Python Project Report

Simple Restaurant Manager

01286121 Computer Programming

Software Engineering Program

By

66011149 Phatthadon Sornplang

Python Project

Simple Restaurant Manager

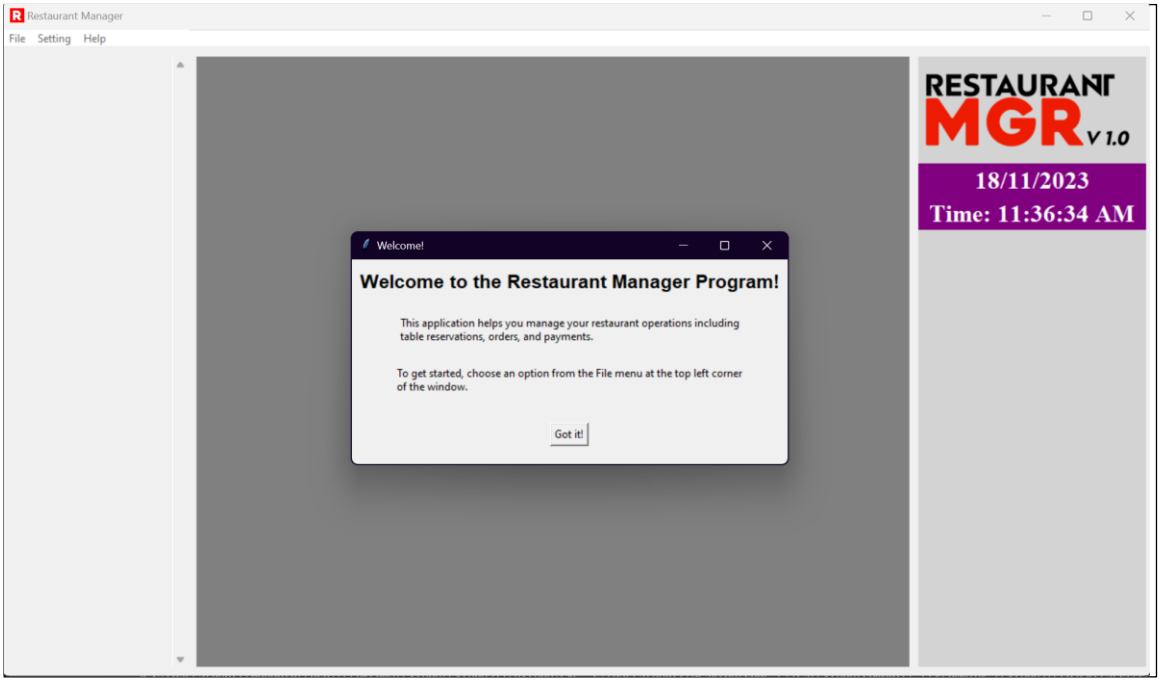
Project Introduction (one page)

A restaurant requires smooth operations to ensure customer satisfaction and retention. A digital system, like a Restaurant Management Application, can help improve many processes that are traditionally tedious and prone to human error. This proposal outlines the development of a Restaurant Management Application using Python's standard GUI (Graphical User Interface) library “TkInter” and Python’s pickle database.

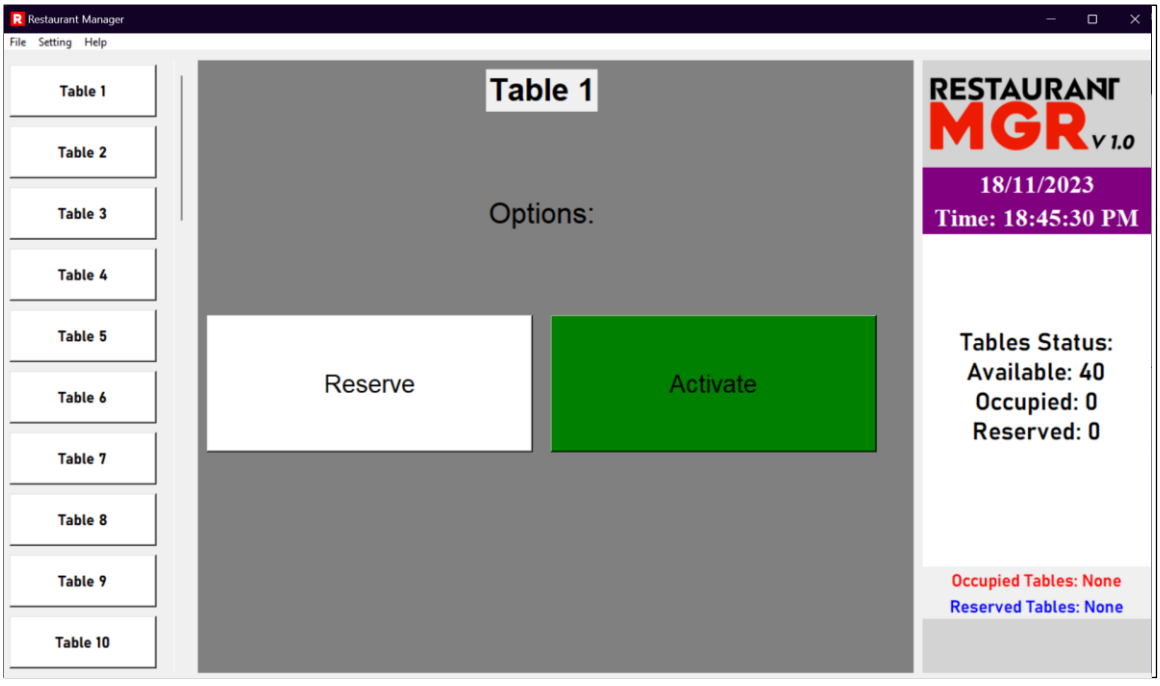
The restaurant managing program is designed to be user-friendly and straightforward, making the daily tasks of a restaurant much more streamline. From the moment you open the application, it's clear where to go and what to do. Whether you're checking the table reservations, updating the menu or calculating the bill, everything is just a click away. You can even create and save your restaurant preset to fit your restaurant operations. (All data will be saved with .pkl file) With this tool in hand, managing a restaurant becomes less about juggling tasks and more about delivering great food and service.

In conclusion, the development and integration of a user-centric restaurant managing program can significantly streamline daily operations and enhance overall efficiency.

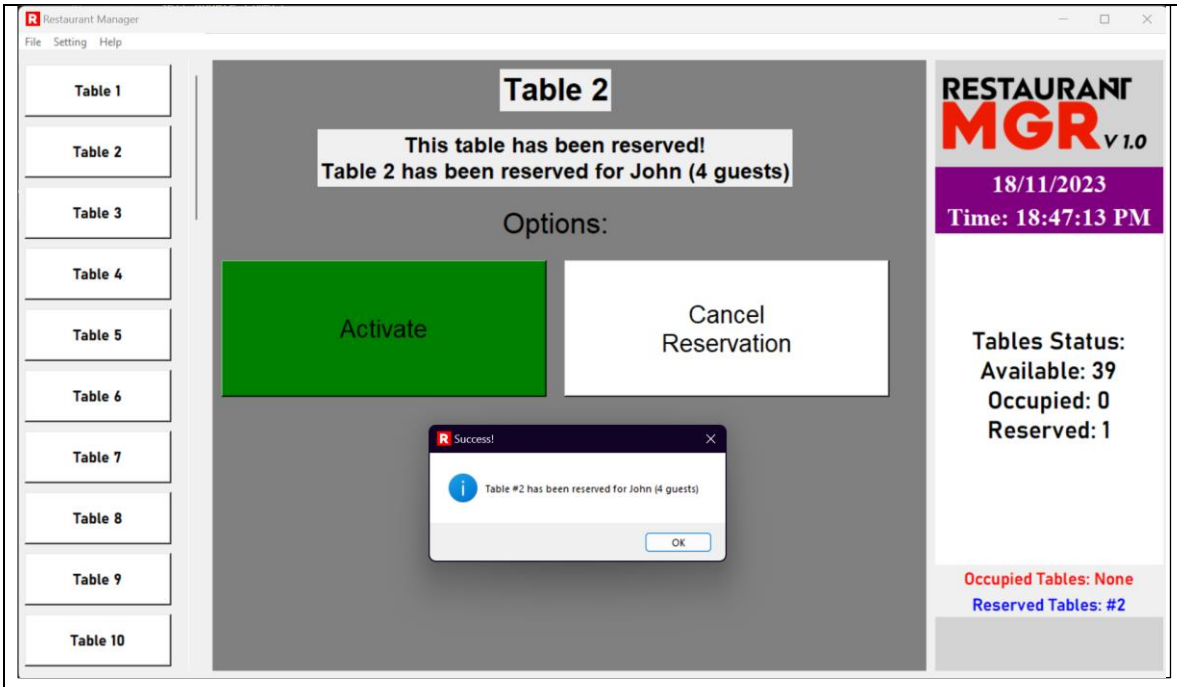
Screen Captures (as many screens as necessary)



