# BIS698

## Group1 Project- Pizza Billing System

Group Members:

* Anurag Gajula
* Goutham Madarapu
* Rajkumar Sriramula
* Sai Satwick Yalavarthi
* Komal Raj Nadella Venkata

Project Code:

Python –

# LOGIN  
class LoginPage:  
 def \_\_init\_\_(self):  
 self.main\_window = tk.Tk()  
 self.main\_window.attributes('-fullscreen', True)  
 self.main\_window.title("Pizza Billing System - Login")  
  
 img222 = Image.open("LOGIN.jpg")  
 screen\_width = self.main\_window.winfo\_screenwidth()  
 screen\_height = self.main\_window.winfo\_screenheight()  
 img222 = img222.resize((screen\_width, screen\_height))  
 self.BGG = ImageTk.PhotoImage(img222)  
 self.imageL = tk.Label(self.main\_window, image=self.BGG)  
 self.imageL.place(x=0, y=0)  
 # Main Label  
 self.store\_label = tk.Label(self.main\_window, text="The Charcoal Pizza Store",  
 font=("Times New Roman", 54, "bold"), bg="#161513",fg="white")  
 self.store\_label.place(relx=0.25, rely=0.01)  
  
 self.top\_frame = tk.Frame(self.main\_window, highlightthickness=2, highlightbackground="#161513", bg="#161513")  
 self.middle\_frame = tk.Frame(self.main\_window, highlightthickness=2, highlightbackground="#161513", bg="#161513")  
 self.middle111\_frame = tk.Frame(self.middle\_frame, bg="#161513", relief="flat")  
 self.middle11\_frame = tk.Frame(self.middle\_frame,bg="#161513",relief="flat")  
 self.bottom\_frame = tk.Frame(self.main\_window, highlightthickness=2, highlightbackground="#161513", bg="#161513")  
 self.BBottom\_frame = tk.Frame(self.main\_window, highlightthickness=2, highlightbackground="#161513", bg="#161513")  
 # create username label and entry  
 self.username\_label = tk.Label(self.top\_frame, text="Username:", font=("Arial", 15, "bold"),fg="white", bg="#161513")  
 self.username\_label.pack(side="left", pady=10)  
 self.username\_entry = tk.Entry(self.top\_frame, font=("Arial", 15, "bold"))  
 self.username\_entry.pack(side="left", pady=10)  
 # create password label and entry  
 self.password\_label = tk.Label(self.middle111\_frame, text="Password:", font=("Arial", 15, "bold"), bg="#161513",fg="white")  
 self.password\_label.pack(side="left", pady=10)  
 self.password\_entry = tk.Entry(self.middle111\_frame, show="\*", font=("Arial", 15, "bold"))  
 self.password\_entry.pack(side="left", pady=10)  
 # create forgot password label  
 self.forgot\_password\_label = tk.Label(self.middle11\_frame, text="Forgot Password?",  
 font=("Arial", 12, "underline"), fg="lightblue", bg="#161513", cursor="hand2")  
 self.forgot\_password\_label.pack(side="bottom", anchor="sw")  
 self.forgot\_password\_label.bind("<Button-1>", lambda event: self.forgot\_password())  
  
 #Admin  
 self.admin\_button = tk.Button(self.BBottom\_frame, text="AdminLogin", height=2, width=15,  
 font=("Arial", 15, "bold"), command=self.adminlogin,fg="#161513",bg="#faf0e6")  
 self.admin\_button.pack()  
  
  
 # create login button  
 self.login\_button = tk.Button(self.bottom\_frame, text="Login", height=2, width=15, bg="#faf0e6",  
 font=("Arial", 15, "bold"), command=self.login,fg="#161513")  
 self.login\_button.grid(row=1, column=0, padx=(0, 0)) # add extra padding to the right  
 # create signup button  
 self.signup\_button = tk.Button(self.bottom\_frame, text="Sign Up", height=2, width=15, bg="#faf0e6",  
 font=("Arial", 15, "bold"), command=self.signup,fg="#161513")  
 self.signup\_button.grid(row=1, column=1, padx=(0, 0)) # add extra padding to the left  
 # Packing the frmes  
 self.top\_frame.place(relx=0.5, rely=0.3, anchor=tkinter.CENTER)  
 self.middle\_frame.place(relx=0.5, rely=0.4, anchor=tkinter.CENTER)  
 self.middle111\_frame.pack()  
 self.middle11\_frame.pack()  
 self.bottom\_frame.place(relx=0.5, rely=0.5, anchor=tkinter.CENTER)  
 self.BBottom\_frame.place(relx=0.5, rely=0.6, anchor=tkinter.CENTER)  
 #close BUtton  
 self.close\_button = tk.Button(self.main\_window, text="Close", command=self.close, width=10, fg="white",  
 font=("Arial", 15, "bold"), bg="red")  
 self.close\_button.place(relx=0.9, rely=0.9, anchor=tk.CENTER)  
  
 tk.mainloop()  
  
 def close(self): # closing the application  
 self.main\_window.destroy()  
  
 def adminlogin(self):  
  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 # fetch stored credentials from the database  
 cursor.execute("SELECT Username, Password FROM EmployeeInformation")  
 stored\_credentials = cursor.fetchall()  
  
 entered\_username = self.username\_entry.get().strip()  
 entered\_password = hashlib.sha256(self.password\_entry.get().strip().encode()).hexdigest()  
  
 for username, password in stored\_credentials:  
 if username == entered\_username and password == entered\_password:  
 cursor.execute("SELECT employee\_id, CONCAT(Last\_Name, ', ', First\_Name, ' ', Middle\_Name) AS full\_name FROM EmployeeInformation WHERE Username = %s",  
 (entered\_username,))  
  
 result = cursor.fetchone()  
 employee\_id = result[0]  
 full\_name = result[1]  
 # if username and password match, go to main menu  
 self.main\_window.destroy()  
 EmpMenu(employee\_id, full\_name)  
 return  
 # if username and password do not match, show error message  
 error\_message = tk.Label(self.main\_window, text="Error: Invalid username or password.")  
 error\_message.pack()  
  
 mydb.close()  
  
 def login(self):  
  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
  
 # fetch stored credentials from the database  
 cursor.execute("SELECT Username, Password FROM CustomersInformation")  
 stored\_credentials = cursor.fetchall()  
  
 entered\_username = self.username\_entry.get().strip()  
 entered\_password = hashlib.sha256(self.password\_entry.get().strip().encode()).hexdigest()  
  
 for username, password in stored\_credentials:  
 if username == entered\_username and password == entered\_password:  
 cursor.execute(  
 "SELECT Customer\_ID, CONCAT(Last\_Name, ', ', First\_Name, ' ', Middle\_Name) AS full\_name FROM CustomersInformation WHERE Username = %s",  
 (entered\_username,)  
 )  
  
 result = cursor.fetchone()  
 customer\_id = result[0]  
 full\_name = result[1]  
 # if username and password match, go to main menu  
 self.main\_window.destroy()  
 MainMenu(customer\_id, full\_name)  
 return  
 # if username and password do not match, show error message  
 tk.messagebox.showerror("Invaid Credentials","Invalid username or password")  
  
  
 mydb.close()  
  
 def signup(self):  
 self.main\_window.destroy()  
 SignUpPage()  
  
 # FORGOT PASSWORD  
 def forgot\_password(self):  
  
 self.forgot\_window = tk.Toplevel(self.main\_window)  
 self.forgot\_window.title("Forgot Password")  
 self.forgot\_window.configure(background='#161513')  
 self.forgot\_window.geometry("400x600")  
 # rest of your code here  
  
 self.forframe = tk.Frame(self.forgot\_window,bg="#161513")  
 self.forframe.pack()  
 self.forframe1 = tk.Frame(self.forgot\_window,bg="#161513")  
 self.forframe1.pack()  
 email\_label = tk.Label(self.forframe, text="Enter your email address:", font=("Arial", 15, "bold"),bg="#161513",fg="white")  
 email\_label.pack(pady=10)  
 email\_entry = tk.Entry(self.forframe, font=("Arial", 15, "bold"))  
 email\_entry.pack(pady=10)  
  
  
 submit\_button = tk.Button(self.forframe, text="Submit", font=("Arial", 15, "bold"),bg="green",fg="white",  
 command=lambda: self.send\_otp(email\_entry.get()))  
 submit\_button.pack(pady=10)  
 close\_button = tk.Button(self.forframe, text="Close", font=("Arial", 15, "bold"), bg="green", fg="white",  
 command=self.forgot\_window.destroy)  
 close\_button.pack(pady=10)  
  
  
  
 def send\_otp(self, email):  
 if email == " " or email == "":  
 tk.messagebox.showerror("InValid Input","Please enter an email id")  
 self.forgot\_window.focus()  
 return  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 c = mydb.cursor()  
 c.execute("SELECT Username,Email\_ID FROM CustomersInformation WHERE Email\_ID =%s",(email,))  
 result = c.fetchone()  
 mydb.close()  
  
 if result is not None:  
 # function to send OTP to email address  
 otp = str(random.randint(100000, 999999))  
  
 sender\_email = "friendscmu2022@gmail.com"  
 password = "oznhefehqurvpmjz"  
 message= f'Your OTP is {otp}. Please enter this code to reset your password.'  
 msg = MIMEText(message)  
 msg['Subject'] = 'Your OTP for password reset'  
 msg['From'] = 'sender\_email'  
 msg['To'] = email  
 with smtplib.SMTP('smtp.gmail.com', 587) as server:  
 server.starttls()  
 server.login(sender\_email, password)  
 server.sendmail(sender\_email, email, msg.as\_string())  
  
 self.reset\_password(otp, email)  
 else:  
 tk.messagebox.showerror("NoMatch","Email ID dosent exits in the databse")  
  
 def reset\_password(self, otp, email):  
  
 otp\_label = tk.Label(self.forframe1, text="Enter the OTP sent to your email:", font=("Arial", 15, "bold"),bg="#161513",fg="white")  
 otp\_label.pack(pady=10)  
 otp\_entry = tk.Entry(self.forframe1, font=("Arial", 15, "bold"))  
 otp\_entry.pack(pady=10)  
  
 new\_password\_label = tk.Label(self.forframe1, text="Enter your new password:", font=("Arial", 15, "bold"),bg="#161513",fg="white")  
 new\_password\_label.pack(pady=10)  
 new\_password\_entry = tk.Entry(self.forframe1, show="\*", font=("Arial", 15, "bold"))  
 new\_password\_entry.pack(pady=10)  
  
 confirm\_button = tk.Button(self.forframe1, text="Confirm", font=("Arial", 15, "bold"),  
 command=lambda: self.update\_password(otp, otp\_entry.get(), email,  
 new\_password\_entry.get()))  
 confirm\_button.pack(pady=10)  
  
 def update\_password(self, expected\_otp, entered\_otp, email, new\_password):  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 c = mydb.cursor()  
 if entered\_otp == expected\_otp:  
 hashed\_password = hashlib.sha256(new\_password.encode()).hexdigest()  
 c.execute("UPDATE CustomersInformation SET Password =%s WHERE Email\_ID =%s ",(hashed\_password,email))  
 c.execute("SELECT Username FROM CustomerSInformation WHERE Email\_ID = %s",(email,))  
 result = c.fetchone()[0]  
 mydb.commit()  
 mydb.close()  
  
 print(result)  
 #Mailing The user name to the email  
 sender\_email = "friendscmu2022@gmail.com"  
 password = "oznhefehqurvpmjz"  
 message = f'Your Password Has Been Reset.\n Your Username :{result}'  
 msg = MIMEText(message)  
 msg['Subject'] = 'Your OTP for password reset'  
 msg['From'] = 'sender\_email'  
 msg['To'] = email  
 with smtplib.SMTP('smtp.gmail.com', 587) as server:  
 server.starttls()  
 server.login(sender\_email, password)  
 server.sendmail(sender\_email, email, msg.as\_string())  
 tk.messagebox.showinfo("Success","You Have Successfully Change Your Password")  
 self.forgot\_window.destroy()  
 else:  
 mydb.close()  
 tk.messagebox.showerror("OTP NoMatch","OTP Sent to you dosent match\n Sorry Please Try Again")  
 self.forgot\_window.destroy()  
  
# SIGN-UP  
class SignUpPage:  
 def \_\_init\_\_(self):  
 self.main\_window = tk.Tk()  
 self.main\_window.title("Pizza Billing System - Sign Up")  
 self.main\_window.attributes('-fullscreen', True)  
  
 img222 = Image.open("LOGIN.jpg")  
 screen\_width = self.main\_window.winfo\_screenwidth()  
 screen\_height = self.main\_window.winfo\_screenheight()  
 img222 = img222.resize((screen\_width, screen\_height))  
 self.BGG = ImageTk.PhotoImage(img222)  
 self.imageL = tk.Label(self.main\_window, image=self.BGG)  
 self.imageL.place(x=0, y=0)  
  
  
 self.store\_label = tk.Label(self.main\_window, text="Registration", font=("Times New Roman", 64, "bold"),  
 bg="#161513",fg="white")  
 self.store\_label.pack(side="top",pady=5)  
 self.signup\_frame = tk.Frame(self.main\_window, highlightthickness=2,  
 highlightbackground="black", bg="#161513")  
 self.signup\_frame.pack(side="top", pady=10)  
  
  
 ## creating first name last name contact  
 self.first\_label = tk.Label(self.signup\_frame, text="First Name\*:", font=("Arial", 12, "bold"), bg="#161513",fg="#800000")  
 self.first\_entry = tk.Entry(self.signup\_frame, font=("Arial", 12))  
 self.first\_label.grid(row=0, column=0, pady=10,sticky="w")  
 self.first\_entry.grid(row=0, column=1, pady=10,sticky="w")  
  
 self.middle\_label = tk.Label(self.signup\_frame, text="Middle Name:", font=("Arial", 12, "bold"), bg="#161513",fg="white")  
 self.middle\_entry = tk.Entry(self.signup\_frame, font=("Arial", 12))  
 self.middle\_label.grid(row=1, column=0, pady=10,sticky="w")  
 self.middle\_entry.grid(row=1, column=1, pady=10,sticky="w")  
  
 self.last\_label = tk.Label(self.signup\_frame, text="Last Name\*:", font=("Arial", 12, "bold"), bg="#161513",fg="#800000")  
 self.last\_entry = tk.Entry(self.signup\_frame, font=("Arial", 12))  
 self.last\_label.grid(row=2, column=0, pady=10,sticky="w")  
 self.last\_entry.grid(row=2, column=1, pady=10,sticky="w")  
  
 self.contact\_label = tk.Label(self.signup\_frame, text="Contact Number\*:", font=("Arial", 12, "bold"), bg="#161513",fg="#800000")  
 self.contact\_entry = tk.Entry(self.signup\_frame, font=("Arial", 12))  
 self.contact\_label.grid(row=3, column=0, pady=10,sticky="w")  
 self.contact\_entry.grid(row=3, column=1, pady=10,sticky="w")  
  
 # create Adress label and entry  
 self.Address\_label = tk.Label(self.signup\_frame, text="Address:", font=("Arial", 12, "bold"), bg="#161513",fg="white")  
 self.Address\_entry = tk.Entry(self.signup\_frame, font=("Arial", 12))  
 self.Address\_label.grid(row=4, column=0, pady=10,sticky="w")  
 self.Address\_entry.grid(row=4, column=1, pady=10,sticky="w")  
  
 usa\_state\_names = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL", "IN", "IA", "KS", "KY",  
 "LA", "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM", "NY", "NC", "ND",  
 "OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"]  
  
  
 self.State\_label = tk.Label(self.signup\_frame, text="State:", font=("Arial", 12, "bold"), bg="#161513",fg="white")  
 self.State\_label.grid(row=6, column=0, pady=10,sticky="w")  
  
 self.State\_combo = ttk.Combobox(self.signup\_frame, values=usa\_state\_names, font=("Arial", 12), width=18)  
 self.State\_combo.grid(row=6, column=1, pady=10,sticky="w")  
  
 self.City\_label = tk.Label(self.signup\_frame, text="City:", font=("Arial", 12, "bold"), bg="#161513",fg="white")  
 self.City\_entry = tk.Entry(self.signup\_frame, font=("Arial", 12))  
 self.City\_label.grid(row=5, column=0, pady=10,sticky="w")  
 self.City\_entry.grid(row=5, column=1, pady=10,sticky="w")  
  
 self.Zip\_label = tk.Label(self.signup\_frame, text="ZipCode:", font=("Arial", 12, "bold"), bg="#161513",fg="white")  
 self.Zip\_entry = tk.Entry(self.signup\_frame, font=("Arial", 12))  
 self.Zip\_label.grid(row=7, column=0, pady=10,sticky="w")  
 self.Zip\_entry.grid(row=7, column=1, pady=10,sticky="w")  
  
 self.Email\_label = tk.Label(self.signup\_frame, text="Email\*:", font=("Arial", 12, "bold"), bg="#161513",fg="#800000")  
 self.Email\_entry = tk.Entry(self.signup\_frame, font=("Arial", 12))  
 self.Email\_label.grid(row=8, column=0, pady=10,sticky="w")  
 self.Email\_entry.grid(row=8, column=1, pady=10,sticky="w")  
  
 # create password label and entry  
 self.password\_label = tk.Label(self.signup\_frame, text="Password\*:", font=("Arial", 12, "bold"), bg="#161513",fg="#800000")  
 self.password\_entry = tk.Entry(self.signup\_frame, show="\*", font=("Arial", 12))  
 self.password\_label.grid(row=9, column=0, pady=10,sticky="w")  
 self.password\_entry.grid(row=9, column=1, pady=10,sticky="w")  
  
 # create confirm password label and entry  
 self.confirm\_password\_label = tk.Label(self.signup\_frame, text="Confirm Password\*:", font=("Arial", 12, "bold")  
 , bg="#161513", fg="#800000")  
 self.confirm\_password\_entry = tk.Entry(self.signup\_frame, show="\*", font=("Arial", 12))  
 self.confirm\_password\_label.grid(row=10, column=0, pady=10,sticky="w")  
 self.confirm\_password\_entry.grid(row=10, column=1, pady=10,sticky="w")  
  
 # create signup button  
 self.signup\_button = tk.Button(self.signup\_frame, text="Sign Up", command=self.signup, width=10,  
 fg="white", font=("Arial", 15, "bold"), bg="green")  
 self.signup\_button.grid(row=11, column=0, columnspan=2, pady=10)  
  
 llabel = tk.Label(self.signup\_frame, text ="\*Kindly note that all color fields are mandatory and must be filled",font=("Arial", 8, "bold"),fg="#800000",bg="#161513" )  
 llabel.grid(row=12,columnspan=2)  
  
 # create back button  
 self.back\_button = tk.Button(self.main\_window, text="Back", command=self.back, width=10, fg="white",  
 font=("Arial", 15, "bold"), bg="red")  
 self.back\_button.place(relx=0.9, rely=0.9, anchor=tk.CENTER)  
  
 tk.mainloop()  
 def signup(self):  
  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 ## checking the first name and last name fields arent empty  
 if len(self.first\_entry.get()) == 0 or len(self.last\_entry.get()) == 0:  
 tk.messagebox.showinfo("Error", "First Name and Last Name Fields Must Not Be Empty")  
 return  
  
  
 first = self.first\_entry.get().lower().strip()  
 middle = self.middle\_entry.get().lower().strip()  
 last = self.last\_entry.get().lower().strip()  
 fi = first[0].lower()  
 uid = ' '  
 num = 0  
 if middle != "":  
 mi = middle[0].lower()  
 else:  
 mi = ""  
  
 if len(last) > 5:  
 first\_p = last[:5]  
 else:  
 first\_p = last  
  
 pattern = first\_p + '%'  
 # check if the username already exists in the database  
 select\_stmt = "SELECT COUNT(\*) FROM customersinformation WHERE Last\_Name LIKE %s"  
 cursor.execute(select\_stmt, (pattern,))  
  
 num = cursor.fetchone()[0]  
 uid = first\_p.lower() + str(num + 1) + fi + mi  
  
 ## password and confirm password match check  
 if self.password\_entry.get() == "":  
 tk.messagebox.showerror("Password Required", "Please enter a password")  
 return  
 elif self.password\_entry.get() != self.confirm\_password\_entry.get():  
 tk.messagebox.showerror("Password Mismatch", "Password and Confirm Password Don't Match")  
 return  
  
 #######  
 # Query the database for the highest existing customer ID  
 cursor.execute("SELECT MAX(Customer\_ID) FROM CustomersInformation")  
 max\_id = cursor.fetchone()[0]  
 if max\_id is None:  
 # If there are no existing customers, start the customer ID sequence at 1  
 max\_id = 0  
 else:  
 # Convert the max\_id value to an integer  
 max\_id = int(max\_id.replace("CUS", ""))  
  
 # Increment the highest customer ID to generate a new unique customer ID  
 new\_id = max\_id + 1  
  
 # Add "CUS" back to the beginning of the new customer ID  
 customer\_id = f"CUS{new\_id}"  
 first = self.first\_entry.get().lower().strip()  
 middle = self.middle\_entry.get().lower().strip()  
 last = self.last\_entry.get().lower().strip()  
 address = self.Address\_entry.get().lower()  
 city = self.City\_entry.get().lower().strip()  
 state = self.State\_combo.get().lower().strip()  
 zip\_code = self.Zip\_entry.get().lower().strip()  
 phone = self.contact\_entry.get().lower().strip()  
 email = self.Email\_entry.get().lower().strip()  
 username = uid  
  
 ## phone number format chenk  
 if not phone.isdigit():  
 tk.messagebox.showinfo("Error", "Phone Number should contain only numeric characters")  
 return  
 if len(phone) != 10:  
 tk.messagebox.showinfo("Error","Phone Number should be in 10 digits only ")  
 return  
 # Check if the email ID matches the required format.  
 regex = re.compile('^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$')  
 if not regex.match(email):  
 tk.messagebox.showinfo("Error", "Email Should be in Proper format\n Eg: abc@pizzastore.com ")  
 return  
  
 # hash the password  
 hashed\_password = hashlib.sha256(self.password\_entry.get().encode()).hexdigest()  
  
 # check if the contact already exists in the database  
 cursor.execute("SELECT Phone\_Number FROM CustomersInformation WHERE Phone\_Number = %s",  
 (self.contact\_entry.get(),))  
  
 result = cursor.fetchone()  
 if result is not None:  
 tkinter.messagebox.showinfo("Error", " Contact already exists.")  
 return  
 # check if the Email already exists in the database  
 cursor.execute("SELECT Email\_ID FROM CustomersInformation WHERE Email\_ID = %s",  
 (self.Email\_entry.get().lower(),))  
 result1 = cursor.fetchone()  
 if result1 is not None:  
 tkinter.messagebox.showinfo("Error", " E-mail already exists.")  
 return  
  
 ## Formats and numerics and alphabits check  
 # Validate zip code format  
 if zip\_code and not zip\_code.isnumeric():  
 tk.messagebox.showerror("Invalid Zip Code", "Please enter a valid zip code")  
 return  
 city\_pattern = r'^[a-zA-Z\s]+$'  
 if city and not re.match(city\_pattern, city):  
 tk.messagebox.showerror("Invalid City Name", "Please enter a valid city name")  
 return  
  
 if state and (not state.isalpha() or len(state) != 2):  
 tk.messagebox.showerror("Invalid State Code", "Please enter a valid two-letter state code")  
 return  
  
 cursor.execute(  
 "INSERT INTO CustomersInformation (Customer\_ID, First\_Name, Middle\_Name, Last\_Name, Address, City, State, ZipCode, Phone\_Number, Username, Password, Email\_ID) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)",  
 (customer\_id, first, middle, last, address, city, state, zip\_code, phone, username, hashed\_password, email)  
 )  
  
 ### Sending confirmation E-mail  
 # send confirmation email to user  
 sender\_email = "friendscmu2022@gmail.com"  
 receiver\_email = self.Email\_entry.get()  
 password = "oznhefehqurvpmjz"  
 message = f"Thank you for signing up! Your details:\n\nName: {first} {middle} {last} \nContact Number: {phone}\nAddress:{address}\n{city}\n{zip\_code}\n{state}\n\nCustomer-ID: {customer\_id}\nUserName: {username} \n Password: {self.password\_entry.get()}"  
 msg = MIMEText(message)  
 msg['Subject'] = 'Confirmation of Sign Up'  
 msg['From'] = sender\_email  
 msg['To'] = receiver\_email  
 with smtplib.SMTP('smtp.gmail.com', 587) as server:  
 server.starttls()  
 server.login(sender\_email, password)  
 server.sendmail(sender\_email, receiver\_email, msg.as\_string())  
  
 # add the new user to the database  
  
 mydb.commit()  
 mydb.close()  
  
 # Showing a top up message of details using showmessage function  
 title = "Registered Successfully"  
 message = (  
 f"You account has been created successfully and Your login details are mailed to: {self.Email\_entry.get()}\n\n"  
 f"Thank you for registering with Pizza Restaurant! We are excited to have you as part of our family.\n"  
 f"We look forward to serving you soon!"  
  
