**ONLINE FOOD ORDERDING SYSTEM**

A MINI PROJECT REPORT

**SUBMITTED BY**

**ROSHINI D** **220701228**

**RETHINAATH S** **220701222**

In partial fulfilment for the award of the degree of BACHELOR OF ENGINEERING

IN COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS) THANDALAM

CHENNAI-602105

2023 – 2024

Bonafide Certificate

Certified that this project report “**ONLINE FOOD ORDERDING SYSTEM**” is a bona fide work of “**ROSHINI D (220701228), RETHINAATH S (220701222)**” who carried out the project work under my supervision.

Submitted for the Practical Examination held on

|  |  |
| --- | --- |
| **SIGNATURE**  **Dr . R. Sabitha**  **Professor and II Year Academic Head**  **Science and Engineering, Rajalakshmi Engineering College,(Autonomous)**  **Chennai – 602 105** | **Signature**  **Mrs. K. Maheshmeena Assistant Professor (SG) Computer Science and Engineering**  **Rajalakshmi Engineering College(Autonomous)**  **Chennai – 602 105** |

|  |  |
| --- | --- |
| **Internal Examiner** | **External Examiner** |

**Abstract**

The ever-growing demand for convenience has fueled the popularity of online food ordering systems. This project proposes the development of a comprehensive online food ordering system leveraging a relational database management system to manage customer data, restaurant information, and order details. The system will cater to both customers seeking a convenient way to order food and restaurants aiming to expand their reach through an online platform.

**CHAPTER1: INTRODUCTION**

### INTRODUCTION

Online Food Ordering Systems (OFOS) are revolutionizing the food industry. By offering user-friendly interfaces and real-time order tracking, OFOS enhance the customer experience. For restaurants, OFOS streamline order management, menu control, and provide valuable customer insights. This translates to increased sales, reduced costs, and happier customers for both restaurants and their patrons.

### OBJECTIVE

OFOS (Online Food Ordering Systems) put convenience and efficiency at the forefront. Customers enjoy a smooth experience with easy browsing, ordering, and tracking. For restaurants, OFOS automates tasks, boosts sales, and offers valuable customer insights. This centralized system simplifies restaurant management and provides tiered access for users. Diners and restaurants alike benefit from a streamlined food ordering experience.

### MODULES

* + - LOGIN
    - VIEW AND SELECT RESTAURANT
    - VIEW MENU ITEMS
    - ADD ITEMS TO CART
    - VIEW CART DETAILS
    - VIEW BILL DETAILS

**LOGIN**

Purpose:

* To get basic details from the user

### Functionality:

### Error Handling: Provides feedback for incorrect login attempts.

### Authentication: Checks whether all fields are filled with appropriate data.

**ORDERING FOOD**

Purpose:

* The place where actual ordering starts to take place

### Functionality:

### View and Select Restaurant: Here, the restaurants are displayed with their images and user can select their restaurant of choice.

### View Menu Items: It displays the menu for the selected restaurant.

### Add Items to Cart: The items can then be selected according to quantity and added to cart.

### View Cart Details: The details of the cart involving the items and their respective price and quantity.

### View Bill Details: The details of the bill which contains the total amount including taxes and platform charges.

# CHAPTER 2: SURVEY OF TECHNOLOGY

### SOFTWARE DESCRIPTION

**Visual Studio Code:**

### VS Code with Python and Streamlit offers a powerful development environment for OFOS. VS Code's extensions enhance coding with formatting, debugging, testing, and collaboration features, while Streamlit simplifies UI creation directly in Python. This combination streamlines development with intelligent code completion, debugging tools, version control, and the ability to rapidly build user-friendly interfaces with interactive elements.

### PROGRAMMING LANGUAGES

#### Python:

#### Python powers the logic of online food ordering system: order processing, database interactions, and calculations. Streamlit, a Python library, simplifies UI creation within code. It offers basic elements like buttons, menus, and order details. This combo is ideal for rapid development and a clean codebase, making it suitable for building a basic OFOS UI or prototyping. However, complex UIs or highly interactive features might require dedicated front-end frameworks.

#### **MySQL:**

### MySQL serves as the heart of online food ordering system, storing crucial data like menus, user accounts, order details, and payment records. This secure and scalable database efficiently handles large datasets, allowing system to grow alongside user base and order volume. MySQL's strength lies in its ability to establish relationships between different data tables. This lets us connect users with their orders, items with their corresponding restaurants, and so on. By leveraging SQL (Structured Query Language), we can efficiently retrieve, update, and manipulate data within the database, ensuring the smooth operation of online food ordering system.

### FRAMEWORKS / MODULES USED

### MODULE USED IN PYTHON

#### Streamlit

#### Streamlit acts as a powerful Python library that simplifies the creation of user interfaces (UIs) directly within your Python code. This eliminates the need for separate front-end development tools, streamlining the development process. Streamlit empowers you to build essential UI elements like buttons, menus, and order displays directly within your Python codebase. This approach fosters a clean and maintainable project structure, particularly valuable during rapid prototyping stages or when building basic UIs.

# CHAPTER 3: REQUIREMENTS AND ANALYSIS

### REQUIREMENT SPECIFICATIONS

##### User Requirements:

# The OFOS prioritize an intuitive and user-friendly interface that allows customers to navigate menus, place orders, track deliveries, and manage their accounts with ease.

# The OFOS excel at managing the entire order lifecycle, from customer browsing and order placement.

##### System Requirements:

# A web browser supporting streamlit.

### HARDWARE SPECIFICATIONS

##### Software Requirements:

* + - Frontend - Streamlit with Python
    - Backend – MySQL with Python

##### System Requirements:

* + - Operating Systems: Windows 7 or later, macOS 10.12 or later, Ubuntu 16.04 or later
    - Web browser: Google Chrome version 88 or later, Mozilla Firefox version 85 or later, Safari version 14 or later, Microsoft Edge version 88 or later
    - Internet Connections: Stable broadband internet connection
    - Hardware: Desktop with at least 2GB RAM and dual-core processor, laptop with similar specifications.
      * 64-bit OS to utilize the full capabilities of the hardware
      * 1920 X 1080 resolution monitor for optimal display of application

### FUNCTIONAL/NON-FUNCTIONAL REQUIREMENTS

**Customer Management:** Customers can conveniently submit their details for easy order processing.

**Restaurant Selection:** A user-friendly interface will allow customers to select their desired restaurant.

**Detailed Menu Display:** Restaurants will have the flexibility to showcase their menus with clear descriptions, enticing images, and accurate pricing information.