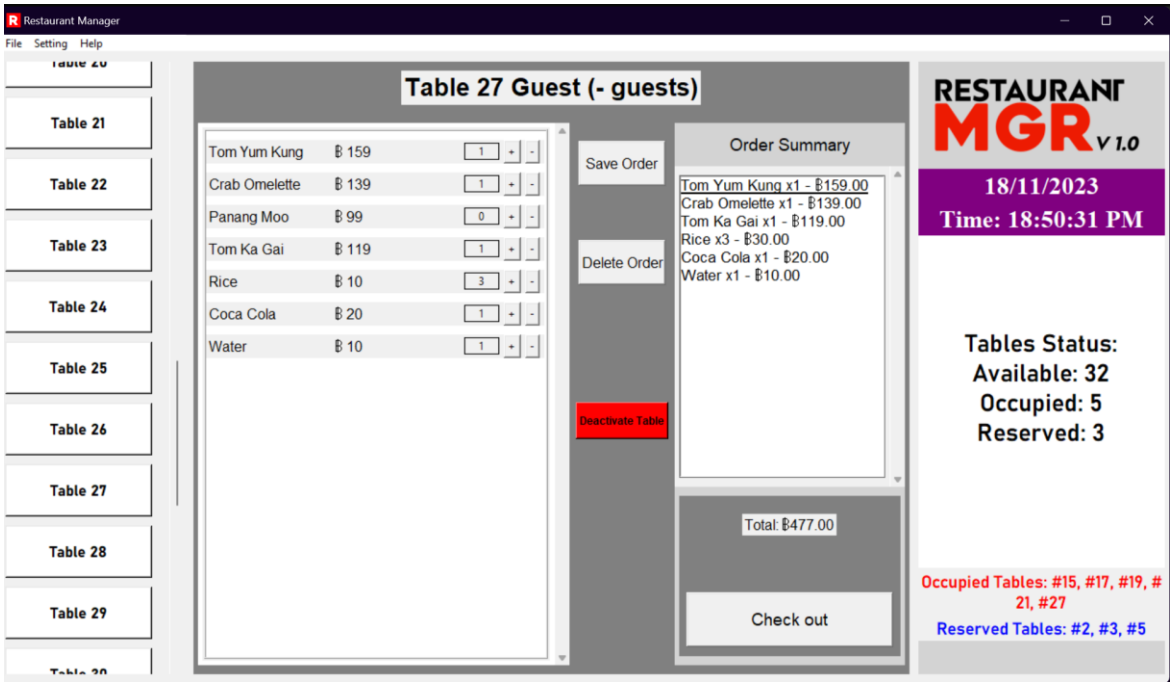
Screen 1



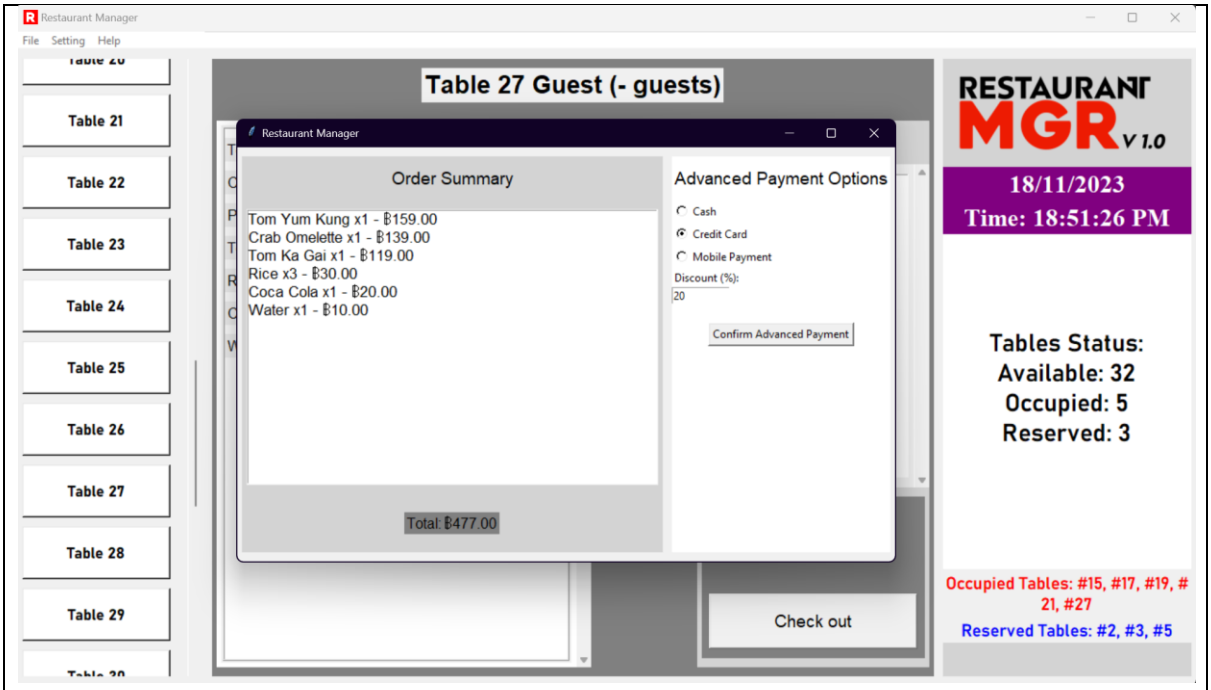
Screen 2



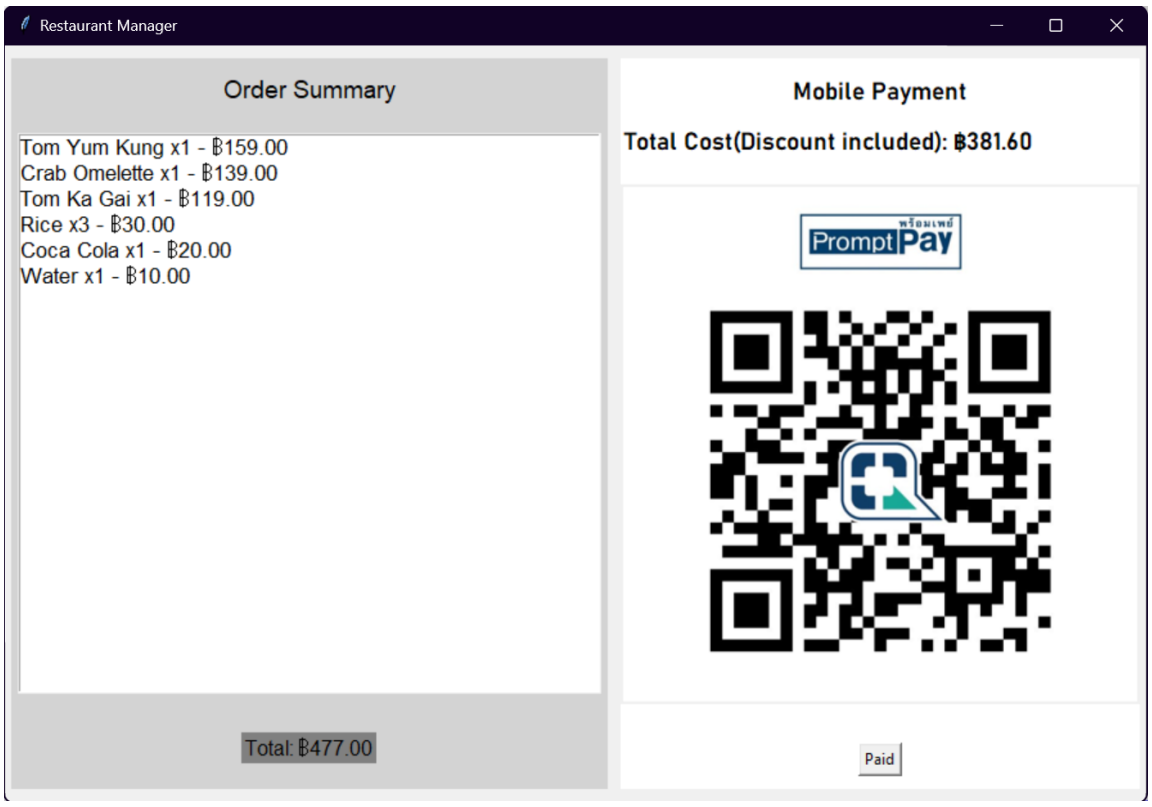
Screen 3



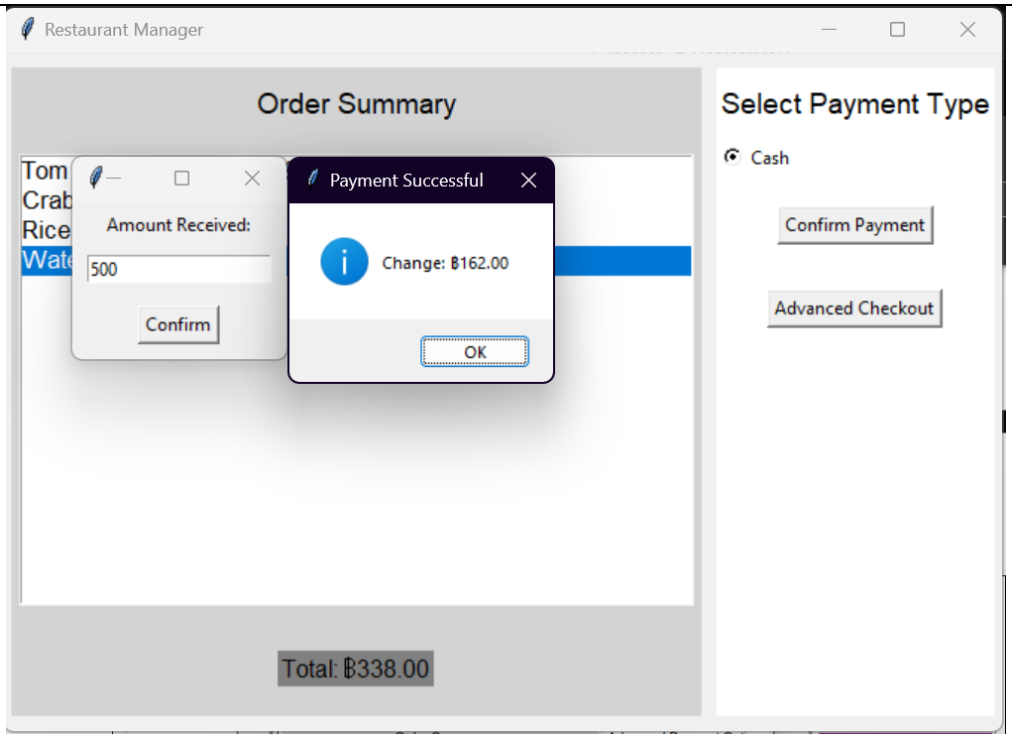
Screen 4



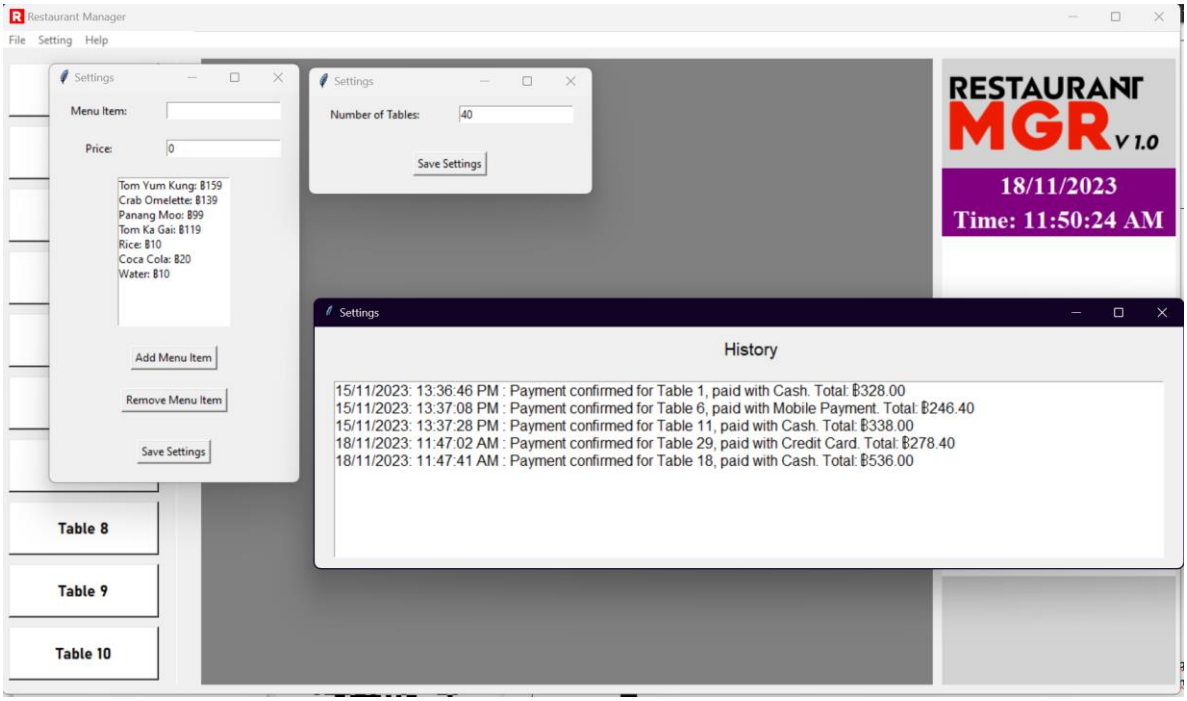
