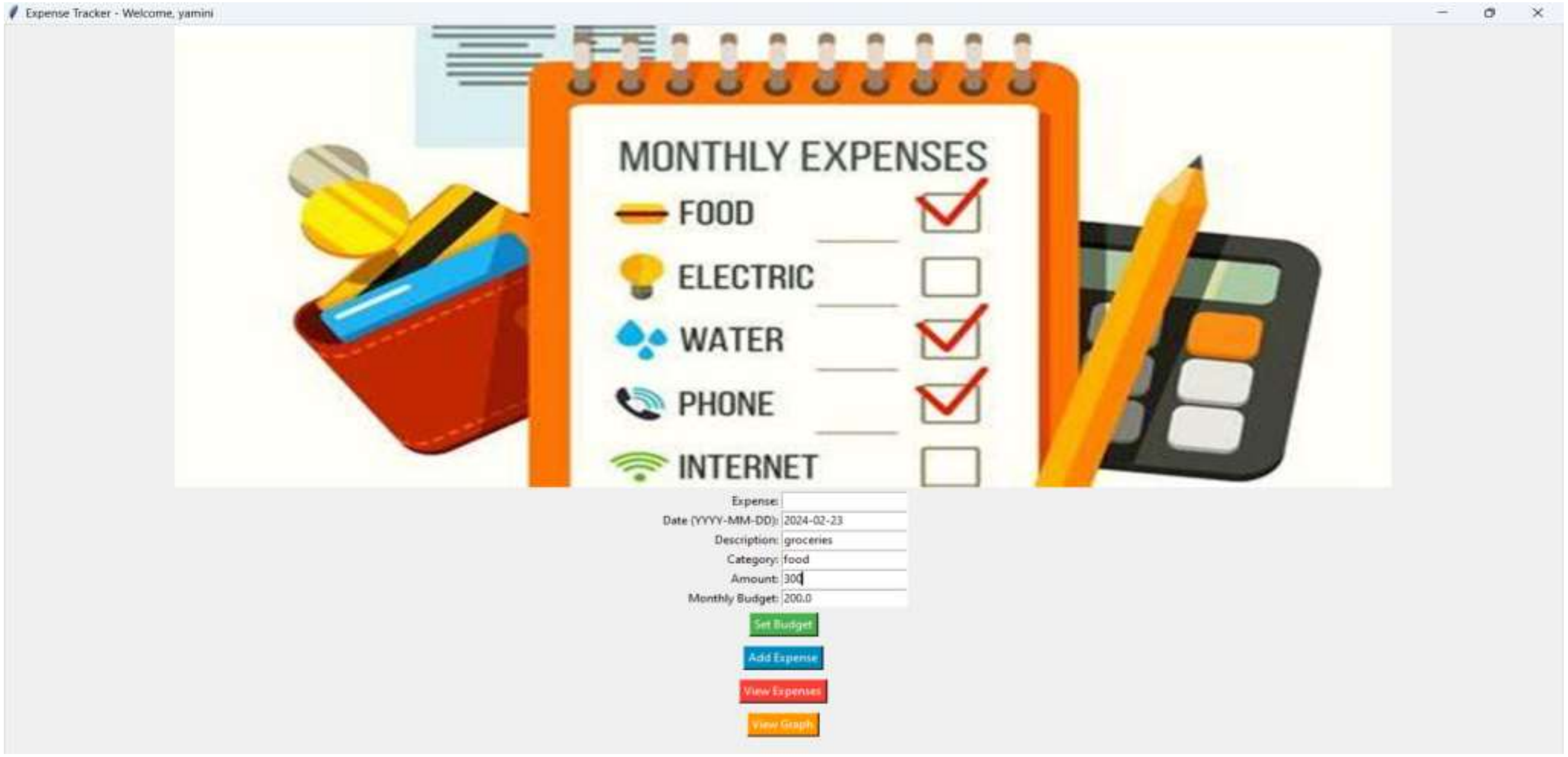
```
def view_graph(self):
    expenses = self.users[self.username]["expenses"]
    if expenses:
        # Calculate total expenses for each month
        monthly_totals = defaultdict(float)
        for expense in expenses:
            try:
                date = datetime.strptime(expense["date"], "%Y-%m-%d")
                month = date.strftime("%Y-%m")
                monthly_totals[month] += expense["amount"]
            except ValueError:
                print("Invalid date format for expense:", expense["date"])

        # Prepare data for plotting
        months = sorted(monthly_totals.keys())
        total_expenses = [monthly_totals[month] for month in months]

        # Plot the graph
        plt.figure(figsize=(10, 6))
        plt.bar(months, total_expenses, color='#4CAF50')  # Graph color
        plt.title('Total Expenses by Month')
        plt.xlabel('Month')
        plt.ylabel('Total Expenses ($)')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()
    else:
        messagebox.showinfo("View Graph", "No expenses to display.")