 )  
  
 font = ("Arial", 12, "bold")  
 self.show\_custom\_messagebox(title, message, font)  
  
 def show\_custom\_messagebox(self, title, message, font):  
 custom\_box = tk.Toplevel()  
 custom\_box.title(title)  
  
 # create label widget to display message  
 message\_label = tk.Label(custom\_box, text=message, font=font)  
 message\_label.pack(padx=20, pady=20)  
  
 # create ok button to close message box  
 ok\_button = tk.Button(custom\_box, text="OK",  
 command=lambda: [custom\_box.destroy(), self.main\_window.destroy(), LoginPage()])  
 ok\_button.pack(padx=10, pady=10)  
  
 custom\_box.mainloop()  
  
 def back(self):  
 # switch to login page  
 self.main\_window.destroy()  
 LoginPage()  
  
  
#########VEG  
#########VEG#########VEG#########VEG  
#########VEG#########VEG#########VEG#########VEG  
#########VEG#########VEG#########VEG#########VEG#########VEG  
#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG  
#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG  
#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG  
#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG#########VEG  
# VEG- PIZZA MENU  
class VegMenu:  
 def \_\_init\_\_(self, parent, pizzasV, pizzasVL, Cart, id, name):  
 self.parent = parent  
 self.main\_window = tk.Toplevel(self.parent)  
  
 self.pizzasV = pizzasV  
 self.pizzasVL = pizzasVL  
 self.id = id  
 self.name = name  
  
 self.main\_window.attributes('-fullscreen', True)  
 self.main\_window.title("Pizza Billing System - Vegetarian Pizzas")  
 self.cart = Cart  
 ## BacgGround Image  
 img222 = Image.open("PIZZA.jpg")  
 screen\_width = self.main\_window.winfo\_screenwidth()  
 screen\_height = self.main\_window.winfo\_screenheight()  
 img222 = img222.resize((screen\_width, screen\_height))  
 self.BGG = ImageTk.PhotoImage(img222)  
 self.imageL = tk.Label(self.main\_window, image=self.BGG)  
 self.imageL.place(x=0, y=0)  
  
 ## Creting Frames and Main Label to load the pizza data  
 # Adding Heading with Animation from left to middle top of screen  
 self.veg\_label = tk.Label(self.main\_window, text="Vegetarian Pizzas Menu", font=("Arial", 32, "bold")  
 , bg="black",fg="#2e8b57")  
 self.veg\_label.place(x=-500, y=-200)  
  
 # Animate the veg label to slide in from the left to the center of the window  
 def slide\_in():  
 x = self.veg\_label.winfo\_x()  
 if x < (screen\_width / 2) - 250:  
 self.veg\_label.place(x=x + 10, y=85)  
 self.main\_window.after(10, slide\_in)  
  
 slide\_in()  
  
 self.regular\_frame = tk.Frame(self.main\_window,relief='flat', bg="black")  
 self.regular\_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)  
  
 ###LOADING IMAGES PIZZA  
 images = [] # create an empty list to store the images  
  
 cheese = Image.open("cheesepizza.png")  
 cheese = cheese.resize((200, 200)) # fix the size argument to a tuple  
 cheese\_image = ImageTk.PhotoImage(cheese)  
 images.append(cheese\_image)  
  
 marg = Image.open("marg.png")  
 marg = marg.resize((200, 200)) # fix the size argument to a tuple  
 marg\_image = ImageTk.PhotoImage(marg)  
 images.append(marg\_image)  
  
 mrrom = Image.open("mrrom.png")  
 mrrom = mrrom.resize((200, 200)) # fix the size argument to a tuple  
 mrrom\_image = ImageTk.PhotoImage(mrrom)  
 images.append(mrrom\_image)  
  
 grillv = Image.open("grillv.png")  
 grillv = grillv.resize((200, 200)) # fix the size argument to a tuple  
 grillv\_image = ImageTk.PhotoImage(grillv)  
 images.append(grillv\_image)  
  
 # Create pizza frames for each pizza  
 for pizza, image, pizza1 in zip(self.pizzasV, images, self.pizzasVL):  
 self.create\_pizza\_reg\_frame(pizza, image, pizza1)  
  
  
 self.cart\_button = tk.Button(self.main\_window, text="View Cart", command=lambda: self.view\_cart(), width=10,  
 bg="green", fg="white", font=("Arial", 15, "bold"))  
 self.cart\_button.place(relx=0.85, rely=0.92, anchor=tk.E)  
 self.add\_more\_button = tk.Button(self.main\_window, text="Add More", command=self.add\_more, width=10, bg="green",  
 fg="white", font=("Arial", 15, "bold"))  
 self.add\_more\_button.place(relx=0.75, rely=0.92, anchor=tk.E)  
 ##close BUtton  
 self.close\_button = tk.Button(self.main\_window, text="Close", command=self.main\_window.destroy, width=8,  
 bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.place(relx=0.95, rely=0.92, anchor=tk.E)  
  
 tk.mainloop()  
  
 def add\_more(self):  
 self.main\_window.destroy()  
  
 def create\_pizza\_reg\_frame(self, pizza, image, pizza1):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.regular\_frame, highlightthickness=5,  
 highlightbackground="black", bg="black", relief='sunken')  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20)  
 ## Loading Images  
 img\_lbl = tk.Label(pizza\_frame, image=image, bg="black")  
 img\_lbl.pack()  
 # Add the pizza name label  
 pizza\_name\_label = tk.Label(pizza\_frame, text=pizza.name, font=("Ariel", 18, "bold"), background="#2e8b57")  
 pizza\_name\_label.pack()  
 pizza1\_frame = tk.Frame(pizza\_frame, bg="black", relief='sunken')  
 pizza1\_frame.pack(side="left", padx=10, pady=10, ipadx=10, anchor="center")  
 pizza2\_frame = tk.Frame(pizza\_frame, bg="black", relief='sunken')  
 pizza2\_frame.pack(side="right", padx=10, pady=10, ipadx=10, anchor="center")  
 # Add the pizza price label  
 pizza\_price\_label = tk.Label(pizza1\_frame, text="${:.2f}".format(pizza.price), font=("Ariel", 14, "bold")  
 , bg="black",fg="white")  
 pizza\_price\_label.pack(side="top")  
 # # Add the pizza price label  
 pizza1\_price\_label = tk.Label(pizza2\_frame, text="${:.2f}".format(pizza1.price), font=("Ariel", 14, "bold"),  
 bg="black",fg="white")  
 pizza1\_price\_label.pack(side="top")  
 # Add the "Add to Cart" button  
  
 add\_to\_cart\_button = tk.Button(pizza1\_frame, text="Regular", command=lambda: self.add\_to\_cart(pizza),  
 font=("Ariel", 15, "bold"), bg="#2e8b57", fg="white", relief="flat")  
 add\_to\_cart\_button.pack(side="bottom", anchor="center")  
  
 add1\_to\_cart\_button = tk.Button(pizza2\_frame, text="Large", command=lambda: self.add\_to\_cart(pizza1),  
 font=("Ariel", 15, "bold"), bg="#2e8b57", fg="white", relief="flat")  
 add1\_to\_cart\_button.pack(side="bottom", anchor="center")  
  
 def add\_to\_cart(self, pizza):  
 pizza\_price = pizza.price  
 # Create a new Pizza object with the selected size and price  
 selected\_pizza = Pizza(pizza.id, pizza.name, pizza\_price, pizza.type, pizza.size)  
 selected\_pizza.quantity.set(pizza.quantity.get())  
 # Check if the pizza already exists in the cart  
 for item in self.cart.items:  
 if item.id == selected\_pizza.id:  
 # Update the quantity of the existing pizza item  
 item.quantity.set(item.quantity.get() + selected\_pizza.quantity.get())  
 break  
 else:  
 # Add the pizza copy to the cart  
 self.cart.add\_item(selected\_pizza)  
  
 def view\_cart(self):  
 # Create a new window for the cart  
 self.cart\_window = tk.Toplevel(self.main\_window)  
 self.cart\_window.geometry("700x600")  
 self.cart\_window.title("View Cart")  
 self.cart\_window.configure(background="#3e4c4f")  
 self.cart\_window.resizable(width=False, height=False)  
  
 frame1 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame1.pack()  
 # Create a canvas to hold the cart items frame and add a vertical scrollbar  
 cart\_canvas = tk.Canvas(self.cart\_window, bg="#3e4c4f", bd=0, highlightthickness=0)  
 cart\_scrollbar = tk.Scrollbar(cart\_canvas, orient="vertical", command=cart\_canvas.yview)  
 cart\_canvas.configure(yscrollcommand=cart\_scrollbar.set)  
 cart\_canvas.pack(fill="both", expand=True)  
 cart\_scrollbar.pack(side="right", fill="y")  
  
 frame3 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame3.pack()  
  
 # Add a label for the cart items  
 cart\_items\_label = tk.Label(frame1, text="Cart Items", font=("Ariel", 25, "bold"), bg="#3e4c4f", fg="white")  
 cart\_items\_label.pack()  
  
 # Add a frame for the cart items  
 # Create a frame to hold the cart items and add it to the canvas  
 cart\_items\_frame = tk.Frame(cart\_canvas, bg="#3e4c4f")  
 cart\_canvas.create\_window((0, 0), window=cart\_items\_frame, anchor="nw")  
 self.cart\_items\_frame = cart\_items\_frame  
 # Add a label for the cart total  
 self.cart\_total\_label = tk.Label(frame3, text="Total: ${:.2f}".format(self.get\_cart\_total()),  
 font=("Ariel", 20, "bold"), bg="#3e4c4f", fg="white")  
 self.cart\_total\_label.pack()  
  
 # Add a "Checkout" button  
 checkout\_button = tk.Button(frame3, text="Checkout", command=self.checkout, width=15, bg="green",  
 fg="white", font=("Arial", 20, "bold"))  
 checkout\_button.pack(pady=10)  
  
 ##close BUtton  
 self.close\_button = tk.Button(frame3, text="Close", command=self.cart\_window.destroy, width=8,  
 bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.pack(pady=10)  
 # Add a label and Spinbox for each pizza item in the cart  
 for i, pizza in enumerate(self.cart.get\_items()):  
 # Create a frame to hold the product label and Spinbox  
 product\_frame = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame.grid(row=i, column=0, sticky='w', pady=10)  
  
 product\_frame1 = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame1.grid(row=i, column=2, sticky='w', pady=10)  
  
 pizza.product\_frame = product\_frame  
 pizza.product\_frame1 = product\_frame1  
  
 # Add a label for the product name, type, and size  
 product\_label = tk.Label(product\_frame, text="{} ({} {})".format(pizza.name, pizza.type, pizza.size),  
 font=('Ariel', 15, "bold"), bg="#3e4c4f", fg="white")  
 product\_label.grid(row=0, column=0, sticky='w')  
  
 # Add a Spinbox for the product quantity  
 quantity\_spinbox = tk.Spinbox(product\_frame1, from\_=1, to=10, width=5, textvariable=pizza.quantity,  
 command=lambda: self.update\_cart\_total(), font=('Ariel', 15, "bold"))  
 quantity\_spinbox.grid(row=0, column=0, padx=(10, 0))  
  
 # Add a label for the product price  
 price\_label = tk.Label(product\_frame, text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())),  
 font=('Ariel', 15, "bold"), bg="#3e4c4f", fg="white")  
 price\_label.grid(row=0, column=1, padx=(10, 0))  
  
 # Add a "Remove" button for the pizza item  
 remove\_button = tk.Button(product\_frame1, text="Remove",  
 command=lambda pizza\_item=pizza: self.remove\_from\_cart(pizza\_item),  
 font=('Ariel', 15, "bold"), bg="red")  
 remove\_button.grid(row=0, column=1, padx=(10, 0))  
  
 # Add the price label to the pizza item for later use  
 pizza.price\_label = price\_label  
  
 # Update the canvas to adjust to the size of the cart items frame  
 cart\_items\_frame.update\_idletasks()  
 cart\_canvas.config(scrollregion=cart\_canvas.bbox("all"))  
  
 def update\_cart\_total(self):  
 # Update the price labels for each pizza item in the cart  
 for pizza in self.cart.get\_items():  
 pizza.price\_label.config(text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())))  
  
 # Update the cart total label with the new total  
 self.cart\_total\_label.config(text="Total: ${:.2f}".format(self.get\_cart\_total()))  
  
 def remove\_from\_cart(self, pizza\_item):  
 # Check if the pizza item is in the items list  
 if pizza\_item in self.cart.items:  
 # Remove the pizza item from the cart's items list  
 self.cart.items.remove(pizza\_item)  
 # Update the cart items and total labels  
 self.update\_cart\_total()  
 # Destroy the product frames associated with the pizza item  
 pizza\_item.product\_frame.destroy()  
 pizza\_item.product\_frame1.destroy()  
  
 def get\_cart\_total(self):  
 return self.cart.get\_total()  
  
 def checkout(self):  
 self.main\_window.destroy()  
 checkout\_window = Checkout(self.cart, self.id)  
  
  
### NON VEG  
### NON VEG### NON VEG### NON VEG  
### NON VEG### NON VEG### NON VEG### NON VEG  
### NON VEG### NON VEG### NON VEG### NON VEG### NON VEG  
### NON VEG### NON VEG### NON VEG### NON VEG### NON VEG### NON VEG  
### NON VEG### NON VEG### NON VEG### NON VEG### NON VEG### NON VEG### NON VEG### NON VEG  
### NON VEG  
  
# NON VEG PIZZA  
class NonVegMenu:  
 def \_\_init\_\_(self, parent, pizzasNV, pizzasNVL, Cart, id, name):  
 self.parent = parent  
 self.main\_window = tk.Toplevel(self.parent)  
  
 self.pizzasNV = pizzasNV  
 self.pizzasNVL = pizzasNVL  
 self.id = id  
 self.name = name  
  
 self.main\_window.attributes('-fullscreen', True)  
 self.main\_window.title("Pizza Billing System - Non-Vegetarian Pizzas")  
 self.cart = Cart  
 ## BacgGround Image  
 img222 = Image.open("PIZZA.jpg")  
 screen\_width = self.main\_window.winfo\_screenwidth()  
 screen\_height = self.main\_window.winfo\_screenheight()  
 img222 = img222.resize((screen\_width, screen\_height))  
 self.BGG = ImageTk.PhotoImage(img222)  
 self.imageL = tk.Label(self.main\_window, image=self.BGG)  
 self.imageL.place(x=0, y=0)  
  
 ## Creting Frames and Main Label to load the pizza data  
 # Adding Heading with Animation from left to middle top of screen  
 self.nonveg\_label = tk.Label(self.main\_window, text="Non-Vegetarian Pizzas Menu", font=("Arial", 32, "bold")  
 , bg="black",fg="#b22222")  
 self.nonveg\_label.place(x=-500, y=-200)  
  
  
  
 # Animate the veg label to slide in from the left to the center of the window  
 def slide\_in():  
 x = self.nonveg\_label.winfo\_x()  
 if x < (screen\_width / 2) - 250:  
 self.nonveg\_label.place(x=x + 10, y=85)  
 self.main\_window.after(10, slide\_in)  
  
 slide\_in()  
  
 self.regular\_frame = tk.Frame(self.main\_window,relief='flat', bg="#000000")  
 self.regular\_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)  
  
 ## Loading Images  
 images = []  
  
 chicken\_tandoori = Image.open("CT.png")  
 chicken\_tandoori = chicken\_tandoori.resize((200, 200)) # fix the size argument to a tuple  
 chicken\_tandoori\_image = ImageTk.PhotoImage(chicken\_tandoori)  
 images.append(chicken\_tandoori\_image)  
  
 pepperoni = Image.open("Pepperoni.png")  
 pepperoni = pepperoni.resize((200, 200)) # fix the size argument to a tuple  
 pepperoni\_image = ImageTk.PhotoImage(pepperoni)  
 images.append(pepperoni\_image)  
  
 grillnv = Image.open("grillnv.png")  
 grillnv = grillnv.resize((200, 200)) # fix the size argument to a tuple  
 grillnv\_image = ImageTk.PhotoImage(grillnv)  
 images.append(grillnv\_image)  
  
 hawaaian = Image.open("Hawaiian-1.png")  
 hawaaian = hawaaian.resize((200, 200)) # fix the size argument to a tuple  
 hawaaian\_image = ImageTk.PhotoImage(hawaaian)  
 images.append(hawaaian\_image)  
  
 # Create pizza frames for each pizza  
 for pizza, image, pizza1 in zip(self.pizzasNV, images, self.pizzasNVL):  
 self.create\_pizza\_reg\_frame(pizza, image, pizza1)  
 # Add a "View Cart" button at the bottom of the window  
 self.cart\_button = tk.Button(self.main\_window, text="View Cart", command=lambda: self.view\_cart(), width=10,  
 bg="green", fg="white", font=("Arial", 15, "bold"))  
 self.cart\_button.place(relx=0.85, rely=0.92, anchor=tk.E)  
 self.add\_more\_button = tk.Button(self.main\_window, text="Add More", command=self.add\_more, width=10, bg="green",  
 fg="white", font=("Arial", 15, "bold"))  
 self.add\_more\_button.place(relx=0.75, rely=0.92, anchor=tk.E)  
 ##close BUtton  
 self.close\_button = tk.Button(self.main\_window, text="Close", command=self.main\_window.destroy, width=8,  
 bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.place(relx=0.95, rely=0.92, anchor=tk.E)  
  
 tk.mainloop()  
  
 def add\_more(self):  
 self.main\_window.destroy()  
  
 def create\_pizza\_reg\_frame(self, pizza, image, pizza1):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.regular\_frame, highlightthickness=5,  
 highlightbackground="black", bg="black", relief='sunken')  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20)  
 ## Loading Images  
 img\_lbl = tk.Label(pizza\_frame, image=image, bg="black")  
 img\_lbl.pack()  
 # Add the pizza name label  
 pizza\_name\_label = tk.Label(pizza\_frame, text=pizza.name, font=("Ariel", 18, "bold"), background="#b22222")  
 pizza\_name\_label.pack()  
 pizza1\_frame = tk.Frame(pizza\_frame, bg="black", relief='sunken')  
 pizza1\_frame.pack(side="left", padx=10, pady=10, ipadx=10, anchor="center")  
 pizza2\_frame = tk.Frame(pizza\_frame, bg="black", relief='sunken')  
 pizza2\_frame.pack(side="right", padx=10, pady=10, ipadx=10, anchor="center")  
 # Add the pizza price label  
 pizza\_price\_label = tk.Label(pizza1\_frame, text="${:.2f}".format(pizza.price), font=("Ariel", 14, "bold"),  
 bg="black",fg="white")  
 pizza\_price\_label.pack(side="top")  
 # # Add the pizza price label  
 pizza1\_price\_label = tk.Label(pizza2\_frame, text="${:.2f}".format(pizza1.price), font=("Ariel", 14, "bold"),  
 bg="black",fg="white")  
 pizza1\_price\_label.pack(side="top")  
 # Add the "Add to Cart" button  
  
 add\_to\_cart\_button = tk.Button(pizza1\_frame, text="Regular", command=lambda: self.add\_to\_cart(pizza),  
 font=("Ariel", 15, "bold"), bg="#b22222", fg="white", relief="flat")  
 add\_to\_cart\_button.pack(side="bottom", anchor="center")  
  
 add1\_to\_cart\_button = tk.Button(pizza2\_frame, text="Large", command=lambda: self.add\_to\_cart(pizza1),  
 font=("Ariel", 15, "bold"), bg="#b22222", fg="white", relief="flat")  
 add1\_to\_cart\_button.pack(side="bottom", anchor="center")  
  
 def add\_to\_cart(self, pizza):  
 pizza\_price = pizza.price  
 # Create a new Pizza object with the selected size and price  
 selected\_pizza = Pizza(pizza.id, pizza.name, pizza\_price, pizza.type, pizza.size)  
 selected\_pizza.quantity.set(pizza.quantity.get())  
 # Check if the pizza already exists in the cart  
 for item in self.cart.items:  
 if item.id == selected\_pizza.id:  
 # Update the quantity of the existing pizza item  
 item.quantity.set(item.quantity.get() + selected\_pizza.quantity.get())  
 break  
 else:  
 # Add the pizza copy to the cart  
 self.cart.add\_item(selected\_pizza)  
  
 def view\_cart(self):  
 # Create a new window for the cart  
 self.cart\_window = tk.Toplevel(self.main\_window)  
 self.cart\_window.geometry("700x600")  
 self.cart\_window.title("View Cart")  
 self.cart\_window.configure(background="#3e4c4f")  
 self.cart\_window.resizable(width=False, height=False)  
  
 frame1 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame1.pack()  
 # Create a canvas to hold the cart items frame and add a vertical scrollbar  
 cart\_canvas = tk.Canvas(self.cart\_window, bg="#3e4c4f", bd=0, highlightthickness=0)  
 cart\_scrollbar = tk.Scrollbar(cart\_canvas, orient="vertical", command=cart\_canvas.yview)  
 cart\_canvas.configure(yscrollcommand=cart\_scrollbar.set)  
 cart\_canvas.pack(fill="both", expand=True)  
 cart\_scrollbar.pack(side="right", fill="y")  
  
 frame3 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame3.pack()  
  
 # Add a label for the cart items  
 cart\_items\_label = tk.Label(frame1, text="Cart Items", font=("Ariel", 25, "bold"), bg="#3e4c4f", fg="white")  
 cart\_items\_label.pack()  
  
 # Add a frame for the cart items  
 # Create a frame to hold the cart items and add it to the canvas  
 cart\_items\_frame = tk.Frame(cart\_canvas, bg="#3e4c4f")  
 cart\_canvas.create\_window((0, 0), window=cart\_items\_frame, anchor="nw")  
 self.cart\_items\_frame = cart\_items\_frame  
 # Add a label for the cart total  
 self.cart\_total\_label = tk.Label(frame3, text="Total: ${:.2f}".format(self.get\_cart\_total()),  
 font=("Ariel", 20, "bold"), bg="#3e4c4f", fg="white")  
 self.cart\_total\_label.pack()  
  
 # Add a "Checkout" button  
 checkout\_button = tk.Button(frame3, text="Checkout", command=self.checkout, width=15, bg="green",  
 fg="white", font=("Arial", 20, "bold"))  
 checkout\_button.pack(pady=10)  
  
 ##close BUtton  
 self.close\_button = tk.Button(frame3, text="Close", command=self.cart\_window.destroy, width=8,  
 bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.pack(pady=10)  
 # Add a label and Spinbox for each pizza item in the cart  
 for i, pizza in enumerate(self.cart.get\_items()):  
 # Create a frame to hold the product label and Spinbox  
 product\_frame = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame.grid(row=i, column=0, sticky='w', pady=10)  
  
 product\_frame1 = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame1.grid(row=i, column=2, sticky='w', pady=10)  
  
 pizza.product\_frame = product\_frame  
 pizza.product\_frame1 = product\_frame1  
  
 # Add a label for the product name, type, and size  
 product\_label = tk.Label(product\_frame, text="{} ({} {})".format(pizza.name, pizza.type, pizza.size),  
 font=('Ariel', 15, "bold"), bg="#3e4c4f", fg="white")  
 product\_label.grid(row=0, column=0, sticky='w')  
  
 # Add a Spinbox for the product quantity  
 quantity\_spinbox = tk.Spinbox(product\_frame1, from\_=1, to=10, width=5, textvariable=pizza.quantity,  
 command=lambda: self.update\_cart\_total(), font=('Ariel', 15, "bold"))  
 quantity\_spinbox.grid(row=0, column=0, padx=(10, 0))  
  
 # Add a label for the product price  
 price\_label = tk.Label(product\_frame, text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())),  
 font=('Ariel', 15, "bold"), bg="#3e4c4f", fg="white")  
 price\_label.grid(row=0, column=1, padx=(10, 0))  
  
 # Add a "Remove" button for the pizza item  
 remove\_button = tk.Button(product\_frame1, text="Remove",  
 command=lambda pizza\_item=pizza: self.remove\_from\_cart(pizza\_item),  
 font=('Ariel', 15, "bold"), bg="red")  
 remove\_button.grid(row=0, column=1, padx=(10, 0))  
  