Screen 5



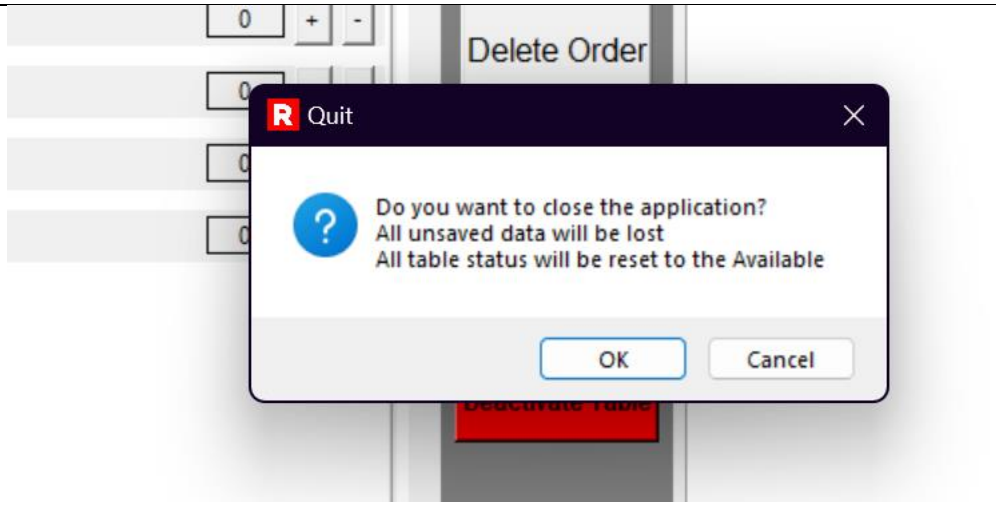
Screen 6



Screen 7



Screen 8


Delete Order
Quit
Do you want to close the application?
All unsaved data will be lost
All table status will be reset to the Available
OK
Cancel