main():
root = tk.Tk()
root.title("Expense Tracker")

# Adjust canvas size according to window dimensions
window_width = 1200  # Adjust as needed
window_height = 500  # Adjust as needed
canvas = tk.Canvas(root, width=window_width, height=window_height)
canvas.pack()
```

```
    # Load and resize the background image
    background_image_path = "background.jpeg"  # Change this to the actual path of your image
    background_image = Image.open(background_image_path)
    background_image = background_image.resize((window_width, window_height), Image.BICUBIC)  # Resize image
    background_image = ImageTk.PhotoImage(background_image)

    # Create the background image on canvas
    canvas.create_image(0, 0, anchor=tk.NW, image=background_image)

    AuthenticationPage(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Authentication Page:

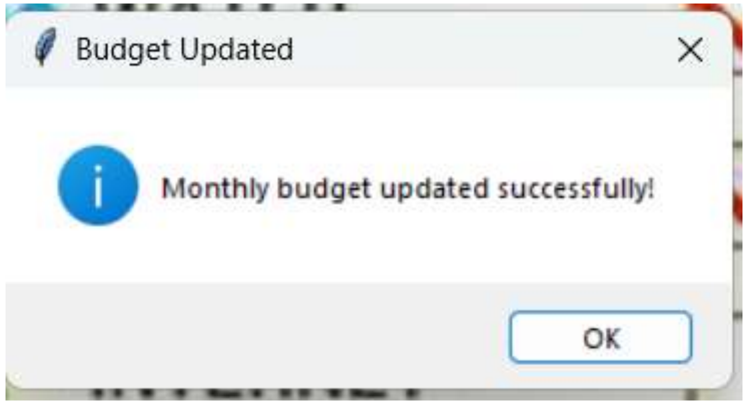


Expenses Entry Page:



Expense:	
Date (YYYY-MM-DD):	2024-02-23
Description:	groceries
Category:	food
Amount:	300
Monthly Budget:	200.0

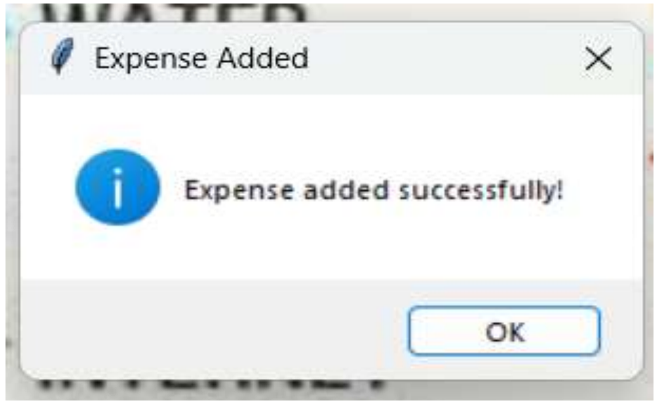
Monthly Budget:



Budget Updated

Monthly budget updated successfully!

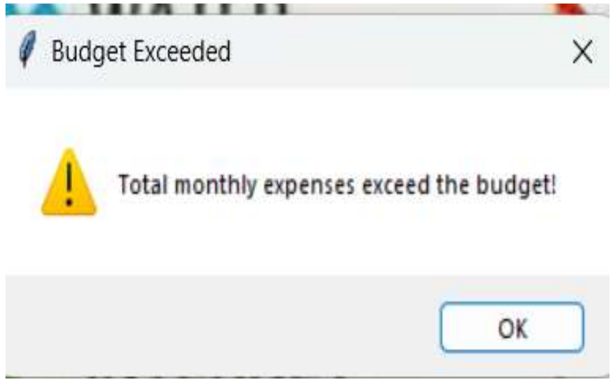
Add Expenses:



Expense Added

Expense added successfully!

