Ryan Jarrett  
Kacie Jordan  
Drashaun Morrow  
George Xiao

**Team 6 Final Project Code**

import os

import tkinter as tk

from tkinter import messagebox

from datetime import datetime

import re

class BackButton(tk.Button):

def \_\_init\_\_(self, master, callback):

super().\_\_init\_\_(master, text="Back", command=callback)

self.pack(pady=20)

class WelcomePage:

def \_\_init\_\_(self, root, show\_menu\_callback):

self.root = root

self.show\_menu\_callback = show\_menu\_callback

self.welcome\_label = tk.Label(root, text="Welcome to Our Restaurant!", font=('Helvetica', 16))

self.welcome\_label.pack(pady=20)

# Display current time

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label = tk.Label(root, text=f"Current Time: {current\_time}", font=('Helvetica', 12))

self.time\_label.pack()

# Path to the image file

image\_path = os.path.join("dining-7000638\_1280.gif")

if os.path.exists(image\_path):

self.open\_image = tk.PhotoImage(file=image\_path)

self.image\_label = tk.Label(root, image=self.open\_image)

self.image\_label.pack(pady=20)

else:

print("Image file not found.")

self.menu\_button = tk.Button(root, text="See Today's Menu", command=self.on\_menu\_button\_click)

self.menu\_button.pack(pady=20)

def on\_menu\_button\_click(self):

# Call the show\_menu\_callback

self.show\_menu\_callback()

# Destroy the image label

if hasattr(self, 'image\_label'):

self.image\_label.destroy()

# Disable and hide the "See Today's Menu" button after it's pressed

self.menu\_button.config(state=tk.DISABLED)

self.menu\_button.pack\_forget()

def show\_menu(self):

self.show\_menu\_callback()

# Disable and hide the "See Today's Menu" button after it's pressed

self.menu\_button.config(state=tk.DISABLED)

self.menu\_button.pack\_forget()

class MainMenuPage(WelcomePage):

def \_\_init\_\_(self, root, show\_menu\_callback):

super().\_\_init\_\_(root, show\_menu\_callback)

class OrderSummaryPage:

def \_\_init\_\_(self, root, items, total\_price, show\_payment\_page\_callback, reset\_callback):

self.root = root

self.items = items

self.total\_price = total\_price

self.show\_payment\_page\_callback = show\_payment\_page\_callback

self.reset\_callback = reset\_callback

drink\_image\_path = os.path.join("drink.png")

if os.path.exists(drink\_image\_path):

drink\_image = tk.PhotoImage(file=drink\_image\_path)

drink\_label = tk.Label(root, image=drink\_image)

drink\_label.image = drink\_image # Keep a reference to avoid garbage collection

drink\_label.pack(pady=20)

else:

print("Drink image file not found.")

self.title\_label = tk.Label(root, text="Order Summary", font=('Helvetica', 16))

self.title\_label.pack(pady=20)

for item\_info in items:

item\_label = tk.Label(root, text=f"{item\_info['name']} - ${item\_info['price']:.2f}\n{item\_info['description']}", font=('Helvetica', 12))

item\_label.pack()

total\_label = tk.Label(root, text=f"Total: ${total\_price:.2f}", font=('Helvetica', 14))

total\_label.pack(pady=20)

pay\_now\_button = tk.Button(root, text="Pay Now", command=self.show\_payment\_page\_callback)

pay\_now\_button.pack()

def reset(self):

# Call the reset callback to restart the application

self.reset\_callback()

class PaymentPage:

def \_\_init\_\_(self, root, show\_payment\_confirmation\_callback, reset\_callback):

self.root = root

self.show\_payment\_confirmation\_callback = show\_payment\_confirmation\_callback

self.reset\_callback = reset\_callback

self.title\_label = tk.Label(root, text="Enter Payment Information", font=('Helvetica', 16))

self.title\_label.pack(pady=20)

