```
import tkinter as tk
from tkinter import messagebox
from tkinter import font
import qrcode
from PIL import Image,ImageTk
import cv2

class RestaurantManagementSystem:
    def __init__(self, root):

        self.root = root
        self.root.title("GrubToGo")
        self.root.configure(bg="#F0F0F0")
        self.custom_font = tk.font.Font(family="Helvetica", size=12)
        self.background_image = tk.PhotoImage(file=r"C:\Users\abgan
\Downloads\valley.gif")
        self.background_label = tk.Label(self.root,
image=self.background_image)
        self.background_label.place(x=0,y=0,relwidth=1, relheight=1)
        self.customer_name = tk.StringVar()
        self.customer_contact = tk.StringVar()

        payment_info = "Your Payment Info"  # Replace with the payment
information
        self.generate_qr_code(payment_info)

        self.root.rowconfigure(0, weight=1)
        self.root.rowconfigure(1, weight=1)
        self.root.rowconfigure(3, weight=1)
        self.root.rowconfigure(4, weight=1)
        self.root.rowconfigure(5, weight=1)
        self.root.rowconfigure(6, weight=1)

        self.root.columnconfigure(0, weight=1)
        self.root.columnconfigure(1, weight=1)
        self.root.columnconfigure(2, weight=1)
        self.root.columnconfigure(3, weight=1)
        self.items = {
            "mac & cheese": "Rs.50",
            "croissant": "Rs.40",
            "noodles": "Rs.40",
            "fried rice": "Rs.60",
            "sandwich": "Rs.50",
            "kathi roll": "Rs.30",
            "samosa": "Rs.20",
            "kachori": "Rs.20",
            "burger": "Rs.40",
            "pizza": "Rs.65",
            "chocolate ice-cream": "Rs.45",
            "tea": "Rs.20",
            "coffee": "Rs.20",
```

```
            "fruit punch": "Rs.30"
        }

        self.orders = {}

        self.gst_percentage = 18

        self.create_gui()

    def create_gui(self):
        background_color = "#f1b2c7"
        label_text_color = "black"  # Dark gray
        button_bg_color = "#ff698f"  # Coral
        button_text_color = "black"
        entry_bg_color = "#ffffff"  # White
        entry_text_color = "#333333"
        details_frame = tk.LabelFrame(self.root, text="Customer
Details", background=background_color,
                                      foreground=label_text_color,
font=self.custom_font)

        details_frame.grid(row=0, column=0, padx=10, pady=2,
sticky="n")

        name_label = tk.Label(details_frame, text="Name:",
background=background_color, foreground=label_text_color,
                              font=self.custom_font)
        name_label.grid(row=0, column=0, padx=5, pady=5, sticky="e")
        name_entry = tk.Entry(details_frame,
textvariable=self.customer_name, font=self.custom_font, bg="#ffffff",
                              fg="#333333")
        name_entry.grid(row=0, column=1, padx=5, pady=5, sticky="w")

        contact_label = tk.Label(details_frame, text="Contact:",
background=background_color,
                                 foreground=label_text_color)
        contact_label.grid(row=1, column=0, padx=5, pady=2, sticky="e")
        contact_entry = tk.Entry(details_frame,
textvariable=self.customer_contact)
        contact_entry.grid(row=1, column=1, padx=5, pady=2, sticky="w")
        contact_entry.configure(validate="key")

contact_entry.configure(validatecommand=(contact_entry.register(self.va
lidate_contact), "%P"))

        menu_frame = tk.LabelFrame(self.root, text="Menu",
background="#f1b2c7")
        menu_frame.grid(row=1, column=0, padx=10, pady=5, sticky="n")

        item_header = tk.Label(menu_frame, text="Items",
background=background_color, foreground=label_text_color)
```

```
        item_header.grid(row=0, column=0, padx=5, pady=10, sticky="w")
        quantity_header = tk.Label(menu_frame, text="Quantity",
background=background_color,
                                   foreground=label_text_color)
        quantity_header.grid(row=0, column=1, padx=5, pady=5,
sticky="w")

        row = 1
        for item, price in self.items.items():
            item_var = tk.IntVar()
            item_label = tk.Label(menu_frame, text=f"{item} -
{self.convert_to_inr(price)}",
                                  background=background_color,
foreground=label_text_color)
            item_label.grid(row=row, column=0, padx=5, pady=5,
sticky="w")

            quantity_entry = tk.Entry(menu_frame, width=5,
background=entry_bg_color, foreground=entry_text_color)
            quantity_entry.grid(row=row, column=1, padx=5, pady=5,
sticky="w")

            self.orders[item] = {"var": item_var, "quantity":
quantity_entry}

            row += 1

        buttons_frame = tk.Frame(self.root,
background=background_color)
        buttons_frame.grid(row=2, column=0, padx=10, pady=10,
sticky="n")

        print_bill_button = tk.Button(buttons_frame, text="Print Bill",
command=self.show_bill_popup,
                                      background=button_bg_color,
foreground=button_text_color)
        print_bill_button.pack(side="left", padx=5)

        clear_selection_button = tk.Button(buttons_frame, text="Clear
Selection", command=self.clear_selection,
                                           background=button_bg_color,
foreground=button_text_color)
        clear_selection_button.pack(side="left", padx=5)



        self.sample_bill_text = tk.Text(self.root, height=15,
background="#f7d3df", foreground=entry_text_color)
        self.sample_bill_text.grid(row=0, column=2, rowspan=2, padx=10,
pady=10, sticky="n")
        self.sample_bill_text.config(width=40)
```

```
        # Update sample bill when quantity or item is selected
        for item, info in self.orders.items():
            info["quantity"].bind("<FocusOut>", lambda event,
item=item: self.update_sample_bill())
            info["quantity"].bind("<Return>", lambda event, item=item:
self.update_sample_bill())
            info["quantity"].bind("<KeyRelease>", lambda event,
item=item: self.update_sample_bill())
            info["var"].trace("w", lambda *args, item=item:
self.update_sample_bill())

    def show_bill_popup(self):
        # Check if customer name is provided
        if not self.customer_name.get().strip():
            messagebox.showwarning("Warning", "Please enter customer
name.")