Screen 9

Python Source codes

```
from tkinter import *
from tkinter import Tk, Label,messagebox,filedialog
from PIL import Image, ImageTk
import time
import datetime
import pickle


class Mainui(object):
    def __init__(self, window):
        self.window = window
        img = PhotoImage(file='C:/Users/phatt/Desktop/Code
Files/Python/Computer Prgramming (Python)/Projekt/images/Rlogo.png')
        self.window.iconphoto(False, img)
        self.window.title("Restaurant Manager")
        self.window.geometry('1280x720')

        self.num_tables = 0
        self.menu = {}
        self.table_vars = {}
        self.reservationlist = {i: "Guest" for i in range(1, self.num_tables +
1)}
        self.reservationguest = {i: '-' for i in range(1, self.num_tables +
1)}
        self.current_file = None
        self.table_status = {i: "dormant" for i in range(1, self.num_tables +
1)}
        self.table_orders = {}
        self.payment_his = ["This is the start of the History"]
        self.revenue = 0

        menu_callbacks = {
            "new_data": self.new_data,
            "save_data_as": self.save_data_as,
            "load_data": self.load_data,
            "open_setting": self.open_setting,
            "about": self.about
        }
        self.menuUI = MenuUI(self.window, menu_callbacks)

        self.create_tables()

        self. middle_frame = Frame(window,height=720, bg="gray")
        self.middle_frame.pack_propagate(0)
        self.middle_frame.pack(side=LEFT, fill=BOTH,expand=True, padx=5,
pady=10)
```

```
        self.right_frame = Frame(window, width=280,height=720, bg="lightgray")
        self.middle_frame.pack_propagate(0)
        self.right_frame.pack(side= RIGHT, fill=Y, expand=False, padx=5,
pady=10)
        self.display_image()
        self.display_datetime()
        self.show_intro_popup()
        self.window.protocol("WM_DELETE_WINDOW", self.on_close)


    def on_close(self):
        if messagebox.askokcancel("Quit", "Do you want to close the
application?\nAll unsaved data will be lost\nAll table status will be reset to
the Available"):
            self.window.destroy()


    def show_intro_popup(self):
        intro_window = Toplevel(self.window)
        intro_window.title("Welcome!")

        Label(intro_window, text=" Welcome to the Restaurant Manager Program!
", font=("Arial", 16, "bold")).pack(pady=10)
        Label(intro_window, text="This application helps you manage your
restaurant operations including table reservations, orders, and payments.",
wraplength=400, justify="left").pack(pady=10)
        Label(intro_window, text="To get started, choose an option from the
File menu at the top left corner of the window.", wraplength=400,
justify="left").pack(pady=10)

        Button(intro_window, text="Got it!",
command=intro_window.destroy).pack(pady=20)


    def about(self):
        about_window = Toplevel(self.window)
        about_window.title("About")
        Label(about_window, text="Restaurant Manager\nVersion 1.0\nBy
WDTX1402\n\nContact: 66011149@kmitl.ac.th").pack(pady=20, padx=20)
        Button(about_window, text="Close",
command=about_window.destroy).pack(pady=20)


    def display_image(self):
        image_path = 'C:/Users/phatt/Desktop/Code Files/Python/Computer
Prgramming (Python)/Projekt/images/logo.png'
        original_image = Image.open(image_path)

        desired_width = 250
```

```
        aspect_ratio = original_image.height / original_image.width
        desired_height = int(desired_width * aspect_ratio)
        resized_image = original_image.resize((desired_width, desired_height))

        img = ImageTk.PhotoImage(resized_image)

        img_label = Label(self.right_frame, image=img ,bg= 'lightgray')
        img_label.image = img
        img_label.grid(row=0, pady=10)

    def display_datetime(self):
        currentDate =
f"{str(datetime.datetime.now().day).zfill(2)}/{str(datetime.datetime.now().mon
th).zfill(2)}/{str(datetime.datetime.now().year)}"
        self.date_label = Label(self.right_frame, text= currentDate,
font=('Times', 20, 'bold'), background='purple', foreground='white')
        self.date_label.grid(row=1,sticky = NSEW)
        self.time_label = Label(self.right_frame, font=('Times', 20, 'bold'),
background='purple', foreground='white')
        self.time_label.grid(row=2,sticky = NSEW)
        self.update_time()

    def display_status(self):
        status_counts = {"Available": 0, "Occupied": 0, "Reserved": 0}
        status_tables = {"Occupied": [], "Reserved": []}

        for table_num, status in self.table_status.items():
            if status == "dormant":
                status_counts["Available"] += 1
            elif status == "active":
                status_counts["Occupied"] += 1
                status_tables["Occupied"].append(table_num)
            elif status == "reserved":
                status_counts["Reserved"] += 1
                status_tables["Reserved"].append(table_num)

            def format_table_list(tables, max_tables_per_line=15):
                tables_str = ", #".join(str(table) for table in tables)
                return '\n'.join([tables_str[i:i+max_tables_per_line] for i in
range(0, len(tables_str), max_tables_per_line)])

        status_str = "Tables Status:\n"
        for status, count in status_counts.items():
            status_str += f"{status}: {count}\n"

        if not hasattr(self, 'status_label'):
            self.status_label = Label(self.right_frame, font=('Bahnschrift
SemiBold', 20), background='white', foreground='black', pady=100)
```

```
            self.status_label.grid(row=3, sticky=NSEW)
        self.status_label.config(text=status_str)


        if not hasattr(self, 'occupied_tables_label'):
            self.occupied_tables_label = Label(self.right_frame,
font=('Bahnschrift SemiBold', 14), foreground='red')
            self.occupied_tables_label.grid(row=4, sticky=NSEW)

        if not hasattr(self, 'reserved_tables_label'):
            self.reserved_tables_label = Label(self.right_frame,
font=('Bahnschrift SemiBold', 14), foreground='blue')
            self.reserved_tables_label.grid(row=5, sticky=NSEW)

        occupied_tables_str = format_table_list(status_tables["Occupied"])
        reserved_tables_str = format_table_list(status_tables["Reserved"])

        self.occupied_tables_label.config(text=f"Occupied Tables:
#{occupied_tables_str}" if status_tables["Occupied"] else "Occupied Tables:
None")
        self.reserved_tables_label.config(text=f"Reserved Tables:
#{reserved_tables_str}" if status_tables["Reserved"] else "Reserved Tables:
None")


    def update_time(self):
        current_time = time.strftime('%H:%M:%S %p')
        self.time_label.config(text=f" Time: {current_time} ")
        self.window.after(1000, self.update_time)

    def generate_tables(self):
        for widget in self.inner_tables_frame.winfo_children():
            widget.destroy()

        for i in range(1, self.num_tables + 1):
            table = Button(self.inner_tables_frame, text=f"Table {i}",
font=('Bahnschrift Bold', 13 ), bg="white", width=17, height=2,
                           command=lambda table_num=i:
self.toggle_table_display(table_num))
            table.pack(pady=5)

        self.table_status = {i: "dormant" for i in range(1, self.num_tables +
1)}
        self.display_status()
        self.reservationlist = {i: "Guest" for i in range(1, self.num_tables +
1)}
        print(self.reservationlist)
        self.reservationguest = {i: '-' for i in range(1, self.num_tables +
1)}
```

```
    def toggle_table_display(self, table_num):
        if self.table_status[table_num] == "dormant":
            self.display_dormant_table(table_num)
        elif self.table_status[table_num] == "active":
            self.display_activated_table(table_num)
        elif self.table_status[table_num] == "reserved":
            self.display_reserved_table(table_num)