**Dynamic Cart Management:** Customers can effortlessly add and remove menu items from their cart, adjust quantities, and view the real-time total cost reflecting chosen items and their prices.

**Seamless Order Placement:** A streamlined process will enable customers to confirm their orders, with the system automatically transferring cart selections to the order database while generating a detailed bill for the chosen items.

## ) ER DIAGRAM

## 

# CHAPTER 4: Program Code

#main.py

from connection import conn

import MySQLdb

import streamlit as st

from customer\_details import customer\_details

from choose\_restaurant import choose\_restaurant

from display\_menu\_items import display\_menu\_items

from display\_cart\_details import display\_cart\_details

from bill import display\_bill

def insert\_customer(connection, customer\_name, contact\_number, address, email):

try:

cursor = connection.cursor()

query = "INSERT INTO customer (customer\_name, contact\_number, address, email) VALUES (%s, %s, %s, %s)"

cursor.execute(query, (customer\_name, contact\_number, address, email))

customer\_id = cursor.lastrowid

query\_cart = "INSERT INTO cart\_info (customer\_id, total\_price) VALUES (%s, %s)"

cursor.execute(query\_cart, (customer\_id, 0))

cart\_id = cursor.lastrowid

st.session\_state['cart\_id'] = cart\_id

connection.commit()

st.success("Customer inserted successfully")

except MySQLdb.Error as e:

st.error(f"Error: '{e}'")

def get\_restaurants():

restaurants = []

try:

if conn.is\_connected():

cursor = conn.cursor()

cursor.execute("SELECT restaurant\_name, images, restaurant\_id FROM restaurants")

rows = cursor.fetchall()

restaurants = [(row[0], row[1], row[2]) for row in rows]

except MySQLdb.Error as e:

st.error(f"Error retrieving restaurant names: {e}")

return restaurants

if conn:

if 'page' not in st.session\_state:

st.session\_state['page'] = 'customer\_details'

if st.session\_state['page'] == 'customer\_details':

customer\_details(conn, insert\_customer)

elif st.session\_state['page'] == 'choose\_restaurant':

restaurants = get\_restaurants()

selected\_restaurant = choose\_restaurant(conn, restaurants,st.session\_state['cart\_id'])

if selected\_restaurant:

print(selected\_restaurant)

st.session\_state['selected\_restaurant'] = selected\_restaurant

st.session\_state['page'] = 'display\_menu\_items'

elif st.session\_state['page'] == 'display\_menu\_items':

display\_menu\_items(conn, st.session\_state['selected\_restaurant'],st.session\_state['cart\_id'])

elif st.session\_state['page'] == 'display\_cart\_details':

display\_cart\_details(conn,st.session\_state['cart\_id'])

elif st.session\_state['page'] == 'bill':

display\_bill(conn,st.session\_state['cart\_id'])

#connection.py

import mysql.connector

import MySQLdb

def create\_connection():

connection = None

try:

connection = mysql.connector.connect(

host="localhost",

user="root",

password="Rethinaath123",

database="sql12707743"

)

if connection.is\_connected():

print("Connection to MySQL DB successful")

except MySQLdb.Error as e:

print(f"The error '{e}' occurred")

return connection

conn = create\_connection()

#choose\_restaurant.py

import streamlit as st

import os

import MySQLdb

from streamlit\_option\_menu import option\_menu

def update\_cart\_info(connection, restaurant\_id, cart\_id):

try:

print("called")

cursor = connection.cursor()

print(cart\_id)

query = "UPDATE cart\_info SET restaurant\_id = %s WHERE cart\_id = %s"

cursor.execute(query, (restaurant\_id, cart\_id))

connection.commit()

st.success("Cart info updated successfully")

except MySQLdb.Error as e:

st.error(f"Error: '{e}'")

def choose\_restaurant(connection, restaurants, cart\_id):

with st.sidebar:

selected = option\_menu("Restaurants", ["La Trattoria", "Sushi World","Taco Fiesta","Burger Haven","Dosa Delight"], icons=['house','house','house','house','house'], menu\_icon="houses", default\_index=0)

st.title(selected)

image\_path = None

selected\_restaurant\_id = None

for row in restaurants:

if row[0] == selected:

selected\_restaurant\_id = row[2]

image\_path = row[1]

break

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

st.image(image\_path, caption=f"Image of {selected}", use\_column\_width=True)

else:

st.error(f"Image not found for {selected}")

if st.button("View Menu Items"):

st.session\_state['selected\_restaurant'] = selected

st.session\_state['selected\_restaurant\_id'] = selected\_restaurant\_id

st.session\_state['page'] = 'display\_menu\_items'

update\_cart\_info(connection, selected\_restaurant\_id, cart\_id)

st.rerun()

#customer\_details.py

import streamlit as st

def customer\_details(conn, insert\_customer):

st.title("Please Enter Your Details")

customer\_name = st.text\_input("Name")

contact\_number = st.text\_input("Number")

email = st.text\_input("Email")

address = st.text\_input("Address")

if st.button("Submit"):

if not customer\_name or not contact\_number or not email:

st.error("Please fill in all fields.")

else:

insert\_customer(conn, customer\_name, contact\_number, email, address)

st.session\_state['page'] = 'choose\_restaurant'

st.rerun()