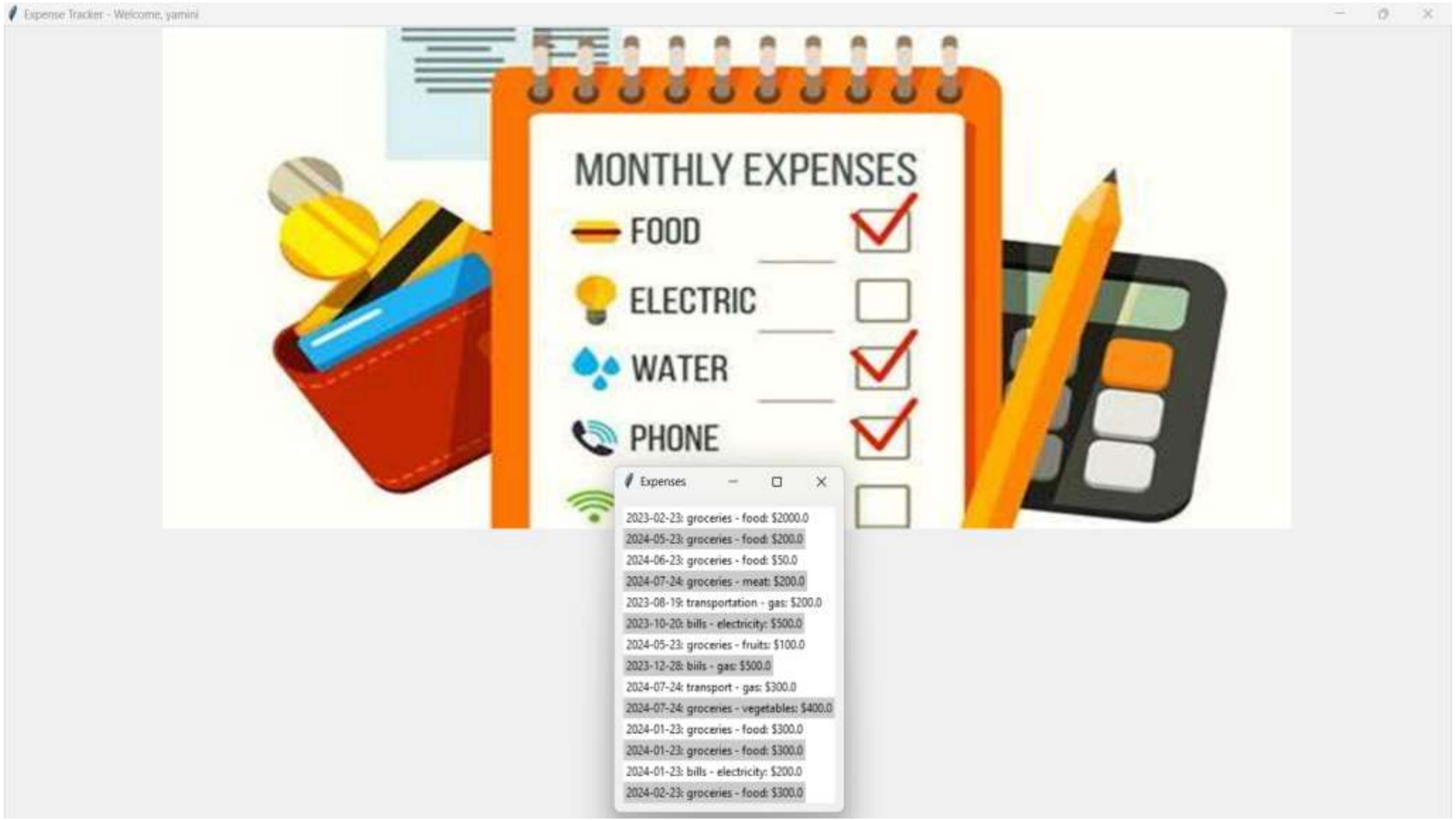
Notification:



Budget Exceeded

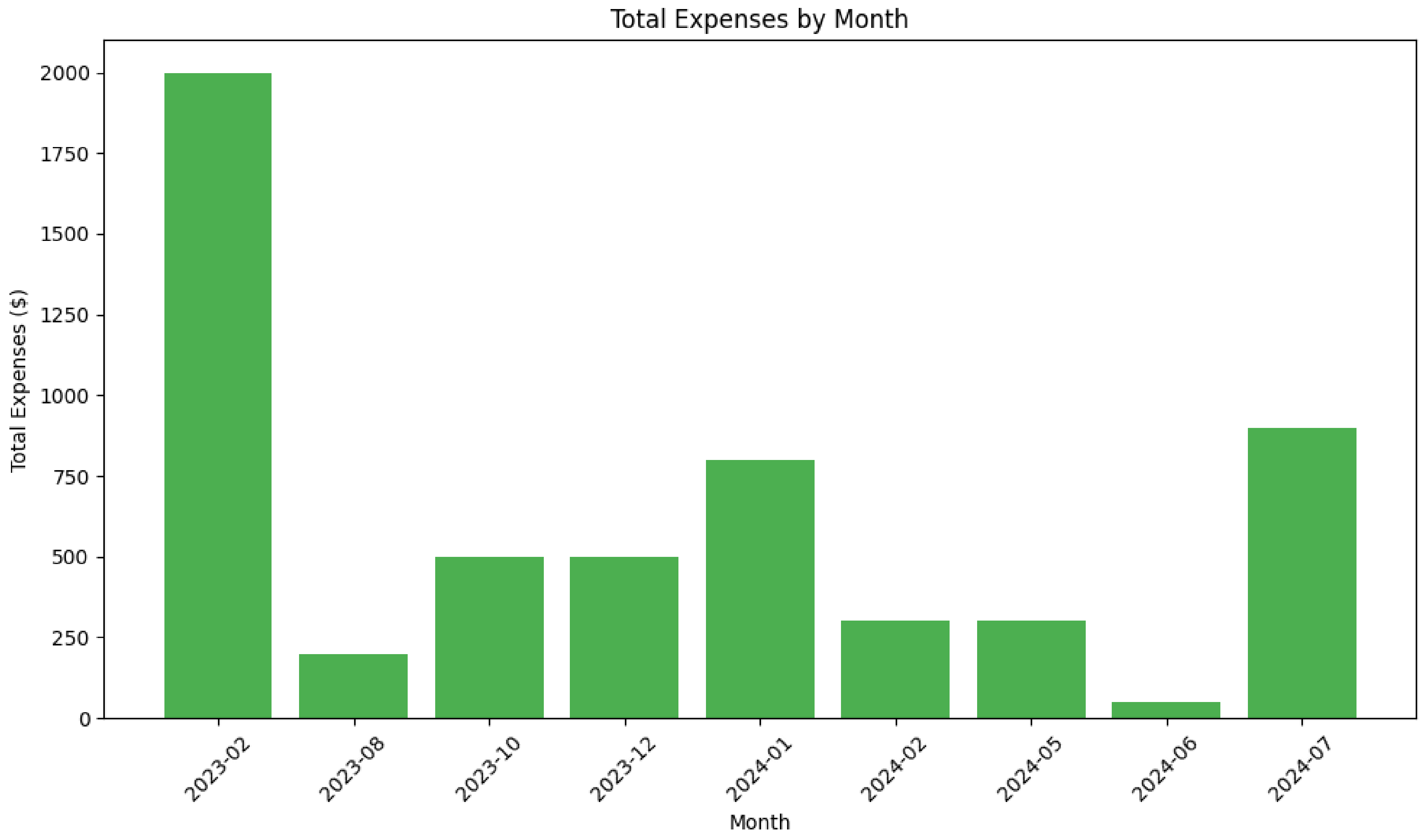
Total monthly expenses exceed the budget!

View Expenses:



```
2023-02-23: groceries - food: $2000.0
2024-05-23: groceries - food: $200.0
2024-06-23: groceries - food: $50.0
2024-07-24: groceries - meat: $200.0
2023-08-19: transportation - gas: $200.0
2023-10-20: bills - electricity: $500.0
2024-05-23: groceries - fruits: $100.0
2023-12-28: biils - gas: $500.0
2024-07-24: transport - gas: $300.0
2024-07-24: groceries - vegetables: $400.0
2024-01-23: groceries - food: $300.0
2024-01-23: groceries - food: $300.0
2024-01-23: bills - electricity: $200.0
2024-02-23: groceries - food: $300.0
```

View Graph:


Total Expenses by Month
Total Expenses ($)
2000
1750
1500
1250
1000
750
500
250
0
2023-02
2023-08
2023-10
2023-12
2024-01
2024-02
2024-05
2024-06
2024-07
Month

Sprint#4 Conclusion:

- User Interface Refinement:
  - Polishing the appearance of the application.
  - Ensuring consistency in layout, colors, and fonts.
  - Adjust widget sizes and positions for better usability.
  - Use simple icons and labels to guide users through the application.
  - Exception Handling: The error handling mechanisms help ensure that the program gracefully handles unexpected inputs or errors during runtime and provides feedback to the user to correct their input or understand any issues encountered.