 # Add the price label to the pizza item for later use  
 pizza.price\_label = price\_label  
  
 # Update the canvas to adjust to the size of the cart items frame  
 cart\_items\_frame.update\_idletasks()  
 cart\_canvas.config(scrollregion=cart\_canvas.bbox("all"))  
  
 def update\_cart\_total(self):  
 # Update the price labels for each pizza item in the cart  
 for pizza in self.cart.get\_items():  
 pizza.price\_label.config(text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())))  
  
 # Update the cart total label with the new total  
 self.cart\_total\_label.config(text="Total: ${:.2f}".format(self.get\_cart\_total()))  
  
 def remove\_from\_cart(self, pizza\_item):  
 # Check if the pizza item is in the items list  
 if pizza\_item in self.cart.items:  
 # Remove the pizza item from the cart's items list  
 self.cart.items.remove(pizza\_item)  
 # Update the cart items and total labels  
 self.update\_cart\_total()  
 # Destroy the product frames associated with the pizza item  
 pizza\_item.product\_frame.destroy()  
 pizza\_item.product\_frame1.destroy()  
  
 def get\_cart\_total(self):  
 return self.cart.get\_total()  
  
 def checkout(self):  
 self.main\_window.destroy()  
 checkout\_window = Checkout(self.cart, self.id)  
  
  
## BEV Sides  
## BEV Sides## BEV Sides## BEV Sides  
## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides  
## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides  
## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides## BEV Sides  
## BEV Sides  
# BEVERAGES PIZZA  
class BevSideMenu:  
 def \_\_init\_\_(self, parent, bev, sides, Cart, id, name):  
 self.parent = parent  
 self.main\_window = tk.Toplevel(self.parent)  
  
 self.bev = bev  
 self.sides = sides  
 self.id = id  
 self.name = name  
 self.main\_window.attributes('-fullscreen', True)  
  
 self.main\_window.title("Pizza Billing System - Beverages & Sides")  
 self.cart = Cart  
  
 ## BacgGround Image  
 img222 = Image.open("PIZZA.jpg")  
 screen\_width = self.main\_window.winfo\_screenwidth()  
 screen\_height = self.main\_window.winfo\_screenheight()  
 img222 = img222.resize((screen\_width, screen\_height))  
 self.BGG = ImageTk.PhotoImage(img222)  
 self.imageL = tk.Label(self.main\_window, image=self.BGG)  
 self.imageL.place(x=0, y=0)  
  
 ## Creting Frames and Main Label to load the pizza data  
 # Adding Heading with Animation from left to middle top of screen  
 self.veg\_label = tk.Label(self.main\_window, text="Beverages & Sides Menu", font=("Arial", 32, "bold")  
 , bg="black",fg="white")  
 self.veg\_label.place(x=-500, y=-200)  
  
 # Animate the veg label to slide in from the left to the center of the window  
 def slide\_in():  
 x = self.veg\_label.winfo\_x()  
 if x < (screen\_width / 2) - 250:  
 self.veg\_label.place(x=x + 10, y=85)  
 self.main\_window.after(10, slide\_in)  
  
 slide\_in()  
  
 frame = tk.Frame(self.main\_window,relief='flat', bg="black")  
 frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)  
  
 self.bev\_frame = tk.Frame(frame, relief='flat', bg="black")  
 self.bev\_frame.pack()  
  
 self.sides\_frame = tk.Frame(frame, relief='flat', bg="black")  
 self.sides\_frame.pack()  
 ### Bev Images  
  
 images1 = [] # create an empty list to store the images  
  
 ditecoke = Image.open("DietCoke.png")  
 ditecoke = ditecoke.resize((200, 150)) # fix the size argument to a tuple  
 ditecoke\_image = ImageTk.PhotoImage(ditecoke)  
 images1.append(ditecoke\_image)  
  
 coke = Image.open("coke (2).png")  
 coke = coke.resize((200, 150)) # fix the size argument to a tuple  
 coke\_image = ImageTk.PhotoImage(coke)  
 images1.append(coke\_image)  
  
 mdw = Image.open("MDEW.png")  
 mdw = mdw.resize((200, 150)) # fix the size argument to a tuple  
 mdw\_image = ImageTk.PhotoImage(mdw)  
 images1.append(mdw\_image)  
  
 lmde = Image.open("DRP.png")  
 lmde = lmde.resize((200, 150)) # fix the size argument to a tuple  
 lmde\_image = ImageTk.PhotoImage(lmde)  
 images1.append(lmde\_image)  
 # Create pizza frames for each pizza  
 for pizza, image in zip(self.bev, images1):  
 self.create\_bev\_frame(pizza, image)  
  
 ## loading sides images  
 images2 = [] # create an empty list to store the images  
  
 gbread = Image.open("garlic.png")  
 gbread = gbread.resize((200, 100)) # fix the size argument to a tuple  
 gbread\_image = ImageTk.PhotoImage(gbread)  
 images2.append(gbread\_image)  
  
 pwedges = Image.open("potatowedges.png")  
 pwedges = pwedges.resize((200, 100)) # fix the size argument to a tuple  
 pwedges\_image = ImageTk.PhotoImage(pwedges)  
 images2.append(pwedges\_image)  
  
 cwings = Image.open("cwings.png")  
 cwings = cwings.resize((200, 100)) # fix the size argument to a tuple  
 cwings\_image = ImageTk.PhotoImage(cwings)  
 images2.append(cwings\_image)  
  
 clava = Image.open("clava.png")  
 clava = clava.resize((200, 100)) # fix the size argument to a tuple  
 clava\_image = ImageTk.PhotoImage(clava)  
 images2.append(clava\_image)  
  
 colors = ["#2e8b57", "#2e8b57", "#b22222", "#612D08"]  
 for pizza, color, image in zip(self.sides, colors, images2):  
 self.create\_sides\_frame(pizza, color, image)  
 # Add a "View Cart" button at the bottom of the window  
 self.cart\_button = tk.Button(self.main\_window, text="View Cart", command=lambda: self.view\_cart(), width=10,  
 bg="green", fg="white", font=("Arial", 15, "bold"))  
 self.cart\_button.place(relx=0.85, rely=0.92, anchor=tk.E)  
 self.add\_more\_button = tk.Button(self.main\_window, text="Add More", command=self.add\_more, width=10, bg="green",  
 fg="white", font=("Arial", 15, "bold"))  
 self.add\_more\_button.place(relx=0.75, rely=0.92, anchor=tk.E)  
  
 ##close BUtton  
 self.close\_button = tk.Button(self.main\_window, text="Close", command=self.main\_window.destroy, width=8,  
 bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.place(relx=0.95, rely=0.92, anchor=tk.E)  
  
 tk.mainloop()  
  
 def add\_more(self):  
 self.main\_window.destroy()  
  
 def create\_bev\_frame(self, pizza, image):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.bev\_frame, width=300, height=200, highlightthickness=2,  
 highlightbackground="black", bg="black")  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20)  
 ## Loading Images  
 img\_lbl = tk.Label(pizza\_frame, image=image, bg="black")  
 img\_lbl.pack()  
 # Add the pizza name label  
 pizza\_name\_label = tk.Label(pizza\_frame, text=pizza.name, font=("Ariel", 20, "bold"), background="#71a6d2")  
 pizza\_name\_label.pack()  
 # Add the pizza price label  
 pizza\_price\_label = tk.Label(pizza\_frame, text="${:.2f}".format(pizza.price), font=("Ariel", 14, "bold")  
 , bg="black",fg="white")  
 pizza\_price\_label.pack()  
 # Add the "Add to Cart" button  
 add\_to\_cart\_button = tk.Button(pizza\_frame, text="Add to Cart", command=lambda: self.add\_to\_cart(pizza),  
 font=("Ariel", 14, "bold"), bg="#71a6d2", fg="white", relief="flat")  
 add\_to\_cart\_button.pack()  
  
 def create\_sides\_frame(self, pizza, color, image):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.sides\_frame, width=300, height=200, highlightthickness=2,  
 highlightbackground="black", bg="black")  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20)  
 ## Loading Images  
 img\_lbl = tk.Label(pizza\_frame, image=image, bg="black")  
 img\_lbl.pack()  
 # Add the pizza name label  
 pizza\_name\_label = tk.Label(pizza\_frame, text=pizza.name, font=("Ariel", 20, "bold"), background=color)  
 pizza\_name\_label.pack()  
  
 # Add the pizza price label  
 pizza\_price\_label = tk.Label(pizza\_frame, text="${:.2f}".format(pizza.price), font=("Ariel", 14, "bold")  
 , bg="black",fg="white")  
 pizza\_price\_label.pack()  
 # Add the "Add to Cart" button  
 add\_to\_cart\_button = tk.Button(pizza\_frame, text="Add to Cart", command=lambda: self.add\_to\_cart(pizza),  
 font=("Ariel", 14, "bold"), bg=color, fg="white", relief="flat")  
 add\_to\_cart\_button.pack()  
  
 def add\_to\_cart(self, pizza):  
 pizza\_price = pizza.price  
 # Create a new Pizza object with the selected size and price  
 selected\_pizza = Pizza(pizza.id, pizza.name, pizza\_price, pizza.type, pizza.size)  
 selected\_pizza.quantity.set(pizza.quantity.get())  
 # Check if the pizza already exists in the cart  
 for item in self.cart.items:  
 if item.id == selected\_pizza.id:  
 # Update the quantity of the existing pizza item  
 item.quantity.set(item.quantity.get() + selected\_pizza.quantity.get())  
 break  
 else:  
 # Add the pizza copy to the cart  
 self.cart.add\_item(selected\_pizza)  
  
 def view\_cart(self):  
 # Create a new window for the cart  
 self.cart\_window = tk.Toplevel(self.main\_window)  
 self.cart\_window.geometry("700x600")  
 self.cart\_window.title("View Cart")  
 self.cart\_window.configure(background="#3e4c4f")  
 self.cart\_window.resizable(width=False, height=False)  
  
 frame1 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame1.pack()  
 # Create a canvas to hold the cart items frame and add a vertical scrollbar  
 cart\_canvas = tk.Canvas(self.cart\_window, bg="#3e4c4f", bd=0, highlightthickness=0)  
 cart\_scrollbar = tk.Scrollbar(cart\_canvas, orient="vertical", command=cart\_canvas.yview)  
 cart\_canvas.configure(yscrollcommand=cart\_scrollbar.set)  
 cart\_canvas.pack(fill="both", expand=True)  
 cart\_scrollbar.pack(side="right", fill="y")  
  
 frame3 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame3.pack()  
  
 # Add a label for the cart items  
 cart\_items\_label = tk.Label(frame1, text="Cart Items", font=("Ariel", 25, "bold"), bg="#3e4c4f", fg="white")  
 cart\_items\_label.pack()  
  
 # Add a frame for the cart items  
 # Create a frame to hold the cart items and add it to the canvas  
 cart\_items\_frame = tk.Frame(cart\_canvas, bg="#3e4c4f")  
 cart\_canvas.create\_window((0, 0), window=cart\_items\_frame, anchor="nw")  
 self.cart\_items\_frame = cart\_items\_frame  
 # Add a label for the cart total  
 self.cart\_total\_label = tk.Label(frame3, text="Total: ${:.2f}".format(self.get\_cart\_total()),  
 font=("Ariel", 20, "bold"), bg="#3e4c4f", fg="white")  
 self.cart\_total\_label.pack()  
  
 # Add a "Checkout" button  
 checkout\_button = tk.Button(frame3, text="Checkout", command=self.checkout, width=15, bg="green",  
 fg="white", font=("Arial", 20, "bold"))  
 checkout\_button.pack(pady=10)  
  
 ##close BUtton  
 self.close\_button = tk.Button(frame3, text="Close", command=self.cart\_window.destroy, width=8,  
 bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.pack(pady=10)  
 # Add a label and Spinbox for each pizza item in the cart  
 for i, pizza in enumerate(self.cart.get\_items()):  
 # Create a frame to hold the product label and Spinbox  
 product\_frame = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame.grid(row=i, column=0, sticky='w', pady=10)  
  
 product\_frame1 = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame1.grid(row=i, column=2, sticky='w', pady=10)  
  
 pizza.product\_frame = product\_frame  
 pizza.product\_frame1 = product\_frame1  
  
 # Add a label for the product name, type, and size  
 product\_label = tk.Label(product\_frame, text="{} ({} {})".format(pizza.name, pizza.type, pizza.size),  
 font=('Ariel', 15, "bold"), bg="#3e4c4f", fg="white")  
 product\_label.grid(row=0, column=0, sticky='w')  
  
 # Add a Spinbox for the product quantity  
 quantity\_spinbox = tk.Spinbox(product\_frame1, from\_=1, to=10, width=5, textvariable=pizza.quantity,  
 command=lambda: self.update\_cart\_total(), font=('Ariel', 15, "bold"))  
 quantity\_spinbox.grid(row=0, column=0, padx=(10, 0))  
  
 # Add a label for the product price  
 price\_label = tk.Label(product\_frame, text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())),  
 font=('Ariel', 15, "bold"), bg="#3e4c4f", fg="white")  
 price\_label.grid(row=0, column=1, padx=(10, 0))  
  
 # Add a "Remove" button for the pizza item  
 remove\_button = tk.Button(product\_frame1, text="Remove",  
 command=lambda pizza\_item=pizza: self.remove\_from\_cart(pizza\_item),  
 font=('Ariel', 15, "bold"), bg="red")  
 remove\_button.grid(row=0, column=1, padx=(10, 0))  
  
 # Add the price label to the pizza item for later use  
 pizza.price\_label = price\_label  
  
 # Update the canvas to adjust to the size of the cart items frame  
 cart\_items\_frame.update\_idletasks()  
 cart\_canvas.config(scrollregion=cart\_canvas.bbox("all"))  
  
 def update\_cart\_total(self):  
 # Update the price labels for each pizza item in the cart  
 for pizza in self.cart.get\_items():  
 pizza.price\_label.config(text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())))  
  
 # Update the cart total label with the new total  
 self.cart\_total\_label.config(text="Total: ${:.2f}".format(self.get\_cart\_total()))  
  
 def remove\_from\_cart(self, pizza\_item):  
 # Check if the pizza item is in the items list  
 if pizza\_item in self.cart.items:  
 # Remove the pizza item from the cart's items list  
 self.cart.items.remove(pizza\_item)  
 # Update the cart items and total labels  
 self.update\_cart\_total()  
 # Destroy the product frames associated with the pizza item  
 pizza\_item.product\_frame.destroy()  
 pizza\_item.product\_frame1.destroy()  
  
 def get\_cart\_total(self):  
 return self.cart.get\_total()  
  
 def checkout(self):  
 self.main\_window.destroy()  
 checkout\_window = Checkout(self.cart, self.id)  
  
  
# MainMenu ## Cart ## Pizza # Receipt  
# MainMenu ## Cart ## Pizza # Receipt# MainMenu ## Cart ## Pizza # Receipt# MainMenu ## Cart ## Pizza # Receipt  
# MainMenu ## Cart ## Pizza # Receipt# MainMenu ## Cart ## Pizza # Receipt# MainMenu ## Cart ## Pizza # Receipt# MainMenu ## Cart ## Pizza # Receipt  
# MainMenu ## Cart ## Pizza # Receipt# MainMenu ## Cart ## Pizza # Receipt  
# MainMenu ## Cart ## Pizza # Receipt# MainMenu ## Cart ## Pizza # Receipt# MainMenu ## Cart ## Pizza # Receipt  
  
  
## Pizza CLass  
# PIZZA CLASS  
class Pizza:  
 def \_\_init\_\_(self, id, name, price, type, size):  
 self.id = id  
 self.name = name  
 self.price = price  
 self.type = type  
 self.size = size  
 self.quantity = tk.IntVar()  
 self.quantity.set(1)  
  
  
# CART  
class Cart:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 self.cart\_items\_label = None  
  
 def add\_item(self, item):  
 self.items.append(item)  
  
 self.update\_cart\_items\_label()  
  
 def remove\_item(self, item):  
 self.items.remove(item)  
 self.update\_cart\_items\_label()  
  
 def update\_cart\_items\_label(self):  
 if self.cart\_items\_label:  
 self.cart\_items\_label.configure(text="Cart Items: {}".format(len(self.items)))  
  
 def get\_items(self):  
 return self.items  
  
 def get\_total\_quantity(self):  
 total\_quantity = 0  
 for item in self.items:  
 total\_quantity += item.quantity.get()  
 return total\_quantity  
  
 def get\_total(self):  
 total = 0  
 for item in self.items:  
 total += item.price \* item.quantity.get()  
 return total  
  
  
### CUSTOMER CHECKOUT AND RECEIPT  
### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT  
### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT  
### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT  
### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT### CUSTOMER CHECKOUT AND RECEIPT  
# CHECKOUT  
class Checkout:  
 def \_\_init\_\_(self, cart, id):  
 self.cart = cart  
 self.id = id  
 self.payment\_method = "" # To store the payment method option  
 self.window = tk.Tk()  
 self.window.title("Checkout")  
 self.window.geometry("400x250")  
  
 # Add a label for the checkout total  
 total\_label = tk.Label(self.window, text="Total: ${:.2f}".format(self.cart.get\_total()),  
 font=("Ariel", 20, "bold"))  
 total\_label.pack()  
  
 # Add the payment method options  
 payment\_label = tk.Label(self.window, text="Select Payment Method:", font=("Ariel", 20, "bold"))  
 payment\_label.pack(pady=5)  
 # Add the "Cash" button  
 cash\_button = tk.Button(self.window, text="Cash", command=self.select\_cash, width=10, bg="green",  
 fg="white", font=("Arial", 10, "bold"))  
 cash\_button.pack(pady=10)  
  
 # Add the "Card" button  
 card\_button = tk.Button(self.window, text="Card", command=self.select\_card, width=10, bg="green",  
 fg="white", font=("Arial", 10, "bold"))  
 card\_button.pack(pady=10)  
  
 def select\_cash(self):  
 self.payment\_method = "Cash"  
 self.checkout()  
  
 def select\_card(self):  
 self.payment\_method = "Card"  
 self.checkout()  
  
 def checkout(self):  
  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 payment\_method = self.payment\_method  
 totalitems = self.cart.get\_total\_quantity()  
 total = self.cart.get\_total()  
 if total == 0:  
 tk.messagebox.showerror("Error", "Your cart is empty.")  
 else:  
 # Connect to the database and get the highest order ID  
 cursor.execute("SELECT MAX(Order\_ID) FROM Orders")  
 result = cursor.fetchone()  
 order\_id = result[0] + 1 if result[  
 0] else 1 # Increment the highest ID by one or start at 1 if the table is empty  
  
 # Generate an invoice code based on the current time and date  
 invoice\_code = time.strftime("%Y%m%d%H%M%S") + str(order\_id)  
  
 tk.messagebox.showinfo("Checkout",  
 "Thank you for your order! Your total is ${:.2f}. Your order will be ready soon.\n\nInvoice Code: {}. Your order ID is {}.".format(  
 total, invoice\_code, order\_id))  
  
 # Insert the order into the database  
 cursor.execute(  
 "INSERT INTO Orders (Order\_ID,Customer\_ID,Order\_Date,Order\_Total,Order\_Qty) VALUES (%s, %s, %s, %s, %s)",  
 (order\_id, self.id, time.strftime('%Y-%m-%d %H:%M:%S'), '{:.2f}'.format(total), totalitems))  
 # Update the Order\_Status table  
 cursor.execute("INSERT INTO Order\_Status (Order\_ID, Order\_Status) VALUES (%s, %s)", (order\_id, 'Pending'))  
  
 for item in self.cart.items:  
 cursor.execute("INSERT INTO Order\_Item (Order\_ID, item\_ID, item\_quantity) VALUES (%s, %s, %s)",  
 (order\_id, item.id, item.quantity.get()))  
  
 cursor.execute(  
 "INSERT INTO Payment (Invoice\_Num,MOP,Order\_ID,Customer\_ID,Order\_Date,Amount\_Paid) VALUES(%s, %s, %s, %s, %s, %s)",  
 (invoice\_code, payment\_method, order\_id, self.id, time.strftime('%Y-%m-%d'), '{:.2f}'.format(total)))  
  
 ## Getting the Email id  
 cursor.execute("SELECT Email\_ID FROM CustomersInformation WHERE Customer\_ID = %s",(self.id,))  
 email\_result = cursor.fetchone()[0]  
  
 mydb.commit()  
  
 mydb.close()  
 # Create a copy of the cart items list  
 cart\_items\_copy = self.cart.items.copy()  
  
 # Create the receipt using the copy of the cart items  
 receipt = Receipt(cart\_items\_copy, total, dt.datetime.now(), invoice\_code, payment\_method, order\_id,email\_result)  
  
 # Clear the cart after checkout  
 self.cart.items.clear()  
  
 # Close the window  
 self.window.destroy()  
  
  
  
### EMPLOYEE CHECKOUT AND RECEIPT  
### EMPLOYEE CHECKOUT AND RECEIPT### EMPLOYEE CHECKOUT AND RECEIPT### EMPLOYEE CHECKOUT AND RECEIPT### EMPLOYEE CHECKOUT AND RECEIPT  
### EMPLOYEE CHECKOUT AND RECEIPT### EMPLOYEE CHECKOUT AND RECEIPT### EMPLOYEE CHECKOUT AND RECEIPT  
### EMPLOYEE CHECKOUT AND RECEIPT### EMPLOYEE CHECKOUT AND RECEIPT### EMPLOYEE CHECKOUT AND RECEIPT### EMPLOYEE CHECKOUT AND RECEIPT  
# EMPLOYEE CHECKOUT  
class EmpCheckout:  
 def \_\_init\_\_(self,parent, cart, id, empid):  
 self.cart = cart  
 self.id = id  
 self.empid = empid  
 self.payment\_method = "" # To store the payment method option  
 self.parent = parent  
 self.window = tk.Toplevel(parent)  
 self.window.title("Checkout")  
 self.window.geometry("400x250")  
  
 # Add a label for the checkout total  
 total\_label = tk.Label(self.window, text="Total: ${:.2f}".format(self.cart.get\_total()), font=("default", 16))  
 total\_label.pack()  
  
 # Add the payment method options  
 payment\_label = tk.Label(self.window, text="Select Payment Method:", font=("default", 12))  
 payment\_label.pack(pady=5)  
 # Add the "Cash" button  
 cash\_button = tk.Button(self.window, text="Cash", command=self.select\_cash, width=10, bg="green",  
 fg="white", font=("Arial", 10, "bold"))  
 cash\_button.pack(pady=10)  
  
 # Add the "Card" button  
 card\_button = tk.Button(self.window, text="Card", command=self.select\_card, width=10, bg="green",  
 fg="white", font=("Arial", 10, "bold"))  
 card\_button.pack(pady=10)  
  
 def select\_cash(self):  
 self.payment\_method = "Cash"  
 self.checkout()  
  
 def select\_card(self):  
 self.payment\_method = "Card"  
 self.checkout()  
  
 def checkout(self):  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 payment\_method = self.payment\_method  
 totalitems = self.cart.get\_total\_quantity()  
 total = self.cart.get\_total()  
 if total == 0:  
 tk.messagebox.showerror("Error", "Your cart is empty.")  
 else:  
 # Connect to the database and get the highest order ID  
 cursor.execute("SELECT MAX(Order\_ID) FROM Orders")  
 result = cursor.fetchone()  
 order\_id = result[0] + 1 if result[  
 0] else 1 # Increment the highest ID by one or start at 1 if the table is empty  
  
 # Generate an invoice code based on the current time and date  
 invoice\_code = time.strftime("%Y%m%d%H%M%S") + str(order\_id)  
  
 tk.messagebox.showinfo("Checkout",  
 "Thank you for your order! Your total is ${:.2f}. Your order will be ready soon.\n\nInvoice Code: {}. Your order ID is {}.".format(  
 total, invoice\_code, order\_id))  
  
 # Insert the order into the database  
 cursor.execute(  
 "INSERT INTO Orders (Order\_ID,Customer\_ID,Employee\_ID,Order\_Date,Order\_Total,Order\_Qty) VALUES (%s, %s, %s, %s, %s, %s)",  
 (order\_id, self.id, self.empid, time.strftime('%Y-%m-%d %H:%M:%S'), '{:.2f}'.format(total), totalitems))  
  