#display\_menu\_items.py

import MySQLdb

import streamlit as st

from update\_cart import update\_cart

from delete\_item\_from\_cart import delete\_item\_from\_cart

def insert\_cart(connection, cart\_id, item\_id, quantity, initial\_price):

try:

cursor = connection.cursor()

total\_price = initial\_price \* quantity

query = "INSERT INTO cart\_items (cart\_id, item\_id, quantity, total\_price) VALUES (%s, %s, %s, %s)"

cursor.execute(query, (cart\_id, item\_id, quantity, total\_price))

connection.commit()

st.success("Cart inserted successfully")

except MySQLdb.Error as e:

st.error(f"Error inserting cart: {e}")

def display\_menu\_items(connection, selected\_restaurant, cart\_id):

col1, col2, col3 = st.columns([8, 20, 10])

with col1:

if st.button("Back"):

st.session\_state['page'] = 'choose\_restaurant'

st.rerun()

with col3:

if st.button("Cart"):

st.session\_state['page'] = 'display\_cart\_details'

st.rerun()

try:

cursor = connection.cursor()

query = "SELECT restaurant\_id FROM restaurants WHERE restaurant\_name = %s"

cursor.execute(query, (selected\_restaurant,))

result = cursor.fetchone()

if result:

restaurant\_id = result[0]

cursor.execute("SELECT item\_id, item\_name, price, description, is\_veg FROM menu\_items WHERE restaurant\_id = %s", (restaurant\_id,))

menu\_items = cursor.fetchall()

if 'cart' not in st.session\_state:

st.session\_state['cart'] = []

st.title(f"Menu Items for {selected\_restaurant}")

if menu\_items:

for item in menu\_items:

item\_id, item\_name, item\_price, item\_des, item\_veg = item

container = st.container(height=250)

with container:

col1, col2, col3 = st.columns([30, 14, 12])

with col1:

st.subheader(f"{item\_name}", divider='rainbow')

with col3:

st.subheader(f"Rs.{item\_price}", divider='rainbow')

st.write(f'{item\_des}')

if item\_veg == 1:

st.caption(':green\_heart:')

else:

st.caption(":heart:")

quantity = st.number\_input(f"Quantity for {item\_name}:", min\_value=0, step=1, key=item\_name)

if quantity > 0:

item\_in\_cart = next((i for i in st.session\_state['cart'] if i['item\_id'] == item\_id), None)

if item\_in\_cart:

if item\_in\_cart['quantity'] != quantity:

item\_in\_cart['quantity'] = quantity

update\_cart(connection, cart\_id, item\_id, quantity)

else:

st.session\_state['cart'].append({

'item\_id': item\_id,

'item\_name': item\_name,

'price': item\_price,

'quantity': quantity

})

insert\_cart(connection, cart\_id, item\_id, quantity, item\_price)

elif quantity == 0:

cursor.execute("SELECT cart\_item\_id FROM cart\_items WHERE cart\_id = %s AND item\_id = %s", (cart\_id, item\_id))

result = cursor.fetchone()

if result:

cart\_item\_id = result[0]

delete\_item\_from\_cart(connection, cart\_item\_id)

st.session\_state['cart'] = [i for i in st.session\_state['cart'] if i['item\_id'] != item\_id]

else:

st.write("No menu items found for this restaurant.")

else:

st.write("Restaurant not found.")

except MySQLdb.Error as e:

st.error(f"Error retrieving menu items: {e}")

#display\_cart\_details.py

import streamlit as st

import MySQLdb

from update\_cart import update\_cart

from delete\_item\_from\_cart import delete\_item\_from\_cart

def insert\_bill(conn, cart\_id):

try:

cursor = conn.cursor()

select\_query = "SELECT cart\_id, customer\_id, total\_price FROM cart\_info WHERE cart\_id = %s"

cursor.execute(select\_query, (cart\_id,))

result = cursor.fetchone()

if result:

cart\_id, customer\_id, total\_price = result

insert\_query = "INSERT INTO bill (cart\_id, customer\_id, amount) VALUES (%s, %s, %s)"

cursor.execute(insert\_query, (cart\_id, customer\_id, total\_price))

bill\_id = cursor.lastrowid

cursor.callproc('calculate\_gst', (total\_price, bill\_id))

cursor.callproc('calculate\_service\_charge', (total\_price, bill\_id))

query = "SELECT amount, cgst, sgst, service\_charge from bill where bill\_id = %s"

cursor.execute(query, (bill\_id,))

calc = cursor.fetchone()

amount, cgst, sgst, service\_charge = calc

cursor.callproc('calculate\_final\_amount', (amount, cgst, sgst, service\_charge, bill\_id))

cursor.callproc('calculate\_total\_quantity', [cart\_id])

conn.commit()

else:

st.error("Cart not found.")

except MySQLdb.Error as e:

st.error(f"Error: '{e}'")

def display\_cart\_details(connection, cart\_id):

try:

cursor = connection.cursor()

if cart\_id is None:

st.error("Cart ID not found in session state.")

return

query = "SELECT item\_id, quantity, total\_price FROM cart\_items WHERE cart\_id = %s"

cursor.execute(query, (cart\_id,))

cart\_items = cursor.fetchall()

if st.button("Back"):

st.session\_state['page'] = 'display\_menu\_items'

st.rerun()

st.title("Cart Details")

if cart\_items:

if 'cart\_quantities' not in st.session\_state:

st.session\_state.cart\_quantities = {}

st.session\_state.cart = []

for item in cart\_items:

item\_id, quantity, total\_price\_per\_item = item

query = "SELECT item\_name, price FROM menu\_items WHERE item\_id = %s"

cursor.execute(query, (item\_id,))

item\_name\_result = cursor.fetchone()

if item\_name\_result:

item\_name, item\_price = item\_name\_result

else:

item\_name = "Unknown"

item\_price = 0

if item\_id not in st.session\_state.cart\_quantities:

st.session\_state.cart\_quantities[item\_id] = quantity

container = st.container(height=280)

with container:

st.subheader(f"{item\_name}",divider="rainbow")

current\_quantity = st.number\_input(

f"Quantity for {item\_name}:",

min\_value=0,

step=1,

key=f"quantity\_{item\_id}",

value=st.session\_state.cart\_quantities[item\_id]

)

st.session\_state.cart\_quantities[item\_id] = current\_quantity

if current\_quantity > 0:

item\_in\_cart = next((i for i in st.session\_state['cart'] if i['item\_id'] == item\_id), None)

if item\_in\_cart:

if item\_in\_cart['quantity'] != current\_quantity:

item\_in\_cart['quantity'] = current\_quantity

update\_cart(connection, cart\_id, item\_id, current\_quantity)

else:

st.session\_state['cart'].append({

'item\_id': item\_id,

'item\_name': item\_name,

'price': item\_price,

'quantity': current\_quantity

})

elif current\_quantity == 0:

cursor.execute("SELECT cart\_item\_id FROM cart\_items WHERE cart\_id = %s AND item\_id = %s",