# Entry fields for user information

self.first\_name\_label = tk.Label(root, text="First Name:")

self.first\_name\_label.pack()

self.first\_name\_entry = tk.Entry(root)

self.first\_name\_entry.pack()

self.last\_name\_label = tk.Label(root, text="Last Name:")

self.last\_name\_label.pack()

self.last\_name\_entry = tk.Entry(root)

self.last\_name\_entry.pack()

self.phone\_number\_label = tk.Label(root, text="Phone Number:")

self.phone\_number\_label.pack()

self.phone\_number\_entry = tk.Entry(root)

self.phone\_number\_entry.pack()

self.email\_label = tk.Label(root, text="Email Address:")

self.email\_label.pack()

self.email\_entry = tk.Entry(root)

self.email\_entry.pack()

self.card\_number\_label = tk.Label(root, text="Card Number")

self.card\_number\_label.pack()

self.card\_number\_entry = tk.Entry(root)

self.card\_number\_entry.pack()

self.card\_expire\_label = tk.Label(root, text="Card Expire MM/YY")

self.card\_expire\_label.pack()

self.card\_expire\_entry = tk.Entry(root)

self.card\_expire\_entry.pack()

self.card\_security\_label = tk.Label(root, text="Security Code")

self.card\_security\_label.pack()

self.card\_security\_entry = tk.Entry(root)

self.card\_security\_entry.pack()

confirm\_button = tk.Button(root, text="Confirm Order", command=self.confirm\_order)

confirm\_button.pack()

def confirm\_order(self):

# Get entered information

first\_name = self.first\_name\_entry.get()

last\_name = self.last\_name\_entry.get()

phone\_number = self.phone\_number\_entry.get()

email = self.email\_entry.get()

card\_number = self.card\_number\_entry.get()

card\_expire = self.card\_expire\_entry.get()

card\_security = self.card\_security\_entry.get()

# Validate first name and last name (only alphabet characters allowed)

if not first\_name.isalpha() or not last\_name.isalpha():

messagebox.showerror("Error", "Please enter valid alphabet characters for First Name and Last Name.")

return

# Validate phone number (only numeric characters allowed, length between 10 and 11)

if not phone\_number.isdigit() or not (10 <= len(phone\_number) <= 11):

messagebox.showerror("Error", "Please enter a valid phone number with only numeric characters, between 10 and 11 digits.")

return

# Validate email address

if not re.match(r"[^@]+@[^@]+\.[^@]+", email):

messagebox.showerror("Error", "Please enter a valid email address.")

return

# Validate card number

if not card\_number.isdigit() or not (16 <= len(card\_number) <= 17):

messagebox.showerror("Error", "Please enter a valid 16 digit card number.")

return

# Validate card expire

if "/" not in card\_expire or not (5 <= len(card\_expire) <= 6):

messagebox.showerror("Error", "Please enter a valid expiration ex: 05/28.")

return

# Validate card security

if not (3 <= len(card\_security) <= 5):

messagebox.showerror("Error", "Please enter a valid security code ex: 123.")

return

# Call the callback function to show payment confirmation

self.show\_payment\_confirmation\_callback(first\_name, last\_name, phone\_number, email)

def destroy\_current\_menu(self):

for widget in self.root.winfo\_children():

widget.destroy()

self.root.destroy()

def reset(self):

self.destroy\_current\_menu()

# Create a new instance of the RestaurantApp class

new\_app = RestaurantApp(self.root)

class RestaurantApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Restaurant Menu Viewer")

self.current\_menu = None

self.order\_total = 0.0

self.order\_items = [] # To store the items added to the order

self.total\_label = tk.Label(root, text="Total: $0.00", font=('Helvetica', 14))

self.show\_welcome()

def show\_welcome(self):

if self.current\_menu:

self.current\_menu.pack\_forget()

self.welcome\_page = WelcomePage(self.root, self.show\_menu)

def show\_menu(self, existing\_order\_items=None, existing\_order\_total=None):

if self.current\_menu:

self.current\_menu.pack\_forget()

if existing\_order\_items is not None:

self.order\_items = existing\_order\_items

if existing\_order\_total is not None:

self.order\_total = existing\_order\_total

if datetime.now().hour < 12:

menu\_title = "Breakfast"

items = [

{

"name": "Eggs",

"price": 3.99,

"description": "Scrambled, fried, or poached eggs"

},

{

"name": "Bacon",

"price": 4.99,

"description": "Crispy strips of bacon"

},

{

"name": "Toast",

"price": 2.99,

"description": "White or whole wheat, buttered"

},

# Add more breakfast items here

]

elif datetime.now().hour < 18:

menu\_title = "Lunch"

items = [

{

"name": "Burger",

"price": 6.99,

"description": "Juicy beef patty with toppings"

},

{

"name": "Salad",

"price": 5.99,

"description": "Fresh greens with dressing"