 # Update the Order\_Status table  
 cursor.execute("INSERT INTO Order\_Status (Order\_ID, Order\_Status) VALUES (%s, %s)", (order\_id, 'Pending'))  
  
 for item in self.cart.items:  
 cursor.execute("INSERT INTO Order\_Item (Order\_ID, item\_ID, item\_quantity) VALUES (%s, %s, %s)",  
 (order\_id, item.id, item.quantity.get()))  
  
 cursor.execute(  
 "INSERT INTO Payment (Invoice\_Num,MOP,Order\_ID,Customer\_ID,Order\_Date,Amount\_Paid) VALUES(%s, %s, %s, %s, %s, %s)",  
 (invoice\_code, payment\_method, order\_id, self.id, time.strftime('%Y-%m-%d'), '{:.2f}'.format(total)))  
  
 mydb.commit()  
 mydb.close()  
 # Create a copy of the cart items list  
 cart\_items\_copy = self.cart.items.copy()  
  
 # Create the receipt using the copy of the cart items  
 receipt = EmpReceipt(cart\_items\_copy, total, dt.datetime.now(), invoice\_code, payment\_method, order\_id,parent=self.parent)  
  
 # Clear the cart after checkout  
 self.cart.items.clear()  
  
# RECEIPT  
class Receipt:  
 def \_\_init\_\_(self, items, total, date\_time, invoice\_code, payment\_mode, order\_id,email):  
 self.items = items  
 self.total = total  
 self.date\_time = date\_time  
 self.invoice\_code = invoice\_code  
 self.payment\_mode = payment\_mode  
 self.order\_id = order\_id  
 self.email\_id = email  
 # create the receipt window  
 self.receipt\_window = tk.Tk()  
 self.receipt\_window.title("Receipt")  
 self.receipt\_window.configure(background="white")  
 self.receipt\_window.geometry("555x600")  
 # Add a label for the receipt items  
 receipt\_items\_label = tk.Label(self.receipt\_window, text="Receipt Items", font=("Ariel", 16, "bold"), bg="white")  
 receipt\_items\_label.pack(anchor=tk.CENTER)  
 # Add a label for the invoice code  
 invoice\_label = tk.Label(self.receipt\_window, text="Invoice Code: {}".format(self.invoice\_code),  
 font=("Ariel", 14, "bold"), bg="white")  
 invoice\_label.pack(anchor=tk.W)  
 # Add a label for the date and time  
 date\_time\_label = tk.Label(self.receipt\_window, text="Date and Time: {}".format(self.date\_time.strftime("%Y-%m-%d %H:%M:%S")[:19]),  
 font=("Ariel", 14, "bold"), bg="white")  
 date\_time\_label.pack(anchor=tk.W)  
 # Add a label for the payment mode  
 MOP\_label = tk.Label(self.receipt\_window, text="Mode of Payment: {}".format(self.payment\_mode),  
 font=("Ariel", 14, "bold"), bg="white")  
 MOP\_label.pack(anchor=tk.W)  
  
 # Create a canvas to hold the cart items frame and add a vertical scrollbar  
 cart1\_canvas = tk.Canvas(self.receipt\_window, bg="white", bd=0, highlightthickness=0)  
 cart1\_scrollbar = tk.Scrollbar(cart1\_canvas, orient="vertical", command=cart1\_canvas.yview)  
 cart1\_canvas.configure(yscrollcommand=cart1\_scrollbar.set)  
 cart1\_canvas.pack(fill="both", expand=True)  
 cart1\_scrollbar.pack(side="right", fill="y",anchor=tk.W)  
  
 canvas\_frame = tk.Frame(cart1\_canvas, bg="white")  
 cart1\_canvas.create\_window((0, 0), window=canvas\_frame, anchor="nw")  
  
 name\_frame = tk.LabelFrame(canvas\_frame, text="Name",font=("Ariel", 14, "bold"))  
 name\_frame.pack(side='left')  
 qty\_frame = tk.LabelFrame(canvas\_frame, text="Quantity",font=("Ariel", 14, "bold"))  
 qty\_frame.pack(side='left')  
 total\_frame = tk.LabelFrame(canvas\_frame, text="Price",font=("Ariel", 14, "bold"))  
 total\_frame.pack(side='left')  
  
  
  
  
 # Add a label for each pizza item in the cart  
 for pizza in self.items:  
  
 name\_label = tk.Label(name\_frame, font=("Ariel", 15),bg="white",  
 text="{} ({} {})".format(pizza.name, pizza.type,pizza.size))  
 name\_label.pack(anchor=tk.W, pady=10)  
 qty\_label = tk.Label(qty\_frame, font=("Ariel", 15), bg="white",  
 text="{}".format(pizza.quantity.get()))  
 qty\_label.pack(anchor=tk.CENTER, pady=10)  
 price\_label = tk.Label(total\_frame, font=("Ariel", 15), bg="white",  
 text="{}".format(pizza.price \* int(pizza.quantity.get())))  
 price\_label.pack(anchor=tk.CENTER, pady=10)  
  
  
  
 # Add a label for the cart total  
 total\_label = tk.Label(self.receipt\_window, text="Total: ${:.2f}".format(self.total), font=("Ariel", 16),bg="white")  
 total\_label.pack()  
 button11\_frame = tk.Frame(self.receipt\_window,bg="white")  
 button11\_frame.pack()  
 # Add a "Print" button  
 print\_button = tk.Button(button11\_frame, text="Print",  
 command=lambda: self.print\_receipt(self.email\_id), width=10, bg="blue",  
 fg="white", font=("Arial", 10, "bold"))  
 print\_button.pack(pady=10,side="left")  
  
 # Add a "EMAIL" button  
 email\_button = tk.Button(button11\_frame, text="E-mail",  
 command=lambda: self.print\_receipt(self.email\_id), width=10, bg="blue",  
 fg="white", font=("Arial", 10, "bold"))  
 email\_button.pack(pady=10,side="left")  
  
 # add a label for rating input  
 rating\_label = tk.Label(self.receipt\_window, text="Please rate your order from 1 to 5:",  
 font=("Ariel", 15, 'bold'),bg="white")  
 rating\_label.pack(pady=10)  
  
 # add a rating scale  
 rating\_scale = tk.Scale(self.receipt\_window, from\_=0, to=5, orient=tk.HORIZONTAL,bg="white")  
 rating\_scale.config(width=20, length=200)  
 rating\_scale.pack(pady=10)  
  
 # add a submit button  
 submit\_button = tk.Button(self.receipt\_window, text="Submit",font=("Arial", 10, "bold"),  
 command=lambda: self.update\_order\_rating(rating\_scale.get()))  
 submit\_button.pack()  
 # Update the canvas to adjust to the size of the canvas\_frame frame  
 canvas\_frame.update\_idletasks()  
 cart1\_canvas.config(scrollregion=cart1\_canvas.bbox("all"))  
  
 def update\_order\_rating(self, rating):  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 # update the order rating in the database  
 try:  
 cursor.execute("UPDATE Orders SET Order\_Ratings = %s WHERE Order\_ID = %s", (rating, self.order\_id))  
 mydb.commit()  
 mydb.close()  
 tk.messagebox.showinfo("Success", "Order rating has been updated successfully!")  
  
 except Exception as e:  
 tk.messagebox.showerror("Error", "An error occurred while updating order rating: {}".format(str(e)))  
 finally:  
 self.receipt\_window.destroy()  
  
 def print\_receipt(self,email):  
 #BODY of the Confirmation Mail  
 body = """Dear Customer,  
  
Thank you for your order. Your receipt is attached below.  
  
Invoice Code: {}  
Date and Time: {}  
Mode of Payment: {}  
  
--------------------------------------------------  
{}  
--------------------------------------------------  
  
Total: ${:.2f}  
  
If you have any questions or concerns, please don't hesitate to contact us.  
  
Best regards,  
The Pizza Shop""".format(self.invoice\_code, self.date\_time.strftime("%Y-%m-%d %H:%M:%S")[:19], self.payment\_mode,  
 "\n".join("{} ({} {}), Quantity: {}, Price: ${:.2f}".format(pizza.name, pizza.type,  
 pizza.size,  
 pizza.quantity.get(),  
 pizza.price \* int(  
 pizza.quantity.get()))  
 for pizza in self.items),  
 self.total)  
  
 # Mailing The user name to the email  
 sender\_email = "friendscmu2022@gmail.com"  
 password = "oznhefehqurvpmjz"  
 message = body  
 msg = MIMEText(message)  
 msg['Subject'] = 'Order Confirmation'  
 msg['From'] = 'sender\_email'  
 msg['To'] = email  
 with smtplib.SMTP('smtp.gmail.com', 587) as server:  
 server.starttls()  
 server.login(sender\_email, password)  
 server.sendmail(sender\_email, email, msg.as\_string())  
 self.receipt\_window.destroy()  
  
# EMPLOYEE RECEIPT  
class EmpReceipt:  
 def \_\_init\_\_(self, items, total, date\_time, invoice\_code, payment\_mode, order\_id,parent):  
 self.items = items  
 self.total = total  
 self.date\_time = date\_time  
 self.invoice\_code = invoice\_code  
 self.payment\_mode = payment\_mode  
 self.order\_id = order\_id  
 self.parent = parent  
 # create the receipt window  
 self.receipt\_window = tk.Toplevel(parent)  
 self.receipt\_window.title("Receipt")  
 self.receipt\_window.configure(background="white")  
 self.receipt\_window.geometry("555x600")  
 # Add a label for the receipt items  
 receipt\_items\_label = tk.Label(self.receipt\_window, text="Receipt Items", font=("Ariel", 16, "bold"),  
 bg="white")  
 receipt\_items\_label.pack(anchor=tk.CENTER)  
 # Add a label for the invoice code  
 invoice\_label = tk.Label(self.receipt\_window, text="Invoice Code: {}".format(self.invoice\_code),  
 font=("Ariel", 14, "bold"), bg="white")  
 invoice\_label.pack(anchor=tk.W)  
 # Add a label for the date and time  
 date\_time\_label = tk.Label(self.receipt\_window,  
 text="Date and Time: {}".format(self.date\_time.strftime("%Y-%m-%d %H:%M:%S")[:19]),  
 font=("Ariel", 14, "bold"), bg="white")  
 date\_time\_label.pack(anchor=tk.W)  
 # Add a label for the payment mode  
 MOP\_label = tk.Label(self.receipt\_window, text="Mode of Payment: {}".format(self.payment\_mode),  
 font=("Ariel", 14, "bold"), bg="white")  
 MOP\_label.pack(anchor=tk.W)  
  
 # Create a canvas to hold the cart items frame and add a vertical scrollbar  
 cart1\_canvas = tk.Canvas(self.receipt\_window, bg="white", bd=0, highlightthickness=0)  
 cart1\_scrollbar = tk.Scrollbar(cart1\_canvas, orient="vertical", command=cart1\_canvas.yview)  
 cart1\_canvas.configure(yscrollcommand=cart1\_scrollbar.set)  
 cart1\_canvas.pack(fill="both", expand=True)  
 cart1\_scrollbar.pack(side="right", fill="y", anchor=tk.W)  
  
 canvas\_frame = tk.Frame(cart1\_canvas, bg="white")  
 cart1\_canvas.create\_window((0, 0), window=canvas\_frame, anchor="nw")  
  
 name\_frame = tk.LabelFrame(canvas\_frame, text="Name", font=("Ariel", 14, "bold"))  
 name\_frame.pack(side='left')  
 qty\_frame = tk.LabelFrame(canvas\_frame, text="Quantity", font=("Ariel", 14, "bold"))  
 qty\_frame.pack(side='left')  
 total\_frame = tk.LabelFrame(canvas\_frame, text="Price", font=("Ariel", 14, "bold"))  
 total\_frame.pack(side='left')  
  
 # Add a label for each pizza item in the cart  
 for pizza in self.items:  
 name\_label = tk.Label(name\_frame, font=("Ariel", 15), bg="white",  
 text="{} ({} {})".format(pizza.name, pizza.type, pizza.size))  
 name\_label.pack(anchor=tk.W, pady=10)  
 qty\_label = tk.Label(qty\_frame, font=("Ariel", 15), bg="white",  
 text="{}".format(pizza.quantity.get()))  
 qty\_label.pack(anchor=tk.CENTER, pady=10)  
 price\_label = tk.Label(total\_frame, font=("Ariel", 15), bg="white",  
 text="{}".format(pizza.price \* int(pizza.quantity.get())))  
 price\_label.pack(anchor=tk.CENTER, pady=10)  
  
 # Add a label for the cart total  
 total\_label = tk.Label(self.receipt\_window, text="Total: ${:.2f}".format(self.total), font=("Ariel", 16),  
 bg="white")  
 total\_label.pack()  
 button11\_frame = tk.Frame(self.receipt\_window, bg="white")  
 button11\_frame.pack()  
 # Add a "Print" button  
 print\_button = tk.Button(button11\_frame, text="Print",  
 command=lambda: self.print\_receipt(), width=10, bg="blue",  
 fg="white", font=("Arial", 10, "bold"))  
 print\_button.pack(pady=10, side="left")  
  
 # Add a "EMAIL" button  
 email\_button = tk.Button(button11\_frame, text="E-mail",  
 command=lambda: self.print\_receipt(), width=10, bg="blue",  
 fg="white", font=("Arial", 10, "bold"))  
 email\_button.pack(pady=10, side="left")  
  
 # add a label for rating input  
 rating\_label = tk.Label(self.receipt\_window, text="Please rate your order from 1 to 5:",  
 font=("Ariel", 15, 'bold'), bg="white")  
 rating\_label.pack(pady=10)  
  
 # add a rating scale  
 rating\_scale = tk.Scale(self.receipt\_window, from\_=0, to=5, orient=tk.HORIZONTAL, bg="white")  
 rating\_scale.config(width=20, length=200)  
 rating\_scale.pack(pady=10)  
  
 # add a submit button  
 submit\_button = tk.Button(self.receipt\_window, text="Submit", font=("Arial", 10, "bold"),  
 command=lambda: self.update\_order\_rating(rating\_scale.get()))  
 submit\_button.pack()  
 # Update the canvas to adjust to the size of the canvas\_frame frame  
 canvas\_frame.update\_idletasks()  
 cart1\_canvas.config(scrollregion=cart1\_canvas.bbox("all"))  
  
 def update\_order\_rating(self, rating):  
  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 # update the order rating in the database  
 try:  
 cursor.execute("UPDATE Orders SET Order\_Ratings = %s WHERE Order\_ID = %s", (rating, self.order\_id))  
 mydb.commit()  
 mydb.close()  
  
 tk.messagebox.showinfo("Success", "Order rating has been updated successfully!")  
 except Exception as e:  
 tk.messagebox.showerror("Error", "An error occurred while updating order rating: {}".format(str(e)))  
 finally:  
 self.receipt\_window.focus()  
  
  
 def print\_receipt(self):  
 # Print the receipt  
 print("Invoice Code: {}".format(self.invoice\_code))  
 print("Date and Time: {}".format(self.date\_time))  
 print("Mode of Payment: {}".format(self.payment\_mode))  
 print("--------------------------------------------------")  
 for pizza in self.items:  
 print("{} ({} {}), Quantity: {}, Price: ${:.2f}".format(pizza.name, pizza.type, pizza.size,  
 pizza.quantity.get(),  
 pizza.price \* int(pizza.quantity.get())))  
 print("--------------------------------------------------")  
 print("Total: ${:.2f}".format(self.total))  
 self.parent.destroy()  
 self.receipt\_window.destroy()  
  
 ## Customer MainMenu  
  
# MAIN MENU  
class MainMenu:  
 def \_\_init\_\_(self, id, name):  
 ## Customer Name And ID  
 self.id = id  
 self.name = name  
  
 ### It creates the main window that will contain all of the other graphical elements  
 # Main window screen  
 self.main\_window = tk.Tk()  
 self.main\_window.attributes('-fullscreen', True)  
 self.main\_window.title("Employee Main Menu")  
 ## Creating an instances of Cart class to store items added to carts.  
 self.cart = Cart()  
  
 ## BacgGround Image  
 img222 = Image.open("LOGIN.jpg")  
 screen\_width = self.main\_window.winfo\_screenwidth()  
 screen\_height = self.main\_window.winfo\_screenheight()  
 img222 = img222.resize((screen\_width, screen\_height))  
 self.BGG = ImageTk.PhotoImage(img222)  
 self.imageL = tk.Label(self.main\_window, image=self.BGG)  
 self.imageL.place(x=0, y=0)  
  
 self.store\_label = tk.Label(self.main\_window, text="Welcome To The Charcoal Pizza Store!!",  
 font=("Times New Roman", 30, "bold"),  
 bg="#161513",fg="white")  
 self.store\_label.place(relx=0.25, rely=0.05,anchor=tk.CENTER)  
  
  
  
 self.firstframe = tk.Frame(self.main\_window)  
 self.firstframe.place(relx=0.2, rely=0.5, anchor=tk.CENTER)  
 self.secondframe = tk.Frame(self.main\_window)  
 self.secondframe.place(relx=0.5, rely=0.5, anchor=tk.CENTER)  
 self.thirdframe = tk.Frame(self.main\_window)  
 self.thirdframe.place(relx=0.8, rely=0.5, anchor=tk.CENTER)  
  
 #Loading cart image  
 cart\_image = Image.open(  
 "Cart.png") ## adding th image  
 resized\_cart\_image = cart\_image.resize((75, 75)) # resizeing the image  
 cart\_photo = ImageTk.PhotoImage(resized\_cart\_image)  
 # assigning cart image to a button  
 cart\_items\_Button = tk.Button(self.main\_window, image=cart\_photo  
 , command=lambda: self.view\_cart(), height=75, width=75,  
 bg="#161513", relief="flat")  
 cart\_items\_Button.place(relx=0.95, rely=0.85, anchor=tk.CENTER)  
  
 #Loading Account info logo  
 AI\_image = Image.open(  
 "AI.png") ## adding th image  
 resized\_AI\_image = AI\_image.resize((75, 75)) # resizeing the image  
 AII\_photo = ImageTk.PhotoImage(resized\_AI\_image)  
 # Assigning AI logo to Ai button  
 AI\_Button= tk.Button(self.main\_window, image=AII\_photo  
 , command=lambda: self.edit\_customer\_details(self.id), height=75, width=75,  
 bg="#161513", relief="flat")  
 AI\_Button.place(relx=0.95, rely=0.05, anchor=tk.CENTER)  
  
 ## Loading Pizza data & Beverages Sides  
 # Loading Pizza data in pizza class and storing the instances in the below lists  
 self.pizzasV = [] # loading veg regular  
 self.pizzasVL = [] # loading veg large  
 self.pizzasNV = [] # loading non veg regular  
 self.pizzasNVL = [] # loading non veg large  
 self.bev = [] # loading bev  
 self.sides = [] # loading sides  
 self.load\_pizza\_data()  
  
 self.pizza\_button = tk.Button(self.firstframe, text="Vegetarian Menu",  
 command=lambda: self.open\_veg\_menu(self.main\_window, self.pizzasV, self.pizzasVL,  
 self.cart, self.id, self.name), width=18,  
 height=3, bg="green", fg="white", font=("Arial", 20, "bold")).pack()  
 self.pizza\_button = tk.Button(self.secondframe, text="Non-Vegetarian Menu",  
 command=lambda: self.open\_non\_veg\_menu(self.main\_window, self.pizzasNV,  
 self.pizzasNVL, self.cart, self.id,  
 self.name), width=18, height=3, bg="green",  
 fg="white", font=("Arial", 20, "bold")).pack()  
 self.pizza\_button = tk.Button(self.thirdframe, text="Beverages & Sides",  
 command=lambda: self.open\_bevsides\_menu(self.main\_window, self.bev, self.sides,  
 self.cart, self.id, self.name), width=18,  
 height=3, bg="green", fg="white", font=("Arial", 20, "bold")).pack()  
  
  
 ##LogOUT BUtton  
 self.logout\_button = tk.Button(self.main\_window, text="LogOut", command=self.logout, width=8, bg="red",  
 fg="white", font=("Arial", 15, "bold"))  
 self.logout\_button.place(relx=0.85, rely=0.05, anchor=tk.CENTER)  
  
 ##close BUtton  
 self.close\_button = tk.Button(self.main\_window, text="Close", command=self.close, width=8, bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.place(relx=0.76, rely=0.05, anchor=tk.CENTER)  
  
 # Add a button to show the order history  
 history\_button = tk.Button(self.main\_window, text="Order History", command=self.show\_order\_history, bg="red",  
 fg="white", font=("Arial", 15, "bold"))  
 history\_button.place(relx=0.65, rely=0.05, anchor=tk.CENTER)  
  
 tk.mainloop()  
  
 def show\_order\_history(self):  
  
 # Connect to the database  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 # Connect to the database and retrieve all the orders  
 cursor.execute(  
 "SELECT Orders.Order\_ID, Orders.Order\_Date, Orders.Order\_Total, Order\_Status.Order\_Status FROM Orders INNER JOIN Order\_Status ON "  
 "Orders.Order\_ID = Order\_Status.Order\_ID WHERE Orders.Customer\_ID = %s ORDER BY Orders.Order\_ID DESC",  
 (self.id,))  
 orders = cursor.fetchall()  
  
 # Create a new window to display the orders  
 window = tk.Toplevel(self.main\_window)  
 window.title("Order History")  
 window.geometry("1000x700")  
 window.configure(background="#7e4b1c")  
 # Add a label for the order history  
 label = tk.Label(window, text="Order History", font=("default", 16))  
 label.pack(pady=10)  
 s = ttk.Style()  
 s.theme\_use('classic')  
 # Configure the style of Heading in Treeview widget  
 s.configure('Treeview.Heading', background="#ff8c00", font=("Ariel", 15, "bold"))  
 s.configure('Treeview', font=('Arial', 12))  
  
 # Add a treeview to display the orders  
 columns = ("Order ID", "Order Date", "Order Total", "Order Status")  
 tree = ttk.Treeview(window, columns=columns, show="headings", height=20)  
  
 tree.column("Order Date", width=300)  
 for col in columns:  
 tree.heading(col, text=col)  
 if col in ["Order ID", "Order Total", "Order Status"]:  
 tree.column(col, anchor="center")  
 tree.pack()  
  
 # Define the tags for each status  
 tree.tag\_configure("pending", background="yellow")  
 tree.tag\_configure("preparing", background="orange")  
 tree.tag\_configure("ready", background="green")  
 tree.tag\_configure("cancelled", background="red")  
  
 # Add the orders to the treeview  
 for order in orders:  
 order\_id, order\_date, order\_total, order\_status = order  
 order\_total\_formatted = f"{order\_total:.2f}"  
 order\_item\_id = f"{order\_id}\_item"  
 tree.insert("", "end", values=(order\_id, order\_date, order\_total\_formatted, order\_status), iid=order\_id,  
 tags=(order\_status.lower(),))  
  
 # Retrieve the order items for the current order  
 cursor.execute(  
 "SELECT CONCAT(Items.item\_name, ' ', IFNULL(CONCAT(' ', Items.item\_size), '')) AS name, Order\_Item.item\_Quantity, (Order\_Item.item\_Quantity \* Items.item\_price) AS price FROM Items INNER JOIN Order\_Item ON Items.item\_ID = Order\_Item.item\_ID WHERE Order\_Item.Order\_ID = %s",  
 (order\_id,)  
 )  
  
 order\_items = cursor.fetchall()  
  
 # Add the order items to the treeview as sub-items  
 for index, order\_item in enumerate(order\_items, 1):  
 name, item\_quantity, price = order\_item  
 item\_id = f"{order\_item\_id}\_{index}"  
  
 tree.insert(order\_id, "end", values=("", f"{name} ({item\_quantity})", "{:.2f}".format(price)),  
 iid=item\_id)  
  
 def clear\_selection(event):  
 selected\_items = event.widget.selection()  
 for item in selected\_items:  
 event.widget.selection\_remove(item)  
  
 # Bind the TreeviewSelect event to clear\_selection  
 tree.bind("<<TreeviewSelect>>", clear\_selection)  
  
 close\_button = tk.Button(window, text="Close", font=("Ariel", 20, "bold"), background="#ffebcd",  
 command=window.destroy)  
 close\_button.pack(side="right")  
 mydb.close()  
  
 def close(self): ## closing the application  
 self.main\_window.destroy()  
  
 def logout(self):  
 # switch to login page  
 self.main\_window.destroy()  
 LoginPage()  
  