    def create_tables(self):
        tables_frame = Frame(self.window, width=180, height=720,
bg="lightgray")
        tables_frame.pack(side=LEFT, fill=Y, padx=5, pady=10)

        scrollbar = Scrollbar(tables_frame, orient=VERTICAL)
        scrollbar.pack(side=RIGHT, fill=Y)

        canvas = Canvas(tables_frame, width=180, yscrollcommand=scrollbar.set)
        canvas.pack(side=LEFT, fill=BOTH, expand=True)
        scrollbar.config(command=canvas.yview)
        self.inner_tables_frame = Frame(canvas)
        canvas.create_window((0, 0), window=self.inner_tables_frame,
anchor="nw")
        self.inner_tables_frame.bind("<Configure>", lambda e:
canvas.configure(scrollregion=canvas.bbox("all")))


    def display_dormant_table(self, table_number):
        self.table_status[int(table_number)] = "dormant"
        self.display_status()
        for widget in self.middle_frame.winfo_children():
            widget.destroy()

        table_label = Label(self.middle_frame, text=f"Table {table_number}",
font=('Arial', 26, 'bold'))
        table_label.grid(row=0, column=0, columnspan=3, pady=10, padx=10)

        options_label = Label(self.middle_frame, text="Options:",
font=('Arial', 24), bg="gray",height= 5)
        options_label.grid(row=1, column=0, columnspan=3, pady=10, padx=10)

        reserve_btn = Button(self.middle_frame, text="Reserve", font=('Arial',
20), bg="white",width= 22 ,height= 4,
                             command=lambda: self.reservation(table_number))
        reserve_btn.grid(row=2, column=0, pady=10, padx=10)

        activ_btn = Button(self.middle_frame, text="Activate", font=('Arial',
20), bg="green",width= 22 ,height= 4,
```

```
                        command=lambda: self.activate_table(table_number))
        activ_btn.grid(row=2, column=2, pady=10, padx=10)

        # clreserve_btn = Button(self.middle_frame, text=f"
Cancel\nReservation ", font=('Arial', 20), bg="white", width=13, height=3)
        # clreserve_btn.grid(row = 4, column = 1, padx= 10)


    def reservation(self,table_num):

        self.reserve_window = Toplevel(self.window)
        self.reserve_window.title(f"Reservation Info Table #{table_num}")

        Label(self.reserve_window, text="Enter customer's name").pack(padx=10,
pady=10)
        name_entry = Entry(self.reserve_window, bd=5, width=35)
        name_entry.pack(pady=10)
        Label(self.reserve_window, text="Enter number of
guests").pack(padx=10, pady=10)
        seats = Entry(self.reserve_window, bd=5, width=5)
        seats.pack(pady=10)


        Button(self.reserve_window, text="save", command=lambda:
self.save_reservation(table_num, name_entry,seats)).pack(pady=20)
        Button(self.reserve_window, text="Close",
command=self.reserve_window.destroy).pack(pady=20)


        # reserve_window.update_idletasks()
        # width = reserve_window.winfo_width()
        # height = reserve_window.winfo_height()
        # x = (reserve_window.winfo_screenwidth() // 2) - (width // 2)
        # y = (reserve_window.winfo_screenheight() // 2) - (height // 2)
        # reserve_window.geometry('{}x{}+{}+{}'.format(width, height, x, y))

    def save_reservation(self, table_num, name_entry, guests):

        customer_name = name_entry.get()
        guestsnum = guests.get()

        if customer_name.strip() and guestsnum.strip():
            self.reservationlist[table_num] = customer_name
            self.reservationguest[table_num] = guestsnum
            print(self.reservationlist)
            self.table_status[table_num] = 'reserved'
            self.display_reserved_table(table_num)
            messagebox.showinfo("Success!", f"Table #{table_num} has been
reserved for {customer_name} ({guestsnum} guests)")
```

```
            self.reserve_window.destroy()
        else:
            messagebox.showerror("Error", "All box must be filled!")

    def display_reserved_table(self, table_number):
        self.table_status[int(table_number)] = "reserved"
        self.display_status()
        for widget in self.middle_frame.winfo_children():
            widget.destroy()

        table_label = Label(self.middle_frame, text=f"Table {table_number}",
font=('Arial', 26, 'bold'))
        table_label.grid(row=0, column=0, columnspan=2, pady=10, padx=10)

        infobox = Label(self.middle_frame,
                        text=f"This table has been reserved!\nTable
{table_number} has been reserved for {self.reservationlist[table_number]}
({self.reservationguest[table_number]} guests)",
                        font=('Arial', 18, 'bold'))
        infobox.grid(row=1, column=0, columnspan=2, pady=10, padx=10)

        options_label = Label(self.middle_frame, text="Options:",
font=('Arial', 24), bg="gray")
        options_label.grid(row=2, column=0, columnspan=2, pady=10, padx=10)

        activ_btn = Button(self.middle_frame, text="Activate", font=('Arial',
20), bg="green",width= 22 ,height= 4,
                        command=lambda: self.activate_table(table_number))
        activ_btn.grid(row=3, column=0, pady=10, padx=10)

        clreserve_btn = Button(self.middle_frame, text="Cancel\nReservation",
font=('Arial', 20), bg="white",width= 22 ,height= 4,
                               command=lambda:
self.cancel_reservation(table_number))
        clreserve_btn.grid(row=3, column=1, pady=10, padx=10)

        # for i in range(1, table_number+ 1):
        #     reserve_btn = Button(self.middle_frame, text=f" Reserve ",
font=('Arial', 20), bg="white", width=13, height=3,
        #                          command=lambda i=i: self.reservation(i))
        #     reserve_btn.grid(row = 4, column = 0, padx= 10)

    def activate_table(self, table_num):
        self.display_activated_table(table_num)
        self.table_status[table_num] = 'active'
        self.display_status()

    def cancel_reservation(self, table_num):
```

```
        self.display_dormant_table(table_num)
        self.reservationlist[table_num] = 'Guest'
        self.reservationguest[table_num] = '-'


    def display_activated_table(self, table_number):
        print(f"Received table number: {table_number}
{self.reservationlist[table_number]} ({self.reservationguest[table_number]})")

        for widget in self.middle_frame.winfo_children():
            widget.destroy()

        table_label = Label(self.middle_frame, text=f"Table {table_number}
{self.reservationlist[table_number]} ({self.reservationguest[table_number]}
guests)",
                            font=('Arial', 20, 'bold'), )
        table_label.pack(pady=10)

        #create the order frame
        order_frame = Frame(self.middle_frame, bg='white')
        order_frame.pack(side=LEFT, fill=BOTH, expand=True, padx=5, pady=10)

        #scrollbar inside order_frame
        scrollbar = Scrollbar(order_frame, orient=VERTICAL)
        scrollbar.pack(side=RIGHT, fill=Y)

        #canvas inside order_frame
        self.menucanvas = Canvas(order_frame, yscrollcommand=scrollbar.set,
bg='white', bd=2, relief='ridge')
        self.menucanvas.pack(side=LEFT, fill=BOTH, expand=True, padx=5,
pady=5)
        scrollbar.config(command=self.menucanvas.yview)

        #frame inside Canvas
        self.menu_frame = Frame(self.menucanvas, bg='white', pady=5)
        self.menucanvas.create_window((0, 0), window=self.menu_frame,
anchor="nw")
        self.menu_frame.bind("<Configure>", lambda e:
self.menucanvas.configure(scrollregion=self.menucanvas.bbox("all")))


        self.summary_frame = Frame(self.middle_frame, bg='lightgray')
        self.summary_frame.pack(side=RIGHT, fill=BOTH, padx=5, pady=10,
expand=True)

        #Packing the Order Summary label at the top of the summary_frame
        order_summary_label = Label(self.summary_frame, text="Order Summary",
font=('Arial', 14),bg='lightgray')
        order_summary_label.pack(pady=10)
```

```
        #Create a frame to hold the listbox and its scrollbar