(cart\_id, item\_id))

result = cursor.fetchone()

if result:

cart\_item\_id = result[0]

delete\_item\_from\_cart(connection, cart\_item\_id)

st.session\_state['cart'] = [i for i in st.session\_state['cart'] if i['item\_id'] != item\_id]

cursor.execute("SELECT total\_price FROM cart\_items WHERE cart\_id = %s AND item\_id = %s", (cart\_id, item\_id))

total\_price\_result = cursor.fetchone()

if total\_price\_result:

updated\_total\_price\_per\_item = total\_price\_result[0]

st.subheader(f"Price: {updated\_total\_price\_per\_item}")

if st.button(f"Delete", key=f"delete\_{item\_id}"):

cursor.execute("SELECT cart\_item\_id FROM cart\_items WHERE cart\_id = %s AND item\_id = %s",

(cart\_id, item\_id))

result = cursor.fetchone()

if result:

cart\_item\_id = result[0]

delete\_item\_from\_cart(connection, cart\_item\_id)

st.session\_state['cart'] = [i for i in st.session\_state['cart'] if i['item\_id'] != item\_id]

st.experimental\_rerun()

query = "SELECT total\_price FROM cart\_info WHERE cart\_id = %s"

cursor.execute(query, (cart\_id,))

total\_price\_result = cursor.fetchone()

if total\_price\_result:

total\_price = total\_price\_result[0]

st.header(f"Total Price: {total\_price}")

if st.button("Place Order"):

print(1)

insert\_bill(connection,cart\_id)

st.session\_state['page'] = 'bill'

st.rerun()

else:

st.write("Your cart is empty.")

except MySQLdb.Error as e:

st.error(f"Error retrieving cart details: {e}")

#update\_cart.py

import MySQLdb

import streamlit as st

def update\_cart(connection, cart\_id, item\_id, quantity):

try:

cursor = connection.cursor()

query = "UPDATE cart\_items SET quantity = %s WHERE cart\_id = %s AND item\_id = %s"

cursor.execute(query, (quantity, cart\_id, item\_id))

connection.commit()

except MySQLdb.Error as e:

st.error(f"Error updating cart: {e}")

#delete\_item\_from\_cart.py

import MySQLdb

import streamlit as st

def delete\_item\_from\_cart(connection, cart\_item\_id):

try:

cursor = connection.cursor()

query = "DELETE FROM cart\_items WHERE cart\_item\_id = %s"

cursor.execute(query, (cart\_item\_id,))

connection.commit()

except MySQLdb.Error as e:

st.error(f"Error deleting item from cart: {e}")

#bill.py

import MySQLdb

import streamlit as st

import pandas as pd

from delete\_from\_bill import delete\_from\_bill

def display\_bill(connection, cart\_id):

try:

cursor = connection.cursor()

st.title("Bill Details")

name\_lt = []

qty\_lt = []

price\_lt = []

query = "SELECT item\_id, quantity, total\_price FROM cart\_items WHERE cart\_id = %s"

cursor.execute(query, (cart\_id,))

cart\_items = cursor.fetchall()

for item in cart\_items:

query = "SELECT item\_name FROM menu\_items WHERE item\_id = %s"

cursor.execute(query, (item[0],))

name = cursor.fetchone()

if name:

item\_name, = name

name\_lt.append(item\_name)

qty\_lt.append(item[1])

price\_lt.append(item[2])

table = {

"Item Name":name\_lt,

"Quantity":qty\_lt,

"Total Price":price\_lt

}

df = pd.DataFrame(table)

st.dataframe(df,hide\_index=True)

query = """

SELECT bill\_id, cart\_id, final\_amount, cgst, sgst, service\_charge, total\_quantity

FROM bill

WHERE cart\_id = %s

"""

cursor.execute(query, (cart\_id,))

bills = cursor.fetchall()

if bills:

for bill in bills:

container = st.container(height=490)

with container:

col1, col2, col3 = st.columns([30, 20, 12])

with col1:

st.subheader(f"Bill ID: {bill[0]}")

with col3:

st.subheader(f"Cart ID: {bill[1]}")

st.divider()

st.subheader(f"Total Quantity: {bill[6]}")

st.subheader(f"Final Amount: {bill[2]}")

with st.expander("Show Extras"):

st.subheader(f"CGST: {bill[3]}")

st.subheader(f"SGST: {bill[4]}")

st.subheader(f"Service Charge: {bill[5]}")

else:

st.write("No bills found.")

if st.button("Back"):

delete\_from\_bill(connection, cart\_id)

st.session\_state['page'] = 'display\_cart\_details'

st.rerun()

except MySQLdb.Error as e:

st.error(f"Error retrieving bill details: {e}")

#delete\_from\_bill.py

import MySQLdb

import streamlit as st

def delete\_from\_bill(connection, cart\_id):

try:

cursor = connection.cursor()

query = "DELETE FROM bill WHERE cart\_id = %s"

cursor.execute(query, (cart\_id,))

connection.commit()

st.success("Bill deleted successfully.")

except MySQLdb.Error as e:

st.error(f"Error deleting bill: {e}")

#data.sql

use sql12707743;

show tables;

desc cart\_items;

create table restaurants (restaurant\_id INT(2) PRIMARY KEY, restaurant\_name varchar(30), cuisine varchar(30), category varchar(30));

INSERT INTO restaurants (restaurant\_id, restaurant\_name, cuisine, category) VALUES (1, 'La Trattoria', 'Italian', 'Casual Dining');

INSERT INTO restaurants (restaurant\_id, restaurant\_name, cuisine, category) VALUES (2, 'Sushi World', 'Japanese', 'Fine Dining');

INSERT INTO restaurants (restaurant\_id, restaurant\_name, cuisine, category) VALUES (3, 'Taco Fiesta', 'Mexican', 'Fast Food');

INSERT INTO restaurants (restaurant\_id, restaurant\_name, cuisine, category) VALUES (4, 'Burger Haven', 'American', 'Fast Food');

INSERT INTO restaurants (restaurant\_id, restaurant\_name, cuisine, category) VALUES (5, 'Dosa Delight', 'South Indian', 'Casual Dining');