 ## Veg NonVeg BevSides - calling the class  
 def open\_veg\_menu(self, parent, pizzasV, pizzasVL, Cart, id, name):  
 self.veg\_menu = VegMenu(parent, pizzasV, pizzasVL, Cart, id, name)  
  
 def open\_non\_veg\_menu(self, parent, pizzasNV, pizzasNVL, Cart, id, name):  
 self.non\_veg\_menu = NonVegMenu(parent, pizzasNV, pizzasNVL, Cart, id, name)  
  
 def open\_bevsides\_menu(self, parent, bev, sides, Cart, id, name):  
 self.bevsides\_menu = BevSideMenu(parent, bev, sides, Cart, id, name)  
  
 def get\_cart\_total(self):  
 return self.cart.get\_total()  
  
 def checkout(self):  
 checkout\_window = Checkout(self.cart, self.id)  
  
 ## Loading the data of the menu itesm from the database  
 def load\_pizza\_data(self):  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 cursor = mydb.cursor()  
 # Retrieve the pizza data from the database Vege-R  
 cursor.execute("SELECT \* FROM Items WHERE item\_type = %s AND item\_size = %s", ("Vegetarian", "Regular"))  
 rows = cursor.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 pizza = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.pizzasV.append(pizza)  
  
 # Retrieve the pizza data from the database Veg-L  
 cursor.execute("SELECT \* FROM Items WHERE item\_type = %s AND item\_size = %s", ("Vegetarian", "Large"))  
 rows = cursor.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 pizza = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.pizzasVL.append(pizza)  
  
 # Retrieve the pizza data from the database NON Veg-R  
 cursor.execute("SELECT \* FROM Items WHERE item\_type = %s AND item\_size = %s", ("NonVegetarian", "Regular"))  
 rows = cursor.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 pizza = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.pizzasNV.append(pizza)  
  
 # Retrieve the pizza data from the database NON Veg-L  
 cursor.execute("SELECT \* FROM Items WHERE item\_type = %s AND item\_size = %s", ("NonVegetarian", "Large"))  
 rows = cursor.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 pizza = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.pizzasNVL.append(pizza)  
  
 # Retrieve the pizza data from the database BevSide  
 cursor.execute(  
 "SELECT item\_id, item\_name, item\_price, item\_type, IFNULL(CONCAT(' ', item\_size), '') FROM Items WHERE item\_type = %s",  
 ("Beverages",))  
 rows = cursor.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 bev = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.bev.append(bev)  
  
 # Retrieve the pizza data from the database Side  
 cursor.execute(  
 "SELECT item\_id, item\_name, item\_price, item\_type, IFNULL(CONCAT(' ', item\_size), '') FROM Items WHERE item\_type = %s",  
 ("Sides",))  
  
 rows = cursor.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 side = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.sides.append(side)  
  
 mydb.close()  
  
 def view\_cart(self):  
 # Create a new window for the cart  
 self.cart\_window = tk.Toplevel(self.main\_window)  
 self.cart\_window.geometry("700x600")  
 self.cart\_window.title("View Cart")  
 self.cart\_window.configure(background="#3e4c4f")  
 self.cart\_window.resizable(width=False, height=False)  
  
 frame1 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame1.pack()  
 # Create a canvas to hold the cart items frame and add a vertical scrollbar  
 cart\_canvas = tk.Canvas(self.cart\_window, bg="#3e4c4f", bd=0, highlightthickness=0)  
 cart\_scrollbar = tk.Scrollbar(cart\_canvas, orient="vertical", command=cart\_canvas.yview)  
 cart\_canvas.configure(yscrollcommand=cart\_scrollbar.set)  
 cart\_canvas.pack(fill="both", expand=True)  
 cart\_scrollbar.pack(side="right", fill="y")  
  
 frame3 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame3.pack()  
  
 # Add a label for the cart items  
 cart\_items\_label = tk.Label(frame1, text="Cart Items", font=("Ariel", 25, "bold"),bg="#3e4c4f",fg="white")  
 cart\_items\_label.pack()  
  
 # Add a frame for the cart items  
 # Create a frame to hold the cart items and add it to the canvas  
 cart\_items\_frame = tk.Frame(cart\_canvas, bg="#3e4c4f")  
 cart\_canvas.create\_window((0, 0), window=cart\_items\_frame, anchor="nw")  
 self.cart\_items\_frame = cart\_items\_frame  
 # Add a label for the cart total  
 self.cart\_total\_label = tk.Label(frame3, text="Total: ${:.2f}".format(self.get\_cart\_total()),  
 font=("Ariel", 20, "bold"),bg="#3e4c4f",fg="white")  
 self.cart\_total\_label.pack()  
  
 # Add a "Checkout" button  
 checkout\_button = tk.Button(frame3, text="Checkout", command=self.checkout, width=15, bg="green",  
 fg="white", font=("Arial", 20, "bold"))  
 checkout\_button.pack(pady=10)  
  
 ##close BUtton  
 self.close\_button = tk.Button(frame3, text="Close", command=self.cart\_window.destroy, width=8,  
 bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.pack(pady=10)  
 # Add a label and Spinbox for each pizza item in the cart  
 for i, pizza in enumerate(self.cart.get\_items()):  
 # Create a frame to hold the product label and Spinbox  
 product\_frame = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame.grid(row=i, column=0, sticky='w', pady=10)  
  
 product\_frame1 = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame1.grid(row=i, column=2, sticky='w', pady=10)  
  
 pizza.product\_frame = product\_frame  
 pizza.product\_frame1 = product\_frame1  
  
 # Add a label for the product name, type, and size  
 product\_label = tk.Label(product\_frame, text="{} ({} {})".format(pizza.name, pizza.type, pizza.size),  
 font=('Ariel', 15, "bold"),bg="#3e4c4f",fg="white")  
 product\_label.grid(row=0, column=0, sticky='w')  
  
 # Add a Spinbox for the product quantity  
 quantity\_spinbox = tk.Spinbox(product\_frame1, from\_=1, to=10, width=5, textvariable=pizza.quantity,  
 command=lambda: self.update\_cart\_total(), font=('Ariel', 15, "bold"))  
 quantity\_spinbox.grid(row=0, column=0, padx=(10, 0))  
  
 # Add a label for the product price  
 price\_label = tk.Label(product\_frame, text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())),  
 font=('Ariel', 15, "bold"),bg="#3e4c4f",fg="white")  
 price\_label.grid(row=0, column=1, padx=(10, 0))  
  
 # Add a "Remove" button for the pizza item  
 remove\_button = tk.Button(product\_frame1, text="Remove",  
 command=lambda pizza\_item=pizza: self.remove\_from\_cart(pizza\_item),  
 font=('Ariel', 15, "bold"), bg="red")  
 remove\_button.grid(row=0, column=1, padx=(10, 0))  
  
 # Add the price label to the pizza item for later use  
 pizza.price\_label = price\_label  
  
 # Update the canvas to adjust to the size of the cart items frame  
 cart\_items\_frame.update\_idletasks()  
 cart\_canvas.config(scrollregion=cart\_canvas.bbox("all"))  
  
 def update\_cart\_total(self):  
 # Update the price labels for each pizza item in the cart  
 for pizza in self.cart.get\_items():  
 pizza.price\_label.config(text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())))  
  
 # Update the cart total label with the new total  
 self.cart\_total\_label.config(text="Total: ${:.2f}".format(self.get\_cart\_total()))  
  
 def remove\_from\_cart(self, pizza\_item):  
 # Check if the pizza item is in the items list  
 if pizza\_item in self.cart.items:  
 # Remove the pizza item from the cart's items list  
 self.cart.items.remove(pizza\_item)  
 # Update the cart items and total labels  
 self.update\_cart\_total()  
 # Destroy the product frames associated with the pizza item  
 pizza\_item.product\_frame.destroy()  
 pizza\_item.product\_frame1.destroy()  
  
 #Customer Account Information: Editing  
 def edit\_customer\_details(self,id):  
 self.iid = id  
 # Create a new window for editing customer details  
 edit\_window = tk.Toplevel(self.main\_window)  
 edit\_window.title("Edit Customer Details")  
 edit\_window.configure(background="#3a4447")  
  
 fframe1 = tk.Frame(edit\_window,background="#3a4447")  
 fframe1.pack()  
 fflabel = tk.Label(fframe1,text="Account Information",font=("Ariel",25,"bold"),bg="#3a4447",fg="white")  
 fflabel.pack(pady=10,padx=10)  
 fframe2 = tk.Frame(edit\_window,background="#212129")  
 fframe2.pack(pady=10,padx=10)  
 fframe3 = tk.Frame(edit\_window,background="#3a4447")  
 fframe3.pack(pady=10,padx=10)  
 for widget in fframe2.winfo\_children():  
 widget.destroy()  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
  
 # Perform database operations  
 cursor = mydb.cursor()  
 # Retrieve customer details from database based on search criteria  
 search\_term = self.iid  
 cursor.execute("SELECT \* FROM CustomersInformation WHERE Customer\_ID=%s OR \  
 Phone\_Number=%s OR Email\_ID=%s", (search\_term, search\_term, search\_term))  
 result = cursor.fetchone()  
 mydb.close()  
 if result:  
 # Display customer information in a form for editing  
 self.customer\_id = tk.StringVar(value=result[0])  
 self.first\_name = tk.StringVar(value=result[1])  
 self.middle\_name = tk.StringVar(value=result[2])  
 self.last\_name = tk.StringVar(value=result[3])  
 self.address = tk.StringVar(value=result[4])  
 self.city = tk.StringVar(value=result[5])  
 self.state = tk.StringVar(value=result[6])  
 self.zipcode = tk.StringVar(value=result[7])  
 self.phone\_number = tk.StringVar(value=result[8])  
 self.username = tk.StringVar(value=result[9])  
 self.email\_id = tk.StringVar(value=result[11])  
  
 frame1 = tk.Frame(fframe2,bg="black")  
 frame1.pack()  
  
 # Create a form for editing customer details  
 customer\_id\_label = tk.Label(frame1, text="Customer ID:", font=("Arial", 11,"bold"),bg="black",fg="red")  
 customer\_id\_label.grid(row=0, column=0, padx=5, pady=5,sticky="W")  
 customer\_id\_entry = tk.Entry(frame1, textvariable=self.customer\_id, state='readonly', font=("Arial", 12,"bold"))  
 customer\_id\_entry.grid(row=0, column=1, padx=5, pady=5)  
  
 first\_name\_label = tk.Label(frame1, text="First Name\*:", font=("Arial", 11,"bold"),bg="black",fg="white")  
 first\_name\_label.grid(row=1, column=0, padx=5, pady=5,sticky="W")  
 first\_name\_entry = tk.Entry(frame1, textvariable=self.first\_name, font=("Arial", 12,"bold"))  
 first\_name\_entry.grid(row=1, column=1, padx=5, pady=5)  
  
 middle\_name\_label = tk.Label(frame1, text="Middle Name:", font=("Arial", 11,"bold"),bg="black",fg="white")  
 middle\_name\_label.grid(row=2, column=0, padx=5, pady=5,sticky="W")  
 middle\_name\_entry = tk.Entry(frame1, textvariable=self.middle\_name, font=("Arial", 12,"bold"))  
 middle\_name\_entry.grid(row=2, column=1, padx=5, pady=5)  
  
 last\_name\_label = tk.Label(frame1, text="Last Name\*:", font=("Arial", 11,"bold"),bg="black",fg="white")  
 last\_name\_label.grid(row=3, column=0, padx=5, pady=5,sticky="W")  
 last\_name\_entry = tk.Entry(frame1, textvariable=self.last\_name, font=("Arial", 12,"bold"))  
 last\_name\_entry.grid(row=3, column=1, padx=5, pady=5)  
  
 address\_label = tk.Label(frame1, text="Address:", font=("Arial", 11,"bold"),bg="black",fg="white")  
 address\_label.grid(row=4, column=0, padx=5, pady=5,sticky="W")  
 address\_entry = tk.Entry(frame1, textvariable=self.address, font=("Arial", 12,"bold"))  
 address\_entry.grid(row=4, column=1, padx=5, pady=5)  
  
 city\_label = tk.Label(frame1, text="City:", font=("Arial", 11,"bold"),bg="black",fg="white")  
 city\_label.grid(row=5, column=0, padx=5, pady=5,sticky="W")  
 city\_entry = tk.Entry(frame1, textvariable=self.city, font=("Arial", 12,"bold"))  
 city\_entry.grid(row=5, column=1, padx=5, pady=5)  
  
 usa\_state\_names = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL", "IN", "IA",  
 "KS", "KY",  
 "LA", "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM", "NY",  
 "NC", "ND",  
 "OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI",  
 "WY"]  
  
 state\_label = tk.Label(frame1, text="State:", font=("Arial", 11, "bold"), bg="black", fg="white")  
 state\_label.grid(row=6, column=0, padx=5, pady=5, sticky="W")  
  
 State\_combo = ttk.Combobox(frame1, values=usa\_state\_names,textvariable=self.state, font=("Arial", 12), width=18)  
 State\_combo.grid(row=6,column=1, padx=5, pady=5)  
  
 zipcode\_label = tk.Label(frame1, text="Zipcode:", font=("Arial", 11,"bold"),bg="black",fg="white")  
 zipcode\_label.grid(row=7, column=0, padx=5, pady=5,sticky="W")  
 zipcode\_entry = tk.Entry(frame1, textvariable=self.zipcode, font=("Arial", 12,"bold"))  
 zipcode\_entry.grid(row=7, column=1, padx=5, pady=5)  
  
 phone\_number\_label = tk.Label(frame1, text="Phone Number\*:", font=("Arial", 11,"bold"),bg="black",fg="white")  
 phone\_number\_label.grid(row=8, column=0, padx=5, pady=5,sticky="W")  
 phone\_number\_entry = tk.Entry(frame1, textvariable=self.phone\_number, font=("Arial", 12,"bold"))  
 phone\_number\_entry.grid(row=8, column=1, padx=5, pady=5)  
  
 username\_label = tk.Label(frame1, text="Username:", font=("Arial", 11,"bold"),bg="black",fg="red")  
 username\_label.grid(row=9, column=0, padx=5, pady=5,sticky="W")  
 username\_entry = tk.Entry(frame1, textvariable=self.username, state='readonly', font=("Arial", 12,"bold"))  
 username\_entry.grid(row=9, column=1, padx=5, pady=5)  
  
 email\_id\_label = tk.Label(frame1, text="Email ID\*:", font=("Arial", 11, "bold"), bg="black", fg="white")  
 email\_id\_label.grid(row=10, column=0, padx=5, pady=5,sticky="W")  
 email\_id\_entry = tk.Entry(frame1, textvariable=self.email\_id, font=("Arial", 12,"bold"))  
 email\_id\_entry.grid(row=10, column=1, padx=5, pady=5)  
  
 redlabel= tk.Label(frame1,text="\*Please note that fields in red cannot be changed. Thank you.",font=("Arial", 8,"bold"),fg="red",bg="black")  
 redlabel.grid(row=11, column=0, padx=5, pady=5,columnspan=2)  
 llabel = tk.Label(frame1,  
 text="\*Kindly note that all '\*' fields are mandatory and must be filled",  
 font=("Arial", 8, "bold"),fg="red",bg="black")  
 llabel.grid(row=12, column=0, padx=5, pady=5,columnspan=2)  
  
  
 # Create a button to save changes to the database  
 def save\_changes():  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 cursor = mydb.cursor()  
 ## phone number format chenk and first and last name check  
 if len(first\_name\_entry.get()) == 0 or len(last\_name\_entry.get()) == 0:  
 tk.messagebox.showinfo("Error", "First Name and Last Name Fields Must Not Be Empty")  
 return  
 phone = phone\_number\_entry.get()  
 if not phone.isdigit():  
 tk.messagebox.showinfo("Error", "Phone Number should contain only numeric characters")  
 edit\_window.focus()  
 return  
 if len(phone\_number\_entry.get()) != 10:  
 tk.messagebox.showinfo("Error", "Phone Number should be in 10 digits only ")  
 edit\_window.focus()  
 return  
 # Check if the email ID matches the required format.  
 regex = re.compile('^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$')  
 if not regex.match(email\_id\_entry.get()):  
 tk.messagebox.showinfo("Error", "Email Should be in Proper format\n Eg: abc@pizzastore.com ")  
 edit\_window.focus()  
 return  
 ## Formats and numerics and alphabits check  
 # Validate zip code format  
 zip\_code = zipcode\_entry.get()  
 if zip\_code and not zip\_code.isnumeric():  
 tk.messagebox.showerror("Invalid Zip Code", "Please enter a valid zip code")  
 return  
 city = city\_entry.get()  
 city\_pattern = r'^[a-zA-Z\s]+$'  
 if city and not re.match(city\_pattern, city):  
 tk.messagebox.showerror("Invalid City Name", "Please enter a valid city name")  
 return  
 state = State\_combo.get()  
 if state and (not state.isalpha() or len(state) != 2):  
 tk.messagebox.showerror("Invalid State Code", "Please enter a valid two-letter state code")  
 return  
  
 query = f"UPDATE CustomersInformation SET \  
 First\_Name='{first\_name\_entry.get().strip()}', \  
 Middle\_Name='{middle\_name\_entry.get().strip()}', \  
 Last\_Name='{last\_name\_entry.get().strip()}', \  
 Address='{address\_entry.get()}', \  
 City='{city\_entry.get().strip()}', \  
 State='{State\_combo.get().strip()}', \  
 ZipCode='{zipcode\_entry.get().strip()}', \  
 Phone\_Number='{phone\_number\_entry.get().strip()}', \  
 Username='{username\_entry.get()}', \  
 Email\_ID='{email\_id\_entry.get().strip()}' \  
 WHERE Customer\_ID='{customer\_id\_entry.get()}'"  
 cursor.execute(query)  
 mydb.commit()  
 mydb.close()  
 tk.messagebox.showinfo("Success", "Changes saved to database.")  
 edit\_window.destroy()  
  
 save\_button = tk.Button(fframe3, text="Save Changes", command=save\_changes, font=("Ariel", 15, "bold"),bg="green",fg="white")  
 save\_button.pack()  
 else:  
 tk.messagebox.showwarning("Warning", "No customer found with the provided search criteria.")  
  
  
 # Create a button to close the edit window  
 close\_button = tk.Button(edit\_window, text="Close", font=("Ariel", 15, "bold"), background="red", fg="white", command=edit\_window.destroy)  
 close\_button.pack(side="bottom")  
  
# Notes:  
##The self.cart\_items\_frame instance variable is necessary to keep track of the frame holding the pizza items in the cart.  
##When the view\_cart method is called, it creates a cart\_items\_frame frame and adds the pizza items to it.  
## When the remove\_from\_cart method is called to remove a pizza item,  
## it needs to access the cart\_items\_frame frame to search for and remove the appropriate pizza item widget.  
  
## If we did not use self.cart\_items\_frame and just used cart\_items\_frame as a local variable in view\_cart,  
## it would not be accessible in the remove\_from\_cart method because it would not be defined in that method's scope.  
## By assigning cart\_items\_frame to self.cart\_items\_frame,  
##we make it an instance variable that is accessible throughout the class, including in the remove\_from\_cart method.  
  
## So, using self.cart\_items\_frame as an instance variable is necessary to keep track of the  
##cart items frame and access it from other methods in the class.  
# IMPORTS  
import random  
import calendar  
import csv  
import datetime as dt  
import hashlib  
import smtplib  
import time  
import tkinter as tk  
import tkinter.messagebox  
from email.mime.text import MIMEText  
from tkinter import filedialog  
from tkinter import ttk  
import matplotlib.pyplot as plt  
import mysql.connector  
import pandas as pd  
from PIL import Image, ImageTk  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
from tkcalendar import Calendar, DateEntry  
import matplotlib.cm as cm  
import regex as re  
  
  
### Employee admin### Employee admin  
### Employee admin### Employee admin### Employee admin### Employee admin  
### Employee admin### Employee admin### Employee admin### Employee admin### Employee admin  
### Employee admin### Employee admin### Employee admin### Employee admin### Employee admin### Employee admin  
### Employee admin### Employee admin### Employee admin### Employee admin### Employee admin### Employee admin### Employee admin  
# EMPLOYEE SCREENS  
class EmpMenu():  
 def \_\_init\_\_(self, id, name):  
 self.id = id  
 self.name = name  
 self.main\_window = tk.Tk()  
 self.main\_window.title("Admin-Menu")  
 self.main\_window.attributes("-fullscreen", True)  
 self.main\_window.configure(background="#7e4b1c")  
  
 # Create the frame and center it  
 firstframe = tk.Frame(self.main\_window,bg="#7e4b1c")  
 firstframe.place(relx=0.1, rely=0.4)  
 secondframe = tk.Frame(self.main\_window)  
 secondframe.place(relx=0.4, rely=0.4)  
 self.thirdframe = tk.Frame(self.main\_window,bg="#7e4b1c")  
 self.thirdframe.place(relx=0.7, rely=0.4)  
  
 # Create the buttons  
 orderButton = tk.Button(firstframe, text="New-Order", width=14, relief='raised',  
 font=("default", 20, "bold"), bg="green", fg="yellow", command=self.new\_order)  
 orderButton.pack(ipady=10, ipadx=10)  
 pending\_button = tk.Button(secondframe, text="Pending Orders", command=lambda: self.show\_orders("Pending"),  
 width=14, relief='raised', font=("default", 20, "bold"), bg="#fcf75e")  
 pending\_button.pack()  
 preparing\_button = tk.Button(secondframe, text="Preparing Orders",  
 command=lambda: self.show\_orders("Preparing"), width=14, relief='raised',  
 font=("default", 20, "bold"), bg="orange")  
 preparing\_button.pack()  
 ready\_button = tk.Button(secondframe, text="Ready Orders", command=lambda: self.show\_orders("Ready"),  
 width=14, relief='raised', font=("default", 20, "bold"), bg="#32cd32")  
 ready\_button.pack()  
 cancelled\_button = tk.Button(secondframe, text="Cancelled Orders", command=lambda: self.show\_orders("Cancelled"),  
 width=14, relief='raised', font=("default", 20, "bold"), bg="red")  
 cancelled\_button.pack()  
  
 ReportsButton = tk.Button(self.thirdframe, text="Reports", width=14, relief='raised',  
 font=("default", 20, "bold"), bg="#ffebcd", command=self.show\_reports)  
 ReportsButton.pack(ipady=10, ipadx=10)  
 self.reportframe = tk.Frame(self.thirdframe)  
 self.reportframe.pack(padx=5)  
 edit\_customer = tk.Button(firstframe, text="Customers", width=12,  
 command=self.edit\_customer\_details, font=("Ariel", 18, "bold"), background="#ffebcd")  
 edit\_customer.pack(pady=10)  
  
  
 self.close\_button = tk.Button(self.main\_window, text="Close", command=self.main\_window.destroy, width=10,  
 fg="white",  
 font=("Arial", 15, "bold"), bg="red")  
 self.close\_button.place(relx=0.76, rely=0.9, anchor=tk.CENTER)  
  
 # Start the main loop  
 self.main\_window.mainloop()  
 ####### Reports  
 ####### Reports####### Reports####### Reports  
 def show\_reports(self):  
 if hasattr(self, 'show\_reports\_window') and self.show\_reports\_window.winfo\_exists():  
 self.show\_reports\_window.tkraise()  
 return  
  
 # Create a new window to display the reports  
 window = self.reportframe  
 self.show\_reports\_window = window  
 # window.title("Reports")  
 window.configure(background="#7e4b1c")  
 # Create buttons for different types of reports  
 customer\_reports\_button = tk.Button(self.show\_reports\_window, text="Customer Reports",  
 command=self.show\_customer\_reports  
 , font=("Ariel", 15, "bold"), background="#ffebcd")  
 sales\_reports\_button = tk.Button(self.show\_reports\_window, text="Sales Reports",  
 command=self.show\_sales\_reports  
 , font=("Ariel", 15, "bold"), background="#ffebcd")  
 sales1\_visual\_button = tk.Button(self.show\_reports\_window, text="Date-Dashboard",  
 command=lambda :self.view\_sales\_reports("Daily")  
 , font=("Ariel", 15, "bold"), background="#ffebcd")  
 sales\_visual\_button = tk.Button(self.show\_reports\_window, text="Month- Dashboard",  
 command=lambda: self.view\_sales\_reports("Monthly")  
 , font=("Ariel", 15, "bold"), background="#ffebcd")  
  