            return

        selected_items = [item for item, info in self.orders.items() if
info["var"].get() > 0]
        if not selected_items:
            messagebox.showwarning("Warning", "Please select items to
order.")
            return

        bill_details = self.generate_bill(selected_items)
        self.sample_bill_text.delete(1.0, tk.END)
        self.sample_bill_text.insert(tk.END, bill_details)

    def clear_selection(self):
        self.customer_name.set("")
        self.customer_contact.set("")
        for item, info in self.orders.items():
            info["var"].set(0)
            info["quantity"].delete(0, tk.END)

    def update_sample_bill(self):
        selected_items = []
        total_price = 0

        for item, info in self.orders.items():
            quantity_str = info["quantity"].get()
            if quantity_str:
                quantity = int(quantity_str)
                selected_items.append((item, quantity))
                price = int(self.items[item].split("Rs.")[1])  #
Extract the numeric value
                total_price += price * quantity

        gst_amount = (total_price * self.gst_percentage) / 100
```

```
        bill = f"Customer Name: {self.customer_name.get()}\n"
        bill += f"Customer Contact: {self.customer_contact.get()}\n\n"
        bill += "Selected Items:\n"

        for item, quantity in selected_items:
            bill += f"{item} x {quantity} - {self.convert_to_inr(price*
quantity)}\n"

        bill += f"\nTotal Price: {self.convert_to_inr(total_price)}\n"
        bill += f"GST ({self.gst_percentage}%):
{self.convert_to_inr(gst_amount)}\n"
        bill += f"Grand Total: {self.convert_to_inr(total_price +
gst_amount)}"

        self.sample_bill_text.delete("1.0", tk.END)  # Clear previous
contents
        self.sample_bill_text.insert(tk.END, bill)

    def generate_bill(self, selected_items):
        customer_name = self.customer_name.get()
        customer_contact = self.customer_contact.get()

        bill_details = f"Customer Name: {customer_name}\nContact:
{customer_contact}\n\n"

        total_amount = 0
        for item in selected_items:
            quantity = int(self.orders[item]["quantity"].get())
            price_per_item = float(self.items[item].split("Rs.")[1])
            total_amount += quantity * price_per_item
            bill_details += f"{item} - {quantity} x
{self.convert_to_inr(price_per_item)}\n"

        gst_amount = (self.gst_percentage / 100) * total_amount
        total_amount_with_gst = total_amount + gst_amount

        bill_details += f"\nTotal Amount:
{self.convert_to_inr(total_amount)}"
        bill_details += f"\nGST ({self.gst_percentage}%):
{self.convert_to_inr(gst_amount)}"
        bill_details += f"\nTotal Amount (including GST):
{self.convert_to_inr(total_amount_with_gst)}"

        return bill_details

    def convert_to_inr(self, amount):
        try:
            amount = float(amount)
            return f"Rs. {amount:.2f}"
        except ValueError:
```

```
            return amount
    def validate_contact(self, new_text):
        return new_text.isdigit() and len(new_text) ≤ 10


    def generate_qr_code(self, data):
        # Generate QR code
        qr = qrcode.QRCode(
            version=1,
            error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=10,
            border=4,
        )
        qr.add_data(data)
        qr.make(fit=True)
        qr_img = qr.make_image(fill_color="black", back_color="white")

        # Convert the QR code image to Tkinter PhotoImage
        qr_img = qr_img.resize((150, 150))  # Adjust the size as needed
        qr_code_image = ImageTk.PhotoImage(qr_img)

        # Display the QR code in a Label
        qr_code_label = tk.Label(self.root, image=qr_code_image,
background="#f1b2c7")
        qr_code_label.photo = qr_code_image  # To prevent the image
from being garbage collected
        qr_code_label.grid(row=0, column=1, padx=10, pady=10,
sticky="n")

        # Bind a click event to decode the QR code
        qr_code_label.bind("<Button-1>", lambda event, data=data:
self.decode_qr_code(data))

    def decode_qr_code(self, expected_data):
        # Capture a frame from the camera (you may need to adjust the
camera index)
        cap = cv2.VideoCapture(0)
        _, frame = cap.read()

        # Decode the QR code using OpenCV
        decoded_objects = cv2.QRCodeDetector().detect(frame)

        # Release the camera capture
        cap.release()

        if decoded_objects:
            decoded_data = decoded_objects[0][0]

            # Check if the decoded data matches the expected data
            if decoded_data == expected_data:
                messagebox.showinfo("Payment Successful", "Payment has
```

```
been successfully received.")
            else:
                messagebox.showerror("Error", "Invalid QR code. Payment
unsuccessful.")
        else:
            messagebox.showerror("Error", "Unable to detect QR code.
Payment unsuccessful.")

if __name__ == "__main__":
    root = tk.Tk()
    app = RestaurantManagementSystem(root)
```