        self.summarylist_frame = Frame(self.summary_frame)
        self.summarylist_frame.pack(fill=BOTH, expand=True)

        #create the scrollbar inside summarylist_frame
        order_listbox_scrollbar = Scrollbar(self.summarylist_frame,
orient=VERTICAL)
        order_listbox_scrollbar.pack(side=RIGHT, fill=Y)

        #create the listbox associated with the scrollbar inside
summarylist_frame
        self.order_listbox = Listbox(self.summarylist_frame, font=('Arial',
13), height=15, width=25, yscrollcommand=order_listbox_scrollbar.set)
        self.order_listbox.pack(side=LEFT, fill=BOTH, expand=True, padx=5,
pady=10)

        #connect the scrollbar to the listbox
        order_listbox_scrollbar.config(command=self.order_listbox.yview)


        self.checkout_frame = Frame(self.summary_frame, bg = 'gray')
        self.checkout_frame.pack(side=BOTTOM, fill=BOTH, padx=5, pady=10,
expand=True)

        self.total_cost_var = StringVar(value="Total: ฿0.00")
        self.total_cost_label = Label(self.checkout_frame,
textvariable=self.total_cost_var, font=('Arial', 12))
        self.total_cost_label.pack(side = TOP,pady=20, anchor='s')

        if not hasattr(self, 'table_orders'):
            self.table_orders = {}

        if table_number not in self.table_orders:
            self.table_orders[table_number] = {}

        #Clear the listbox for order summary:
        self.order_listbox.delete(0, END)

        for item, qty in self.table_orders[table_number].items():
            if qty > 0:
                price = self.menu[item]
                self.order_listbox.insert(END, f"{item} x{qty} - ฿{price *
qty:.2f}")

        for menu_item , menu_price in self.menu.items():
            item_frame = Frame(self.menu_frame)
            item_frame.pack(fill=X, pady=5)
```

```
            menu_label = Label(item_frame, text=menu_item,font=('Arial', 12),
width=15, anchor=W)
            menu_label.pack(side=LEFT)

            price_label = Label(item_frame, text=f"฿
{menu_price}",font=('Arial', 12), width=15, anchor=W)
            price_label.pack(side=LEFT)

            qty_var =
IntVar(value=self.table_orders[table_number].get(menu_item, 0))
            qty_label = Label(item_frame, textvariable=qty_var, width=5,
relief="solid", bd=1)
            qty_label.pack(side=LEFT, padx=5)

            add_btn = Button(item_frame, text="+", command=lambda var=qty_var,
item=menu_item: self.update_order(var, item, table_number, 1))
            add_btn.pack(side=LEFT)

            sub_btn = Button(item_frame, text="-", command=lambda var=qty_var,
item=menu_item: self.update_order(var, item, table_number, -1))
            sub_btn.pack(side=LEFT, padx=5)

            self.table_vars[menu_item] = qty_var
            self.table_orders[table_number][menu_item] = qty_var.get()

            saved_qty = self.table_orders[table_number].get(menu_item, 0)
            qty_var.set(saved_qty)

        total_cost = self.compute_total_cost(table_number)
        self.total_cost_var.set(f"Total: ฿{total_cost:.2f}")

        save_btn = Button(self.middle_frame, text="Save Order",font=('Arial',
12), width=20, height=2, command=lambda: self.save_order(table_number))
        save_btn.pack(padx= 5,pady=30)

        delete_btn = Button(self.middle_frame, text="Delete
Order",font=('Arial', 12), width=20, height=2, command=lambda:
self.delete_order(table_number))
        delete_btn.pack(padx= 5,pady=30)

        deactiv_btn = Button(self.middle_frame, text="Deactivate Table",
font=('Arial', 9, 'bold'), bg='red', width=20, height=2,
                     command=lambda: self.deactivate_table(table_number))
        deactiv_btn.pack(padx=2, pady=100)

        checkout_btn = Button(self.checkout_frame, text="Check
out",font=('Arial', 14), width=20, height=2, command=lambda:
self.check_order_empty(table_number))
```

```
        checkout_btn.pack(side = BOTTOM , pady= 10)

    def check_order_empty(self,table_number):
        if self.order_listbox.size() == 0:
            messagebox.showerror("Error", "This table has no order")
        else:
            self.open_checkout(table_number)


    def update_order(self, var, item, table_number, add):
        new_value = max(0, var.get() + add)
        var.set(new_value)
        self.table_orders[table_number][item] = new_value


    def save_order(self, table_number):
        self.order_listbox.delete(0, END)
        total_cost = 0.0

        for menu_item, qty_var in self.table_vars.items():
            try:
                qty = int(qty_var.get())
                self.table_orders[table_number][menu_item] = qty
                item_cost = self.menu.get(menu_item, 0)
                menu_item_total = qty * item_cost
                total_cost += menu_item_total

                if qty > 0:
                    self.order_listbox.insert(END, f"{menu_item} x{qty} - ฿
{menu_item_total:.2f}")

            except ValueError:
                self.table_orders[table_number][menu_item] = 0

        self.total_cost_var.set(f"Total: ฿{total_cost:.2f}")


    def delete_order(self, table_number):
        selected_index = self.order_listbox.curselection()
        if selected_index:
            selected_item = self.order_listbox.get(selected_index)

            menu_name = selected_item.split(" x")[0]

            if table_number in self.table_orders and menu_name in
self.table_orders[table_number]:
                del self.table_orders[table_number][menu_name]

            self.order_listbox.delete(selected_index)
```

```
            total_cost = self.compute_total_cost(table_number)
            self.total_cost_var.set(f"Total: ฿{total_cost:.2f}")


    def compute_total_cost(self, table_num):
        total = 0.0
        for menu_item, qty in self.table_orders[table_num].items():
            item_cost = self.menu.get(menu_item, 0)
            total += item_cost * qty
        return total

    def deactivate_table(self, table_num):
        self.display_dormant_table(table_num)
        self.table_status[table_num] = 'dormant'
        self.reservationlist[table_num] = 'Guest'
        self.reservationguest[table_num] = '-'
        self.table_orders.pop(table_num, None)


    def open_setting(self, setting_type):
        setting_window = Toplevel(self.window)
        Setting(setting_window, self, setting_type, self.revenue)

    def open_checkout(self, table_num):
        checkout_window = Toplevel(self.window)
        #print(f"open debug {self.revenue}")
        Checkout(self.menu, self.table_orders, self.payment_his,self.revenue ,
checkout_window, table_num,self)

    def save_data_as(self):
        file_name = filedialog.asksaveasfilename(defaultextension=".pkl",
filetypes=[("Pickle files", "*.pkl")])
        if file_name:
            with open(file_name, 'wb') as f:
                pickle.dump(self.num_tables, f)
                pickle.dump(self.menu, f)
                pickle.dump(self.payment_his, f)
                pickle.dump(self.revenue, f)
            print(f"Data saved to {file_name}.")
            self.current_file = file_name

        pass


    def load_data(self):
        file_name = filedialog.askopenfilename(defaultextension=".pkl",
filetypes=[("Pickle files", "*.pkl")])
```

```
        if file_name:
            try:
                self.menu = {}
                for widget in self.middle_frame.winfo_children():
                    widget.destroy()

                with open(file_name, 'rb') as f:
                    self.num_tables = pickle.load(f)
                    self.menu = pickle.load(f)
                    self.payment_his = pickle.load(f)
                    self.revenue = pickle.load(f)
                self.generate_tables()
                print(f"Data loaded from {file_name}.")
                self.current_file = file_name

                return self.num_tables

            except (FileNotFoundError, EOFError, pickle.UnpicklingError) as e:
                messagebox.showerror('Error', 'Data could not be loaded')
                print(f"Error loading data: {e}")

        pass


    def new_data(self):
        for widget in self.middle_frame.winfo_children():
            widget.destroy()

        self.menu = {}
        self.num_tables = 0
        self.payment_his = []
        self.revenue = 0
        self.generate_tables()
        self.current_file = None
        print("Started a new file.")

        pass



class Checkout(object):
    def __init__(self, menu, table_orders, payment_his,revenue, window,
table_num,main):
        self.window = window
        self.menu = menu
        self.table_orders = table_orders
        self.payment_his = payment_his
        self.table_num = table_num
        self.revenue = revenue
```

```
        self.main = main


        self.setup_ui()
        self.display_orders()

    def setup_ui(self):
        self.summary_frame = Frame(self.window, bg='lightgray')
        self.summary_frame.pack(side=LEFT, fill=BOTH, expand=True, padx=5,
pady=10)

        Label(self.summary_frame, text="Order Summary", font=('Arial', 14),
bg='lightgray').pack(pady=10)

        self.order_listbox = Listbox(self.summary_frame, font=('Arial', 13),
height=15, width=50)
        self.order_listbox.pack(fill=BOTH, expand=True, padx=5, pady=10)

        self.total_cost_var = StringVar(value="Total: ฿0.00")
        Label(self.summary_frame, textvariable=self.total_cost_var,
font=('Arial', 12), bg='gray').pack(pady=20)

        self.payment_frame = Frame(self.window, bg='white')
        self.payment_frame.pack(side=RIGHT, fill=BOTH, padx=5, pady=10,
expand=True)

        Label(self.payment_frame, text="Select Payment Type", font=('Arial',
14), bg='white').pack(pady=10)

        self.payment_type = StringVar(value="cash")
        Radiobutton(self.payment_frame, text="Cash",
variable=self.payment_type, value="cash", bg='white').pack(anchor='w')

        Button(self.payment_frame, text="Confirm Payment",
command=self.confirm_payment).pack(pady=20)
        Button(self.payment_frame, text="Advanced Checkout",
command=self.open_advanced_checkout).pack(pady=10)


    def open_advanced_checkout(self):
        self.window.destroy()

        advanced_checkout_window = Toplevel()
        self.advanced_checkout = AdvancedCheckout(self.menu,
self.table_orders, self.payment_his, self.revenue, advanced_checkout_window,
self.table_num, self.main)
        self.advanced_checkout.setup_advanced_ui()
```

```
    def display_orders(self):
        self.order_listbox.delete(0, END)
        total_cost = 0

        orders = self.table_orders.get(self.table_num, {})

        for item, qty in orders.items():
            if qty > 0:
                price = self.menu.get(item, 0)
                cost = price * qty
                total_cost += cost
                self.order_listbox.insert(END, f"{item} x{qty} - ฿{cost:.2f}")

        self.total_cost_var.set(f"Total: ฿{total_cost:.2f}")


    def confirm_payment(self):
        chosen_payment_type = self.payment_type.get()
        if chosen_payment_type == "cash":
            self.cash_payment_popup()
        else:
            ()

    def cash_payment_popup(self):
        popup = Toplevel(self.window)
        popup.title("Cash Payment")

        Label(popup, text="Amount Received:").pack(padx=10, pady=5)
        amount_entry = Entry(popup)
        amount_entry.pack(padx=10, pady=5)

        Button(popup, text="Confirm", command=lambda:
self.process_cash_payment(amount_entry.get(), popup)).pack(pady=10)

    def process_cash_payment(self, amount_received, popup):
        try:
            amount_received = float(amount_received)
            total_cost = float(self.total_cost_var.get().split(": ฿")[1])
            if amount_received < total_cost:
                messagebox.showerror("Error", "Insufficient amount received!",
parent=popup)
                return

            change = amount_received - total_cost
            messagebox.showinfo("Payment Successful", f"Change: ฿
{change:.2f}", parent=popup)
```

```
            currentDate =
f"{str(datetime.datetime.now().day).zfill(2)}/{str(datetime.datetime.now().mon
th).zfill(2)}/{str(datetime.datetime.now().year)}"
            self.payment_his.append(f"{currentDate}: {time.strftime('%H:%M:%S
%p')} : Payment confirmed for Table {self.table_num}, paid with Cash. Total: ฿
{total_cost:.2f}")

            self.revenue += total_cost

            self.window.destroy()
            self.main.deactivate_table(self.table_num)
            popup.destroy()


        except ValueError:
            messagebox.showerror("Error", "Invalid input! Please enter a
numeric value.", parent=popup)




class AdvancedCheckout(Checkout):
    def __init__(self, menu, table_orders, payment_his, revenue, window,
table_num, main):
        super().__init__(menu, table_orders, payment_his, revenue, window,
table_num, main)

    def calculate_total_cost(self):
        total_cost = float(self.total_cost_var.get().split(": ฿")[1])
        discount = self.discount_var.get()

        if discount:
            total_cost -= total_cost * (discount / 100)

        return total_cost


    def setup_advanced_ui(self):
        for widget in self.payment_frame.winfo_children():
            widget.destroy()
        Label(self.payment_frame, text="Advanced Payment Options",
font=('Arial', 14), bg='white').pack(pady=10)

        self.payment_type.set("cash")
        Radiobutton(self.payment_frame, text="Cash",
variable=self.payment_type, value="cash", bg='white').pack(anchor='w')
        Radiobutton(self.payment_frame, text="Credit Card",
variable=self.payment_type, value="card", bg='white').pack(anchor='w')
```

```
        Radiobutton(self.payment_frame, text="Mobile Payment",
variable=self.payment_type, value="mobile", bg='white').pack(anchor='w')

        #self.split_bill_var = BooleanVar()
        #Checkbutton(self.payment_frame, text="Split Bill",
variable=self.split_bill_var, bg='white').pack(anchor='w')

        self.discount_var = DoubleVar()
        Label(self.payment_frame, text="Discount (%):",
bg='white').pack(anchor='w')
        Entry(self.payment_frame, textvariable=self.discount_var,
width=10).pack(anchor='w')

        confirm_btn = Button(self.payment_frame, text="Confirm Advanced
Payment", command=self.confirm_advanced_payment)
        confirm_btn.pack(pady=20)


    def setup_credit_card_ui(self):
        total_cost = self.calculate_total_cost()

        for widget in self.payment_frame.winfo_children():
            widget.destroy()

        Label(self.payment_frame, text="Credit Card Payment",
font=('Bahnschrift SemiBold', 14), bg='white').pack(pady=10)

        Label(self.payment_frame, text=f"Total Cost(Discount included): ฿
{total_cost:.2f}",font=('Bahnschrift SemiBold', 14),
bg='white').pack(anchor='w')


        paid_btn = Button(self.payment_frame, text="Paid",
command=self.credit_card_payment)
        paid_btn.pack(pady=20)


    def setup_mobile_payment_ui(self):
        total_cost = self.calculate_total_cost()

        for widget in self.payment_frame.winfo_children():
            widget.destroy()

        Label(self.payment_frame, text="Mobile Payment", font=('Bahnschrift
SemiBold', 14), bg='white').pack(pady=10)
```

```
        Label(self.payment_frame, text=f"Total Cost(Discount included): ฿
{total_cost:.2f}",font=('Bahnschrift SemiBold', 14),
bg='white').pack(anchor='w')


        qr_image = Image.open('C:/Users/phatt/Desktop/Code
Files/Python/Computer Prgramming (Python)/Projekt/images/qr.png')
        qr_image = qr_image.resize((400, 400))
        qr_image = ImageTk.PhotoImage(qr_image)
        qr_label = Label(self.payment_frame, image=qr_image)
        qr_label.image = qr_image
        qr_label.pack(pady=20)


        paid_btn = Button(self.payment_frame, text="Paid",
command=self.mobile_payment)
        paid_btn.pack(pady=10)