 # Add the buttons to the window  
 customer\_reports\_button.pack(pady=10)  
 sales\_reports\_button.pack(pady=10)  
 sales1\_visual\_button.pack(pady=10)  
 sales\_visual\_button.pack(pady=10)  
  
 close\_button = tk.Button(window, text="Close", font=("Ariel", 15, "bold"), background="red", fg="white",  
 command= self.dynamic1)  
 close\_button.pack(side="bottom")  
  
 def dynamic1(self):  
 self.reportframe.destroy() # destroy the reportframe widget  
 self.reportframe = tk.Frame(self.thirdframe)  
 self.reportframe.pack(padx=5) # create and show a new reportframe widget  
 self.thirdframe.update()  
  
 def edit\_customer\_details(self):  
 # Create a new window for editing customer details  
 edit\_window = tk.Toplevel(self.main\_window)  
 edit\_window.title("Edit Customer Details")  
 edit\_window.configure(background="#7e4b1c")  
  
 fframe1 = tk.Frame(edit\_window,background="#7e4b1c")  
 fframe1.pack()  
 fframe2 = tk.Frame(edit\_window,background="#7e4b1c")  
 fframe2.pack()  
 fframe3 = tk.Frame(edit\_window,background="#7e4b1c")  
 fframe3.pack()  
  
 # Create a search field for the user to enter the search criteria  
 search\_label = tk.Label(fframe1, text="Enter customer ID, phone number, or email ID:", font=("Ariel",20,"bold"))  
 search\_label.pack(pady=10)  
 search\_entry = tk.Entry(fframe1,width=15,font=("Ariel",20,"bold"))  
 search\_entry.pack()  
  
  
  
 def search\_customer():  
 for frame in [fframe2, fframe3]:  
 for widget in frame.winfo\_children():  
 widget.destroy()  
  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
  
 # Perform database operations  
 cursor = mydb.cursor()  
 # Retrieve customer details from database based on search criteria  
 search\_term = search\_entry.get().strip()  
 cursor.execute("SELECT \* FROM CustomersInformation WHERE Customer\_ID=%s OR \  
 Phone\_Number=%s OR Email\_ID=%s", (search\_term, search\_term, search\_term))  
 result = cursor.fetchone()  
 mydb.close()  
 if result:  
 # Display customer information in a form for editing  
 self.customer\_id = tk.StringVar(value=result[0])  
 self.first\_name = tk.StringVar(value=result[1])  
 self.middle\_name = tk.StringVar(value=result[2])  
 self.last\_name = tk.StringVar(value=result[3])  
 self.address = tk.StringVar(value=result[4])  
 self.city = tk.StringVar(value=result[5])  
 self.state = tk.StringVar(value=result[6])  
 self.zipcode = tk.StringVar(value=result[7])  
 self.phone\_number = tk.StringVar(value=result[8])  
 self.username = tk.StringVar(value=result[9])  
 self.email\_id = tk.StringVar(value=result[11])  
  
 frame1 = tk.Frame(fframe2,bg="#212129")  
 frame1.pack()  
  
 # Create a form for editing customer details  
 customer\_id\_label = tk.Label(frame1, text="Customer ID:", font=("Arial", 11, "bold"), bg="#212129",  
 fg="red")  
 customer\_id\_label.grid(row=0, column=0, padx=5, pady=5, sticky="W")  
 customer\_id\_entry = tk.Entry(frame1, textvariable=self.customer\_id, state='readonly',  
 font=("Arial", 12, "bold"))  
 customer\_id\_entry.grid(row=0, column=1, padx=5, pady=5)  
  
 first\_name\_label = tk.Label(frame1, text="First Name\*:", font=("Arial", 11, "bold"), bg="#212129",  
 fg="white")  
 first\_name\_label.grid(row=1, column=0, padx=5, pady=5, sticky="W")  
 first\_name\_entry = tk.Entry(frame1, textvariable=self.first\_name, font=("Arial", 12, "bold"))  
 first\_name\_entry.grid(row=1, column=1, padx=5, pady=5)  
  
 middle\_name\_label = tk.Label(frame1, text="Middle Name:", font=("Arial", 11, "bold"), bg="#212129",  
 fg="white")  
 middle\_name\_label.grid(row=2, column=0, padx=5, pady=5, sticky="W")  
 middle\_name\_entry = tk.Entry(frame1, textvariable=self.middle\_name, font=("Arial", 12, "bold"))  
 middle\_name\_entry.grid(row=2, column=1, padx=5, pady=5)  
  
 last\_name\_label = tk.Label(frame1, text="Last Name\*:", font=("Arial", 11, "bold"), bg="#212129",  
 fg="white")  
 last\_name\_label.grid(row=3, column=0, padx=5, pady=5, sticky="W")  
 last\_name\_entry = tk.Entry(frame1, textvariable=self.last\_name, font=("Arial", 12, "bold"))  
 last\_name\_entry.grid(row=3, column=1, padx=5, pady=5)  
  
 address\_label = tk.Label(frame1, text="Address:", font=("Arial", 11, "bold"), bg="#212129", fg="white")  
 address\_label.grid(row=4, column=0, padx=5, pady=5, sticky="W")  
 address\_entry = tk.Entry(frame1, textvariable=self.address, font=("Arial", 12, "bold"))  
 address\_entry.grid(row=4, column=1, padx=5, pady=5)  
  
 city\_label = tk.Label(frame1, text="City:", font=("Arial", 11, "bold"), bg="#212129", fg="white")  
 city\_label.grid(row=5, column=0, padx=5, pady=5, sticky="W")  
 city\_entry = tk.Entry(frame1, textvariable=self.city, font=("Arial", 12, "bold"))  
 city\_entry.grid(row=5, column=1, padx=5, pady=5)  
  
 usa\_state\_names = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL", "IN","IA","KS", "KY",  
 "LA", "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM",  
 "NY","NC", "ND","OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV","WI","WY"]  
  
 state\_label = tk.Label(frame1, text="State:", font=("Arial", 11, "bold"), bg="#212129", fg="white")  
 state\_label.grid(row=6, column=0, padx=5, pady=5, sticky="W")  
  
 State\_combo = ttk.Combobox(frame1, values=usa\_state\_names, textvariable=self.state, font=("Arial", 12),  
 width=18)  
 State\_combo.grid(row=6, column=1, padx=5, pady=5)  
  
 zipcode\_label = tk.Label(frame1, text="Zipcode:", font=("Arial", 11, "bold"), bg="#212129", fg="white")  
 zipcode\_label.grid(row=7, column=0, padx=5, pady=5, sticky="W")  
 zipcode\_entry = tk.Entry(frame1, textvariable=self.zipcode, font=("Arial", 12, "bold"))  
 zipcode\_entry.grid(row=7, column=1, padx=5, pady=5)  
  
 phone\_number\_label = tk.Label(frame1, text="Phone Number\*:", font=("Arial", 11, "bold"), bg="#212129",  
 fg="white")  
 phone\_number\_label.grid(row=8, column=0, padx=5, pady=5, sticky="W")  
 phone\_number\_entry = tk.Entry(frame1, textvariable=self.phone\_number, font=("Arial", 12, "bold"))  
 phone\_number\_entry.grid(row=8, column=1, padx=5, pady=5)  
  
 username\_label = tk.Label(frame1, text="Username:", font=("Arial", 11, "bold"), bg="#212129", fg="red")  
 username\_label.grid(row=9, column=0, padx=5, pady=5, sticky="W")  
 username\_entry = tk.Entry(frame1, textvariable=self.username, state='readonly',  
 font=("Arial", 12, "bold"))  
 username\_entry.grid(row=9, column=1, padx=5, pady=5)  
  
 email\_id\_label = tk.Label(frame1, text="Email ID\*:", font=("Arial", 11, "bold"), bg="#212129",  
 fg="white")  
 email\_id\_label.grid(row=10, column=0, padx=5, pady=5, sticky="W")  
 email\_id\_entry = tk.Entry(frame1, textvariable=self.email\_id, font=("Arial", 12, "bold"))  
 email\_id\_entry.grid(row=10, column=1, padx=5, pady=5)  
  
 redlabel = tk.Label(frame1, text="\*Please note that fields in red cannot be changed. Thank you.",  
 font=("Arial", 8, "bold"), fg="red", bg="#212129")  
 redlabel.grid(row=11, column=0, padx=5, pady=5, columnspan=2)  
 llabel = tk.Label(frame1,  
 text="\*Kindly note that all '\*' fields are mandatory and must be filled",  
 font=("Arial", 8, "bold"), fg="red", bg="#212129")  
 llabel.grid(row=12, column=0, padx=5, pady=5, columnspan=2)  
  
 # Create a button to save changes to the database  
 def save\_changes():  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 cursor = mydb.cursor()  
 ## phone number format chenk and first and last name check  
 if len(first\_name\_entry.get()) == 0 or len(last\_name\_entry.get()) == 0:  
 tk.messagebox.showinfo("Error", "First Name and Last Name Fields Must Not Be Empty")  
 edit\_window.focus()  
 return  
 if len(phone\_number\_entry.get()) != 10:  
 tk.messagebox.showinfo("Error", "Phone Number should be in 10 digits only ")  
 edit\_window.focus()  
 return  
 # Check if the email ID matches the required format.  
 regex = re.compile('^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$')  
 if not regex.match(email\_id\_entry.get()):  
 tk.messagebox.showinfo("Error", "Email Should be in Proper format\n Eg: abc@pizzastore.com ")  
 edit\_window.focus()  
 return  
 # Validate zip code format  
 zip\_code = zipcode\_entry.get()  
 if zip\_code and not zip\_code.isnumeric():  
 tk.messagebox.showerror("Invalid Zip Code", "Please enter a valid zip code")  
 edit\_window.focus()  
 return  
  
 # Validate city format  
 city = city\_entry.get()  
 city\_pattern = r'^[a-zA-Z\s]+$'  
 if city and not re.match(city\_pattern, city):  
 tk.messagebox.showerror("Invalid City Name", "Please enter a valid city name")  
 edit\_window.focus()  
 return  
  
 # Validate state format  
 state = State\_combo.get()  
 if state and (not state.isalpha() or len(state) != 2):  
 tk.messagebox.showerror("Invalid State Code", "Please enter a valid two-letter state code")  
 edit\_window.focus()  
 return  
  
 query = f"UPDATE CustomersInformation SET \  
 First\_Name='{first\_name\_entry.get().strip()}', \  
 Middle\_Name='{middle\_name\_entry.get().strip()}', \  
 Last\_Name='{last\_name\_entry.get().strip()}', \  
 Address='{address\_entry.get()}', \  
 City='{city\_entry.get().strip()}', \  
 State='{State\_combo.get().strip()}', \  
 ZipCode='{zipcode\_entry.get().strip()}', \  
 Phone\_Number='{phone\_number\_entry.get().strip()}', \  
 Username='{username\_entry.get()}', \  
 Email\_ID='{email\_id\_entry.get().strip()}' \  
 WHERE Customer\_ID='{customer\_id\_entry.get()}'"  
 cursor.execute(query)  
 mydb.commit()  
 mydb.close()  
 tk.messagebox.showinfo("Success", "Changes saved to database.")  
 edit\_window.destroy()  
  
 save\_button = tk.Button(fframe3, text="Save Changes", command=save\_changes,font=("Ariel", 15, "bold"))  
 save\_button.pack(pady=10)  
 else:  
 tk.messagebox.showwarning("Warning", "No customer found with the provided search criteria.")  
 edit\_window.focus()  
 # Create a button to close the edit window  
 close\_button = tk.Button(edit\_window, text="Close", font=("Ariel", 15, "bold"), background="red", fg="white", command=edit\_window.destroy)  
 close\_button.pack(side="bottom",pady=10)  
 # Create a button to trigger the customer search  
 search\_button = tk.Button(fframe1, text="Search", command=search\_customer, font=("Ariel", 15, "bold"))  
 search\_button.pack(pady=10)  
  
 def show\_customer\_reports(self):  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
  
 # Execute the SQL query to retrieve customer reports  
 cursor.execute("SELECT \* FROM CustomersInformation")  
 customer\_data = cursor.fetchall()  
  
 # Close the database connection  
 mydb.close()  
 s = ttk.Style()  
 s.theme\_use('clam')  
 # Configure the style of Heading in Treeview widget  
 s.configure('Treeview.Heading', background="#ff8c00", font=("Ariel", 15, "bold"))  
 s.configure('Treeview', font=('Arial', 12, 'bold'))  
  
 # Create a new window to display the customer reports  
 window = tk.Toplevel(self.show\_reports\_window)  
 window.title("Customer Reports")  
 window.configure(background="#7e4b1c")  
  
 name\_label = tk.Label(window,text="Customers Data",font=("Times Now Roman",25,"bold"))  
 name\_label.pack(side="top", anchor=tk.CENTER,pady=5)  
 date = dt.datetime.now()  
 date\_label = tk.Label(window, text=f"{date:%A, %B %d, %Y}", font="Ariel, 25", anchor='e')  
 date\_label.pack(side="top", anchor=tk.CENTER,pady=5)  
 # Create a Treeview widget to display the customer data  
 tree = ttk.Treeview(window, columns=("Customer ID", "First Name", "Middle Name", "Last Name", "Phone Number",  
 "Username", "Email ID"))  
 tree.heading("#0", text="Index")  
 tree.heading("Customer ID", text="Customer ID")  
 tree.heading("First Name", text="First Name")  
 tree.heading("Middle Name", text="Middle Name")  
 tree.heading("Last Name", text="Last Name")  
 tree.heading("Phone Number", text="Phone Number")  
 tree.heading("Username", text="Username")  
 tree.heading("Email ID", text="Email ID")  
 tree.column("#0", width=65, minwidth=50, anchor=tk.CENTER)  
 tree.column("Customer ID", width=140, minwidth=100, anchor=tk.CENTER)  
 tree.column("First Name", width=140, minwidth=100, anchor=tk.CENTER)  
 tree.column("Middle Name", width=160, minwidth=100, anchor=tk.CENTER)  
 tree.column("Last Name", width=140, minwidth=100, anchor=tk.CENTER)  
 tree.column("Phone Number", width=170, minwidth=120, anchor=tk.CENTER)  
 tree.column("Username", width=120, minwidth=100, anchor=tk.CENTER)  
 tree.column("Email ID", width=250, minwidth=200, anchor=tk.CENTER)  
  
 # Define tags for alternate row colors  
 tree.tag\_configure('evenrow', background='#addfad')  
 tree.tag\_configure('oddrow', background='white')  
  
 # Insert the customer data into the Treeview widget  
 for i, customer in enumerate(customer\_data):  
 if i % 2 == 0:  
 tree.insert(parent="", index="end", iid=i, text=i, values=(  
 customer[0], customer[1], customer[2], customer[3], customer[8], customer[9], customer[11]),  
 tags=('evenrow',))  
 else:  
 tree.insert(parent="", index="end", iid=i, text=i, values=(  
 customer[0], customer[1], customer[2], customer[3], customer[8], customer[9], customer[11]),  
 tags=('oddrow',))  
  
 tree.pack(fill=tk.BOTH, expand=1)  
  
 # Create a button to export the data to an Excel file  
 def export\_to\_excel():  
 # Ask the user to choose a file location to save the Excel file  
 filepath = filedialog.asksaveasfilename(defaultextension='.xlsx')  
 if filepath:  
 # Create a CSV file with the customer data  
 with open('customer\_data.csv', 'w', newline='') as csvfile:  
 writer = csv.writer(csvfile)  
 writer.writerow(  
 ['Customer ID', 'First Name', 'Middle Name', 'Last Name', 'Phone Number', 'Username',  
 'Email ID'])  
 for customer in customer\_data:  
 writer.writerow((customer[0], customer[1], customer[2], customer[3], customer[8], customer[9],  
 customer[11]))  
 df = pd.read\_csv('customer\_data.csv')  
 df.to\_excel(filepath, index=False)  
  
 window.destroy()  
 self.show\_reports\_window.focus()  
  
 export\_button = tk.Button(window, text="Export to Excel", font=("Ariel", 15, "bold"), background="#ffebcd",  
 command=export\_to\_excel)  
 export\_button.pack(side="bottom", anchor=tk.E)  
 close\_button = tk.Button(window, text="Close", font=("Ariel", 15, "bold"), background="red", fg="white",  
 command=window.destroy)  
 close\_button.pack(side="bottom")  
  
 # Bind the Enter key to the search button  
 window.bind('<Return>', lambda event: search\_button.invoke())  
 # Create an Entry widget to enter the search term  
 search\_entry = tk.Entry(window, font=("Arial", 12))  
 search\_entry.pack(side="top", padx=10, pady=5)  
 search\_label = tk.Label(window, text="Search by ID, Name, Email, or Phone number", font=("Arial", 12,"bold"), bg='#7e4b1c')  
 search\_label.pack(side="top", padx=10, pady=5)  
  
 # Create a button to search for the customers  
 def search\_customers():  
 search\_term = search\_entry.get()  
 # Perform database operations  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 cursor = mydb.cursor()  
  
 # Execute the SQL query to retrieve the customer data  
 cursor.execute(f"SELECT \* FROM CustomersInformation WHERE First\_Name LIKE '%{search\_term}%' OR Middle\_Name LIKE '%{search\_term}%' OR Phone\_Number LIKE '%{search\_term}%' OR "  
 f"Last\_Name LIKE '%{search\_term}%' OR Email\_ID LIKE '%{search\_term}%'OR Customer\_ID LIKE '%{search\_term}%'")  
 customer\_data = cursor.fetchall()  
  
 # Close the database connection  
 mydb.close()  
  
 # Delete the existing data in the Treeview widget  
 tree.delete(\*tree.get\_children())  
  
 # Insert the updated customer data into the Treeview widget  
 for i, customer in enumerate(customer\_data):  
 if i % 2 == 0:  
 tree.insert(parent="", index="end", iid=i, text=i, values=(  
 customer[0], customer[1], customer[2], customer[3], customer[8], customer[9], customer[11]),  
 tags=('evenrow',))  
 else:  
 tree.insert(parent="", index="end", iid=i, text=i, values=(  
 customer[0], customer[1], customer[2], customer[3], customer[8], customer[9], customer[11]),  
 tags=('oddrow',))  
  
  
 def refresh\_screen():  
 window.destroy()  
 self.show\_customer\_reports()  
  
  
  
  
 frame\_frame = tk.Frame(window,background="#7e4b1c",relief="flat")  
 frame\_frame.pack(side="top", padx=10, pady=5)  
  
  
  
 search\_button = tk.Button(frame\_frame, text="Search", font=("Arial", 15, 'bold'), command=search\_customers)  
 search\_button.pack(side="left", padx=10, pady=5)  
 refresh\_button = tk.Button(frame\_frame, text="Refresh", font=("Arial", 15, 'bold'), command=refresh\_screen)  
 refresh\_button.pack(side="left", padx=10, pady=5)  
  
  
 # Bind the Enter key to the search button  
 search\_entry.bind('<Return>', lambda event: search\_button.invoke())  
  
  
 # Create a label for the search entry  
  
 def show\_sales\_reports(self):  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 cursor = mydb.cursor()  
 # Create a new window to display the customer reports  
 window = tk.Toplevel(self.show\_reports\_window)  
 window.title("Sales Reports")  
 window.configure(background="#7e4b1c")  
  
 ##Main Frames  
 fframe1 = tk.Frame(window,bg="#7e4b1c")  
 fframe1.pack()  
  
 fframe2 = tk.Frame(window,bg="#7e4b1c")  
 fframe2.pack()  
  
 fframe3 = tk.Frame(window,bg="#7e4b1c")  
 fframe3.pack()  
  
  
  
 frame1 = tk.Frame(fframe1,bg="#7e4b1c")  
 frame1.pack(side="left")  
  
 frame2 = tk.Frame(fframe1,bg="#7e4b1c")  
 frame2.pack(side="left")  
 # Create labels for the start and end dates  
 self.start\_label = tk.Label(frame1, text="Start Date:", font="Arial 12")  
 self.start\_label.pack(side="top", padx=10, pady=5)  
 self.end\_label = tk.Label(frame2, text="End Date:", font="Arial 12")  
 self.end\_label.pack(side="top", padx=10, pady=5)  
  
 self.start\_date = DateEntry(frame1, width=12, font=("Arial", 12),  
 date\_pattern="yyyy-mm-dd", foreground="black")  
 self.start\_date.pack(side="top", padx=10, pady=5)  
  
  
 tomorrow1 = (dt.date.today() + dt.timedelta(days=1)).strftime('%Y-%m-%d')  
 tomorrow11 = dt.datetime.strptime(tomorrow1, '%Y-%m-%d').date()  
 self.end\_date = DateEntry(frame2, width=12, font=("Arial", 12), date\_pattern="yyyy-mm-dd", foreground="black")  
 self.end\_date.set\_date(tomorrow11)  
 self.end\_date.pack(side="top", padx=10, pady=5)  
  
 # Create a button for retrieving the sales reports  
 get\_button = tk.Button(fframe2, text="Get Reports", font=("Arial", 12, "bold"), command=lambda: get\_sales\_reports())  
 get\_button.pack(side="top", pady=5)  
  
 # Configure the style of Heading in Treeview widget  
 s = ttk.Style()  
 s.theme\_use('clam')  
 s.configure('Treeview.Heading', background="#ff8c00", font=("Ariel", 15, "bold"))  
 s.configure('Treeview', font=('Arial', 12, 'bold'))  
  
 # Create a Treeview widget to display the sales data  
 tree = ttk.Treeview(fframe2, columns=("Order\_Date", "Total\_Sales","Cash\_Total","Card\_Total"))  
 tree.heading("#0", text="Index")  
 tree.heading("Order\_Date", text="Order Date")  
 tree.heading("Total\_Sales", text="Total Sales")  
 tree.heading("Cash\_Total", text="Card Total")  
 tree.heading("Card\_Total", text="Card Total")  
 tree.column("#0", width=65, minwidth=50, anchor="center")  
 tree.column("Order\_Date", width=140, minwidth=100, anchor="center")  
 tree.column("Total\_Sales", width=140, minwidth=100, anchor="center")  
 tree.column("Cash\_Total",width=140, minwidth=100, anchor="center")  
 tree.column("Card\_Total",width=140, minwidth=100, anchor="center")  
 tree.pack(fill="both", expand=1)  
 def export\_to\_excel():  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 cursor = mydb.cursor()  
 # Get the start and end dates from the DateEntry widgets  
 start = self.start\_date.get\_date().strftime("%Y-%m-%d")  
 end = self.end\_date.get\_date().strftime("%Y-%m-%d")  
 # Execute the SQL query to retrieve sales reports for the given date range  
 cursor.execute(  
 "SELECT Order\_Date as Date,"  
 "ROUND(SUM(CASE WHEN MOP IN ('card', 'cash') THEN Amount\_Paid ELSE 0 END), 2) as Total\_Sales,"  
 "ROUND(SUM(CASE WHEN MOP ='cash' THEN Amount\_Paid ELSE 0 END), 2) as Cash\_Total,"  
 "ROUND(SUM(CASE WHEN MOP ='card' THEN Amount\_Paid ELSE 0 END), 2) as Card\_Total"  
 " FROM Payment WHERE Order\_Date BETWEEN %s AND %s AND Amount\_Paid != %s GROUP BY Date",  
 (start, end,0))  
 # Fetch the results and store them in the total\_sales list  
 total\_sales = cursor.fetchall()  
 mydb.close()  
 # Ask the user to choose a file location to save the Excel file  
 filepath = filedialog.asksaveasfilename(defaultextension='.xlsx')  
 if filepath:  
 # Create a CSV file with the sales data  
 with open('sales\_data.csv', 'w', newline='') as csvfile:  
 writer = csv.writer(csvfile)  
 writer.writerow(["Order Date", "Total Sales","Cash Total","Card Total"])  
 for sales in total\_sales:  
 writer.writerow((sales[0], sales[1],sales[2],sales[3]))  
  
 # Create an Excel file from the CSV file  
 df = pd.read\_csv('sales\_data.csv')  
 df.to\_excel(filepath, index=False)  
 tk.messagebox.showinfo("Success", "Sales data exported to Excel file.")  
 else:  
 tk.messagebox.showerror("Error", "Please enter a file name.")  
 self.show\_reports\_window.focus()  
 window.destroy()  
  
 def get\_sales\_reports():  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 cursor = mydb.cursor()  
 # Get the start and end dates from the DateEntry widgets  
 start = self.start\_date.get\_date().strftime("%Y-%m-%d")  
 end = self.end\_date.get\_date().strftime("%Y-%m-%d")  
 # Execute the SQL query to retrieve sales reports for the given date range  
 cursor.execute(  
 "SELECT Order\_Date as Date,"  
 "ROUND(SUM(CASE WHEN MOP IN ('card', 'cash') THEN Amount\_Paid ELSE 0 END), 2) as Total\_Sales,"  
 "ROUND(SUM(CASE WHEN MOP ='cash' THEN Amount\_Paid ELSE 0 END), 2) as Cash\_Total,"  
 "ROUND(SUM(CASE WHEN MOP ='card' THEN Amount\_Paid ELSE 0 END), 2) as Card\_Total"  
 " FROM Payment WHERE Order\_Date BETWEEN %s AND %s AND Amount\_Paid != %s GROUP BY Date",  
 (start, end, 0))  
 # Fetch the results and store them in the total\_sales list  
 total\_sales = cursor.fetchall()  
 mydb.close()  
  