CREATE TABLE customer (

Customer\_id INT AUTO\_INCREMENT PRIMARY KEY,

Customer\_name VARCHAR(30),

Contact\_Number VARCHAR(30),

Address VARCHAR(100),

Email VARCHAR(50)

);

CREATE TABLE menu\_items (

item\_id INT PRIMARY KEY,

item\_name VARCHAR(50),

restaurant\_id INT,

Description VARCHAR(255),

Price DECIMAL(10, 2),

is\_veg BOOLEAN,

category VARCHAR(50),

FOREIGN KEY (restaurant\_id) REFERENCES restaurants(restaurant\_id)

);

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

-- Appetizers

(1, 'Bruschetta', 1, 'Toasted bread topped with diced tomatoes, garlic, basil, and olive oil', 595.00, TRUE, 'Appetizer'),

(2, 'Garlic Bread', 1, 'Toasted bread slices rubbed with garlic and butter', 447.00, TRUE, 'Appetizer'),

(3, 'Antipasto Platter', 1, 'Assorted cured meats, cheeses, olives, and roasted vegetables', 1367.50, TRUE, 'Appetizer'),

(4, 'Garlic Knots', 1, 'Soft bread knots brushed with garlic butter and herbs, served with marinara sauce', 525.00, TRUE, 'Appetizer'),

-- Salads

(5, 'Caprese Salad', 1, 'Fresh salad made with tomatoes, mozzarella cheese, basil, olive oil, and balsamic glaze', 787.50, TRUE, 'Salad'),

(6, 'Caesar Salad', 1, 'Classic salad made with romaine lettuce, croutons, Parmesan cheese, and Caesar dressing', 825.00, TRUE, 'Salad'),

(7, 'Greek Salad', 1, 'Fresh salad made with mixed greens, tomatoes, cucumbers, olives, feta cheese, and Greek dressing', 850.00, TRUE, 'Salad'),

(8, 'Fruit Salad', 1, 'Fresh fruit salad with seasonal fruits, honey, and lime dressing', 750.00, TRUE, 'Salad'),

-- Pasta/Risotto

(9, 'Penne Arrabbiata', 1, 'Pasta tossed in a spicy tomato sauce with garlic and chili flakes', 934.00, TRUE, 'Pasta/Risotto'),

(10, 'Fettuccine Alfredo', 1, 'Creamy pasta dish made with fettuccine noodles and Alfredo sauce', 897.00, FALSE, 'Pasta/Risotto'),

(11, 'Mushroom Risotto', 1, 'Creamy risotto rice cooked with mushrooms, onions, white wine, and Parmesan cheese', 975.00, TRUE, 'Pasta/Risotto'),

(12, 'Spinach Risotto', 1, 'Creamy risotto rice cooked with spinach, garlic, onions, and Parmesan cheese', 925.00, TRUE, 'Pasta/Risotto'),

(13, 'Carbonara', 1, 'Pasta dish made with spaghetti, eggs, cheese, pancetta, and black pepper', 950.00, FALSE, 'Pasta/Risotto'),

-- Pizza

(14, 'Margherita Pizza', 1, 'Classic pizza topped with tomato sauce, mozzarella cheese, and fresh basil leaves', 745.00, TRUE, 'Pizza'),

(15, 'Vegetarian Pizza', 1, 'Pizza topped with assorted vegetables and mozzarella cheese', 825.00, TRUE, 'Pizza'),

(16, 'Pepperoni Pizza', 1, 'Pizza topped with pepperoni slices and mozzarella cheese', 875.00, FALSE, 'Pizza'),

(17, 'BBQ Chicken Pizza', 1, 'Pizza topped with BBQ sauce, grilled chicken, red onions, and mozzarella cheese', 950.00, FALSE, 'Pizza'),

-- Desserts

(18, 'Tiramisu', 1, 'Classic Italian dessert made with layers of coffee-soaked ladyfingers and mascarpone cheese', 850.00, FALSE, 'Dessert'),

(19, 'Cannoli', 1, 'Sicilian pastry filled with sweet ricotta cheese and chocolate chips', 765.00, FALSE, 'Dessert'),

(20, 'Chocolate Fondant', 1, 'Warm chocolate cake with a molten chocolate center, served with vanilla ice cream', 975.00, FALSE, 'Dessert');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(26, 'Vegetable Tempura', 2, 'Assorted vegetables coated in a light tempura batter and deep-fried until crispy', 750.00, TRUE, 'Agemono and Yakimono'),

(27, 'Agedashi Tofu', 2, 'Deep-fried tofu served in a dashi-based sauce with grated daikon radish and green onions', 550.00, TRUE, 'Agemono and Yakimono'),

(28, 'Chicken Teriyaki', 2, 'Grilled chicken glazed with a sweet and savory teriyaki sauce, served with steamed rice', 900.00, FALSE, 'Agemono and Yakimono'),

(29, 'Beef Negimaki', 2, 'Thinly sliced beef rolled with scallions (green onions), grilled, and served with teriyaki sauce', 1000.00, FALSE, 'Agemono and Yakimono'),

(30, 'Ebi Fry', 2, 'Breaded and deep-fried shrimp, served with tonkatsu sauce and shredded cabbage', 950.00, FALSE, 'Agemono and Yakimono');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(31, 'Vegetable Yakisoba', 2, 'Stir-fried noodles with assorted vegetables, seasoned with yakisoba sauce', 800.00, TRUE, 'Noodles'),

(32, 'Miso Ramen', 2, 'Ramen noodles served in a miso-based broth with tofu, seaweed, and green onions', 850.00, TRUE, 'Noodles'),

(33, 'Chicken Udon', 2, 'Udon noodles served in a hot dashi broth with grilled chicken, vegetables, and green onions', 900.00, FALSE, 'Noodles'),

(34, 'Beef Ramen', 2, 'Ramen noodles served in a rich beef broth with slices of beef, boiled egg, and vegetables', 950.00, FALSE, 'Noodles'),

(35, 'Seafood Yaki Udon', 2, 'Udon noodles stir-fried with assorted seafood and vegetables', 1000.00, FALSE, 'Noodles');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(36, 'Vegetable Donburi', 2, 'Steamed rice bowl topped with assorted vegetables and teriyaki sauce', 750.00, TRUE, 'Donburi'),