},

{

"name": "Sandwich",

"price": 4.99,

"description": "Classic sandwich varieties"

},

# Add more lunch items here

]

else:

menu\_title = "Dinner"

items = [

{

"name": "Steak",

"price": 12.99,

"description": "Grilled to perfection"

},

{

"name": "Fish",

"price": 10.99,

"description": "Freshly caught, seasoned and baked"

},

{

"name": "Chicken",

"price": 9.99,

"description": "Tender and succulent"

},

# Add more dinner items here

]

self.current\_menu = tk.Frame(self.root)

self.current\_menu.pack(pady=20)

title\_label = tk.Label(self.current\_menu, text=f"{menu\_title} Menu", font=('Helvetica', 16))

title\_label.pack()

# Display drink image

drink\_image\_path = os.path.join("drink.png")

if os.path.exists(drink\_image\_path):

drink\_image = tk.PhotoImage(file=drink\_image\_path)

drink\_label = tk.Label(self.current\_menu, image=drink\_image)

drink\_label.image = drink\_image # Keep a reference to avoid garbage collection

drink\_label.pack(pady=20)

for item\_info in items:

item\_label = tk.Label(self.current\_menu, text=f"{item\_info['name']} - ${item\_info['price']:.2f}\n{item\_info['description']}", font=('Helvetica', 12))

item\_label.pack()

# Add "Add to Order" button for each item

add\_to\_order\_button = tk.Button(self.current\_menu, text="Add to Order", command=lambda item\_info=item\_info: self.add\_to\_order(item\_info))

add\_to\_order\_button.pack()

# Display total label

self.total\_label.pack()

# Add "Send Order" button

send\_order\_button = tk.Button(self.current\_menu, text="Send Order", command=self.send\_order)

send\_order\_button.pack()

clear\_order\_button = tk.Button(self.current\_menu, text="Clear Order", command=self.clear\_order)

clear\_order\_button.pack()

#back button

back\_button = BackButton(self.current\_menu, self.show\_welcome)

# If there's an existing order, add it to the current order

if existing\_order\_items and existing\_order\_total:

self.order\_items.extend(existing\_order\_items)

self.order\_total += existing\_order\_total

# Update total label

self.total\_label.config(text=f"Total: ${self.order\_total:.2f}")

def add\_to\_order(self, item\_info):

print(f"{item\_info['name']} added to the order!")

self.order\_items.append(item\_info)

self.order\_total += item\_info['price']

# Update total label

self.total\_label.config(text=f"Total: ${self.order\_total:.2f}")

def clear\_order(self):

print("Cleared the order!")

self.order\_items.clear()

self.order\_total = 0

# Update total label

self.total\_label.config(text=f"Total: ${self.order\_total:.2f}")

def send\_order(self):

confirmation = messagebox.askquestion("Send Order", "Are you sure you want to send the order?")

if confirmation == 'yes':

print("Order Sent!")

# Display order summary page

self.show\_order\_summary()

else:

# Go back to the menu

self.show\_menu()

def show\_order\_summary(self):

# Hide the current menu

self.destroy\_current\_menu()

# Create and display the order summary page

order\_summary\_page = OrderSummaryPage(

self.root,

self.order\_items,

self.order\_total,

self.show\_payment\_page,

self.reset\_app)

self.current\_menu = order\_summary\_page

def show\_payment\_page(self):

# Display the payment page

payment\_window = tk.Toplevel(self.root)

payment\_page = PaymentPage(

payment\_window,

self.show\_payment\_confirmation,

self.reset\_app)

self.current\_menu = payment\_page

def destroy\_current\_menu(self):

if self.current\_menu:

self.current\_menu.destroy()

def show\_payment\_confirmation(self, first\_name, last\_name, phone\_number, email):

# Placeholder for payment processing

# Print the entered information (replace this with actual payment processing)

print("Payment Information:")

print(f"First Name: {first\_name}")

print(f"Last Name: {last\_name}")

print(f"Phone Number: {phone\_number}")

print(f"Email Address: {email}")

# Display a message indicating that the payment was successful

messagebox.showinfo("Payment", "Payment successful! Thank you for your order.")

# Reset the application

self.reset\_app()

root = tk.Tk()

app = RestaurantApp(root)

root.mainloop()

def reset\_app(self):

self.root.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = RestaurantApp(root)

root.mainloop()