 # Clear the Treeview widget before inserting new data  
 tree.delete(\*tree.get\_children())  
 # Insert the sales data into the Treeview widget  
 for i, sales in enumerate(total\_sales):  
 if i % 2 == 0:  
 tree.insert(parent="", index="end", iid=i, text=i, values=(  
 sales[0], sales[1],sales[2],sales[3]), tags=('evenrow',))  
 else:  
 tree.insert(parent="", index="end", iid=i, text=i, values=(  
 sales[0], sales[1],sales[2],sales[3]), tags=('oddrow',))  
  
 # Create buttons for exporting the sales data to Excel and closing the window  
 export\_button = tk.Button(fframe3, text="Export to Excel", font=("Ariel", 15, "bold"), background="#ffebcd",  
 command=export\_to\_excel)  
 export\_button.pack(side="left", padx=10, pady=10)  
 close\_button = tk.Button(fframe3, text="Close", font=("Ariel", 16, "bold"), background="#ffebcd", bg="red",  
 fg="white", command=window.destroy)  
 close\_button.pack(side="right", padx=10, pady=10)  
  
 # Set the focus on the start date widget  
 self.start\_date.focus\_set()  
  
 # Define tags for alternate row colors in the Treeview widget  
 tree.tag\_configure('evenrow', background='#addfad')  
 tree.tag\_configure('oddrow', background='white')  
 # Bind the Enter key to the get\_sales\_reports function  
 window.bind("<Return>", lambda event: get\_sales\_reports())  
  
 # Call the get\_sales\_reports function to populate the Treeview widget with initial data  
 get\_sales\_reports()  
  
 # Start the main event loop for the window  
 window.mainloop()  
  
 def view\_sales\_reports(self, date):  
  
 Date = date  
 window = tk.Toplevel(self.show\_reports\_window)  
 window.attributes("-fullscreen", True)  
 window.configure(background="#212129")  
 l\_label = tk.Label(window, text="Pizza Store Dashboard", font=("Times Now Roman", 30, "bold"),  
 fg="white", bg="#212129")  
 l\_label.pack(side="top", anchor=tk.CENTER)  
  
 frame1 = tk.Frame(window, bg='#212129')  
 frame1.pack(fill="both", expand=True)  
 ffframe1 = tk.Frame(frame1, bg='#212129')  
 ffframe1.pack(side="left")  
 ffframe2 = tk.Frame(frame1, bg='#212129')  
 ffframe2.pack(side="left")  
 ffframe3 = tk.Frame(frame1, bg='#212129')  
 ffframe3.pack( side="left")  
  
 frame2 = tk.Frame(window, bg="#212129")  
 frame2.pack(fill="both", expand=True)  
 frame3 = tk.Frame(frame2, bg="#212129")  
 frame3.pack(side="right",padx=20)  
 # add a close button to the plot window  
 close\_button = tk.Button(frame3, text="Close", command=window.destroy, width=10, bg="red", fg="white",  
 font=("Ariel", 20, "bold"))  
 close\_button.pack(side="bottom")  
 frame3\_1 = tk.Frame(frame3, bg='#212129')  
 frame3\_1.pack(side="left", fill="both", expand=True)  
 frame3\_2 = tk.Frame(frame3, bg='#212129')  
 frame3\_2.pack(side="left")  
  
 mydb = None  
 try:  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 c = mydb.cursor()  
  
 # Statistics Text Labels  
  
 ## Fetching Total orders and Total Revenue earned from database  
 c.execute("SELECT COUNT(\*) as Total\_Orders,SUM(Amount\_Paid) as total FROM Payment WHERE Amount\_Paid != %s",  
 (0,))  
 text\_query = c.fetchall()  
 for values in text\_query:  
 Total\_Orders = values[0]  
 Total\_sales = values[1]  
  
 ## Fetching Average Customer Ratings  
 c.execute("SELECT AVG(Order\_Ratings) as ratings FROM Orders WHERE Order\_Ratings>=1")  
 Average\_Ratings = c.fetchone()[0]  
  
 ## Fetching Mode Of Payment total  
 c.execute("SELECT SUM(Amount\_Paid) FROM Payment WHERE MOP = 'Cash'")  
 cash\_MOP = c.fetchone()[0]  
 c.execute("SELECT SUM(Amount\_Paid) FROM Payment WHERE MOP = 'Card'")  
 card\_MOP = c.fetchone()[0]  
 ## Assigning the Above value to lables for dashboard display  
 total\_orders = tk.Label(frame3\_1, text='Total Orders', font=("Ariel", 20, "bold"),  
 bg="#212129", fg="white")  
 total\_orders.pack(anchor=tk.W)  
 total\_orders1 = tk.Label(frame3\_2, text=f':{Total\_Orders}', font=("Ariel", 20, "bold"),  
 bg="#212129", fg="white")  
 total\_orders1.pack(anchor=tk.W)  
  
 total\_sales = tk.Label(frame3\_1, text='Total Revenue', font=("Ariel", 20, "bold"),  
 fg="white", bg="#212129")  
 total\_sales.pack(anchor=tk.W)  
 total\_sales1 = tk.Label(frame3\_2, text=f':${Total\_sales:.2f}', font=("Ariel", 20, "bold"),  
 fg="white", bg="#212129")  
 total\_sales1.pack(anchor=tk.W)  
  
 cash\_sales = tk.Label(frame3\_1, text='CashRevenue', font=("Ariel", 20, "bold"), fg="white",  
 bg="#212129")  
 cash\_sales.pack(anchor=tk.W)  
 cash\_sales1 = tk.Label(frame3\_2, text=f':${cash\_MOP:.2f}', font=("Ariel", 20, "bold"), fg="white",  
 bg="#212129")  
 cash\_sales1.pack(anchor=tk.W)  
  
 card\_sales = tk.Label(frame3\_1, text='CardRevenue', font=("Ariel", 20, "bold"), fg="white",  
 bg="#212129")  
 card\_sales.pack(anchor=tk.W)  
 card\_sales1 = tk.Label(frame3\_2, text=f':${card\_MOP:.2f}', font=("Ariel", 20, "bold"), fg="white",  
 bg="#212129")  
 card\_sales1.pack(anchor=tk.W)  
 average\_ratings = tk.Label(frame3\_1, text='Average Ratings',  
 font=("Ariel", 20, "bold"), fg="white", bg="#212129")  
 average\_ratings.pack(anchor=tk.W)  
 average\_ratings1 = tk.Label(frame3\_2, text=f':{Average\_Ratings:.2f}/5',  
 font=("Ariel", 20, "bold"), fg="white", bg="#212129")  
 average\_ratings1.pack(anchor=tk.W)  
  
 if Date == "Daily":  
 ### Sales History By Date  
 # Sales report over changing dates  
 c.execute(  
 "SELECT YEAR(Order\_Date) as Year, MONTH(Order\_Date) as Month,Order\_Date as Date, SUM(Amount\_Paid) as Total\_Sales FROM Payment GROUP BY Year, Month,Date ORDER BY Date")  
 sales\_rows = c.fetchall()  
 sales\_df = {"Year": [], "Month": [], "Date": [], "Total\_Sales": []}  
  
 for row in sales\_rows:  
 month\_name = calendar.month\_name[row[1]]  
 sales\_df["Year"].append(row[0])  
 sales\_df["Month"].append(f"{month\_name} {row[0]}")  
 date\_str = row[2].strftime("%Y-%m-%d")  
 sales\_df["Date"].append(date\_str)  
 sales\_df["Total\_Sales"].append(float(row[3]))  
 fig1, ax1 = plt.subplots()  
 fig1.set\_facecolor('#212129')  
 ax1.set\_facecolor('#16161e')  
 ax1.plot(sales\_df["Date"], sales\_df["Total\_Sales"], '-o', color="orange")  
 ax1.set\_xlabel('Date')  
 ax1.set\_ylabel('Total Sales')  
 ax1.set\_title('Total Sales by Date', fontweight="bold")  
 # add labels for each data point  
 for x, y in zip(sales\_df["Date"], sales\_df["Total\_Sales"]):  
 label = "${:,.2f}".format(y)  
 ax1.annotate(label, (x, y), xytext=(0, 5), textcoords="offset points", ha='center', va='bottom',  
 color='white', fontsize=8, weight='bold')  
 for text in ax1.get\_xticklabels() + ax1.get\_yticklabels() + [ax1.title] + [ax1.xaxis.label,  
 ax1.yaxis.label]:  
 text.set\_color("white")  
 text.set\_weight("bold")  
 for label in ax1.xaxis.get\_ticklabels():  
 label.set\_rotation(5)  
  
 # embed the plot in the Tkinter window  
 canvas = FigureCanvasTkAgg(fig1, master=ffframe1)  
 canvas.draw()  
 canvas.get\_tk\_widget().pack(side="left", fill=tk.BOTH, expand=True, anchor="e")  
 elif Date == "Monthly":  
 ### Sales History By Month  
 # Sales report over changing dates  
 c.execute(  
 "SELECT YEAR(Order\_Date) as Year, MONTH(Order\_Date) as Month,Order\_Date as Date, SUM(Amount\_Paid) as Total\_Sales FROM Payment GROUP BY Year, Month,Date ORDER BY Date")  
 sales\_rows = c.fetchall()  
 sales\_df = {"Year": [], "Month": [], "Date": [], "Total\_Sales": []}  
  
 for row in sales\_rows:  
 month\_name = calendar.month\_name[row[1]]  
 sales\_df["Year"].append(row[0])  
 sales\_df["Month"].append(f"{month\_name} {row[0]}")  
 date\_str = row[2].strftime("%Y-%m-%d")  
 sales\_df["Date"].append(date\_str)  
 sales\_df["Total\_Sales"].append(float(row[3]))  
 fig1, ax1 = plt.subplots()  
 fig1.set\_facecolor('#212129')  
 ax1.set\_facecolor('#16161e')  
 ax1.plot(sales\_df["Month"], sales\_df["Total\_Sales"], '-o', color="orange")  
 ax1.set\_xlabel('Month')  
 ax1.set\_ylabel('Total Sales')  
 ax1.set\_title('Total Sales by Month', fontweight="bold")  
 # add labels for each data point  
 for x, y in zip(sales\_df["Month"], sales\_df["Total\_Sales"]):  
 label = "${:,.2f}".format(y)  
 ax1.annotate(label, (x, y), xytext=(0, 5), textcoords="offset points", ha='center', va='bottom',  
 color='white', fontsize=8, weight='bold')  
 for text in ax1.get\_xticklabels() + ax1.get\_yticklabels() + [ax1.title] + [ax1.xaxis.label,  
 ax1.yaxis.label]:  
 text.set\_color("white")  
 text.set\_weight("bold")  
  
 # embed the plot in the Tkinter window  
 canvas = FigureCanvasTkAgg(fig1, master=ffframe1)  
 canvas.draw()  
 canvas.get\_tk\_widget().pack(side="left", fill=tk.BOTH, expand=True, anchor="e")  
 else:  
 ### Sales History By Year  
 # Sales report over changing dates  
 c.execute(  
 "SELECT YEAR(Order\_Date) as Year, MONTH(Order\_Date) as Month,Order\_Date as Date, SUM(Amount\_Paid) as Total\_Sales FROM Payment GROUP BY Year, Month,Date ORDER BY Date")  
 sales\_rows = c.fetchall()  
 sales\_df = {"Year": [], "Month": [], "Date": [], "Total\_Sales": []}  
  
 for row in sales\_rows:  
 month\_name = calendar.month\_name[row[1]]  
 sales\_df["Year"].append(row[0])  
 sales\_df["Month"].append(f"{month\_name} {row[0]}")  
 date\_str = row[2].strftime("%Y-%m-%d")  
 sales\_df["Date"].append(date\_str)  
 sales\_df["Total\_Sales"].append(float(row[3]))  
 fig1, ax1 = plt.subplots()  
 fig1.set\_facecolor('#212129')  
 ax1.set\_facecolor('#16161e')  
 ax1.plot(sales\_df["Year"], sales\_df["Total\_Sales"], '-o', color="orange")  
 ax1.set\_xlabel('Year')  
 ax1.set\_ylabel('Total Sales')  
 ax1.set\_title('Total Sales by Year', fontweight="bold")  
 # add labels for each data point  
 for x, y in zip(sales\_df["Year"], sales\_df["Total\_Sales"]):  
 label = "${:,.2f}".format(y)  
 ax1.annotate(label, (x, y), xytext=(0, 5), textcoords="offset points", ha='center', va='bottom',  
 color='white', fontsize=8, weight='bold')  
 for text in ax1.get\_xticklabels() + ax1.get\_yticklabels() + [ax1.title] + [ax1.xaxis.label,  
 ax1.yaxis.label]:  
 text.set\_color("white")  
 text.set\_weight("bold")  
  
 # embed the plot in the Tkinter window  
 canvas = FigureCanvasTkAgg(fig1, master=ffframe1)  
 canvas.draw()  
 canvas.get\_tk\_widget().pack(side="left", fill=tk.BOTH, expand=True, anchor="e")  
  
  
  
  
  
  
 ###### MOst Sold Items  
 items\_query = "SELECT CONCAT(item\_name, '', IFNULL(CONCAT(' ', item\_size), '')) AS item\_name, SUM(item\_Quantity) as Total\_Quantity\_Sold FROM Order\_Item oi INNER JOIN Items i ON oi.item\_ID = i.item\_ID GROUP BY item\_name,item\_size ORDER BY Total\_Quantity\_Sold DESC LIMIT 10"  
  
 c.execute(items\_query)  
 items\_rows = c.fetchall()  
  
 items\_df = {"item\_name": [], "Total\_Quantity": []}  
 for row in items\_rows:  
 items\_df["item\_name"].append(row[0])  
 items\_df["Total\_Quantity"].append(row[1])  
 fig2, ax2 = plt.subplots()  
 fig2.set\_facecolor('#212129')  
 ax2.set\_facecolor('#16161e')  
 # create a color map that maps the values to colors  
 colors = cm.ScalarMappable(cmap="YlOrRd").to\_rgba([float(x) for x in items\_df["Total\_Quantity"]])  
 ax2.barh(items\_df["item\_name"], items\_df["Total\_Quantity"], color=colors)  
 ax2.set\_title("Top Selling Items", fontweight="bold")  
 # rotate the x-axis tick labels  
 plt.xticks(rotation=0)  
 # adjust padding around the subplots  
 plt.subplots\_adjust(left=0.2, right=0.95, top=0.9, bottom=0.1)  
 # add the value of each bar to the horizontal bar chart  
 for i, value in enumerate(items\_df["Total\_Quantity"]):  
 ax2.annotate(str(value), xy=(value, i), ha='left', va='center', color='white', weight='bold')  
 # set the color of all text in the plot to white  
 for text in ax2.get\_xticklabels() + ax2.get\_yticklabels() + [ax2.title]:  
 text.set\_color("white")  
 text.set\_weight("bold")  
  
 # embed the plot in the Tkinter window  
 canvas = FigureCanvasTkAgg(fig2, master=frame2)  
 canvas.draw()  
 canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True, anchor=tk.E)  
  
 ##### Order ratings  
 star\_query = "SELECT Order\_Ratings as ratings, COUNT(\*) as Num\_Of\_Orders FROM Orders GROUP BY Order\_Ratings ORDER BY ratings"  
 c.execute(star\_query)  
 star\_rows = c.fetchall()  
 star\_df = {"ratings": [], "Num\_of\_Orders": []}  
 for row in star\_rows:  
 star\_df["ratings"].append(row[0])  
 star\_df["Num\_of\_Orders"].append(row[1])  
 fig3, ax3 = plt.subplots()  
 fig3.set\_facecolor('#212129')  
 ax3.set\_facecolor('#16161e')  
 ax3.set\_ylabel('Ratings', weight='bold')  
 ax3.set\_xlabel('Orders', weight='bold')  
 # create a color map that maps the values to colors  
 colors = cm.ScalarMappable(cmap="YlOrRd").to\_rgba(star\_df["ratings"])  
 ax3.barh(star\_df["ratings"], star\_df["Num\_of\_Orders"], color=colors)  
 ax3.set\_title("Avg Customer Ratings", fontweight="bold")  
 # add the value of each bar to the horizontal bar chart  
 for i, value in enumerate(star\_df["Num\_of\_Orders"]):  
 ax3.annotate(str(value), xy=(value, i), ha='left', va='center', color='white', weight='bold')  
  
 # set the color of all text in the plot to white  
 for text in ax3.get\_xticklabels() + ax3.get\_yticklabels() + [ax3.title] + [ax3.xaxis.label,  
 ax3.yaxis.label]:  
 text.set\_color("white")  
 text.set\_weight("bold")  
 # embed the plot in the Tkinter window  
 canvas = FigureCanvasTkAgg(fig3, master=ffframe2)  
 canvas.draw()  
 canvas.get\_tk\_widget().pack(side="left", fill=tk.BOTH, expand=True, anchor="e")  
  
 # Pie Chart  
 c.execute(  
 "SELECT COUNT(CASE WHEN employee\_id IS NOT NULL THEN order\_id END) \* 100 / COUNT(order\_id) AS store\_order\_percentage, COUNT(CASE WHEN employee\_id IS NULL THEN order\_id END) \* 100 / COUNT(order\_id) AS online\_order\_percentage FROM orders")  
 result = c.fetchone()  
 # Create a dictionary of percentage values  
 order\_dict = {  
 'Store': result[0],  
 'Online': result[1]  
 }  
 fig4, ax4 = plt.subplots()  
 fig4.set\_facecolor('#212129')  
 ax4.set\_facecolor('#16161e')  
 labels = list(order\_dict.keys())  
 values = list(order\_dict.values())  
 textprops = {'color': 'white', 'weight': 'bold', 'fontsize': 9}  
 ax4.pie(values, labels=labels, autopct='%1.1f%%', textprops=textprops)  
 ax4.set\_title('Order Distribution', color='white', weight='bold')  
 canvas = FigureCanvasTkAgg(fig4, master=ffframe3)  
 canvas.draw()  
 canvas.get\_tk\_widget().pack(side="left", fill=tk.BOTH, expand=True, anchor="e")  
 except mysql.connector.Error as err:  
 print("Something went wrong: {}".format(err))  
 finally:  
 if mydb is not None:  
 mydb.close()  
  
  
 ### Order History  
  
 def show\_orders(self, status):  
 # Create a new window to display the orders  
 window = tk.Toplevel(self.main\_window)  
 self.show\_order\_window = window  
 window.title("Order History")  
 window.geometry("1000x700")  
 window.configure(background="#7e4b1c")  
 # Add a label for the order history  
 label = tk.Label(self.show\_order\_window, text="Order History", font=("Ariel", 25), bg="#7e4b1c")  
 label.pack(pady=10)  
 date = dt.datetime.now()  
 date\_label = tk.Label(self.show\_order\_window, text=f"{date:%A, %B %d, %Y}", font="Ariel, 20", anchor='e')  
 date\_label.pack(pady=10, anchor=tk.CENTER)  
  
 s = ttk.Style()  
 s.theme\_use('clam')  
 # Configure the style of Heading in Treeview widget  
 s.configure('Treeview.Heading', background="#ff8c00", font=("Ariel", 15, "bold"))  
 s.configure('Treeview', font=('Arial', 12, 'bold'))  
  
 # Add a treeview to display the orders  
 columns = ("Order ID", "Order Date", "Order Total", "Order Status")  
 tree = ttk.Treeview(self.show\_order\_window, columns=columns, show="headings", height=20)  
  
 tree.column("Order Date", width=300)  
 for col in columns:  
 tree.heading(col, text=col)  
 if col in ["Order ID", "Order Total", "Order Status"]:  
 tree.column(col, anchor="center")  
 tree.pack()  
  
 # Define the tags for each status  
 tree.tag\_configure("pending", background="yellow")  
 tree.tag\_configure("preparing", background="orange")  
 tree.tag\_configure("ready", background="green")  
 tree.tag\_configure("cancelled", background="red")  
  
 # Set the default date to today's date  
 default\_date = dt.date.today().strftime('%Y-%m-%d')  
  
 # Show orders for today's date by default  
 self.show\_orders\_by\_date(tree, default\_date, status)  
  
 # Add a button to select a date  
 date\_button = tk.Button(self.show\_order\_window, text="Select Date", font=("Arial", 20, "bold"),  
 command=lambda: self.select\_date(tree, status), bg="grey")  
 date\_button.pack(side="left")  
  
 if status == "Pending":  
 # Create buttons to update order status  
 prepare\_button = tk.Button(self.show\_order\_window, text="Preparing", font=("Arial", 20, "bold"),  
 command=lambda: self.update\_order\_status(tree, "Preparing"), bg="orange")  
 prepare\_button.pack(side="left")  
  
 cancel\_button = tk.Button(self.show\_order\_window, text="Cancel", font=("Arial", 20, "bold"),  
 command=lambda: self.update\_order\_status(tree, "Cancelled"), bg="green")  
 cancel\_button.pack(side="left", anchor=tk.CENTER)  
 elif status == "Preparing":  
 ready\_button = tk.Button(self.show\_order\_window, text="Ready", font=("Arial", 20, "bold"),  
 command=lambda: self.update\_order\_status(tree, "Ready"), bg="green")  
 ready\_button.pack(side="left", anchor=tk.CENTER)  
 elif status == "Ready":  
 # Create buttons to update order status  
 prepare\_button = tk.Button(self.show\_order\_window, text="Preparing", font=("Arial", 20, "bold"),  
 command=lambda: self.update\_order\_status(tree, "Preparing"), bg="orange")  
 prepare\_button.pack(side="left")  
 else:  
 pass  
  
  
  
  
  
  
 close\_button = tk.Button(self.show\_order\_window, text="Close", font=("Ariel", 20, "bold"), background="red",  
 fg="white",  
 command=window.destroy)  
 close\_button.pack(side="right")  
  
 def select\_date(self, tree, status):  
 def select():  
 selected\_date = cal.selection\_get().strftime('%Y-%m-%d')  
 self.show\_orders\_by\_date(tree, selected\_date, status)  
 top.destroy()  
  
 top = tk.Toplevel(self.main\_window)  
 top.geometry("400x300")  
  
 # Set the default date to today's date  
 default\_date = dt.datetime.today().date()  
  
 # Pass the default date to the Calendar widget  
 cal = Calendar(top, selectmode='day', year=default\_date.year, month=default\_date.month, day=default\_date.day)  
 cal.pack(pady=10)  
  
 button = tk.Button(top, text="Select", command=select)  
 button.pack(pady=10)  
  
 def show\_orders\_by\_date(self, tree, selected\_date, status):  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 cursor.execute(  
 "SELECT Orders.Order\_ID, Orders.Order\_Date, Orders.Order\_Total, Order\_Status.Order\_Status "  
 "FROM Orders INNER JOIN Order\_Status ON Orders.Order\_ID = Order\_Status.Order\_ID"  
 " WHERE date(Orders.Order\_Date) = %s AND Order\_Status.Order\_Status =%s "  
 "ORDER BY Orders.Order\_ID DESC",  
 (selected\_date, status))  
  
 orders = cursor.fetchall()  
 mydb.close()  
 self.display\_orders(tree, orders)  
  
 def display\_orders(self, tree, orders):  
 tree.delete(\*tree.get\_children())  
 for order in orders:  
 order\_id, order\_date, order\_total, order\_status = order  
 order\_total\_formatted = f"{order\_total:.2f}"  
 order\_item\_id = f"{order\_id}\_item"  
 tree.insert("", "end", values=(order\_id, order\_date, order\_total\_formatted, order\_status), iid=order\_id,  
 tags=(order\_status.lower(),))  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
 cursor.execute(  
 "SELECT CONCAT(Items.item\_name, ' ', IFNULL(CONCAT(' ', Items.item\_size), '')) AS name, Order\_Item.item\_Quantity, (Order\_Item.item\_Quantity \* Items.item\_price) AS price FROM Items INNER JOIN Order\_Item ON Items.item\_ID = Order\_Item.item\_ID WHERE Order\_Item.Order\_ID = %s",  
 (order\_id,)  
 )  
  
 order\_items = cursor.fetchall()  
 mydb.close()  
 for index, order\_item in enumerate(order\_items, 1):  
 name, item\_quantity, price = order\_item  
 item\_id = f"{order\_item\_id}\_{index}"  
 tree.insert(order\_id, "end", values=("", f"{name} ({item\_quantity})", "{:.2f}".format(price)),  
 iid=item\_id)  
  
 def update\_order\_status(self, order\_tree, status):  
 # Get the selected order from the Treeview  
 selected\_order = order\_tree.focus()  
  