(37, 'Tofu Donburi', 2, 'Steamed rice bowl topped with grilled tofu and vegetables in a savory sauce', 800.00, TRUE, 'Donburi'),

(38, 'Chicken Katsu Don', 2, 'Breaded and deep-fried chicken cutlet served over a bowl of steamed rice with egg and onions', 850.00, FALSE, 'Donburi'),

(39, 'Beef Teriyaki Don', 2, 'Grilled slices of beef glazed with teriyaki sauce, served over a bowl of steamed rice with vegetables', 900.00, FALSE, 'Donburi'),

(40, 'Spicy Tuna Don', 2, 'Fresh spicy tuna mixed with spicy mayo and served over a bowl of sushi rice', 1000.00, FALSE, 'Donburi');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(21, 'Ebi Tempura Nigiri', 2, 'Nigiri sushi with tempura-fried shrimp on top of a small bed of sushi rice', 950.00, FALSE, 'Sushi and Sashimi'),

(22, 'Sashimi Platter', 2, 'Assorted slices of fresh raw fish (tuna, salmon, yellowtail, scallop)', 1600.00, FALSE, 'Sushi and Sashimi'),

(23, 'Dragon Maki', 2, 'Maki roll filled with eel (unagi), cucumber, and avocado, topped with thinly sliced avocado and eel sauce', 1050.00, FALSE, 'Sushi and Sashimi'),

(24, 'Salmon Nigiri', 2, 'Nigiri sushi with a slice of fresh salmon on top of a small bed of sushi rice', 700.00, FALSE, 'Sushi and Sashimi'),

(25, 'Avocado Maki', 2, 'Maki roll filled with avocado, wrapped in nori seaweed and sushi rice', 550.00, TRUE, 'Sushi and Sashimi');

select \* from menu\_items;

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(41, 'Nachos', 3, 'Crispy tortilla chips topped with melted cheese, jalapeños, and sour cream', 200.00, TRUE, 'Appetizers'),

(42, 'Guacamole', 3, 'Fresh avocado dip served with tortilla chips', 150.00, TRUE, 'Appetizers'),

(43, 'Chicken Quesadilla', 3, 'Grilled tortilla filled with cheese and chicken', 250.00, FALSE, 'Appetizers'),

(44, 'Stuffed Jalapeños', 3, 'Jalapeños stuffed with cream cheese and deep-fried', 180.00, TRUE, 'Appetizers');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(45, 'Vegetable Tacos', 3, 'Soft tacos filled with grilled vegetables, lettuce, and salsa', 150.00, TRUE, 'Tacos'),

(46, 'Chicken Tacos', 3, 'Soft tacos filled with marinated chicken, lettuce, and salsa', 200.00, FALSE, 'Tacos'),

(47, 'Beef Tacos', 3, 'Soft tacos filled with seasoned ground beef, cheese, and salsa', 250.00, FALSE, 'Tacos'),

(48, 'Fish Tacos', 3, 'Soft tacos filled with battered fish, cabbage slaw, and creamy sauce', 300.00, FALSE, 'Tacos');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(49, 'Vegetable Burrito', 3, 'Flour tortilla filled with rice, beans, grilled vegetables, and cheese', 250.00, TRUE, 'Burritos'),

(50, 'Chicken Burrito', 3, 'Flour tortilla filled with rice, beans, chicken, and cheese', 300.00, FALSE, 'Burritos'),

(51, 'Beef Burrito', 3, 'Flour tortilla filled with rice, beans, beef, and cheese', 350.00, FALSE, 'Burritos'),

(52, 'Pork Burrito', 3, 'Flour tortilla filled with rice, beans, pulled pork, and cheese', 350.00, FALSE, 'Burritos');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(53, 'Churros', 3, 'Fried dough pastries rolled in cinnamon sugar', 150.00, TRUE, 'Desserts'),

(54, 'Tres Leches Cake', 3, 'Moist cake made with three types of milk', 200.00, TRUE, 'Desserts'),

(55, 'Flan', 3, 'Creamy caramel custard', 180.00, TRUE, 'Desserts'),

(56, 'Sopapillas', 3, 'Puffy, fried pastries drizzled with honey', 150.00, TRUE, 'Desserts');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(57, 'Horchata', 3, 'Traditional Mexican rice drink with cinnamon and vanilla', 120.00, TRUE, 'Beverages'),

(58, 'Agua Fresca', 3, 'Refreshing fruit water with natural flavors', 100.00, TRUE, 'Beverages'),

(59, 'Mexican Soda', 3, 'Sweet and fruity carbonated drink', 80.00, TRUE, 'Beverages'),

(60, 'Michelada', 3, 'Spicy beer cocktail with lime juice and assorted sauces', 150.00, TRUE, 'Beverages');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(61, 'The Classic Burger', 4, 'A juicy all-beef patty with lettuce, tomato, onion, pickles, ketchup, and mustard on a toasted sesame seed bun.', 8.99, FALSE, 'Burgers'),

(62, 'Cheeseburger', 4, 'The Classic Burger topped with a slice of melted cheddar cheese.', 9.49, FALSE, 'Burgers'),

(63, 'Bacon Cheeseburger', 4, 'The Cheeseburger loaded with crispy bacon.', 10.49, FALSE, 'Burgers'),

(64, 'BBQ Bacon Burger', 4, 'The Bacon Cheeseburger smothered in our sweet and smoky BBQ sauce.', 10.99, FALSE, 'Burgers'),

(65, 'Mushroom Swiss Burger', 4, 'The Classic Burger topped with sauteed mushrooms and melted Swiss cheese.', 9.99, FALSE, 'Burgers'),

(66, 'Veggie Burger', 4, 'A delicious black bean patty topped with lettuce, tomato, onion, pickles, ketchup, and mustard on a toasted sesame seed bun.', 8.49, TRUE, 'Burgers'),

(67, 'Crispy Chicken Sandwich', 4, 'A juicy, crispy chicken breast filet on a toasted brioche bun with lettuce, tomato, mayo, and pickles.', 8.99, FALSE, 'Sandwiches'),

(68, 'Buffalo Chicken Sandwich', 4, 'Crispy Chicken Sandwich tossed in our spicy buffalo sauce with crumbled blue cheese dressing.', 9.49, FALSE, 'Sandwiches'),