    def credit_card_payment(self):
        total_cost = self.calculate_total_cost()
        currentDate =
f"{str(datetime.datetime.now().day).zfill(2)}/{str(datetime.datetime.now().mon
th).zfill(2)}/{str(datetime.datetime.now().year)}"
        self.payment_his.append(f"{currentDate}: {time.strftime('%H:%M:%S
%p')} : Payment confirmed for Table {self.table_num}, paid with Credit Card.
Total: ฿{total_cost:.2f}")

        self.revenue += total_cost

        self.window.destroy()
        self.main.deactivate_table(self.table_num)
        messagebox.showinfo("Payment Confirmation", "Credit Card payment
confirmed.")


    def mobile_payment(self):
        total_cost = self.calculate_total_cost()
        currentDate =
f"{str(datetime.datetime.now().day).zfill(2)}/{str(datetime.datetime.now().mon
th).zfill(2)}/{str(datetime.datetime.now().year)}"
        self.payment_his.append(f"{currentDate}: {time.strftime('%H:%M:%S
%p')} : Payment confirmed for Table {self.table_num}, paid with Mobile
Payment. Total: ฿{total_cost:.2f}")

        self.revenue += total_cost

        self.window.destroy()
        self.main.deactivate_table(self.table_num)
```

```
        messagebox.showinfo("Payment Confirmation", "Mobile payment
confirmed.")


    def confirm_advanced_payment(self):
        chosen_payment_type = self.payment_type.get()
        total_cost = self.calculate_total_cost()

        if chosen_payment_type == "card":
            self.setup_credit_card_ui()
        elif chosen_payment_type == "mobile":
            self.setup_mobile_payment_ui()
        elif chosen_payment_type == "cash":
            self.setup_cash_ui(total_cost)

        payment_info = f"Payment confirmed for Table {self.table_num} with
{chosen_payment_type}. Total: ฿{total_cost:.2f}"
        print(payment_info)


    def setup_cash_ui(self,total_cost):
        total_cost = self.calculate_total_cost()

        for widget in self.payment_frame.winfo_children():
            widget.destroy()

        Label(self.payment_frame, text="Cash Payment", font=('Bahnschrift
SemiBold', 14), bg='white').pack(pady=10)

        Label(self.payment_frame, text=f"Total Cost(Discount included): ฿
{total_cost:.2f}",font=('Bahnschrift SemiBold', 14),
bg='white').pack(anchor='w')


        paid_btn = Button(self.payment_frame, text="Paid",
command=self.open_cash_payment_popup_with_discount(total_cost))
        paid_btn.pack(pady=20)


    def open_cash_payment_popup_with_discount(self, total_cost):
        self.cash_payment_popup()



class MenuUI(Menu):
    def __init__(self, master, callback_map):
        super().__init__(master)

        file_menu = Menu(self, tearoff=0)
```

```
        file_menu.add_command(label="New",
command=callback_map.get("new_data"))
        file_menu.add_command(label="Save",
command=callback_map.get("save_data_as"))
        file_menu.add_command(label="Load",
command=callback_map.get("load_data"))
        file_menu.add_command(label="Exit", command=master.quit)
        self.add_cascade(label="File", menu=file_menu)

        setting_menu = Menu(self, tearoff=0)
        setting_menu.add_command(label="Set Tables", command=lambda:
callback_map.get("open_setting")("tables"))
        setting_menu.add_command(label="Set Menus", command=lambda:
callback_map.get("open_setting")("menus"))
        setting_menu.add_command(label="View History", command=lambda:
callback_map.get("open_setting")("history"))
        self.add_cascade(label="Setting", menu=setting_menu)

        help_menu = Menu(self, tearoff=0)
        help_menu.add_command(label="About",
command=callback_map.get("about"))
        self.add_cascade(label="Help", menu=help_menu)

        master.config(menu=self)



class Setting(object):
    def __init__(self, master, mainapp, setting_type, revenue):
        self.master = master
        self.mainapp = mainapp
        self.revenue = revenue
        self.master.title("Settings")

        if setting_type == "tables":
            self.ui_table_settings()
        elif setting_type == "menus":
            self.ui_menu_settings()
        elif setting_type == "history":
            self.ui_history()


    def ui_table_settings(self):

        self.label_tables = Label(self.master, text="Number of Tables:")
        self.label_tables.grid(row=0, column=0, padx=20, pady=10)

        self.num_tables_var = IntVar(value=self.mainapp.num_tables)
```

```
        self.entry_tables = Entry(self.master,
textvariable=self.num_tables_var)
        self.entry_tables.grid(row=0, column=1, padx=20, pady=10)

        self.save_btn = Button(self.master, text="Save Settings",
command=self.save_setting)
        self.save_btn.grid(row=6, column=0, columnspan=2, padx=20, pady=20)

    def ui_menu_settings(self):

        self.label_menu = Label(self.master, text="Menu Item:")
        self.label_menu.grid(row=1, column=0, padx=20, pady=10)

        self.menu_name_var = StringVar()
        self.entry_menu = Entry(self.master, textvariable=self.menu_name_var)
        self.entry_menu.grid(row=1, column=1, padx=20, pady=10)

        self.label_price = Label(self.master, text="Price:")
        self.label_price.grid(row=2, column=0, padx=20, pady=10)

        self.menu_price_var = IntVar()
        self.entry_price = Entry(self.master,
textvariable=self.menu_price_var)
        self.entry_price.grid(row=2, column=1, padx=20, pady=10)

        self.menu_listbox = Listbox(self.master)
        for menu_item, price in self.mainapp.menu.items():
            self.menu_listbox.insert(END, f"{menu_item}: ฿{price}")
        self.menu_listbox.grid(row=3, column=0, columnspan=2, padx=20,
pady=10)

        self.add_btn = Button(self.master, text="Add Menu Item",
command=self.add_menu_item)
        self.remove_btn = Button(self.master, text="Remove Menu Item",
command=self.remove_menu_item)
        self.add_btn.grid(row=4, column=0, columnspan=2, padx=20, pady=10)
        self.remove_btn.grid(row=5, column=0, columnspan=2, padx=20, pady=10)

        self.save_btn = Button(self.master, text="Save Settings",
command=self.save_setting)
        self.save_btn.grid(row=6, column=0, columnspan=2, padx=20, pady=20)

    def ui_history(self):
        Label(self.master, text="History", font=('Arial', 14)).grid(row=0,
column=0, columnspan=2, padx=20, pady=10)

        history_frame = Frame(self.master)
        history_frame.grid(row=1, column=0, columnspan=2, padx=20, pady=10)
```

```
        scrollbar = Scrollbar(history_frame, orient=VERTICAL)
        scrollbar.pack(side=RIGHT, fill=Y)

        history_listbox = Listbox(history_frame, font=('Arial', 12),
height=10, width=100, yscrollcommand=scrollbar.set)
        history_listbox.pack(side=LEFT, fill=BOTH, expand=True)

        scrollbar.config(command=history_listbox.yview)

        for item in self.mainapp.payment_his:
            history_listbox.insert(END, item)


    def add_menu_item(self):
        menu_name = self.menu_name_var.get().strip()
        menu_price = self.menu_price_var.get()

        if menu_name:
            self.mainapp.menu[menu_name] = menu_price
            self.menu_listbox.insert(END, f"{menu_name}: ฿{menu_price}")
            self.menu_name_var.set('')
            self.menu_price_var.set(0.0)

    def remove_menu_item(self):
        selected_index = self.menu_listbox.curselection()
        if selected_index:
            selected_text = self.menu_listbox.get(selected_index)
            menu_name = selected_text.split(":")[0].strip()

            if menu_name in self.mainapp.menu:
                del self.mainapp.menu[menu_name]

            self.menu_listbox.delete(selected_index)

    def save_setting(self):
        if hasattr(self, 'num_tables_var'):
            self.mainapp.num_tables = self.num_tables_var.get()
            self.mainapp.generate_tables()
        self.master.destroy()


if __name__ == '__main__':
    window = Tk()
    app = Mainui(window)
    window.mainloop()
```