 # Extract the order ID from the selected order  
 order\_id = selected\_order  
  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 cursor = mydb.cursor()  
  
 if status == "Cancelled":  
 cursor.execute("UPDATE Order\_Status SET Order\_Status = %s WHERE Order\_ID = %s", (status, order\_id))  
 cursor.execute("UPDATE Payment SET Amount\_Paid = %s WHERE Order\_ID = %s", (0, order\_id))  
 else:  
 cursor.execute("UPDATE Order\_Status SET Order\_Status = %s WHERE Order\_ID = %s", (status, order\_id))  
  
 # Commit the changes to the database  
 mydb.commit()  
  
 # Remove the selected order from the Treeview  
 order\_tree.delete(selected\_order)  
  
 ### New Orders  
 def new\_order(self):  
 # Create a new window  
 order\_window = tk.Toplevel()  
  
 order\_window.title("New Order")  
 order\_window.configure(background="#7e4b1c")  
 no\_ff\_Frame = tk.Frame(order\_window, bg="#7e4b1c")  
 no\_ff\_Frame.pack(side="left")  
  
 no\_ff11\_Frame = tk.Frame(order\_window)  
 no\_ff11\_Frame.pack(side="left")  
  
 # Create the labels and entry boxes for customer information  
 tk.Label(no\_ff\_Frame, text="First Name: ", font=("Ariel", 15, "bold"), bg="#7e4b1c").pack(anchor=tk.W)  
 first\_name\_entry = tk.Entry(no\_ff11\_Frame, font=("Ariel", 15, "bold"))  
 first\_name\_entry.pack()  
  
 tk.Label(no\_ff\_Frame, text="Middle Name: ", font=("Ariel", 15, "bold"), bg="#7e4b1c").pack(anchor=tk.W)  
 middle\_name\_entry = tk.Entry(no\_ff11\_Frame, font=("Ariel", 15, "bold"))  
 middle\_name\_entry.pack()  
  
 tk.Label(no\_ff\_Frame, text="Last Name: ", font=("Ariel", 15, "bold"), bg="#7e4b1c").pack(anchor=tk.W)  
 last\_name\_entry = tk.Entry(no\_ff11\_Frame, font=("Ariel", 15, "bold"))  
 last\_name\_entry.pack()  
  
 tk.Label(no\_ff\_Frame, text="Email: ", font=("Ariel", 15, "bold"), bg="#7e4b1c").pack(anchor=tk.W)  
 email\_entry = tk.Entry(no\_ff11\_Frame, font=("Ariel", 15, "bold"))  
 email\_entry.pack()  
  
 tk.Label(no\_ff\_Frame, text="Contact Number: ", font=("Ariel", 15, "bold"), bg="#7e4b1c").pack(anchor=tk.W)  
 contact\_entry = tk.Entry(no\_ff11\_Frame, font=("Ariel", 15, "bold"))  
 contact\_entry.pack()  
  
  
  
 ##Close Button  
 close\_button = tk.Button(order\_window, text="Close", font=("Ariel", 16, "bold"), background="#ffebcd",bg="red",fg="white",  
 command=order\_window.destroy)  
 close\_button.pack(side="bottom",pady=10)  
  
 # Create a button for setting default values  
 default\_button = tk.Button(order\_window, text="Set Default", font=("Ariel", 16, "bold"), background="#ffebcd",  
 command=lambda: self.set\_default(first\_name\_entry, middle\_name\_entry,  
 last\_name\_entry, email\_entry, contact\_entry))  
 default\_button.pack(side="bottom",padx=10,pady=10)  
  
 submit\_button = tk.Button(order\_window, text="Submit", font=("Trebuchet MS", 15, "bold"), bg="green",  
 fg="white",  
 command=lambda: self.submit\_order(order\_window, first\_name\_entry.get().lower(),  
 middle\_name\_entry.get().lower(),  
 last\_name\_entry.get().lower(),  
 email\_entry.get().lower(),  
 contact\_entry.get().lower()))  
 submit\_button.pack()  
 self.new\_order\_window = order\_window  
  
 def set\_default(self, first\_name\_entry, middle\_name\_entry, last\_name\_entry, email\_entry, contact\_entry):  
 # Set default values for the customer information fields  
 first\_name\_entry.insert(0, "abc")  
 middle\_name\_entry.insert(0, "abc")  
 last\_name\_entry.insert(0, "abc")  
 email\_entry.insert(0, "abc.abc@pizzastore.com")  
 contact\_entry.insert(0, "9911991199")  
 # Define a function to check if email or contact is empty and display an error message  
 def validate\_customer\_info(self, email, contact):  
 if not email and not contact:  
 tk.messagebox.showerror("Error", "Email or contact number is required.")  
 self.new\_order\_window.focus()  
 return False  
 if email:  
 # Check if the email ID matches the required format.  
 regex = re.compile('^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$')  
 if not regex.match(email):  
 tk.messagebox.showinfo("Error", "Email Should be in Proper format\n Eg: abc@pizzastore.com ")  
 self.new\_order\_window.focus()  
 return False  
 else:  
 ## phone number format chenk  
 if len(contact) != 10:  
 tk.messagebox.showinfo("Error", "Phone Number should be in 10 digits only ")  
 self.new\_order\_window.focus()  
 return False  
 return True  
  
 def submit\_order(self, order\_window, first\_name, middle\_name, last\_name, email, contact):  
 if not self.validate\_customer\_info(email, contact):  
 return  
  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 c = mydb.cursor()  
 c.execute(  
 "SELECT Customer\_ID, CONCAT(Last\_Name, ', ', First\_Name, ' ', Middle\_Name) AS full\_name FROM CustomersInformation WHERE Email\_ID = %s OR Phone\_Number = %s",  
 (email, contact))  
  
 row = c.fetchone()  
 if row:  
 # Customer already exists, retrieve customer id  
 customer\_id = row[0]  
 full\_name = row[1]  
  
 else:  
 ## Validation for mandatory fields throwing errors  
 if not first\_name:  
 tk.messagebox.showerror("Error", "First name is required.")  
 self.new\_order\_window.focus()  
 return False  
 if not last\_name:  
 tk.messagebox.showerror("Error", "Last name is required.")  
 self.new\_order\_window.focus()  
 return False  
 if not email:  
 tk.messagebox.showerror("Error", "Email is required.")  
 self.new\_order\_window.focus()  
 return False  
 if not contact:  
 tk.messagebox.showerror("Error", "contact is required.")  
 self.new\_order\_window.focus()  
 return False  
  
 # Query the database for the highest existing customer ID  
 # Retrieve the last customer ID, sorting by numeric part  
 c.execute(  
 "SELECT Customer\_ID FROM CustomersInformation ORDER BY CAST(substring(Customer\_ID, 4) AS UNSIGNED) DESC LIMIT 1")  
 max\_id = c.fetchone()[0]  
 if max\_id is None:  
 # If there are no existing customers, start the customer ID sequence at 1  
 max\_id = 0  
 else:  
 # Convert the max\_id value to an integer  
 max\_id = int(max\_id.replace("CUS", ""))  
  
 # Increment the highest customer ID to generate a new unique customer ID  
 new\_id = max\_id + 1  
  
 # Add "CUS" back to the beginning of the new customer ID  
 customer\_id = f"CUS{new\_id}"  
 ## creating a new user name and default password  
 first = first\_name.lower()  
 middle = middle\_name.lower()  
 last = last\_name.lower()  
 fi = first[0].lower()  
 uid = ' '  
 num = 0  
 if middle != "":  
 mi = middle[0].lower()  
 else:  
 mi = ""  
  
 if len(last) > 5:  
 first\_p = last[:5]  
 else:  
 first\_p = last  
  
 pattern = first\_p + '%'  
 # check if the username already exists in the database  
 select\_stmt = "SELECT COUNT(\*) FROM customersinformation WHERE Last\_Name LIKE %s"  
 c.execute(select\_stmt, (pattern,))  
 num = c.fetchone()[0]  
 uid = first\_p.lower() + str(num + 1) + fi + mi  
 print(uid)  
 username = uid  
 # setting default password for the customer in store  
 default\_password = "pizzapassword123"  
 hashed\_password = hashlib.sha256(default\_password.encode()).hexdigest()  
 c.execute(  
 "INSERT INTO CustomersInformation (Customer\_ID, First\_Name, Middle\_Name, Last\_Name, Email\_ID, Phone\_Number,Username,Password) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)",  
 (customer\_id, first\_name, middle\_name, last\_name, email, contact,username, hashed\_password))  
 full\_name = f"{last\_name},{first\_name} {middle\_name} ".strip()  
 # Close the database connection  
 mydb.commit()  
  
 mydb.close()  
  
 # Open the main menu with the customer id and full name  
 order\_window.destroy()  
 self.parent = self.main\_window  
 self.mainmenu(customer\_id, full\_name, self.id)  
  
 ### Calling the MainMenu class  
 def mainmenu(self, customer\_id, full\_name, empid):  
 mainmenu = EmpMainMenu(self.parent, customer\_id, full\_name, empid)  
  
# EMPLOYEE MENU SCREEN  
class EmpMainMenu:  
 def \_\_init\_\_(self, parent, id, name, empid):  
 self.parent = parent  
 self.main\_window = tk.Toplevel(self.parent)  
 self.main\_window.attributes("-fullscreen", True)  
 ## Customer Name And ID  
 self.id = id  
 self.name = name  
 self.empid = empid  
 self.main\_window.title("Employee Main Menu")  
 ## Creating an instances of Cart class to store items added to carts.  
 self.cart = Cart()  
 fframe = tk.Frame(self.main\_window)  
 fframe.pack(anchor=tk.W,pady=100,padx=100)  
  
 self.secondframe = tk.LabelFrame(fframe, text="Veg Regular", font=("Arial", 14, "bold"),  
 relief='flat')  
 self.secondframe.pack(anchor=tk.W)  
 self.thirdframe = tk.LabelFrame(fframe, text="Veg Large", font=("Arial", 14, "bold"), relief='flat')  
 self.thirdframe.pack(anchor=tk.W)  
 self.forthframe = tk.LabelFrame(fframe, text="NonVeg Regular", font=("Arial", 14, "bold"),  
 relief='flat')  
 self.forthframe.pack(anchor=tk.W)  
 self.fifthframe = tk.LabelFrame(fframe, text="NonVeg Large", font=("Arial", 14, "bold"),  
 relief='flat')  
 self.fifthframe.pack(anchor=tk.W)  
 self.firstframe = tk.LabelFrame(fframe, text="Beverages", font=("Arial", 14, "bold"), relief='flat')  
 self.firstframe.pack(anchor=tk.W)  
 self.fffirstframe = tk.LabelFrame(fframe, text="Sides", font=("Arial", 14, "bold"), relief='flat')  
 self.fffirstframe.pack(anchor=tk.W)  
  
 # Creating a cart button with image in close frame  
 closeframe = tk.Frame(self.fffirstframe, relief='flat')  
 closeframe.pack(side="right", padx=200)  
 cart\_items\_Button = tk.Button(closeframe, text="Cart"  
 , command=lambda: self.view\_cart(), font=("default", 18, "bold"),  
 background="#ffebcd")  
 cart\_items\_Button.pack(side="top")  
  
 ##Close Button  
 close\_button = tk.Button(closeframe, text="Close", font=("Ariel", 18, "bold"), background="red",  
 command=self.main\_window.destroy)  
 close\_button.pack()  
  
 ## Loading Pizza data & Beverages Sides  
 # Loading Pizza data in pizza and bevsides class and storing the instances in the below lists  
 self.pizzasV = [] # loading veg regular  
 self.pizzasVL = [] # loading veg large  
 self.pizzasNV = [] # loading non veg regular  
 self.pizzasNVL = [] # loading non veg large  
 self.bev = [] # loading bev  
 self.sides = [] # loading sides  
 self.load\_pizza\_data()  
  
 # Create pizza frames for each pizza  
 for pizza in self.bev:  
 self.create\_bev\_frame(pizza)  
  
 colors = ["#2e8b57", "#2e8b57", "#b22222", "#612D08"]  
 for pizza, color in zip(self.sides, colors):  
 self.create\_sides\_frame(pizza, color)  
  
 for pizza in self.pizzasV:  
 self.create\_pizzasV\_frame(pizza)  
  
 for pizza in self.pizzasVL:  
 self.create\_pizzasVL\_frame(pizza)  
 for pizza in self.pizzasNV:  
 self.create\_pizzasNV\_frame(pizza)  
  
 for pizza in self.pizzasNVL:  
 self.create\_pizzasNVL\_frame(pizza)  
  
 tk.mainloop()  
  
 def create\_bev\_frame(self, pizza):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.firstframe, width=300, height=200, highlightthickness=2,  
 highlightbackground="black")  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20, anchor=tk.W)  
 add\_to\_cart\_button = tk.Button(pizza\_frame, text=f"{pizza.name}",  
 command=lambda: self.add\_to\_cart(pizza), font=("Arial", 15, "bold"),  
 bg="#9bddff")  
 add\_to\_cart\_button.pack(anchor=tk.W)  
  
 def create\_sides\_frame(self, pizza, color):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.fffirstframe, width=300, height=200, highlightthickness=2,  
 highlightbackground="black")  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20, anchor=tk.W)  
 add\_to\_cart\_button = tk.Button(pizza\_frame, text=f"{pizza.name}",  
 command=lambda: self.add\_to\_cart(pizza), font=("Arial", 15, "bold"), bg=color)  
 add\_to\_cart\_button.pack(anchor=tk.W)  
  
 def create\_pizzasV\_frame(self, pizza):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.secondframe, width=300, height=200, highlightthickness=2,  
 highlightbackground="black")  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20, anchor=tk.W)  
 add\_to\_cart\_button = tk.Button(pizza\_frame, text=f"{pizza.name} {pizza.size}",  
 command=lambda: self.add\_to\_cart(pizza), font=("Arial", 15, "bold"),  
 bg="#2e8b57")  
 add\_to\_cart\_button.pack(anchor=tk.W)  
  
 def create\_pizzasVL\_frame(self, pizza):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.thirdframe, width=300, height=200, highlightthickness=2,  
 highlightbackground="black")  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20, anchor=tk.W)  
 add\_to\_cart\_button = tk.Button(pizza\_frame, text=f"{pizza.name} {pizza.size}",  
 command=lambda: self.add\_to\_cart(pizza), font=("Arial", 15, "bold"),  
 bg="#2e8b57")  
 add\_to\_cart\_button.pack(anchor=tk.W)  
  
 def create\_pizzasNV\_frame(self, pizza):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.forthframe, width=300, height=200, highlightthickness=2,  
 highlightbackground="black")  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20, anchor=tk.W)  
 add\_to\_cart\_button = tk.Button(pizza\_frame, text=f"{pizza.name} {pizza.size}",  
 command=lambda: self.add\_to\_cart(pizza), font=("Arial", 15, "bold"),  
 bg="#b22222")  
 add\_to\_cart\_button.pack(anchor=tk.W)  
  
 def create\_pizzasNVL\_frame(self, pizza):  
 # Create a frame for the pizza  
 pizza\_frame = tk.Frame(self.fifthframe, width=300, height=200, highlightthickness=2,  
 highlightbackground="black")  
 pizza\_frame.pack(side=tk.LEFT, padx=20, pady=20, anchor=tk.W)  
 add\_to\_cart\_button = tk.Button(pizza\_frame, text=f"{pizza.name} {pizza.size}",  
 command=lambda: self.add\_to\_cart(pizza), font=("Arial", 15, "bold"),  
 bg="#b22222")  
 add\_to\_cart\_button.pack(anchor=tk.W)  
  
 def add\_to\_cart(self, pizza):  
 pizza\_price = pizza.price  
 # Create a new Pizza object with the selected size and price  
 selected\_pizza = Pizza(pizza.id, pizza.name, pizza\_price, pizza.type, pizza.size)  
 selected\_pizza.quantity.set(pizza.quantity.get())  
 # Check if the pizza already exists in the cart  
 for item in self.cart.items:  
 if item.id == selected\_pizza.id:  
 # Update the quantity of the existing pizza item  
 item.quantity.set(item.quantity.get() + selected\_pizza.quantity.get())  
 break  
 else:  
 # Add the pizza copy to the cart  
 self.cart.add\_item(selected\_pizza)  
  
 def get\_cart\_total(self):  
 return self.cart.get\_total()  
  
 def checkout(self):  
 self.cart\_window.destroy()  
 checkout\_window = EmpCheckout(self.main\_window,self.cart, self.id, self.empid)  
  
 ## Loading the data of the menu itesm from the database  
 def load\_pizza\_data(self):  
 # Connect to the database  
 # Connect to the database  
 mydb = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="pizza"  
 )  
 # Perform database operations  
 c = mydb.cursor()  
 # Retrieve the pizza data from the database Vege-R  
 c.execute("SELECT \* FROM Items WHERE item\_type = %s AND item\_size = %s", ("Vegetarian", "Regular"))  
 rows = c.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 pizza = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.pizzasV.append(pizza)  
  
 # Retrieve the pizza data from the database Veg-L  
 c.execute("SELECT \* FROM Items WHERE item\_type = %s AND item\_size = %s", ("Vegetarian", "Large"))  
 rows = c.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 pizza = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.pizzasVL.append(pizza)  
  
 # Retrieve the pizza data from the database NON Veg-R  
 c.execute("SELECT \* FROM Items WHERE item\_type = %s AND item\_size = %s", ("NonVegetarian", "Regular"))  
 rows = c.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 pizza = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.pizzasNV.append(pizza)  
  
 # Retrieve the pizza data from the database NON Veg-L  
 c.execute("SELECT \* FROM Items WHERE item\_type = %s AND item\_size = %s", ("NonVegetarian", "Large"))  
 rows = c.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 pizza = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.pizzasNVL.append(pizza)  
  
 # Retrieve the pizza data from the database BevSide  
 c.execute(  
 "SELECT item\_id, item\_name, item\_price, item\_type, IFNULL(CONCAT(' ', item\_size), '') FROM Items WHERE item\_type = %s",  
 ("Beverages",))  
 rows = c.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 bev = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.bev.append(bev)  
 # Retrieve the pizza data from the database Side  
 c.execute(  
 "SELECT item\_id, item\_name, item\_price, item\_type, IFNULL(CONCAT(' ', item\_size), '') FROM Items WHERE item\_type = %s",  
 ("Sides",))  
 rows = c.fetchall()  
 # Create Pizza objects for each row in the database  
 for row in rows:  
 side = Pizza(row[0], row[1], row[2], row[3], row[4])  
 self.sides.append(side)  
 # Close the database connection  
 mydb.close()  
  
 def view\_cart(self):  
 # Create a new window for the cart  
 self.cart\_window = tk.Toplevel(self.main\_window)  
 self.cart\_window.geometry("700x600")  
 self.cart\_window.title("View Cart")  
 self.cart\_window.configure(background="#3e4c4f")  
 self.cart\_window.resizable(width=False, height=False)  
  
 frame1 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame1.pack()  
 # Create a canvas to hold the cart items frame and add a vertical scrollbar  
 cart\_canvas = tk.Canvas(self.cart\_window, bg="#3e4c4f", bd=0, highlightthickness=0)  
 cart\_scrollbar = tk.Scrollbar(cart\_canvas, orient="vertical", command=cart\_canvas.yview)  
 cart\_canvas.configure(yscrollcommand=cart\_scrollbar.set)  
 cart\_canvas.pack(fill="both", expand=True)  
 cart\_scrollbar.pack(side="right", fill="y")  
  
 frame3 = tk.Frame(self.cart\_window, background="#3e4c4f")  
 frame3.pack()  
  
 # Add a label for the cart items  
 cart\_items\_label = tk.Label(frame1, text="Cart Items", font=("Ariel", 25, "bold"), bg="#3e4c4f", fg="white")  
 cart\_items\_label.pack()  
  
 # Add a frame for the cart items  
 # Create a frame to hold the cart items and add it to the canvas  
 cart\_items\_frame = tk.Frame(cart\_canvas, bg="#3e4c4f")  
 cart\_canvas.create\_window((0, 0), window=cart\_items\_frame, anchor="nw")  
 self.cart\_items\_frame = cart\_items\_frame  
 # Add a label for the cart total  
 self.cart\_total\_label = tk.Label(frame3, text="Total: ${:.2f}".format(self.get\_cart\_total()),  
 font=("Ariel", 20, "bold"), bg="#3e4c4f", fg="white")  
 self.cart\_total\_label.pack()  
  
 # Add a "Checkout" button  
 checkout\_button = tk.Button(frame3, text="Checkout", command=self.checkout, width=15, bg="green",  
 fg="white", font=("Arial", 20, "bold"))  
 checkout\_button.pack(pady=10)  
  
 ##close BUtton  
 self.close\_button = tk.Button(frame3, text="Close", command=self.cart\_window.destroy, width=8,  
 bg="red", fg="white",  
 font=("Arial", 15, "bold"))  
 self.close\_button.pack(pady=10)  
 # Add a label and Spinbox for each pizza item in the cart  
 for i, pizza in enumerate(self.cart.get\_items()):  
 # Create a frame to hold the product label and Spinbox  
 product\_frame = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame.grid(row=i, column=0, sticky='w', pady=10)  
  
 product\_frame1 = tk.Frame(cart\_items\_frame, bg="#3e4c4f")  
 product\_frame1.grid(row=i, column=2, sticky='w', pady=10)  
  
 pizza.product\_frame = product\_frame  
 pizza.product\_frame1 = product\_frame1  
  
 # Add a label for the product name, type, and size  
 product\_label = tk.Label(product\_frame, text="{} ({} {})".format(pizza.name, pizza.type, pizza.size),  
 font=('Ariel', 15, "bold"), bg="#3e4c4f", fg="white")  
 product\_label.grid(row=0, column=0, sticky='w')  
  
 # Add a Spinbox for the product quantity  
 quantity\_spinbox = tk.Spinbox(product\_frame1, from\_=1, to=10, width=5, textvariable=pizza.quantity,  
 command=lambda: self.update\_cart\_total(), font=('Ariel', 15, "bold"))  
 quantity\_spinbox.grid(row=0, column=0, padx=(10, 0))  
  
 # Add a label for the product price  
 price\_label = tk.Label(product\_frame, text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())),  
 font=('Ariel', 15, "bold"), bg="#3e4c4f", fg="white")  
 price\_label.grid(row=0, column=1, padx=(10, 0))  
  
 # Add a "Remove" button for the pizza item  
 remove\_button = tk.Button(product\_frame1, text="Remove",  
 command=lambda pizza\_item=pizza: self.remove\_from\_cart(pizza\_item),  
 font=('Ariel', 15, "bold"), bg="red")  
 remove\_button.grid(row=0, column=1, padx=(10, 0))  
  
 # Add the price label to the pizza item for later use  
 pizza.price\_label = price\_label  
  
 # Update the canvas to adjust to the size of the cart items frame  
 cart\_items\_frame.update\_idletasks()  
 cart\_canvas.config(scrollregion=cart\_canvas.bbox("all"))  
  
 def update\_cart\_total(self):  
 # Update the price labels for each pizza item in the cart  
 for pizza in self.cart.get\_items():  
 pizza.price\_label.config(text="${:.2f}".format(pizza.price \* int(pizza.quantity.get())))  
  
 # Update the cart total label with the new total  
 self.cart\_total\_label.config(text="Total: ${:.2f}".format(self.get\_cart\_total()))  
  
 def remove\_from\_cart(self, pizza\_item):  
 # Check if the pizza item is in the items list  
 if pizza\_item in self.cart.items:  
 # Remove the pizza item from the cart's items list  
 self.cart.items.remove(pizza\_item)  
 # Update the cart items and total labels  
 self.update\_cart\_total()  
 # Destroy the product frames associated with the pizza item  
 pizza\_item.product\_frame.destroy()  
 pizza\_item.product\_frame1.destroy()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # ee = EmpMenu("EMP1","gajula,anurag")  
 lg = LoginPage()  
 # mg = MainMenu("CUS1","gajula, anurag")