(69, 'Grilled Chicken Sandwich', 4, 'A marinated and grilled chicken breast on a toasted wheat bun with lettuce, tomato, and mayo.', 8.49, FALSE, 'Sandwiches'),

(70, 'Fish Sandwich', 4, 'A golden-fried fish fillet on a toasted brioche bun with tartar sauce and lettuce.', 9.49, FALSE, 'Sandwiches'),

(71, 'French Fries', 4, 'A large portion of crispy golden french fries.', 2.49, TRUE, 'Sides'),

(72, 'Onion Rings', 4, 'Hand-battered onion rings fried to a golden crisp.', 3.49, TRUE, 'Sides'),

(73, 'Mozzarella Sticks', 4, 'Gooey mozzarella cheese sticks served with marinara sauce.', 4.49, TRUE, 'Sides'),

(74, 'Chicken Tenders', 4, 'Crispy golden chicken tenders served with your choice of dipping sauce.', 5.99, FALSE, 'Starters'),

(75, 'Boneless Wings', 4, 'Choose your favorite flavor of boneless wings tossed in our signature sauces.', 6.49, FALSE, 'Starters'),

(76, 'Loaded Nachos', 4, 'Crispy tortilla chips piled high with seasoned ground beef, melted cheese, pico de gallo, and jalapenos.', 7.99, FALSE, 'Starters'),

(77, 'Chocolate Milkshake', 4, 'A rich and creamy chocolate milkshake made with real ice cream.', 3.99, TRUE, 'Drinks'),

(78,' Vanilla Milkshake', 4, 'A classic vanilla milkshake made with real ice cream.', 3.99, TRUE, 'Drinks'),

(79, 'Strawberry Milkshake', 4, 'A refreshing strawberry milkshake made with real ice cream.', 3.99, TRUE, 'Drinks'),

(80, 'Soft Drinks', 4, 'A variety of soft drinks available (specify choice during order).', 2.49, TRUE, 'Drinks');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(81, 'Masala Dosa', 5, 'Thin rice crepe filled with spiced potato masala', 120.00, TRUE, 'Dosas'),

(82, 'Onion Dosa', 5, 'Thin rice crepe filled with chopped onions and spices', 100.00, TRUE, 'Dosas'),

(83, 'Mysore Masala Dosa', 5, 'Thin rice crepe spread with spicy chutney and filled with potato masala', 140.00, TRUE, 'Dosas'),

(84, 'Podi Dosa', 5, 'Thin rice crepe sprinkled with flavorful podi (spice mix)', 130.00, TRUE, 'Dosas');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(85, 'Ghee Roast Dosa', 5, 'Thin rice crepe roasted in ghee (clarified butter) until crispy', 160.00, TRUE, 'Dosas'),

(86, 'Kal Dosa', 5, 'Thick rice pancake cooked until golden brown and crispy', 140.00, TRUE, 'Dosas'),

(87, 'Egg Dosa', 5, 'Thin rice crepe topped with a fluffy omelette and spices', 150.00, FALSE, 'Dosas'),

(88, 'Rava Dosa', 5, 'Thin rice and semolina crepe with a crispy texture', 130.00, TRUE, 'Dosas');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(89, 'Idli Sambar', 5, 'Steamed rice cakes served with sambar and chutneys', 100.00, TRUE, 'Idlis'),

(90, 'Poori Masala', 5, 'Deep-fried Indian bread served with spiced potato masala', 120.00, TRUE, 'Poori'),

(91, 'Rava Kesari', 5, 'Semolina pudding flavored with saffron, cardamom, and nuts', 80.00, TRUE, 'Desserts'),

(92, 'Rava Kichadi', 5, 'Semolina (rava) cooked with vegetables and spices', 90.00, TRUE, 'Breakfast');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(93, 'Vegetable Biryani', 5, 'Fragrant basmati rice cooked with mixed vegetables and aromatic spices', 180.00, TRUE, 'Rices'),

(94, 'Jeera Rice', 5, 'Basmati rice tempered with cumin seeds and cooked to perfection', 100.00, TRUE, 'Rices'),

(95, 'Lemon Rice', 5, 'Rice flavored with lemon juice, tempered with mustard seeds, and garnished with peanuts and curry leaves', 120.00, TRUE, 'Rices'),

(96, 'Coconut Rice', 5, 'Rice cooked with grated coconut, tempered with mustard seeds, and garnished with cashews and curry leaves', 130.00, TRUE, 'Rices');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(97, 'Vegetable Pulao', 5, 'Fragrant basmati rice cooked with mixed vegetables and aromatic spices', 160.00, TRUE, 'Rices'),

(98, 'Paneer Pulao', 5, 'Fragrant basmati rice cooked with paneer (cottage cheese) cubes and aromatic spices', 180.00, TRUE, 'Rices');

INSERT INTO menu\_items (item\_id, item\_name, restaurant\_id, Description, Price, is\_veg, category)

VALUES

(99, 'Masala Chai', 5, 'Indian spiced tea made with milk, tea leaves, and aromatic spices', 50.00, TRUE, 'Beverages'),

(100, 'Filter Coffee', 5, 'South Indian coffee brewed with dark roast coffee beans and served with milk and sugar', 60.00, TRUE, 'Beverages'),

(101, 'Mango Lassi', 5, 'Refreshing yogurt-based drink flavored with ripe mango pulp', 70.00, TRUE, 'Beverages'),

(102, 'Water', 5, 'Bottled drinking water', 20.00, TRUE, 'Beverages');

CREATE TABLE cart\_info (

cart\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT,

restaurant\_id INT,

total\_price DECIMAL(10, 2),

FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id),

FOREIGN KEY (restaurant\_id) REFERENCES restaurants(restaurant\_id)

);

CREATE TABLE cart\_items (

cart\_item\_id INT auto\_increment PRIMARY KEY,

cart\_id INT,

item\_id INT,

quantity INT,

total\_price DECIMAL(10,2),

FOREIGN KEY (cart\_id) REFERENCES cart\_info(cart\_id),

FOREIGN KEY (item\_id) REFERENCES menu\_items(item\_id)

);

desc bill;

CREATE TABLE bill (

bill\_id INT auto\_increment PRIMARY KEY,

cart\_id INT,

customer\_id INT,

amount DECIMAL(10, 2),

cgst DECIMAL(5, 2),

sgst DECIMAL(5, 2),

service\_charge DECIMAL(5, 2),

total\_quantity INT,

final\_amount decimal(10,2),

FOREIGN KEY (cart\_id) REFERENCES cart\_info(cart\_id),

FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id)

);

select \* from customer;

drop table bill;

drop table cart\_items;

drop table cart\_info;

drop table customer;

ALTER TABLE menu\_items

ADD COLUMN images VARCHAR(255);

UPDATE restaurants

SET images = 'images\La\_Trattoria.jpeg'

WHERE restaurant\_id = 1;

UPDATE restaurants

SET images = 'images\Sushi\_World.jpeg'

WHERE restaurant\_id = 2;

UPDATE restaurants

SET images = 'images\Tacos\_Fiesta.jpeg'

WHERE restaurant\_id = 3;

UPDATE restaurants

SET images = 'images\Burger\_Haven.jpeg'

WHERE restaurant\_id = 4;

UPDATE restaurants

SET images = 'images\Dosa Delight.jpeg'

WHERE restaurant\_id = 5;

#procedures.sql

DELIMITER //

CREATE PROCEDURE calculate\_gst(

IN total\_price DECIMAL(10, 2),

IN bill\_id\_q INT

)

BEGIN

DECLARE cgst DECIMAL(5, 2);

DECLARE sgst DECIMAL(5, 2);

SET cgst = total\_price \* 0.025;

SET sgst = total\_price \* 0.025;

UPDATE bill

SET cgst = cgst,

sgst = sgst

WHERE bill\_id = bill\_id\_q;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE calculate\_service\_charge(

IN total\_price DECIMAL(10, 2),

IN bill\_id\_q INT

)

BEGIN

DECLARE service\_charge DECIMAL(5, 2);

SET service\_charge = total\_price \* 0.10;

UPDATE bill

SET service\_charge = service\_charge

WHERE bill\_id = bill\_id\_q;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE calculate\_final\_amount(

IN amount DECIMAL(10, 2),

IN cgst DECIMAL(5, 2),

IN sgst DECIMAL(5, 2),

IN service\_charge DECIMAL(5, 2),

IN bill\_id\_q INT

)

BEGIN

DECLARE final\_amount DECIMAL(10, 2);

select service\_charge;

SET final\_amount = amount + cgst + sgst + service\_charge;

UPDATE bill

SET final\_amount = final\_amount

WHERE bill\_id = bill\_id\_q;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE calculate\_total\_quantity(IN p\_cart\_id INT)

BEGIN

DECLARE total\_quantity INT;

SELECT SUM(quantity) INTO total\_quantity

FROM cart\_items

WHERE cart\_id = p\_cart\_id;

update bill

set total\_quantity = total\_quantity

where cart\_id = p\_cart\_id;

END //

DELIMITER ;

#trigger\_update\_price\_in\_cart\_info.sql

DELIMITER //

CREATE TRIGGER update\_total\_price\_after\_insert

AFTER INSERT ON cart\_items

FOR EACH ROW

BEGIN

DECLARE new\_total\_price DECIMAL(10, 2);

SELECT SUM(total\_price) INTO new\_total\_price

FROM cart\_items

WHERE cart\_id = NEW.cart\_id;

UPDATE cart\_info

SET total\_price = new\_total\_price

WHERE cart\_id = NEW.cart\_id;

END //

CREATE TRIGGER update\_total\_price\_after\_update

AFTER UPDATE ON cart\_items

FOR EACH ROW

BEGIN

DECLARE new\_total\_price DECIMAL(10, 2);

SELECT SUM(total\_price) INTO new\_total\_price

FROM cart\_items

WHERE cart\_id = NEW.cart\_id;

UPDATE cart\_info

SET total\_price = new\_total\_price

WHERE cart\_id = NEW.cart\_id;

END //

CREATE TRIGGER update\_total\_price\_after\_delete

AFTER DELETE ON cart\_items

FOR EACH ROW

BEGIN

DECLARE new\_total\_price DECIMAL(10, 2);

SELECT SUM(total\_price) INTO new\_total\_price

FROM cart\_items

WHERE cart\_id = OLD.cart\_id;

UPDATE cart\_info

SET total\_price = new\_total\_price

WHERE cart\_id = OLD.cart\_id;

END //

DELIMITER ;

#trigger\_update\_price\_in\_cart\_items.sql

use sql12707743;

CREATE TABLE cart\_items\_temp (

cart\_item\_id INT PRIMARY KEY,

total\_price DECIMAL(10,2)

);

DELIMITER //

CREATE TRIGGER update\_total\_price

BEFORE UPDATE ON cart\_items

FOR EACH ROW

BEGIN

DECLARE itemPrice DECIMAL(10,2);

SELECT price INTO itemPrice FROM menu\_items WHERE menu\_items.item\_id = NEW.item\_id;

SET NEW.total\_price = itemPrice \* NEW.quantity;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER after\_update\_total\_price

AFTER INSERT ON cart\_items\_temp

FOR EACH ROW

BEGIN

UPDATE cart\_items

SET total\_price = NEW.total\_price

WHERE cart\_item\_id = NEW.cart\_item\_id;

DELETE FROM cart\_items\_temp WHERE cart\_item\_id = NEW.cart\_item\_id;

END //

DELIMITER ;

show triggers;

# CHAPTER 5: RESULT

Login interface

# 

# Restaurants Dashboard

# 

# Menu Items

# 

# Cart Details

# 

# Bill Details

# 

# CHAPTER – 6: CONCLUSION

# This culmination of development efforts has resulted in a robust Online Food Ordering System (OFOS) platform designed to revolutionize the food delivery landscape. This innovative system empowers restaurants with a centralized hub for managing their online presence. The OFOS features a user-friendly admin panel that streamlines restaurant operations. This web-based interface allows restaurants to effortlessly manage menus, receive and process orders, and track delivery status – all in real-time. Additionally, the OFOS integrates seamlessly with payment gateways, ensuring secure and efficient transactions. Furthermore, the system empowers restaurants to leverage valuable customer data for targeted promotions and marketing campaigns, fostering customer loyalty and boosting sales. This comprehensive OFOS paves the way for a more convenient and efficient food ordering experience for both restaurants and their customers.